An Efficient Dynamic and Distributed Cryptographic Accumulator*

Michael T. Goodrich[1], Roberto Tamassia[2], and Jasminka Hasić[2]

[1] Dept. Information & Computer Science, University of California, Irvine
goodrich@acm.org
[2] Dept. Computer Science, Brown University, Providence, Rhode Island
{rt,jh}@cs.brown.edu

Abstract. We show how to use the RSA one-way accumulator to realize an efficient and dynamic authenticated dictionary, where untrusted directories provide cryptographically verifiable answers to membership queries on a set maintained by a trusted source. Our accumulator-based scheme for authenticated dictionaries supports efficient incremental updates of the underlying set by insertions and deletions of elements. Also, the user can optimally verify in constant time the authenticity of the answer provided by a directory with a simple and practical algorithm. This work has applications to certificate revocation in public key infrastructure and end-to-end integrity of data collections published by third parties on the Internet.

1 Introduction

Modern distributed transactions often operate in an asymmetric computational environment. Typically, client applications are deployed on small devices, such as laptop computers and palm devices, whereas the server side of these applications are often deployed on large-scale multiprocessors. Moreover, several client applications communicate with these powerful server farms over wireless connections or slow modem-speed connections. Thus, distributed applications are facilitated by solutions that involve small amounts of computing and communication on the client side, without overly burdening the more-powerful server side of these same applications. The challenge we address in this paper is how to incorporate added levels of information assurance and security into such applications without significantly increasing the amount of computation and communication that is needed at the client (while at the same time keeping the computations on the servers reasonable).

An information security problem arising in the replication of data to mirror sites is the authentication of the information provided by the sites. Indeed, there are applications where the user may require that data coming from a mirror site be cryptographically validated as being as genuine as they would be had

* Work supported in part by the Dynamic Coalitions Program of the Defense Advanced Research Projects Agency under grant F30602–00–2–0509.

the response come directly from the source. For example, a financial speculator that receives NASDAQ stock quotes from the *Yahoo! Finance* Web site would be well advised to get a proof of the authenticity of the data before making a large trade.

For all data replication applications, and particularly for e-commerce applications in wireless computing, we desire solutions that involve short responses from a mirror site that can be quickly verified with low computational overhead.

Problem Definition. More formally, the problem we address involves three groups of related parties: trusted information sources, untrusted directories, and users. An information *source* defines a finite set S of elements that evolves over time through insertions and deletions of items. *Directories* maintain copies of the set S. Each directory storing S receives time-stamped updates from the source for S together with *update authentication information*, such as signed statements about the update and the current elements of the set. A *user* performs membership queries on the set S of the type "is element e in set S?" but instead of contacting the source for S directly, it queries a directory for S instead. The contacted directory provides the user with a yes/no answer to the query together with *query authentication information*, which yields a proof of the answer assembled by combining statements signed by the source. The user then verifies the proof by relying solely on its trust in the source and the availability of public information about the source that allows to check the source's signature. The data structures used by the source directory to maintain set S, together with the protocol for queries and updates is called an *authenticated dictionary* [25].

The design of an authenticated dictionary should address several goals. These goals include low computational cost, so that the computations performed internally by each entity (source, directory, and user) should be simple and fast, and low communication overhead, so that bandwidth utilization is minimized. Since these goals are particularly important for the user, we say that an authenticated dictionary is *size-oblivious* if the query authentication information size and the verification time do not depend in any way on the number of items in the dictionary. Size-oblivious solutions to the authenticated dictionary problem are ideally suited for wireless e-commerce applications, where user devices have low computational power and low bandwidth. In addition, size-oblivious solutions add an extra level of security, since the size of the dictionary is never revealed to users.

Applications of authenticated dictionaries include third-party data publication on the Internet, certificate revocation, and the authentication of Web services.

Previous and Related Work. Authenticated dictionaries are related to research in distributed computing (e.g., data replication in a network [5, 20]), data structure design (e.g., program checking [7, 28] and memory checking [6, 13]), and cryptography (e.g., incremental cryptography [3]).

Previous additional work on authenticated dictionaries has been conducted primarily in the context of certificate revocation. The traditional method for certificate revocation (e.g., see [18]) is for the CA (source) to sign a statement

consisting of a timestamp plus a hash of the set of all revoked certificates, called *certificate revocation list* (CRL), and periodically send the signed CRL to the directories. This approach is secure, but it is inefficient, for it requires the transmission of the entire set of revoked certificates for both source-to-directory and directory-to-user communication. Thus, this solution is clearly not size-oblivious, and even more recent modifications of this solution, which are based on delta-CRLs [12], are not size-oblivious.

The *hash tree* scheme introduced by Merkle [23, 24] can be used to implement a static authenticated dictionary, which supports the initial construction of the data structure followed by query operations, but not update operations. Other certificate revocation schemes, based on variations of cryptographic hashing, have been recently proposed in [8, 14], but like the static hash tree, these schemes have logarithmic verification time.

Dynamic data structures based on hierarchical hashing that efficiently support also the insertion and deletion of elements are presented in [16, 25]. These data structures have logarithmic query, update and verification time.

The software architecture and implementation of an authenticated dictionary based on the above approach of [16] is described in [17]. An efficient data structure for *persistent authenticated dictionaries*, where user can issue historical queries of the type, "was element e in set S at time t?" is introduced in [1].

A general approach to the design of authenticated data structures, which is based on a hashing scheme that digests a search structure into a single hash value, and its applications to multidimensional range searching are presented in [21]. Using a similar approach, efficient data structures for various fundamental graph problems, such as path and k-connectivity queries for $k \leq 3$, and geometric problems, such as intersection and containment queries, are given in [11]. Both these approaches have logarithmic query and verification time and thus are not size-oblivious.

Our Results. In this paper we present a number of size-oblivious solutions to the authenticated dictionary problem. The general approach we follow here is to abandon the approach of the previous methods cited above that are based on applying one-way hash functions to nodes in a data structure. Instead, we make use of one-way accumulators, as advocated by Benaloh and de Mare [4]. Such an approach is immediately size-oblivious, but there is an additional challenge that has to be overcome to make this approach practical. The computations needed at the source and/or directories in a straightforward implementation of the Benaloh-de Mare scheme are inefficient. Our main contribution, therefore, is a mechanism to make the computations at the source and mirrors efficient.

We present a size-oblivious scheme for authenticated dictionaries, based on one-way accumulators, that it is dynamic and distributed, thus supporting efficient updates and balancing the work between the source and the directories. A first variation of our scheme achieves a complete tradeoff between the cost of updates at the source and of queries at the directories, with updates taking $O(p + \log(n/p))$ time and queries taking $O(n/p)$ time, where n is the size of the dictionary and p is any fixed integer parameter such that $1 \leq p \leq n$. For exam-

ple, we can achieve $O(\sqrt{n})$ time for both updates and queries. A second variation of our scheme, suitable for large data sets, achieves $O(n^\epsilon)$-time performance for updates and queries, while keeping $O(1)$ verification time, where $\epsilon > 0$ is any fixed constant.

Throughout the rest of this paper, we denote with n the current number of elements of the set S stored in the authenticated dictionary. Also, we describe the verification of positive answers to membership queries (i.e., validating $e \in S$). The verification of negative answers (i.e., validating $e \notin S$) can be handled with the technique of storing in the dictionary not the items themselves, but instead pairs of consecutive elements [19].

2 Preliminaries

In this section, we discuss some cryptographic concepts used in our approach.

One-Way Accumulators. An important tool for our scheme is that of one-way *accumulator* functions [2, 4, 9, 15, 26]. Such a function allows a source to digitally sign a collection of objects as opposed to a single object.

The use of one-way accumulators originates with Benaloh and de Mare [4]. They show how to utilize an exponential one-way accumulator, which is also known as an RSA accumulator, to summarize a collection of data so that user verification responses have constant-size. Refinements of the RSA accumulator used in our construction are given by Baric and Pfitzmann [2], Gennaro, Halevi and Rabin [15], and Sander, Ta-Shma and Yung [26].

Recently, Camenisch and Lysyanskaya [9] have independently investigated dynamic accumulators. They give a zero-knowledge protocol and a proof that a committed value is in the accumulator with respect to the Pedersen commitment scheme. They also present applications to revocation for group signature, identity escrow schemes and anonymous credentials systems.

As we show in the rest of this section, the RSA accumulator can be used to implement a static authenticated dictionary, where the set of elements is fixed. However, in a dynamic setting where items are inserted and deleted, the standard way of utilizing the RSA accumulator is inefficient. Several other researchers have also noted the inefficiency of this implementation in a dynamic setting (e.g., see [27]). Indeed, our solution can be viewed as refuting this previous intuition to show that a more sophisticated utilization of the exponential accumulator can be made to be efficient even in a dynamic setting.

The most common form of one-way accumulator is defined by starting with a "seed" value y_0, which signifies the empty set, and then defining the accumulation value incrementally from y_0 for a set of values $X = \{x_1, \cdots, x_n\}$, so that $y_i = f(y_{i-1}, x_i)$, where f is a one-way function whose final value does not depend on the order of the x_i's (e.g., see [4]). In addition, one desires that y_i not be much larger to represent than y_{i-1}, so that the final accumulation value, y_n, is not too large. Because of the properties of function f, a source can digitally sign the value of y_n so as to enable a third party to produce a short proof for any element x_i belonging to X—namely, swap x_i with x_n and recompute y_{n-1}

from scratch—the pair (x_i, y_{n-1}) is a cryptographically-secure assertion for the membership of x_i in set X.

A well-known example of a one-way accumulator function is the *exponential accumulator*, $f(y, x) = y^x \bmod N$ for suitably-chosen values of the seed y_0 and modulus N [4]. In particular, choosing $N = pq$ with p and q being two strong primes [22] makes the exponential accumulator function as difficult to invert as RSA cryptography [4].

The difficulty in using the exponential accumulator function in the context of authenticated dictionaries is that it is not associative; hence, any updates to set X require significant recomputations.

Implications of Euler's Theorem. There is an important technicality involved with use of the exponential accumulator function, namely in the choice of the seed $a = y_0$. In particular, we should choose a relatively prime with p and q. This choice is dictated by Euler's Theorem, which states that $a^{\phi(N)} \bmod N = 1$ if $a > 1$ and $N > 1$ are relatively prime. In our use of the exponential accumulator function, the following well-known corollary to Euler's Theorem will prove useful.

Corollary 1. *If $a > 1$ and $N > 1$ are relatively prime, then $a^x \bmod N = a^{x \bmod \phi(N)} \bmod N$, for all $x \geq 0$.*

One implication of this corollary to the authenticated dictionary problem is that the source should never reveal the values of the prime numbers p and q. Such a revelation would allow a directory to compute $\phi(N)$, which in turn could result in a false validation at a compromised directory. So, our approach takes care to keep the values of p and q only at the source.

Two-Universal Hash Functions. As in previous approaches [15, 26], we use the RSA accumulator in conjunction with two-universal hash functions. Such functions were first introduced by Carter and Wegman [10].

A family $H = \{h : A \to B\}$ of functions is *two-universal* if, for all a_1,$a_2 \in A$, $a_1 \neq a_2$ and for a randomly chosen function h from H,

$$\Pr_{h \in H}\{h(a_1) = h(a_2)\} \leq \frac{1}{|B|}.$$

In our scheme, the set A consists of $3k$-bit vectors and the set B consists of k-bit vectors, and we are interested in finding random elements in the preimage of a two-universal function mapping A to B. We can use the two-universal function $h(x) = Ux$, where U is a $k \times 3k$ binary matrix. To get a representations of all the solutions for $h^{-1}(e)$, we need to solve a linear system. Once this is done, picking a random solution can be done by multiplying a bit matrix by a random bit vector, and takes $O(k^2)$ bit operations.

Choosing a Suitable Prime. We are interested in obtaining a prime solution of the linear system that represents a two-universal hash function. The following lemma, of Gennaro *et al.* [15], is useful in this context:

Lemma 1 ([15]). *Let H be a two-universal family from $\{0,1\}^{3k}$ to $\{0,1\}^k$. Then, for all but a 2^{-k} fraction of the functions $h \in H$, for every $e \in \{0,1\}^k$ a fraction of at least $\frac{1}{ck}$ of the elements in $f^{-1}(e)$ are primes, for some small constant c.*

For reasons that will become clear in the security proof given in Section 6, our scheme requires that a prime inverse be greater then $\sqrt{2^{3k}}$. Also, since the domain of H is $\{0,1\}^{3k}$, this prime is less than 2^{3k}. Thus, in order to find a suitable prime with high probability $1-2^{-\Omega(k)}$, we need to sample $O(k^2)$ times.

The Strong RSA Assumption. The proof of security of our scheme uses the *strong RSA assumption*, as defined by Baric and Pfitzmann [2]. Given N and $x \in Z_N^*$, the strong RSA problem consists of finding integers f, with $2 \leq f < N$, and a, such that we have $a^f = x$. The difference between this problem and the standard RSA problem is that the adversary is given the freedom to choose not only the base a but also the exponent f.

> *Strong RSA Assumption:* There exists a probabilistic algorithm B that on input 1^r outputs an RSA modulus N such that, for all probabilistic polynomial-time algorithms D, all $c > 0$, and all sufficiently large r, the probability that algorithm D on a random input $x \in Z_N$ outputs a and $f \geq 2$ such that $a^f = x \bmod N$ is no more than r^{-c}.

In other words, given N and a randomly chosen element x, it is infeasible to find a and f such that $a^f = x \bmod N$.

A Straightforward Accumulator-Based Scheme. Let $S = \{e_1, e_2, \ldots, e_n\}$ be the set of elements stored at the source. Each element e is represented by k bits. The source chooses strong primes [22] p and q that are suitably large, e.g., $p, q > 2^{\frac{3}{2}k}$. It then chooses a suitably-large base a that is relatively prime to $N = pq$. Note that N is at least 2^{3k}. It also chooses a random hash function h from a two-universal family.The source broadcasts once a, N and h to the directories and users, but keeps p and q secret. At periodic time intervals, for each element e_i of S, the source computes the *representative* of e_i, denoted x_i, where $h(x_i) = e_i$ and x_i is a prime chosen as described above. The source then combines the representatives of the elements by computing the RSA accumulation

$$A \leftarrow a^{x_1 x_2 \cdots x_n} \bmod N$$

and broadcasts to the directories a signed message (A, t), where t is the current timestamp.

Query. To prove that a query element e_i is in S, a directory computes the value

$$A_i \leftarrow a^{x_1 x_2 \cdots x_{i-1} x_{i+1} \cdots x_n} \bmod N. \tag{1}$$

That is, A_i is the accumulation of all the representatives of the elements of S besides x_i and is said to be the *witness* of e_i. After computing A_i, the directory returns to the user the representative x_i, the witness A_i and the pair (A, t), signed by the source. Note that this query authentication information has constant size;

hence, this scheme is size-oblivious. However, computing witness A_i is no trivial task for the directory, for it must perform $n-1$ exponentiations to answer a query. Making the simplifying assumption that the number of bits needed to represent N is independent of n, the computation performed to answer a single query takes $O(n)$ time.

Verification. The user checks that timestamp t is current and that (A, t) is indeed signed by the source. It then checks that x_i is a valid representative of e_i, i.e., $h(x_i) = e_i$. Finally, it computes $A' \leftarrow A_i^{x_i} \bmod N$ and compares it to A. If $A' = A$, then the user is reassured of the validity of the answer because of the strong RSA assumption. The verification time is $O(1)$.

Updates. For updates, the above simple approach has an asymmetric performance (for unrestricted values of accumulated elements), with insertions being much easier than deletions. To insert a new element e_{n+1} into the set S, the source simply recomputes the accumulation A as follows

$$A \leftarrow A^{x_{n+1}} \bmod N$$

where x_{n+1} is the representative of e_{n+1}. The computation of x_{n+1} can be done in time that is independent of n (see Section 2), i.e., $O(1)$ time. An updated signed pair (A, t) is then sent to the directories in the next time interval. Thus, an insertion takes $O(1)$ time. The deletion of an element $e_i \in S$, on the other hand, will in general require the source to recompute the new value A by performing $n-1$ exponentiations. That is, a deletion takes $O(n)$ time.

Of course, if a representative x_i is relatively prime with $p-1$ and $q-1$, the source can delete e_i in constant time by computing $x \leftarrow x_i^{-1} \bmod \phi(N)$ (via the extended Euclidean algorithm) and then updating $A \leftarrow A^x \bmod N$. This deletion computation would take $O(1)$ time, but we cannot guarantee that x_i has an inverse in $Z_{\phi(N)}$ if it is an accumulation of a group of elements; hence, we do not advocate using this approach for deletions. Indeed, we will not assume the existence of multiplicative inverses in $Z_{\phi(N)}$ for any of our solutions. Thus, we are stuck with linear deletion time at the source and linear query time at a directory when making this straightforward application of exponential accumulators to the authenticated dictionary problem.

We describe in the next section an alternative approach that can answer queries much faster.

3 Precomputed Accumulations

We present a first improvement that allows for fast query processing. We require the directory to store a precomputed witness A_i for each element e_i of S, as defined in Eq. 1. Thus, to answer a query, a directory looks up the A_i value, rather than computing it from scratch, and then completes the transaction as described in the previous section. Thanks to the precomputation of the witnesses at the source, a directory can process any query in $O(1)$ time while the verification computation for a user remains unchanged.

Unfortunately, a standard way of implementing this approach is inefficient for processing updates. In particular, a directory now takes $O(n)$ time to process a single insertion, since it needs to update all the witnesses, and $O(n \log n)$ time to process a single deletion, for after a deletion the directory must recompute all the witnesses, which can be done using the algorithm given in [26]. Thus, at first glance, this precomputed accumulations approach appears to be quite inefficient when updates to the set S are required.

We can process updates faster than $O(n \log n)$ time, however, by enlisting the help of the source. Our method in fact can be implemented in $O(n)$ time by a simple two-phase approach. The details for the two phases follows.

First Phase. Let S be the set of n items stored at the source after performing all the insertions and deletions required in the previous time interval. We build a complete binary tree T "on top" of the representative values of the elements of S, so that each leaf of T is associated with the representative x_i of an element e_i of S. In the first phase, we perform a post-order traversal of T, so that each node v in T is visited only after its children are visited. The main computation performed during the visit of a node v is to compute a value $x(v)$. If v is a leaf of T, storing some representative x_i, then we compute

$$x(v) \leftarrow x_i \bmod \phi(N).$$

If v is an internal node of T with children u and w (we can assume T is proper, so that each internal node has two children), then we compute

$$x(v) \leftarrow x(u)x(w) \bmod \phi(N).$$

When we have computed $x(r)$, where r denotes the root of T, then we are done with this first phase. Since a post-order traversal takes $O(n)$ time, and each visit computation in our traversals takes $O(1)$ time, this entire first phase runs in $O(n)$ time. We again make the simplifying assumption that the number of bits needed to represent N is independent of n.

Second Phase. In the second phase, we perform a pre-order traversal of T, where the visit of a node v involves the computation of a value $A(v)$. The value $A(v)$ for a node v is defined to be the accumulation of all values stored at nodes that are *not* descendents of v (including v itself if v is a leaf). Thus, if v is a leaf associated with the representative value x_i of some element of S, then $A(v) = A_i$. For the root, r, of T, we define $A(r) = a$. For any non-root node v, let z denote v's parent and let w denote v's sibling (and note that since T is proper, every node but the root has a sibling). Given $A(z)$ and $x(w)$, we can compute the value $A(v)$ for v as follows:

$$A(v) \leftarrow A(z)^{x(w)} \bmod N.$$

By Corollary 1, we can inductively prove that each $A(v)$ equals the accumulation of all the values stored at non-descendents of v. Since a pre-order traversal of T takes $O(n)$ time, and each visit action can be performed in $O(1)$ time, we can compute all the A_i witnesses in $O(n)$ time. Note that implementing this

algorithm requires knowledge of the value $\phi(N)$, which presumably only the source knows. Thus, this computation can only be performed at the source, who must transmit the updated A_i values to the directory.

A variation of the approach presented in this section consists of precomputing at the source the exponents of the witnesses instead of the witnesses themselves. In this way, the arithmetic computations performed at the source consist of a series of modular multiplications (of bit complexity $O(k^2)$) and do not include modular exponentiations (of bit complexity $O(k^3)$). The directory, however, has to perform one modular exponentiation to compute the actual witness from its exponent when answering a query.

The precomputed accumulations approach supports constant-time queries and linear-time updates. In the next section, we show how to combine this approach with the straightforward approach of Section 2 to design a scheme that is efficient for both updates and queries.

4 Parameterized Accumulations

Consider again the problem of designing an accumulator-based authenticated dictionary for a set $S = \{e_1, e_2, \ldots, e_n\}$. In this section, we show how to balance the processing between the source and the directory, depending on their relative computational power. The main idea is to choose an integer parameter $1 \leq p \leq n$ and partition the set S into p groups of roughly n/p elements each, performing the straightforward approach inside each group and the precomputed accumulations approach among the groups. The details are as follows.

Subdividing the Dictionary. Divide the set S into p groups, $Y_1, Y_2, \ldots, Y_p$, of roughly n/p elements each, balancing the size of the groups as much as possible. For group Y_j, let y_j denote the product of the representatives of the elements in Y_j modulo $\phi(N)$. Define B_j as

$$B_j = a^{y_1 y_2 \cdots y_{j-1} y_{j+1} \cdots y_p} \bmod N.$$

That is, B_j is the accumulation of representatives of all the elements that are not in the set Y_j. After any insertion or deletion in a set Y_j, the source can compute a new value y_j in $O(n/p)$ time. Moreover, since the source knows the value of $\phi(N)$, it can update all the B_j values (or their exponents) after such an update in $O(p)$ time. Thus, the source can process an update operation in $O(p + n/p)$ time, assuming that the update does not require redistributing elements among the groups.

Maintaining the size of each set Y_j is not a major overhead. We need only keep the invariant that each Y_j has at least $\lceil n/p \rceil/2$ elements at most $2\lceil n/p \rceil$ elements. If a Y_j set becomes too small, then we either merge it with one of its adjacent sets Y_{j-1} or Y_{j+1}, or (if merging Y_j with such a sets would cause an overflow) we "borrow" some of the elements from an adjacent set to bring the size of Y_j to at least $3\lceil n/p \rceil/4$. Likewise, if a Y_j set grows too large, then we simply split it in two. These simple adjustments take $O(n/p)$ time, and will maintain

the invariant that each Y_j is of size $\Theta(n/p)$. Of course, this assumes that the value of n does not change significantly as we insert and remove elements. But even this condition is easily handled. Specifically, we can maintain the sizes of the Y_j's in a priority queue that keeps track of the smallest and largest Y_j sets. Whenever we increase n by an insertion, we can check the priority queue to see if the smallest set now must do some merging or borrowing to keep from growing too small. Likewise, whenever we decrease n by a deletion, we can check the priority queue to see if the largest set now must split. An inductive argument shows that this approach keeps the size of the groups to be $\Theta(n/p)$.

Turning to the task at a directory, then, we recall that a directory receives all p of the B_j values after an update occurs. Thus, a directory can perform its part of an update computation in $O(p)$ time. It validates that some e_i is in e by first determining the group Y_j containing e_i, which can be done by table look-up. Then, it computes A_i as

$$A_i \leftarrow B_j^{\Pi_{e_m \in Y_j - \{e_i\}} x_m} \bmod N,$$

where x_m is the representative of e_m. Thus, a directory answers a query in $O(n/p)$ time.

Improving the Update Time for the Source. In this section, we show how the source can further improve the performance of an update operation in the parameterized scheme. In the algorithm described above, the source recomputes y_j from scratch after any update occurs, which takes $O(n/p)$ time. We will now describe how this computation can be done in $O(\log(n/p))$ time.

The method is for the source to store the elements of each Y_j in a balanced binary search tree. For each internal node w in T_j, the source maintains the value $y(w)$, which is the product of the representatives of all the items stored at descendents of w, modulo $\phi(N)$. Thus, $y(r(T_j)) = y_j$, where $r(T_j)$ denotes the root of T_j. Any insertion or deletion will affect only $O(\log(n/p))$ nodes w in T_j, for which we can recompute their $x(w)$ values in $O(\log(n/p))$ total time. Therefore, after any update, the source can recompute a y_j value in $O(\log(n/p))$ time, assuming that the size of the Y_j's does not violate the size invariant. Still, if the size of Y_j after an update violates the size invariant, we can easily adjust it by performing appropriate splits and joins on the trees representing Y_j, Y_{j-1}, and/or Y_{j+1}. Moreover, we can rebuild the entire set of trees after every $O(n/p)$ updates, to keep the sizes of the Y_j sets to be $O(n/p)$, with the cost for this periodic adjustment (which will probably not even be necessary in practice for most applications) being amortized over the previous updates.

Theorem 1. *The parameterized accumulations scheme for implementing an authenticated dictionary over a set of size n uses data structures with $O(n)$ space at the source and directories and has the following performance, for a given parameter p such that $1 \leq p \leq n$:*

- *the insertion and deletion times for the source are each $O(p + \log(n/p))$;*
- *the update authentication information has size $O(p)$;*

– *the query time for a directory is* $O(n/p)$*;*
– *the query authentication information has size* $O(1)$*; and*
– *the verification time for the user is* $O(1)$.

Thus, for $p = \sqrt{n}$, one can balance insertion time, deletion time, update authentication information size, and query time to achieve an $O(\sqrt{n})$ bound, while keeping the query authentication information size and the verification time constant.

The parameterized accumulations scheme described in this section significantly improves the overhead at the source and directories for using an exponential accumulator to solve the authenticated dictionary problem. Moreover, this improvement was achieved without any modification to the client from the original straightforward application of the exponential accumulator described in Section 2.

In the next section, we show that if we are allowed to slightly modify the computation at the client, we can further improve performance at the source and directory while still implementing a size-oblivious scheme

5 Hierarchical Accumulations

In this section, we describe a hierarchical accumulation scheme for implementing an authenticated dictionary on a set S with n elements. In this scheme, the verification algorithm consists of performing a series of c exponentiations, where c is a fixed constant for the scheme. Note that the approach of Section 4 assumed that $c = 1$.

Given a fixed constant c, we define $p = n^{1/(c+1)}$ and construct the following hierarchical partition of S:

– We begin by partitioning set S into p subsets of $n_1 = n^{c/(c+1)}$ elements each, called level-1 subsets.
– For $i = 1, \ldots, c-1$, we partition each level-i subset into p subsets of $n^{(c-i)/(c+1)}$ elements each, called level-$(i+1)$ subsets.

Also, we conventionally say that S is the level-0 subset.

Next, we associate a value $\alpha(Y)$ to each subset Y of the above partition, as follows:

– The value of a level-c subset is the accumulation of the representatives of its elements.
– For $i = 0, \ldots, c-1$, the value of a level-i subset is the accumulation of the representatives of the values of its level-$(i+1)$ subsets.

Finally, we store with each level-i subset Y a data structure based on the precomputed accumulations scheme of Section 3 that stores and validates membership in the set $S(Y)$ of the values of the level-$(i+1)$ subsets of Y.

Let e be an element of S. To prove the containment of e in S, the directory determines, for $i = 1, \ldots, c$, the level-i subset Y_i containing e and returns the sequence of values $\alpha(Y_c), \alpha(Y_{c-1}), \ldots, \alpha(Y_0)$ plus witnesses for the following $c+1$ memberships:

– $e \in Y_c$
– $\alpha(Y_i) \in S(Y_{i-1})$ for $i = c, \ldots, 1$

The user can verify each of the above memberships by means of an exponentiation. Thus, the verification time and query authentication information are proportional to c, i.e., they are $O(1)$.

Theorem 2. *The hierarchical accumulations scheme for implementing an authenticated dictionary over a set of size n uses uses data structures with $O(n)$ space at the source and directories and has the following performance, for a given constant ϵ such that $0 < \epsilon < 1$:*

– *the insertion and deletion times for the source are each $O(n^\epsilon)$);*
– *the update authentication information has size $O(n^\epsilon)$;*
– *the query time for a directory is $O(n^\epsilon)$;*
– *the query authentication information has size $O(1)$; and*
– *the verification time for the user is $O(1)$.*

We can extend the hierarchical accumulations scheme by using a more general hierarchical partitioning of the set S while keeping constant the size of the query authentication information and the verification time. The two extreme partitioning strategies are: (i) single-level partition in $O(1)$ groups of size $O(n)$, and (ii) $O(\log n)$-level partition where the size of each partition is $O(1)$ (this corresponds to a hierarchy that can be mapped into a bounded-degree tree). The insertion and deletion times and the update authentication information size are then proportional to $O(\sum_{i=1}^{c-1} g_i)$, where g_i is the size of the partition at the i-th level, and can range from $O(1)$ to $O(n)$. At the same time, the query time is proportional to $O(c + g_{c-1})$, and can range from $O(n)$ to $O(1)$.

6 Security

We now show that an adversarial directory cannot forge a proof of membership for an element that is not in S. Our proof follows a closely related constructions given in [9, 15, 26]. An important property of the scheme comes from representing the elements e of set S with prime numbers. If the accumulator scheme was used without this stage, the scheme would be insecure. An adversarial directory could forge the proof of membership for all the divisors of elements whose proofs it has seen.

Theorem 3. *In the dynamic accumulator schemes for authenticated dictionaries defined in the previous sections, under the strong RSA assumption, a directory whose resources are polynomially bounded can produce a proof of membership only for the elements that are in the dictionary.*

Proof. Our proof is based on related proofs given in [9, 15, 26]. Assume an adversarial directory D has seen proofs of membership for all the elements

$e_1, e_2, \ldots e_n$ of the dictionary S. The trusted source has computed representatives $x_1, x_2, \ldots, x_n$ as suitable primes defined in Section 2. The witnesses $A_1, A_2 \ldots, A_n$ have been computed as well, either solely by the trusted source, or by balancing the work between the trusted source and the directories. The trusted source has distributed a signed pair (A, t). By the definition of the scheme in Section 2, for all $1 \leq i \leq n$, we have

- x_i is the prime representative of $e_i \in S$, i.e., $h(x_i) = e_i$;
- $\sqrt{2^{3k}} < x_i < 2^{3k}$;
- $A_i^{x_i} \bmod N = A$.

We need to show that directory D cannot prove the membership of a an element e_{n+1} that is not in the set S already. The proof is by contradiction. Suppose that D has has found a triplet $(e_{n+1}, x_{n+1}, A_{n+1})$ proving the membership of e_{n+1}. Then, the following must hold and can checked by the user (it is not necessary for x_{n+1} to be a prime):

- $h(x_{n+1}) = e_{n+1}$;
- $\sqrt{2^{3k}} < x_{n+1} < 2^{3k}$;
- $A_{n+1}^{x_{n+1}} \bmod N = A$.

Let $d = \gcd(x_{n+1}, x_1 x_2 \ldots x_n)$. Thus, we have $\gcd(\frac{x_{n+1}}{d}, \frac{x_1 x_2 \ldots x_n}{d}) = 1$. Define $f = \frac{x_{n+1}}{d}$. There are integers u, v such that $v\frac{x_1 x_2 \ldots x_n}{d} + uf = 1$ holds over integers. Directory D can find u and v in polynomial time using the extended Euclidean algorithm. Set $s = A_{n+1}^v a^u$. We have

$$s^f = A_{n+1}^{vf} a^{uf} = A_{n+1}^{v\frac{x_{n+1}}{d}} a^{uf} = A^{\frac{v}{d}} a^{uf} = a^{v\frac{x_1 x_2 \ldots x_n}{d} + uf} = a.$$

Thus, directory D can find in polynomial time a value s that is an f-th root of a. By the strong RSA assumption (Section 2), it must be that $f = 1$. Hence, we have $x_{n+1} = d$ and it follows that x_{n+1} divides $x_1 x_2 \ldots x_n$. But by our assumptions we have $x_{n+1} < 2^{3k}$ and $x_i > \sqrt{2^{3k}}$ for each i, which implies that $x_{n+1} = x_i$, for some $1 \leq i \leq n$. Thus, element e_{n+1} is already in set S, which is a contradiction.

We conclude that the adversarial directory D can find membership proofs only for those elements already in S. □

7 Experimental Results

In this section, we present a preliminary experimental study on the performance of the dynamic accumulator schemes for authenticated dictionaries described in this paper. The main results of this study are summarized in the the charts of Figures 1–2, where the x axis represents the size of the dictionary (number of elements) and the y axis represents the average time of the given operation in microseconds. We denote with $(f_1(n), f_2(n), ..., f_c(n))$ a generalized hierarchical partition scheme of the dictionary with $O(f_i(n))$ elements in the i-th level group (Section 5).

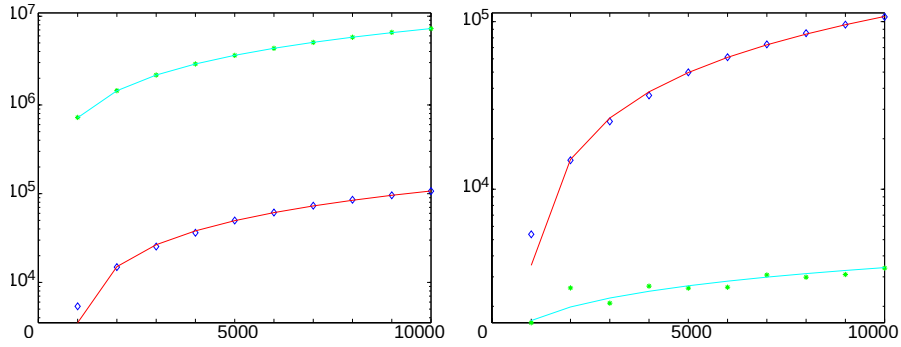


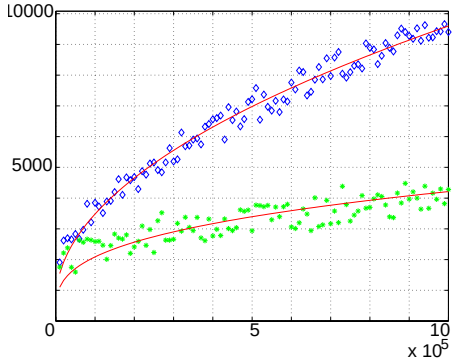
Fig. 1. Performance tradeoffs for dynamic accumulators. The insertion time at the source excludes the computation of the prime representative. Note that we use a logarithmic scale for the y-axis. **(a)** Query time at the directory (stars) when the directory computes the witness from scratch for each query (using modular exponentiations) vs. insertion time at the source (diamonds) when the source precomputes all the exponents of the witnesses (using modular multiplications). **(b)** Insertion time at the source (diamonds) without partitioning, when all the n witnesses's exponents are precomputed, vs. with partitioning (stars), when a 2-level ($n^{1/2}$, $n^{1/4}$) partitioning scheme is used and $O(n^{1/2})$ witnesses's exponents are precomputed.

The dynamic accumulator scheme has been implemented in Java and the experiments have been conducted on an AMD Athlon XP 1700+ 1.47GHz, 512MB running Linux. The items stored in the dictionary and the query values are randomly generated 165-bit integers and the parameter N of the RSA accumulator is a 200-bit integer. The variance between different runs of the query and deletion operations was found to consistently small so only a few runs were done for each dictionary size considered.

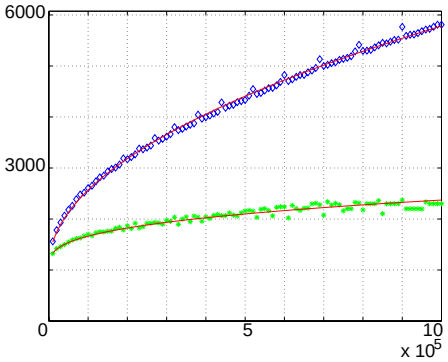
The main performance bottleneck of the scheme was found to be the computation of prime representatives for the elements. In our experiments, finding a prime representative of a 165-bit integer using the standard approach of Section 2 takes about 45 milliseconds and dominates the rest of the insertion time. The computation of prime representatives is a constant overhead that does not depend on the number of elements and has been omitted in the rest of the analysis.

Figure 1 illustrates two performance tradeoffs. Part (a) compares the performance of the two extreme naive approaches where either the source or the directory does essentially all the work. Since the source can use modular multiplication and the directory has to use modular exponentiation, it is more effective to shift as much as possible the insertion work to the source. Part (b) shows the benefits of partitioning, which allows to reduce the computation time at the source.

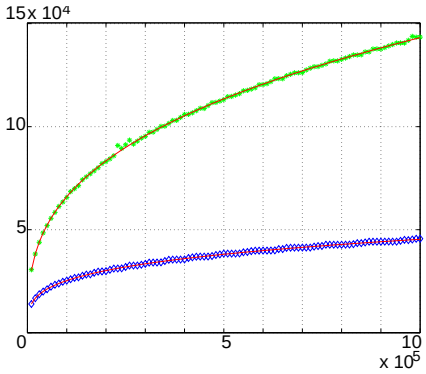
Experimental results on the hierarchical accumulations method (Section 5) are presented in Figure 2. These results show that one can tune the partitioning scheme according to the processing power available at the source and the directory.



(a) insertion time



(b) deletion time



(c) query time

Fig. 2. Insertion and deletion times at the source and query time at the directory for two variants of the hierarchical accumulations approach (Section 5) on dictionaries with up to one million elements. The time for computing the prime representative of an element has been omitted from the insertion time. The stars represent a 2-level ($n^{1/2}$, $n^{1/4}$) partitioning scheme and the diamonds represent a 2-level ($n^{2/3}$, $n^{1/3}$) partitioning scheme.

Acknowledgments

We would like to thank Andrew Schwerin, Giuseppe Ateniese, and Douglas Maughan for several helpful discussions and e-mail exchanges relating to the topics of this paper.

References

1. A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. In *Proc. Information Security Conference (ISC 2001)*, volume 2200 of *LNCS*, pages 379–393. Springer-Verlag, 2001.
2. N. Baric and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology: Proc. EUROCRYPT*, volume 1233 of *LNCS*, pages 480–494. Springer-Verlag, 1997.
3. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *LNCS*, pages 216–233. Springer-Verlag, 1994.
4. J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology—EUROCRYPT 93*, volume 765 of *LNCS*, pages 274–285. Springer-Verlag, 1993.
5. J. J. Bloch, D. S. Daniels, and A. Z. Spector. A weighted voting algorithm for replicated directories. *Journal of the ACM*, 34(4):859–909, 1987.
6. M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.
7. M. Blum and S. Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, Jan. 1995.
8. A. Buldas, P. Laud, and H. Lipmaa. Accountable certificate management using undeniable attestations. In *ACM Conference on Computer and Communications Security*, pages 9–18. ACM Press, 2000.
9. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proc. CRYPTO 2002*. To appear.
10. I. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Proc. ACM Symp. on Theory of Computing*, pages 106–112, 1977.
11. R. Cohen, M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Authenticated data structures for graph and geometric searching. Technical report, Center for Geometric Computing, Brown University, 2001. http://www.cs.brown.edu/cgc/stms/papers/authDatStr.pdf.
12. D. A. Cooper. A more efficient use of delta-CRLs. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 190–202, 2000.
13. Fischlin. Incremental cryptography and memory checkers. In *Proc. EUROCRYPT*, volume 1233 of *LNCS*, pages 393–408. Springer-Verlag, 1997.
14. I. Gassko, P. S. Gemmell, and P. MacKenzie. Efficient and fresh certification. In *Int. Workshop on Practice and Theory in Public Key Cryptography (PKC '2000)*, volume 1751 of *LNCS*, pages 342–353. Springer-Verlag, 2000.
15. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Proc. EUROCRYPT*, volume 1592 of *LNCS*, pages 123–139. Springer-Verlag, 1999.
16. M. T. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical report, Johns Hopkins Information Security Institute, 2000. http://www.cs.brown.edu/cgc/stms/papers/hashskip.pdf.
17. M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. 2001 DARPA Information Survivability Conference and Exposition*, volume 2, pages 68–82, 2001.
18. C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

19. P. C. Kocher. On certificate revocation and validation. In *Proc. Int. Conf. on Financial Cryptography*, volume 1465 of *LNCS*. Springer-Verlag, 1998.
20. B. Kroll and P. Widmayer. Distributing a search tree among a growing number of processors. *ACM SIGMOD Record*, 23(2):265–276, 1994.
21. C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. A general model for authentic data publication, 2001. http://www.cs.ucdavis.edu/~devanbu/files/model-paper.pdf.
22. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
23. R. C. Merkle. Protocols for public key cryptosystems. In *Proc. Symp. on Security and Privacy*, pages 122–134. IEEE Computer Society Press, 1980.
24. R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO '89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1990.
25. M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.
26. T. Sander, A. Ta-Shma, and M. Yung. Blind, auditable membership proofs. In *Proc. Financial Cryptography (FC 2000)*, volume 1962 of *LNCS*. Springer-Verlag, 2001.
27. B. Schneier. *Applied Cryptography: protocols, algorithms, and source code in C*. John Wiley and Sons, Inc., New York, 1994.
28. G. F. Sullivan, D. S. Wilson, and G. M. Masson. Certification of computational results. *IEEE Trans. Comput.*, 44(7):833–847, 1995.