An Efficient Dynamic and Distributed Cryptographic Accumulator

JASMINKA HASIC

in collaboration with Roberto Tamassia and Michael T. Goodrich

May 13, 2002

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. In particular, we show how to perform updates and queries in $O(n^{1/2})$ time while keeping the constant-time verification algorithm exactly the same as in previous inefficient schemes. In addition, at the expense of slightly increasing the conceptual complexity of the verification, we show that there is an accumulator-based approach to the authenticated dictionary problem that achieves $O(n^{\varepsilon})$-time performance for updates and queries, while keeping $O(1)$ verification time, where ε is any fixed constant such that $\varepsilon > 0$. This work has applications to certificate management in public key infrastructure and end-to-end integrity of data collections published by third parties on the Internet.

Keywords authenticated dictionary, one-way accumulator, certificate revocation, third-party data publication, authentication of cached data, dynamic data structure

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).

A major aspect of our approach to this challenge is to replicate the computations of servers throughout mirror sites in the network, so as to reduce the network latency experienced by users in their client applications. This approach is used, for example, by Akamai Technologies to push images and other content to web servers that are close to client browsers. Thus, a user will in general be much closer to one of these mirror sites than to the source of the service, and will therefore experience a faster response time from a mirror than it would by communicating directly with the source. In addition, by off-loading user servicing from the source, this distributed scheme protects the source from denial-of-service attacks and allows for

entries, and design specifications for content providers who wish to outsource the business of publishing this information and processing transactions involving it. In this case, the players in our framework are as follows: the source is a trusted organization (e.g., a stock exchange) that produces and maintains integrity-critical content (e.g., stock prices) and allows third parties (e.g., Web portals), to publish this content on the Internet so that it widely disseminated. The publishers store copies of the content produced by the source and process queries on such content made by the users. In addition to returning the result of a query, a publisher also returns a proof of authenticity of the result, thus providing a validation service. Publishers also perform content updates originating from the source. Even so, the publishers provide this added value and are able to charge for it without the added cost of deploying all the mirror sites in high-security firewall-protected environments. Indeed, the publishers are not assumed to be trustworthy, for a given publisher may be processing updates from the source incorrectly or it may be the victim of a system break-in.

Another application of the authenticated dictionary is in certificate revocation [27, 36, 37, 1, 13, 25, 21], where the source is a *certification authority* (CA) that digitally signs certificates binding entities to their public keys, thus guaranteeing their validity. These certificates are then used to authorize secure socket layer (SSL) connections to e-stores and business-to-business exchanges. Nevertheless, certificates are sometimes revoked (e.g., if if a private key is lost or compromised, or if someone loses their authority to use a particular private key). Thus, the user of a certificate must be able to verify that a given certificate has not been revoked. To facilitate such queries, the set of revoked certificates is distributed to *certificate revocation directories*, which process revocation status queries on behalf of users. The results of such queries need to be trustworthy, for they often form the basis for electronic commerce transactions.

Finally, we highlight how authenticated dictionaries could be used in military and research applications, for they could be used for the authenticated querying of information repositories, such as coalition documents, mission logs, genomic databases [26], and astrophysical databases (like the object catalog of the Sloan Digital Sky Survey [30, 12, 31]). Given the significant defense and scientific benefits that can result from such querying, users need to be certain that the results of their queries are accurate and current.

1.3 Previous Related Work

Authenticated dictionaries are related to research in distributed computing (e.g., data replication in a network [7, 29]), data structure design (e.g., program checking [8, 10, 11, 41] and memory checking [9, 19]), and cryptography (e.g., incremental cryptography [4, 5, 19, 20]).

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 [27]) 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. A directory then just forwards that entire signed CRL to any user who requests the revocation status of a certificate. 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 [16], are not size-oblivious.

Micali [36] proposes an alternate approach, where the source periodically sends to each directory the list of all issued certificates, each tagged with the signed timestamped value of a one-way hash function (e.g., see [40]) that indicates if this certificate has been revoked or not. This approach allows the system to reduce the size of the query authentication information to $O(1)$ words: namely just a certificate identifier and a hash value indicating its status. Unfortunately, this scheme requires the size of the update authentication information to increase to $\Theta(N)$, where N is the number of all nonexpired certificates issued by the certifying authority, which is typically much larger than the number n of revoked certificates. It is size-oblivious for immediate queries, but cannot be used for time stamping for archiving purposes, since no digest of the

The rest of this paper is organized as follows. In Section 2 we review the exponential accumulator [6] and other concepts used in our approach. We also present some basic tools that are used in the rest of the paper, including a description of a straightforward application of the exponential accumulator to the authenticated dictionary problem. We describe an improvement of this scheme that gives constant query and verification times but linear update time in Section 3. This improvement, called *precomputed accumulations*, consists of an efficient precomputation by the source of auxiliary data used by the directories to speed-up query processing. In Section 4, we present our complete solution, which uses a second improvement, called *parameterized accumulations*, to achieve a variety of tradeoffs between the query and update times, while preserving constant verification time by the user. For example, we can balance the two times and achieve $O(\sqrt{n})$ query and update time and $O(1)$ verification time. An alternative solution is presented in Section 5, where we present the *parameterized accumulations* scheme. This scheme, suitable for large data sets, achieves $O(n^{\varepsilon})$ query and update time and $O(1)$ verification time, where ε is any fixed constant such that $\varepsilon > 0$. Section 6 discusses the security of our scheme. Finally, concluding remarks are given in Section 8.

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 validation of positive answers to membership queries (i.e., validating $e \in S$). The validation of negative answers (i.e., validating $e \notin S$) can be handled with a standard method, as discussed in Section 8.

2 Preliminaries

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

2.1 One-Way Accumulators

An important tool for our scheme is that of one-way *accumulator* functions [6, 3, 22, 39]. 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 [6]. 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 [3], Gennaro, Halevi and Rabin [22], and Sander, Ta-Shma and Yung [39].

As we show in the next 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 [40]). 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 [6]). 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 commutative nature of 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*,

$$\exp(y, x) = y^x \bmod N, \tag{1}$$

Recall from Section 2.3 that picking a random solution takes $O(k^2)$ bit operations. Thus, the total running time of finding a suitable prime is equal to running $O(k^2)$ primality tests.

One needs to be careful about choice of primality test because it could happen that the cost of prime generation and verification dominates the cost of signing. One could use Miller-Rabin test, for example. To reduce the probability of mistaking a composite number for a prime one could perform a number of additional Miller-Rabin tests. Performing these tests could be costly. Fortunately, Cramer and Shoup [17] give a fast primality testing algorithm that can be used here. It does additional tests between runs of Miller-Rabin algorithm that reduce the primality checking time. They also state that empirical runs of algorithm indicate running times that are suitable for signing schemes.

2.5 The Strong RSA Assumption

The proof of security of our scheme uses the *strong RSA assumption*, as defined by Baric and Pfitzmann [3]. 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 $a^f = x \bmod N$.

2.6 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 [33] 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 as discussed in Section 2.3). The source broadcasts a, N and h to the directories and users, but keeps the p and q secret. For each element e_i of S, the source computes the *representative* of e_i, denoted x_i, where x_i is a prime chosen as described in Section 2.4. 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 a current timestamp.

2.6.1 Query

When asking for proof of membership of e_i, the user submits e_i to a directory. To prove that some query item e_i is in S, the directory computes the value

$$A_i \leftarrow a^{x_1 x_2 \cdots x_{i-1} x_{i+1} \cdots x_n} \bmod N. \tag{2}$$

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 then returns to the user the representative x_i, the witness A_i and the pair (A, t), signed by the source. 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. Note that the message sent to the user has constant size; hence, this scheme is size-oblivious.

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.

3.1 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. 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.

3.2 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 $A(v) = A_i$. Recall that in a pre-order traversal we perform the visit action on each node v before we perform the respective visit actions for v's children. 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 2, 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.

The performance of the precomputed accumulation scheme is summarized in Table 2.

space	insertion time	deletion time	update info	query time	query info	verify time
$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$

Table 2: Precomputed accumulation scheme for implementing an authenticated dictionary with an exponential accumulator.

The precomputed accumulations approach supports constant-time queries and linear-time updates. If n is large, however, and updates occur frequently, then the linear-time computations at the source can take a

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

The performance of the parameterized accumulation algorithm is summarized in Table 3.

space	insertion time	deletion time	update info	query time	query info	verify time
$O(n)$	$O(p+n/p)$	$O(p+n/p)$	$O(p)$	$O(n/p)$	$O(1)$	$O(1)$

Table 3: Parameterized accumulations scheme for implementing an authenticated dictionary using an exponential accumulator. We denote with p an integer such that $1 \leq p \leq n$.

The parameter p allows us to balance the performance between the source and the directories, and also between the cost for an update and the cost for performing queries. For example, we can balance performance equally by setting $p = \lceil \sqrt{n} \rceil$, which implies that both queries and updates in this scheme take $O(\sqrt{n})$ time. Note that for reasonable values of n, say for n between 10,000 and 1,000,000, $\sqrt{n}$ is between 100 and 1,000. In many cases, this is enough of a reduction to make the dynamic exponential accumulator practical for the source and directories, while still keeping the user computation to be one exponentiation and one signature verification. Indeed, these user computations are simple enough to even be embedded in a smart card, a PDA, or mobile phone.

4.2 Improving the Update Time for the Source

We describe in this section how the source can further improve the performance of an update operation in the parameterized scheme. Recall that in this scheme the set S is partitioned into p subsets, $Y_1, Y_2, \ldots, Y_p$, and the source maintains for each Y_j a value B_j, on behalf of the directories, that is the accumulation of all the values not in Y_j. Also recall that, for each group Y_j, we let y_j denote the product of the items in Y_j modulo $\phi(N)$. In the algorithm description above, the source recomputes y_j from scratch after any update occurs, which takes $O(n/p)$ time. In this section we describe how this 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 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. This performance of the resulting scheme is summarized in Table 4.

space	insertion time	deletion time	update info	query time	query info	verify time
$O(n)$	$O(p+\log(n/p))$	$O(p+\log(n/p))$	$O(p)$	$O(n/p)$	$O(1)$	$O(1)$

Table 4: Enhanced parameterized scheme for implementing an authenticated dictionary using an exponential accumulator. We denote with p an integer such that $1 \leq p \leq n$.

In this version of our scheme, we can achieve a complete tradeoff between the cost of updates at the source and queries at the directories. Tuning the parameter p over time, therefore, could yield the optimal balance between the relative computational powers of the source and directories. It could also be used to balance between the number of queries and updates in the time intervals.

space	insertion time	deletion time	update info	query time	query info	verify time
$O(n)$	$O(n^\varepsilon)$	$O(n^\varepsilon)$	$O(n^\varepsilon)$	$O(n^\varepsilon)$	$O(1)$	$O(1)$

Table 5: Hierarchical accumulations scheme for implementing an authenticated dictionary with an exponential accumulator, where $0 < \varepsilon < 1$ is a fixed constant.

The hierarchical accumulations scheme is likely to outperform in practice the parameterized accumulations scheme only for large-scale authenticated dictionaries (say, containing billions of entries), where the difference between $n^{1/(c+1)}$ and $n^{1/2}$ is significant and offsets the added complication of changing the client code and introducing the $(c+1)$-level accumulation hierarchy

Theorem 5: *The hierarchical accumulations scheme for implementing an authenticated dictionary over a set of size n uses $O(n)$ space and has the following performance, for a given constant ε such that $0 < \varepsilon < 1$:*

- *the insertion and deletion times are $O(n^\varepsilon)$);*
- *the update authentication information has size $O(n^\varepsilon)$;*
- *the query time is $O(n^\varepsilon)$;*
- *the query authentication information has size $O(1)$; and*
- *the verification time is $O(1)$.*

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 [22, 39]. A very important property of the scheme comes from representing elements e of set S with prime numbers x. 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 divisors of elements whose proofs it has seen.

Theorem 6: *In scheme defined in Section 2, 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 S.*

Proof: Our proof is based on related proofs given in [22, 39]. Assume an adversarial directory D has seen proofs of membership for all the elements $e_1, e_2, \ldots e_n$ of S. The trusted source has computed representatives $x_1, x_2, \ldots, x_n$ as suitable primes defined in Section 2.4. 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, for all $1 \leq i \leq n$,

- x_i is the prime representative of $e_i \in S$,
- $\sqrt{2^{3k}} < x_i < 2^{3k}$ (this is one of the conditions from Section 2.4),
- $A_i^{x_i} \bmod N = A$.

We need to show that directory D cannot come up with a triplet $(e_{n+1}, x_{n+1}, A_{n+1})$ proving 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 $(e_{n+1}, x_{n+1}, A_{n+1})$. Then the following must hold and is checked by the user (it is not necessary for x_{n+1} to be a prime):

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

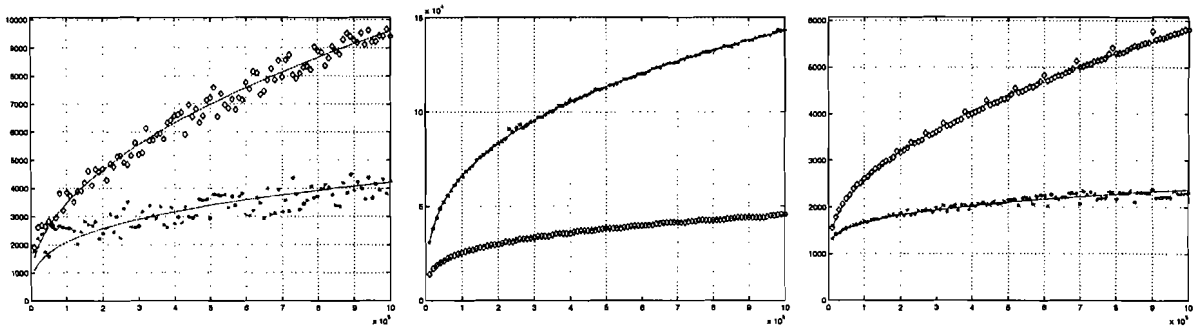

Figure 3: Insert,Quary,Remove

Figure 4 shows the effect of partitioning in the case when central authority performs all the work:

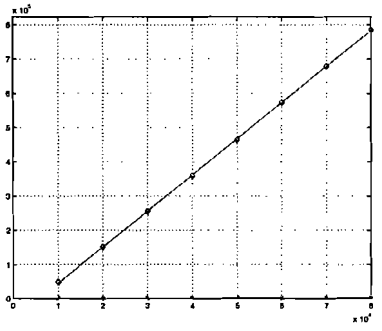

Figure 4: Partioning vs Not

The performance of the overall scheme can be adjusted according to the available resources. The more work that the central authority can perform, the better overall performance. But the amount of work that the central authority can perform if dependent on the update interval. Thus all these parameters need to be accounted for when making choices for the partitioning of the dictionary.

8 Discussion and Conclusion

We have shown how to make the exponential accumulator function the basis for a practical and efficient scheme for authenticated dictionaries, which relies on reasonable cryptographic assumptions similar to those that justify RSA encryption. A distinctive advantage of our approach is that the validation of a query result performed by the user takes constant time and requires computations (a single exponentiation and digital signature verification) simple enough to be performed in devices with very limited computing power, such as a smart card or a mobile phone.

An important aspect of our scheme is 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, for any fixed integer parameter $1 \leq p \leq n$. For example, we can achieve $O(\sqrt{n})$ time for both updates and queries. A second variation of our scheme, suitable for large data sets, achieves $O(n^{\varepsilon})$-time performance for updates and queries, while keeping $O(1)$ verification time, where $\varepsilon > 0$ is any fixed constant.

Our scheme can be easily adapted to contexts, such as certificate revocation queries, where one needs to also validate that an item e is *not* in the set S. In this case, we use the standard method of storing in the

[12] R. J. Brunner, L. Csabai, A. S. Szalay, A. Connolly, G. P. Szokoly, and K. Ramaiyer. The science archive for the Sloan Digital Sky Survey. In *Proceedings of Astronomical Data Analysis Software and Systems Conference V*, 1996.

[13] 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.

[14] I. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Proc. ACM Symp. on Theory of Computing*, pages 106–112, NY, 1977. Association for Computing Machinery.

[15] 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.

[16] 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.

[17] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *ACM Conference on Computer and Communications Security*, pages 46–51, 1999.

[18] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.

[19] Fischlin. Incremental cryptography and memory checkers. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT, LNCS 1233*, pages 393–408, 1997.

[20] M. Fischlin. Lower bounds for the signature size of incremental schemes. In *38th Annual Symposium on Foundations of Computer Science*, pages 438–447, 1997.

[21] I. Gassko, P. S. Gemmell, and P. MacKenzie. Efficient and fresh certification. In *International Workshop on Practice and Theory in Public Key Cryptography '2000 (PKC '2000)*, Lecture Notes in Computer Science, pages 342–353, Melbourne, Australia, 2000. Springer-Verlag, Berlin Germany.

[22] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology: Proc. EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer-Verlag, 1999.

[23] M. T. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical Report, Johns Hopkins Information Security Institute, 2000.

[24] 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.

[25] C. Gunter and T. Jim. Generalized certificate revocation. In *Proc. 27th ACM Symp. on Principles of Programming Languages*, pages 316–329, 2000.

[26] R. M. Karp. Mapping the genome: Some combinatorial problems arising in molecular biology. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 278–285, 1993.

[27] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[28] P. C. Kocher. On certificate revocation and validation. In *Proc. International Conference on Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, 1998.

[29] B. Kroll and P. Widmayer. Distributing a search tree among a growing number of processors. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):265–276, 1994.

[30] R. Lupton, F. M. Maley, and N. Young. Sloan digital sky survey. http://www.sdss.org/sdss.html.