Authenticated Dictionaries for Fresh Attribute Credentials*

Michael T. Goodrich[1], Michael Shin[2,3], Roberto Tamassia[3], and William H. Winsborough[4]

[1] Dept. Info. & Comp. Sci., University of California, Irvine
Irvine, CA 92697 USA
goodrich@acm.org
[2] Department of Computer Science, Johns Hopkins University
Baltimore, MD 21218 USA
mys@cs.jhu.edu
[3] Department of Computer Science, Brown University
Providence, RI 02912 USA
{mys,rt}@cs.brown.edu
[4] Network Associates Laboratories
Rockville, MD 20850 USA
willian_winsborough@nai.com

Abstract. We describe several schemes for efficiently populating an authenticated dictionary with fresh credentials. The thrust of this effort is directed at allowing for many data authors, called sources, to collectively publish information to a common repository, which is then distributed throughout a network to allow for authenticated queries on this information. Authors are assured of their contributions being added to the repository based on cryptographic receipts that the repository returns after performing the updates sent by an author. While our motivation here is the dissemination of credential status data from multiple credential issuers, applications of this technology also include time stamping of documents, document version integrity control, and multiple-CA certificate revocation management, to name just a few.

Keywords authenticated dictionary, certificate revocation, third-party data publication, authentication of cached data, dynamic data structures, digital credential, trust management

1 Introduction

Many problems in distributed systems require that content from various authors be validated for authenticity and integrity. One of these problems is authorization in decentralized systems. Traditional authorization systems assume

* Work supported in part by DARPA through AFRL agreement F30602–00–2–0509 and SPAWAR contract N66001–01–C–8005 and by NSF under grants CCR–0098068 and CDA–9703080.

organizations have a hierarchical structure in which authority emanates from a single entity. However, this assumption breaks down when independent organizations engage in collaboration or commerce, particularly in large numbers. In this case, authority emanates from many entities. Blaze, Feigenbaum, and Lacy [3] introduced *trust management* (TM) as a collection of principles for supporting authorization in such a decentralized environment. TM systems such as KeyNote [2] use credentials or chains of credentials to represent capabilities, that is, rights with respect to specific resources. Others, such as SPKI/SDSI [7] and *RT* [15], enable entities to manage and combine their judgments about one another's authority based on more general entity attributes. What these systems have in common is that authority is distributed among a potentially large number of entities, each of which may issue credentials. Additionally, credentials from several entities may need to be combined to form a proof that a particular resource request is authorized.

In the authorization systems mentioned above, an entity is typically represented by a public/private cryptographic key pair. Using its private key, an entity signs the credentials it issues, enabling authenticity and integrity to be verified. The assertions carried in credentials may be subject to changing circumstances. A credential's validity ends when it expires or is revoked. Updates concerning a credential's validity status can be disseminated by the issuer either as positive revalidations, or as negative revocations.

For convenience here, we assume that Alice is a typical credential issuer. For example, Alice may be a certification authority[5] who has the ability and responsibility of authenticating various entities. Presumably Alice is motivated to make her assertions verifiable, since she issues signed certificates. To accept such a certificate, Bob, a typical agent relying on a certificate issued by Alice, may require proof that as of a certain time in the recent past, Alice had not revoked the certificate. Yet Alice's strengths may not include the ability to answer a large volume of online certificate-status queries. Thus, Alice and Bob are both motivated to utilize a third-party, Charles, who will maintain certificate status information for Alice and provide verifiable answers to queries from users, such as Bob, as quickly and efficiently as possible.

In practice, of course, it is impossible to disseminate credential status updates instantaneously, particularly when invoking a third-party, Charles, to process updates and queries. The longer it takes, the greater the risk of basing decisions on stale information. The rate at which risk accumulates depends on the nature of the information, as well as on the application for which credentials are accepted. Ultimately, Bob must determine what level of risk is tolerable. Yet whatever that level may be, we desire a system that will quickly communicate any updates from Alice to Bob with minimal overhead and minimal additional trust required in Charles.

[5] For our purposes, the terms "certificate" and "credential" are essentially interchangeable. Historically, a "certificate" has typically carried a binding of a name to a public key, while "credential" has often been used more broadly for associations with keys.

1.1 Considering Many Credential Issuers

Prior work in the area of distributed data authentication (see, e.g., [1, 4–6, 10–13, 18]) has focused primarily on disseminating updates from a single, trusted source (merging the roles of Alice and Charles). Providing a high degree of availability through these previous solutions, a small-scale credential issuer would incur significant start-up cost. Economies of scale would suggest that credential issuers could contain their costs by using a shared infrastructure to disseminate credential status data. If we consider credential issuers to be the authors of credential status data, then several authors may use the same publisher, Charles. In this case, the authors entrust to the publisher the dissemination of credential status data to its consumers, the authorization agents. In the current paper, we present algorithms that can be used for this purpose. Our design goals are as follows:

- Authors must be able to determine whether their updates are being published in a timely manner.
- Determination that updates are occurring should be done efficiently and without requiring trust in the responders.
- The repository should provide cryptographic *receipts* verifying that updates are occurring.
- Update receipts should be small, so as to minimize network traffic, and should be easy to verify.

In addition, the following client user-interface design goals are shared with prior work:

- Answers to client queries must be highly available, meaning that it should be possible to obtain answers at any time without prior arrangement.
- Answers must be validated efficiently for authenticity and integrity.
- Client users must be able to reliably determine the time at which data they receive in answers was certified by its authors.
- Clients need not trust responders.

1.2 Our Contributions

In this paper, we describe how to extend prior authenticated dictionary techniques [1, 4–6, 10–13, 18] to show how a publication service can make highly available the current, authenticatable validity status of each credential issued by one of a large number of credential authors. Our techniques for supporting many authors involve the novel use of efficient authenticated data structures. Moreover, we offer an author, Alice, a trade-off between how much state she wishes to maintain for her data and the size of the receipts that the publication repository, Charles, provides to prove he is processing her updates correctly. An important feature of our solutions is they allow Alice to delegate computations and data archiving to Charles without delegating large degrees of trust to him. Indeed, the protection of trust in a distributed environment is an important principle in our work.

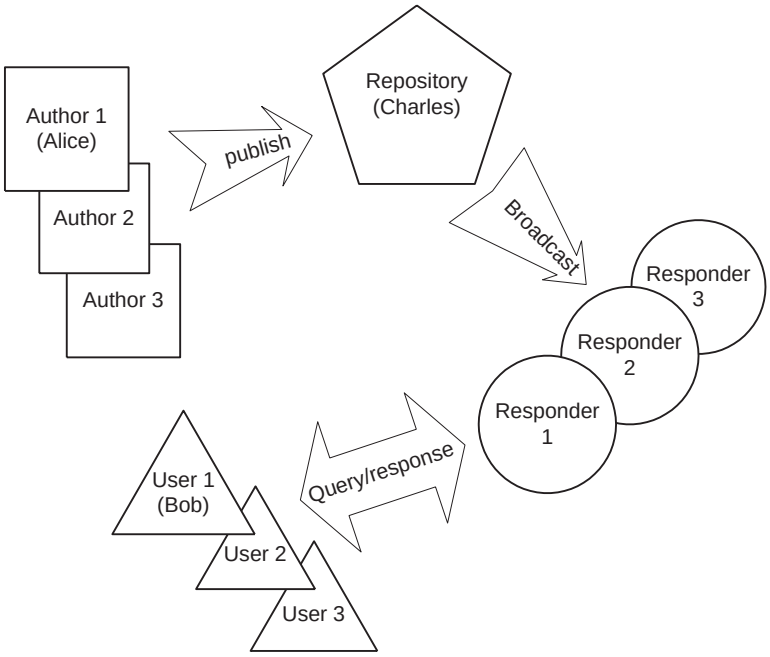

Fig. 1. Entities in a multi-authored authenticated dictionary

The paper is organized as follows. We begin with a general discussion of background and motivation for multi-authored authenticated data repositories. We follow this discussion, in Section 3, with a presentation of two simple-minded schemes for providing for multiple authors in an authenticated setting, showing that both of these schemes have serious drawbacks. We then present, in Section 4, three variations on a novel intermediate scheme, showing its efficiency with respect to several measures. Finally, in Section 5, we describe a number of extensions to our scheme.

2 Background and Motivation

As mentioned above, the framework we are addressing in this paper involves several groups of entities operating in a distributed environment, as illustrated in Figure 1.

- An *author* (Alice). The authors are the issuers of credentials and/or credential status information. They are assumed to be online only during their interaction with the data repository. They may have limited storage and processing abilities; hence, they do not wish to process queries for users.
- A *user* (Bob). The users are consumers of credential and/or credential status information. They desire quick response for a query requesting a credential or

the revocation status of an existing credential. They have trust in the issuer of a credential or credential status information, but they do not necessarily trust the publishing system that is archiving and providing query responses for this information.
- The *publication repository* (Charles). The publication repository stores the credential and/or credential status information for the authors. Charles processes updates for authors, providing them with receipts that confirm his acceptance of these updates. He also broadcasts these updates out to the responders, who perform the actual processing of user requests.
- The *responders*. The responders accept updates from the publication repository, Charles, and answer queries on this data for users. In addition to the answers that responders provide, they also provide cryptographic proofs, derived from *basis* information provided by Alice and Charles, that their answers are as accurate as if they had come directly from Alice and Charles.

We assume Alice has some recourse in the event that Charles fails to fulfill his publishing obligations correctly. It may satisfy Alice's purposes to presume that should Charles fail, Alice would make this known, thus damaging Charles's reputation to such a degree as to act as a sufficient deterrent. However, if Alice's ability to conduct her operations would be significantly compromised, stronger assumptions may be required. For instance, Alice could maintain a minimal online presence, operating a site, referred to in the credentials she issues, that gives instructions about how to contact responders operated by her current publisher. This level of indirection would enable Alice to replace Charles should he fail to meet his publishing obligations. Depending on the contract Alice and Charles enter into, Alice may also have legal recourse. Thus, we have a spectrum of possible recourses.

So as to better motivate this framework, let us briefly discuss some applications.

2.1 Archiving and Time Stamping

An important service in a distributed collaborative environment is that of archiving and time stamping. Such a service stores documents and other data ordered by creation time, and allows users to retrieve these documents and verify their creation times. In our framework, the document and data creators are the authors, the publication repository is the archive, the responders provide answers to queries, and the users request documents and their time stamp verification. For efficiency, Alice may wish to sign only a cryptographic digest (hash) of her documents, and it may actually be this digest that is archived for time stamping purposes. In addition, note that in this application Alice will never delete anything from the archive, since that would violate the principle of time stamping. Her updates in this case are limited to insertions.

Charles's signature is useful for corroborating the date at which Alice makes her assertion. It proves that Alice had signed the assertion by the date Charles

signs it. It prevents Alice signing something later than the date shown, or anyone else claiming that she did so. Note that this is not inherently related to Charles's function as a distribution channel in any way more fundamental than convenience.

2.2 Certificate Validity Status

Digital certificates are a central part of any public-key infrastructure (PKI), in that they bind identification information to a public key. Such mappings could be for individuals, groups, or even roles and permissions within a dynamic coalition. But certificates may need to be revoked before they expire, due to changes in roles, system compromises, or lost private keys. Once a certificate is revoked, it is no longer valid. Thus, an important, but often neglected, step in any protocol involving digital certificates is the verification of the revocation status of a digital certificate. Putting this application into our framework, we see that the authors are the certification authorities (CA's) that issue and revoke certificates. Likewise, the publication repository and its responders provide a certificate status querying service, with verifiable responses to queries. The client users in this case are any entities that wish to verify the status of a given digital certificate. Clients may also need to be able to prove that status to others.

Validity status data can be represented as a collection of certificates valid at a certain time, or as a similarly dated collection of certificate revocations, each giving, say, the serial number of the certificate. Clearly in either case, the representation must support insertion. In the former case, the (verified) presence of the certificate in the collection establishes its validity as of the associated time. Authors must be able to delete a credential to revoke it, thus rendering it invalid after the next time quantum. The acceptor of a credential determines how recently validity must be proven and therefore what amount of latency (introduced by publication or by caching of validity proofs) is acceptable. Verification of negative query responses may be needed in some applications, for instance, to justify (to a third party) denial of access.

When status data is represented by a collection of revocations, the (verified) absence of an unexpired credential's serial number in the collection establishes the credential's validity as of the associated time. Thus, it is essential that the representation of this collection support verifiable negative answers to user queries. Verifiable positive answers may also be necessary in some applications to enable acceptors to justify credential rejection.

The essential functionality of either representation is to enable the acceptor to verify that the issuer has revalidated a certificate at some point in the recent past. Normal operation need not allow authors to "unrevoke" or reinstate a revoked credential; if required, a new certificate carrying the same assertion can be issued instead. However, to maintain efficiency it may be important to be able remove revocations from the data structure, for instance, after the corresponding credentials have expired. After that time, the revocation is no longer needed to invalidate the credential. Yet removal of revocations must be undertaken cautiously, particularly as the revocation time approaches the expiration time.

In many applications it is important to ensure that revocations are not removed before they have been published, as doing so could defeat auditing and the detection of fraudulent transactions [14].

For a certificate to be accepted by Bob, he must have adequate proof of its validity as an assertion made by Alice as of a given time. It may contribute to this proof if the publisher, Charles, signs the association of credential validity status and the publication time. (This reduces the need for Bob to trust Alice not to replace her credential validity status data at a later time while lying about the time of the changed assertion. Again, this might be significant if Bob must justify to third parties decisions he takes based on certificates.) Depending on Bob's purposes, he may require that validity status information be signed by Alice herself rather than accepting it solely on Charles's authority. At minimum, to accept any revocation information on Charles's authority, we would expect Bob to require that Alice identify Charles as her publisher. This could effectively be done, for instance, if Alice maintains a minimal online presence, as suggested above. This would enable Alice to publish a single credential, signed by herself, positively identifying Charles (by his public key) as Alice's current publisher. If Alice should have to replace Charles, the credential would be replaced by one issued to Alice's new publisher and the revocation of the credential previously issued to Charles would be published by Alice's new publisher.

2.3 Data Integrity Verification

Whenever coalition data is hosted in untrusted locations, there is a possibility that it may be tampered with. Thus, in these contexts, there is a need for documents, web sites, and other content to be validated. But if this validation has a large overhead, there will be a temptation to circumvent the validation step. Therefore, we feel that data integrity verification is an ideal application for a distributed multi-author authenticated dictionary. In this case, the authors are the document or web site producers, who, in addition to distributing their content to various sites also publish digital digests of their content to the authentication system run by the publication repository. Whenever a client, Bob, accesses a document of web site, an authorization agent for Bob quickly verifies that the content of that document or web site has not been modified by anyone other than the author. The authorization agent for Bob makes a judgment about how fresh verification data must be to sufficiently limit the risk associated with relying on the credential. For example, the author of the status data, may provide a suggested "shelf life" for updates, which Bob may take into account in his determination. This expiration time data would be contained in the signed basis returned with the query response from a responder for Charles.

2.4 Credential Discovery in Permission Systems

In another application related to distributed authorization, the data structures discussed here can be used to support the discovery of credentials issued by a

given author [16]. In this case, the credentials themselves are stored in the multi-author authenticated dictionary, and their (verified) presence is a testimony to their validity. Should Alice require that a credential be revoked, she can remove it from the authenticated dictionary, thereby preventing its validity being verified after that time. Thus, status updates and discovery can be supported through the same structure.

The authorizer makes the final determination as to how recently valid a credential must be for its purposes. However, Alice may also provide a recommended "shelf life" for credentials she issues. When credential shelf life is sufficiently long in relation to the rate at which credentials are issued and expire, it may reduce Alice's cost to publish credentials for the purpose of supporting discovery in the archival structure discussed in Section 4.2 and to publish revocations separately in one of the structures that support validation of negative answers. In this case, after using the first structure to discover a credential, Bob would check the revocation structure to verify the credential's validity, as discussed in Section 2.2 above.

3 Some Simple-Minded Schemes

We begin our algorithmic discussion by pointing out two simple schemes, which provide two opposite ends of a spectrum of how much trust the authors place in the publication repository. Throughout our discussion of these simple schemes, as well as our improved schemes, we will refer to a typical data author as "Alice," we will refer to a typical client user as "Bob," and we will refer to the publication repository as "Charles." The first such solution we review is one in which a typical author, Alice, places little trust in the publication repository, Charles.

3.1 Independent Authentication

One possible solution for the many-author authenticated dictionary problem is for each author, Alice, to implement her own authenticated dictionary, e.g., using the scheme of Goodrich, Tamassia, and Schwerin [11]. In this case, Alice would maintain a hashed data structure, such as a tree or skip list, would make the updates in this data structure, and would sign and time-stamp a new "root" for this structure at every time quantum; this set of information is called the *basis*. But rather than distributing the updates and newly-signed basis to her own collection of responders, Alice would in this case simply forward the updates and basis to the repository, Charles. Charles would then maintain the same data structure as Alice, and would distribute the updates and new basis to the responders. In other words, Charles is in this case little more than a distribution channel to the responders. He does not sign anything—instead, he just passes the authors' signatures on. Authors, such as Alice, must maintain their own copies of the data structures they publish. On the positive side, this solution supports each user, Bob, to be able to verify the validity status, as of a given update date, of each credential issued by the author.

An author, Alice, signs the root hash of her data structure, together with the date, forming a sub-basis. The date and root hash of Alice's data structure are provided in the clear to the query issuer, Bob (the update consumer), together with the sub-basis and the rest of the authenticated dictionary proof. Bob checks that the signed root hash of the data structure is indeed the computed digest of the hash chain. Finally, Bob can decide whether the date is fresh enough for his purposes. Note that Bob does not have to trust the publisher, Charles, nor the responder, in this case.

The fact that Alice's data structure is incorporated into Charles's larger repository enables the author to verify through an efficient query to an untrusted responder whether all application data (such as credential status information) as of a given update time is being published, and when. For this the author, Alice, poses one query to a responder and compares the hash value at the root in the response to her own root hash. As soon as she finds a responder that answers queries using this root hash as the basis, Alice can verify that her entire update has been published.

There are unfortunately several disadvantages to this simple scheme:

- Each data author, Alice, must be sophisticated enough to implement a complete authenticated dictionary.
- There is a high likelihood that the publication repository, Charles, and his responders, will have to maintain many different kinds and versions of authenticated dictionaries, which makes for a cumbersome software maintenance plan.
- The time quanta for the different authenticated dictionaries are likely to be out of phase, which complicates the authentication guarantees provided by the system when many data authors are involved.

Let us therefore consider the other extreme.

3.2 Fully-Trusted Central Repository

At the other end of the trust spectrum is a scheme in which Alice and Bob fully trust the publication repository, Charles, to publish Alice's update accurately and promptly. In this case, it is sufficient for Charles to implement a single authenticated dictionary for all authors, which he then distributes out to the responders. Alice forwards updates to Charles and fully trusts him to perform those updates (be they insertions or deletions) on the published database. Data and trust is therefore aggregated at the publication repository, Charles. He implements a single authenticated dictionary for all the authors, and his responders answer queries using a proof basis that is signed only by him. The advantage of this scheme is that it has a simple software maintenance plan of keeping everything at the publication repository. This approach has several disadvantages, however, including the following:

- Having authors fully trust the publication repository may be unrealistic. In many applications, authors will at minimum require support for auditing

to detect incorrect omissions or additions to the repository that Charles maintains on Alice's behalf.
- Each user, Bob, must also fully trust Charles, since Charles's basis is the only cryptographic verification for a query's accuracy.
- The publication repository, Charles, becomes a single point of failure of trust, for compromising him compromises the security of the entire system.

Thus, there are significant drawbacks at both extremes of the spectrum of trust regarding the publication repository, Charles. Let us, therefore, consider some novel approaches that place only moderate trust in him.

4 A Multiply-Authored Authenticated Dictionary

In this section, we consider an intermediate solution, which maintains efficiency while requiring only moderate trust in the publication repository, Charles. We offer three versions of this intermediate solution, depending on the needs and abilities of Alice's repository.

In each solution, Charles stores a separate authenticated dictionary for each author, Alice, using a hierarchical hashing scheme (tree-based or skip-list-based) or an accumulator scheme. In each case a response is returned together with a validity *basis* that is signed by Alice and Charles. This basis contains the root hash value together with a time stamp indicating the most recent time quantum(s) in which this hash was considered by Alice and Charles (either together or separately) as the digest for all of Alice's data.

4.1 Minimally Compliant Authors

The first version we consider is the case of an author, Alice, who is minimally compliant with our approach. That is, Alice has minimal additional resources that she wishes to deploy for repository archiving. She may wish, for example, to maintain very little state. Indeed, Alice may not directly participate in the protocol at all, but instead indirectly participate by having a third-party agent pull her updates and transfer them to the publication repository.

In this case, the only state Alice needs to maintain is the current "root" hash of the authenticated dictionary for her data. Let $U = \{u_1, u_2 \ldots, u_k\}$ be a set of updates, that is, insertions and deletions, for Alice's data. The publication repository, Charles, upon receiving this sequence of updates, processes them in order one at a time. Each update, u_i, involves Charles searching a path in the authenticated data structure for Alice, updating that path to do the insertion or deletion, and updating the hash values along the path from the updated element to the "root" (possibly with local structural changes to maintain balance in the structure). Let P_i denote the search path prior to the update and let P_i' denote the search path after, including the hash values of nodes immediately adjacent to these paths. Then, as a receipt of this transaction, Charles returns to Alice the sequence $\mathcal{P} = \{P_1, P_1', P_2, P_2', \ldots, P_k, P_k'\}$. Alice (or her agent) can then verify

inductively that all the updates were performed correctly by starting with her cached hash value for the root (which should correspond to the root hash of P_1), and iteratively checking the computed hash value of P'_1, P_2, P'_2, and so on. Moreover, we require that path siblings be identical in P_i and P'_i, and that path roots be identical in P'_i and P_{i+1}. If all the hashes check out as being computed correctly, then Alice accepts that the updates have occurred, and she caches the root hash of P'_k as the new root hash. Note, in addition, that she can query a responder at any time to verify that this is also the root hash that is being used as the basis to her database for authenticated queries.

Charles and Alice will then mutually sign the new root hash, together with the current date, and this mutually-signed value will serve as the basis. Additionally, Charles can re-sign this root hash with each time quantum if there are no updates during that quantum, provided Alice has the ability to contact other authors in the case of her discovering that an update has been ignored by Charles. Otherwise, Alice will have to re-sign the root hash, along with Charles, in each time quantum.

In terms of the analysis of this scheme, there is a natural trade-off between the state that Alice is willing to maintain and the efficiency of the repository's receipt. For, in this minimal scheme, Alice need only maintain a cache of $O(1)$ size: namely, the root hash value. But, in order to validate a batch of k updates in this scheme, Charles needs to send Alice a receipt of size $O(k \log n)$, where n is the total size of Alice's database. We can design a scheme that is more efficient than this, however, if Alice's updates are all of a certain form.

4.2 Authors with Archive Data

A simplifying case of the multi-authored authenticated dictionary is when all updates involve only insertions and only positive answers to queries (i.e., the query element is present in the dictionary) need to be authenticated. Such a situation could arise, for example, in applications where authors wish to archive the digests of their documents, say, for time-stamping purposes. In the credentialing context, this case arises when credentials are valid for a fixed period and cannot be revoked.

In this case, Charles can store Alice's data in an authenticated tree or skip-list structure ordered by insertion time. Since such a tree is inefficient for searching, he should require that each responder maintain an alternative search structure, ordered by item keys, that maps an item in the key-ordered dictionary to its version in the authenticated data structure. This is needed because the hash values that produce the root digest signed by Charles and Alice are in the authenticated data structure ordered by insertion time.

Moreover, if Charles is operating a general time-stamping service for a whole collection of authors, he can store in a separate "super" data structure the sequence of all historical values of authors' authenticated data structures.

Let $U = \{u_1, u_2 \ldots, u_k\}$ be a set of updates, that is, a sequence of k insertions for Alice's data. Given such a sequence, the publication repository, Charles, adds the elements from this set of updates in the given order to the authenticated

data structure he is maintaining for Alice. Since this data structure is ordered by insertion time, these elements comprise a contiguous sequence of elements at the "end" of the data structure. Thus, the union of all that paths from these elements to the root consists of at most $O(k + \log n)$ nodes. Therefore, Charles can present to Alice a receipt that consists of the union of these paths, together with the hash values of adjacent nodes. This receipt can be checked as in the scheme described in the previous subsection, but it is much smaller in size. In particular, the insertion by Alice of k items requires a receipt of size only $O(k+\log n)$, rather than $O(k \log n)$, where n is the total size of Alice's database. If Charles is additionally maintaining a global archive of updates for m different authors, then there would be an additional portion of size $O(\log m)$ that would be added to the receipt Charles gives to Alice to verify that he has added her changes to the global archive.

Therefore, we are able to achieve an efficient receipt size for the case when all of Alice's updates are insertions, while keeping the state Alice must maintain to be constant. If Alice is willing to maintain state equal to the size of her database, however, we can achieve an efficient receipt size even in the case where updates consist of insertions and deletions.

4.3 Maintaining Limited State at an Author

Suppose that Alice is willing to maintain state equal to the size of her database. In this case, Charles can maintain Alice's authenticated dictionary as a hashed red-black tree (e.g., see [9]). The reason we choose a red-black tree, in this case, is that it has the property that updates and searches can be done in $O(\log n)$ time, but any update, be it an insertion or deletion, requires only $O(1)$ structural changes to the underlying binary search tree.

Let $U = \{u_1, u_2 \ldots, u_k\}$ be a set of updates, that is, a sequence of k insertions and deletions for Alice's data. In this case, Charles performs all of the updates in U on the authenticated red-black tree he is maintaining for Alice. He then sends back to her a receipt R that consists of all the *structural* changes he made to this tree, that is, the sequence of node insertions, removals, and tree rotations. They then both sign the final hash value of the root as the new basis. The total size of the receipt is $O(k)$. The reason this is sufficient is that, given the node changes dictated by Charles, Alice can recompute the hash value at the root of the tree in $O(k+\log n)$ time. Moreover, Alice doesn't even need to know that the binary tree is the underlying structure for a red-black tree. To her, it is simply a binary search tree with hash values stored at the internal nodes, much like a classic Merkle hash tree [17].

4.4 Common Themes

Thus, we have described three variations on an intermediate solution, where the publication repository, Charles, performs some computations on behalf of each data author, Alice, but does not fall in either extreme of taking complete control from Alice nor simply acting as a mere publication channel for Alice. Each of

our schemes have the basis derived from something signed by both Alice and Charles. Moreover, Alice's need only place minimal trust in Charles, for she can always pose as a user, Bob, and request an authenticated query from one of Charles's responders. If that response does not use the correct, up-to-date basis, Alice is free to reveal this breach of trust to the other users. Moreover, she can prove that Charles in error if he does not use the basis derived from her most recent updates, for she will have a signed receipt from Bob that can be used to show what the basis should be. Indeed, if Alice combines all the receipts she has received from Charles, she can prove the entire contents of what her database contains.

The three scenarios presented in this section are summarized in Table 1.

Table 1. Comparison of three scenarios for multi-authored authenticated dictionaries. All bounds are asymptotic, with the "big-Oh" omitted to simplify the notation

Scenario	Operations	State	Receipt Size	Verification Time
Min. Compliant Authors	ins. and del.	1	$k \log n$	$k \log n$
Authors w/ Archive Data	ins.	1	$k + \log n$	$k + \log n$
Authors w/ Limited State	ins. and del.	n	k	$k \log n$

5 Discussion, Conclusions, and Additional Issues

There are several additional issues that surround the publishing of authenticated data by many authors. We investigate several of these issues in this section.

5.1 Push versus Pull for Author Updates

When many authors have the ability to push updates to a publication repository, Charles, this site becomes potentially vulnerable to denial-of-service attacks. This danger is avoided in single-author version of an authenticated data structure by taking off-line the generation of the structure.

In the multiple-author context, we can reduce the risk of denial-of-service attacks against Charles by adding a subscription requirement to the model. That is, the authors who wish to publish information to Charles's repository must subscribe to the service, for which Charles could, for example, charge a fee based on usage. This economic model could reduce overwhelming numbers of updates sent to Charles in an attempt to deny other authors their right to publish.

Because we assume that the time quantum is defined by the publisher, we can strengthen the model further by switching author updates from a push approach to a pull approach. That is, each subscribed author, Alice, could be polled by

Charles (or an agent for Charles) in each time quantum for her updates. The net effect of using a pull approach would be that the freshness of a batch of updates per time quantum would be maintained, but now it would be impossible to flood Charles with a simultaneous stream of updates from many authors. Even a clearly frivolous sequence of updates from a subscribed author could be cutoff mid-stream by Charles in this pull setting.

5.2 User Subscription

In the system we have described, users pull validity proofs from responders. An alternative approach is to enable users to subscribe to updates on specific queries, which would be pushed out to them as they are published. In open systems, the field of potential users is so vast that pushing validity proofs is feasible only based on subscription. While a subscription may permit the consumer to avoid initiating a per-use check for freshness, it requires the user to trust the publisher to provide timely updates on changes in revocation status. This trust can be limited by requiring the publisher to provide a "heartbeat" (see [8]) at regular intervals to indicate that the subscribed-to credential remains valid. In our setting, where the user's trust in the publisher must be minimal, the heartbeat's content could be the validity proof itself. The heartbeat approach is optimized for prolonged user-resource interactions such as login sessions or continuous data feeds. However, the heartbeat approach builds in the limitation on freshness given by the heartbeat period, so the period must be relatively short. As such, it is not well suited to applications where authorization decisions are more sporadic. Moreover, the use of heartbeats begs the question, What does the user do when a fresh validity proof does not arrive or the proof received is not fresh enough to meet the user's needs? In these cases users should have the option of contacting different responders to obtain adequately fresh validity proofs.

5.3 Off-line Versus On-line

The authenticatable data structures we have presented are constructed off-line at periodic intervals defined by the publisher and then pushed out to untrusted responders. While the modest delay this introduces may slightly increase the user's vulnerability to an attack employing recently revoked credentials, it promises defenses against denial of service attacks that are far simpler than are available for on-line issuance and revocation services, where Byzantine failure and secure group communication must be addressed (see for instance [19]). As we have discussed, the time-quantum assumption allows the user protocol to employ untrusted, highly available responders and allows author updates to be pulled rather than pushed.

5.4 Credential Sensitivity

Credentials may be sensitive and therefore protected. In this case, authors should not have to entrust protection to publishers. Instead, the update should not con-

tain potentially sensitive credential content, but rather should contain only credential identifiers, such as a serial number. Thus, given a credential, the publication can be consulted to determine its validity status, while information content of the credential cannot be obtained from the publication.

5.5 Locating Responders

There are two parts to the issue of locating an appropriate responder: locating the publisher of the validity status of a given credential, and locating a responder for that publisher. In addition, a user would naturally desire a responder that is close in the network and lightly loaded with query requests being processed.

5.6 Conclusion

In this paper, we have studied the problem of disseminating an authenticated dictionary through an untrusted or partially trusted publisher. We have considered several architectural issues involved with such publication, including the necessity of receipts, the efficiency of receipts, and methods for pushing updates. In this context, we have presented efficient algorithmic solutions for a variety of levels of trust in the publisher. We have additionally provided an extremely efficient solution for the case when data removal need not be supported.

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. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote trust-management system, version 2. IETF RFC 2704, Sept. 1999.
3. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, May 1996.
4. 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.
5. P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. In *Proc. ACM Conference on Computer and Communications Security*, 2001.
6. P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.
7. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. IETF RFC 2693, Sept. 1999.
8. E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti. dRBAC: Distributed role-based access control for dynamic coalition environments. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. IEEE Computer Society, July 2002.

9. M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis and Internet Examples*. John Wiley & Sons, New York, NY, 2002.
10. M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In *Proc. Int. Security Conference (ISC 2002)*, volume 2433 of *LNCS*, pages 372–388. Springer-Verlag, 2002.
11. 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.
12. M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Proc. RSA Conference, Cryptographers Track (RSA-CT)*, volume 2612 of *LNCS*, pages 295–313. Springer-Verlag, 2003.
13. P. Kocher. A quick introduction to certificate revocation trees (CRTs), 1998. http://www.valicert.com/resources/whitepaper/bodyIntroRevocation.html.
14. N. Li and J. Feigenbaum. Nonmonotonicity, user interfaces, and risk assessment in certificate revocation. In *Proceedings of the 5th Internation Conference on Financial Cryptography (FC'01)*, volume 2339 of *Lecture Notes in Computer Science*, pages 166–177. Springer-Verlag, 2001.
15. N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2002.
16. N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, Feb. 2003.
17. R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO '89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1990.
18. M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.
19. L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems (TOCS)*, 20(4):329–368, Nov. 2002.