Accountable Certificate Management using Undeniable Attestations

Ahto Buldas
Küberneetika AS
Akadeemia tee 21
12618 Tallinn, Estonia
ahto.buldas@cyber.ee

Peeter Laud
Universität des Saarlandes
FB 14 Informatik
Im Stadtwald - Bau 45
Postfach 15 11 50
66041 Saarbrücken, Germany
laud@cs.uni-sb.de

Helger Lipmaa
Helsinki University of Technology
Department of Computer Science
FI-02015 HUT, Espoo, Finland
helger@tml.hut.fi

ABSTRACT

This paper initiates a study of accountable certificate management methods, necessary to support long-term authenticity of digital documents. Our main contribution is a model for accountable certificate management, where clients receive attestations confirming inclusion/removal of their certificates from the database of valid certificates. We explain why accountability depends on the inability of the third parties to create contradictory attestations. After that we define an undeniable attester as a primitive that provides efficient attestation creation, publishing and verification, so that it is intractable to create contradictory attestations. We introduce authenticated search trees and build an efficient undeniable attester upon them. The proposed system is the first accountable long-term certificate management system. Moreover, authenticated search trees can be used in many security-critical applications instead of the (sorted) hash trees to reduce trust in the authorities, without decrease in efficiency. Therefore, the undeniable attester promises looks like a very useful cryptographic primitive with a wide range of applications.

Keywords

accountable certificate management, authenticated search trees, attesters, long-term authenticity, non-repudiation, public-key infrastructure, search trees, time-stamping

1. INTRODUCTION

The concept of public-key cryptography was created in an effort to solve the cryptographic key management problem [11]. While giving an answer to many difficult problems, public-key cryptography also raised several of its own. Not surprisingly, one of the main problematic areas to be solved before the public-key cryptography can be successfully applied in practice is still the key management.

Efficient and accountable identity-based certificate management is necessary (in particular, but not only) to support authenticity of digital documents with a long lifetime. A body of supporting methods for long-term authenticity was developed in the area commonly known as digital time-stamping [13]. Recent work in time-stamping has also shown how to build efficient yet accountable time-stamping systems [6, 16, 7] with minimal trust in the third parties. However, one has to complement the techniques of accountable time-stamping with methods from other areas of applied cryptography to support long-term authenticity and non-repudiation. One of such areas is accountable efficient certificate management. Unfortunately, cryptographic literature has only briefly treated the question of how to achieve the latter [8].

First, we present informal motivation and definition of accountable certificate management, where every validity change of a certificate is accompanied by a transferable attestation ascertaining this act, and a short digest of the (current state of) database of valid certificates is periodically published. In Section 2, we argue *informally* that a certificate management system is accountable if and only if it is intractable for anybody to create a pair of contradictory attestations, so that a certificate would be accepted as valid or not, depending on which certificate is in the possession of the verifier. Under this intractability assumption, our certificate management system has several desirable properties, including that every dispute in court can be solved by the present evidence. Moreover, one can verify the certificate validity at some moment, based only on a short digest of the certificate database, a short certificate-specific attestation and the certificate itself. The rest of the paper focuses on this assumption.

In Section 4, we give formal definition of a new primitive called *undeniable attester*. Informally, an attester is a triple (P, D, V) of algorithms, such that

- The proving algorithm P, given a candidate string x and a set S, outputs an attestation certifying whether $x \in S$.
- The digest algorithm D, given a set S, outputs a short digest $d = D(S)$ of it.
- The verification algorithm V is given a candidate element x, a digest d, and an attestation p. V accepts or rejects depending on whether x belongs to a set with digest d.

We call an attester *undeniable* if it is intractable to generate a digest d, an element x and two attestations p and $\overline{p}$ such that $V(x, d, p)$ accepts but $V(x, d, \overline{p})$ rejects.

Before giving formal definitions, in Section 3 we survey some attesters whose subsystems were considered previously in the certificate management and the public-key infrastructure. In particular, we review attesters based on certificate revocation lists, hash

CCS '00, Athens, Greece.

trees [17], certificate revocation trees [15, 20], certificate revocation system [19], and RSA accumulators [5, 2]. However, since most of the mentioned systems have not been designed with accountability in mind, they all have some implicit trust assumptions. As a result, we conclude in Section 4 that the only known undeniable attester is the trivial one (similar in efficiency to the certificate revocation lists) with attestation lengths $\Theta(|S| \cdot \log |S|)$.

A good example of an attester that is *not* undeniable is the sorted hash tree attester, defined in Section 5. The sorted hash tree attester is based on an efficient construction similar to the certificate revocation trees. We show in 5 that the sorted hash tree attester is not undeniable and therefore a sorted hash tree attester-based certificate management system makes it possible for the CA to cheat the clients. Until now the certificate revocation trees had no known weaknesses.

In Section 6, we propose a very simple efficient *authenticated search tree*-based construction of undeniable attesters that we call an *authenticated search tree attester*. The key difference between the sorted hash tree attester and the proposed construction is that the authenticated search tree attester assigns to every internal node v of a search tree a hash value $S[v]$, taken over the labels of v's children *and* the search keys of v. (In the sorted hash tree attester, $S[v]$ did *not* authenticate the search keys of v.) Moreover, the authenticated search tree attester is in several aspects more intuitive than the sorted hash tree attester: Being directly based on the search trees as they are generally understood in computer science, it allows us to carry over to cryptography the research done in the area of algorithms and data structures [14].

After defining the new attester, we will prove that it is undeniable. As with any new cryptographic primitive—and undeniable attester *is* a new primitive—it is good to know how it relates to the previously known primitives. As such, the proposed undeniable attester might have surprisingly wide applications in different security applications (and not only purely in certificate management).

A proof that undeniable attesters exist if and only if collision-resistant hash functions exist is presented in Section 6. In Section 7, we will provide efficiency analysis of the authenticated search tree attester. The attestation compressing method given in Section 7 might be of independent interest. We show that the attestations in the authenticated search tree can be compressed, ideally, by a factor of 2; This makes authenticated search tree attester almost as space efficient as the sorted hash tree attester. While this method is straightforward, the authors are unaware of any previous constructions that use the same technique to compress search trees. Moreover, it is unusual to apply standard compression methods to make cryptographic primitives more space efficient.

2. MOTIVATIONS

Our research is motivated by the observation that for the long-term authenticity and non-repudiation of digital documents, new methods are necessary for verifying whether identity certificates (bounds between a person and a signature key) were valid at some moment of time. Since many digitally signed documents (e.g., loan agreements) may have important legal value for decades, these methods have to ensure that the validity information of certificates cannot be forged by anybody, including the authorities.

We aim at the construction of an accountable certificate management system, where all forgeries by the third parties can be explicitly proven and all false accusations explicitly disproven [7]. More precisely, we would like the only part of the certificate management (physical visit of a person to an authority) that clearly cannot be mathematically modeled also be the only stage in the system that needs some non-cryptographic solution (i.e., involving physical presence of a client-chosen notary) to the trust problems.

Now, long-term certificate validity can only partially be ensured by the methods of time-stamping [13] only, where absence of a proof that a certificate was issued is implicitly counted as the proof of its nonexistence. Such an assumption is clearly undesirable in many situations. We would like to have not only explicit positive attestations stating that valid certificates are valid, but also explicit negative attestations stating that non-valid certificates are not valid. In this way, all disputes regarding the validity of a certificate could be solved based on the present evidence (a positive or a negative attestation), given that it is intractable for anybody to create a pair of contradictory attestations.

From now on, we will work in the setting where the Certificate Authority (the CA) maintains a dynamic database S of valid certificates. (See [24, 12] for argumentation why a database of valid certificates is better than a database of revoked certificates. In our case, the database of revoked certificates would just add unnecessary complexities to the system. Presence of a central authority lessens the communication complexity of the scheme and simplifies the tracking of the origins of frauds). Our model also includes the Publication Authority [7] and a (possibly huge) number of clients.

We assume that every client receives a positive (resp. negative) attestation from the CA if her certificate belongs (resp. does not belong) to the database. This assumption is not restricting, since some sort of attestation—or receipt—is returned to the client by every CA. In our system, it is in the client's own interest to store the attestation so that he can later explicitly prove or disprove the validity of his certificate at some time. Additionally, everyone can make membership queries of type "$x \in S$" to the CA, who then returns an attestation. Clients who want later to use an attestation p of "$x \in S$" (or of "$x \notin S$") as evidence in court, should obtain it from the CA in some suitable time-frame. (This is very similar to what is done in time-stamping [6, 16].)

A digest of the database S (denoted as $D(S)$) is published by the Publication Authority in some authenticated and widely available medium by using accountable publishing protocols [7]. (Motivations behind this are the same as in time-stamping [13, 6, 7]. First, without authenticated information about the database, the CA can easily create contradictory attestations. Second, long-term authenticity should not depend on the security of private keys [13]. Publishing the digest is the most natural and widely accepted solution in digital time-stamping to achieve the long-term authenticity.)

Third, nobody should be forced to store all the copies of the dynamic database S: The system should still be accountable, if the verifier does not have anything more than an element, a short attestation and a short digest of the database. This is again very similar to the situation in time-stamping, where clients can verify a time stamp, given only the time stamp (equivalent to the attestation), the round stamp (equivalent to the digest), and the candidate element itself [13, 6].

We show that (our model of) accountable certificate management incorporates at least three different algorithms. Motivated by this, we define a new primitive, *attester*, to be a triple (P, D, V) of algorithms. A proving algorithm P, given a candidate string x and a set S, outputs an attestation. A digest algorithm D, given a set S, outputs a short digest $d = D(S)$ of the database. Finally, a verification algorithm V takes as an input a candidate element x, a digest d, and an attestation p, and accepts or rejects depending on whether x belongs to a set specified with digest d.

In the described model, the CA cannot cheat a client. (Note that we assume that the Denial of Service attacks, where the CA does not return an attestation to the client, can be prevented (say) by letting a client-chosen notary to participate in handing over the attestation.) That is, if a client has a (say) positive attestation that his certificate belonged to the database of valid certificates at some time, the CA has no means to generate a contradictory attestation, claiming that the same certificate was not in this database, assuming that the CA is not able to break some underlying cryptographic primitives. More formally, we call an attester *undeniable*, if it is intractable to generate a set S, an element x and two attestations p and $\overline{p}$ such that $V(x, D(S), p)$ accepts but $V(x, D(S), \overline{p})$ rejects.

For the long-term authenticity undeniability is crucial, e.g., when the CA who issued a concrete certificate might have (say) gone bankrupt long before the verification act, so that it is impossible to sue her for cheating. Moreover, if a client has accidentally deleted his attestation, he can at least be sure that nobody else can sue him, based on the contradictory attestation. These properties will significantly increase the trustworthiness of the CAs.

Separation of Duties

Functions of the CA should be divided between at least two authorities, an off-line CA, and an on-line Validation Authority, as it is done also in many other certificate management systems [8]. However, while the distinction between the CA and the Validation Authority is important in practice, it is not a subject of this paper: Since our methods help to prevent forgeries even in the case when one possibly misbehaving party (the CA) has control over the whole system, it also prevents forgeries if there are several third parties. For simplicity, in this paper we will not stress the separation between the authorities. For the same reason, we do not elaborate on the accountable publication protocols but rather refer the reader to [7] for necessary information.

3. SOME KNOWN CONSTRUCTIONS

Next, we will give a short survey of some attesters based on previously proposed ideas. We will briefly explain why those attesters fail to satisfy our requirements.

3.1 List Attester

For any x and a set S, attestation $P(x, S)$ is equal to S (i.e., to the whole set), with length $|P(x, S)| = \Theta(|S| \log |S|)$. The digest $D(S)$ is equal to a short (say) k-bit hash $H(S)$ of S, where H is a collision-resistant hash function. k is also called the *security parameter*. The verification algorithm V, given $S = P(x, S)$, $d = D(S)$ and x, accepts if and only if $d = H(P(x, S))$ and $x \in P(x, S)$. The resulting construction is clearly undeniable.

Unfortunately, the list attester becomes utterly inefficient if the number of simultaneously valid certificates grows, since both storage requirements and verification time are at least linear in $|S|$. One of the possibilities to decrease the verification time is to assume that the CA has sorted the database. Although then the clients can perform a binary search in the database, the attester will cease to be undeniable since the CA may leave the database unsorted. This method would also not reduce the storage requirements.

3.2 One-time Signature Attester

A more efficient attester can be based on one-time signatures (in the context of the public key infrastructure, this idea was proposed in [19] and later refined in [1]), description of which we omit. The one-time signature attester provides both succinct positive and negative attestations with $|P(x, S)| = \Theta(k)$, where k is the security parameter. However, this solution has $D(S) = S$ and therefore results in a completely impractical publishing overhead. See [19, 1] for more information.

3.3 RSA Attester

The RSA attester can be in a natural way built upon the RSA accumulator [5, 2]. Here, the positive attestations have the form

$$P(x, S) = z^{y_1 \cdots y_m} \mod n$$

for some $y_1, \ldots, y_m$, and therefore the attestation length is $\Theta(k)$, where k is again the security parameter. The digest has the same form and therefore also the same length. However, as first pointed out by Nyberg [22, 23], length of the attestations is reduced by introducing built-in trapdoor information known to some coalition of participants, which should therefore be trusted. The best known method [25] of making the RSA accumulator trapdoorless introduces attestation lengths of order $\Theta(k^2)$. Since $k \geq 128 > \log |S|$, the trapdoorless RSA accumulator has longer attestations than the sorted hash tree attester (the latter is described below). Moreover, the negative attestations are all equal to S itself.

3.4 Hash Tree Attester

Hash trees [17] are widely used to authenticate an element as a set member. In the full generality, the hash tree is a labeled tree, with the leaves labeled by different values $x \in S$ and internal nodes labeled by the hash over their children labels, where a fixed collision-resistant hash function is used.

In the *hash tree attester*, a positive attestation consists of the minimal amount of data, necessary to verify the hash path from the leaf labeled by x to the root. We assume that the used hash trees have depth logarithmic in the number of nodes. As a result, the positive attestations have length $\Theta(k \log |S|)$, where k is again the output length of the used collision-resistant hash function. The digest $D(S)$ of length $\Theta(k)$ is equal to the label of the root.

3.5 Sorted Hash Tree Attester

A serious drawback of the simple hash tree construction is that negative attestations are still equal to the whole database S. However, similarly to the case of the list attester, hash tree attester can be made more efficient if the CA sorts the leaves (an idea only recently proposed in [15, 20]). The resulting *sorted hash tree attester* has both negative and positive attestations with length $\Theta(k \log |S|)$ and is therefore succinct. However, as also in the case of (sorted) list attester, the proposed solution hides in itself an implicit assumption that the CA dutifully sorts the leaves. A corrupted CA may easily build an unsorted hash tree without being detected by anyone who does not possess a copy of the whole S. We give an example of that in Section 6.

4. FORMAL DEFINITIONS

4.1 Preliminaries

Let $\Sigma = \{0, 1\}$. As usually, Σ^k denotes the set of k-bit words, $\Sigma^* := \bigcup_{k \geq 0} \Sigma^k$. From now on, k denotes the security parameter, relative to which the security of various schemes is measured. We assume that nil is a special symbol, encoded differently from any $x \in \Sigma^*$. Let $\mathcal{EA}$ be the class of probabilistic algorithms with execution time that is polynomial in the length of their input. A probability family $\mathcal{P} = (\mathcal{P}_k)$, $k \in \mathbb{N}$, is *negligible* if for all $\varepsilon > 0$ there exists a k_ε, such that $\mathcal{P}_k < k^{-\varepsilon}$, for any $k > k_\varepsilon$. Notation $X \leftarrow \mathsf{S}$ means that X is assigned according to the probability space S that may be the output space of some probabilistic algorithm.

A *collision-resistant hash function* (CRHF) $\mathcal{H}$ for some index set $I \subseteq \Sigma^*$ is a pair (G, H), such that (1) $G \in \mathcal{EA}$ is a *generation algorithm*, such that $G(1^k) \in \Sigma^k \cap I$; (2) For an index $i \in I$, $H(i, \cdot) = H_i(\cdot)$ is a function $H_i : \Sigma^{p(|i|)} \to \Sigma^{|i|}$, such that $H \in$

Attester name	Security of the succinct version	Digest length	Positive attestation length	Negative attestation length
List	Attester	$\Theta(k)$	$[\Theta(n \log n)]$	$[\Theta(n \log n)]$
One-time Signature	Attester	$[\Theta(n \log n)]$	$\Theta(k)$	$\Theta(k)$
RSA	Collision-Resistant Prover	$\Theta(k)$	$\Theta(k)$	—
Hash Tree	Collision-Resistant Prover	$\Theta(k)$	$\Theta(k \log n)$	—
Sorted Hash Tree	Collision-Resistant Attester	$\Theta(k)$	$\Theta(k \log n)$	$\Theta(k \log n)$
Authenticated Search Tree	Undeniable Attester	$\Theta(k)$	$\Theta(k \log n)$	$\Theta(k \log n)$

Table 1: Some known succinct attesters, i.e., security is given only for the succinct versions (see Section 4.3). For example, while the list attester is an undeniable attester, it is only a succinct attester. Here $n = |S|$, and $k > \log n$ is the security parameter.

$\mathcal{EA}$, for some polynomial p, where $p(k) > k$; (3) For all algorithms $A \in \mathcal{EA}$, the probability family $\mathsf{CRH}_{\mathcal{H}}(A)$ is negligible in k, where

$$\mathsf{CRH}_{\mathcal{H},k}(A) := \Pr[i \leftarrow G(1^k), (x_1, x_2) \leftarrow A(1^k, i) : x_1 \neq x_2 \wedge H_i(x_1) = H_i(x_2)] \ .$$

4.2 Definition of Attester

We have already given informal definitions of attesters. Next, we go on with the full formalism followed by discussions. Just note that in the definition of attesters the role of generating function and indices is the same as in the definition of hash functions. Namely, they are not necessary unless we discuss strong security properties like collision-resistancy and undeniability (defined later in this Section).

Definition 1. A quadruple $\mathcal{A} = (G, P, D, V)$ is an *attester* for an index set $I \subseteq \Sigma^*$, if there is a polynomial p, $p(k) > k$, such that

1. A *generating algorithm* $G \in \mathcal{EA}$ takes as input a security parameter 1^k and outputs an index $i \in \Sigma^k \cap I$.
2. A *proving algorithm* $P \in \mathcal{EA}$ takes as input an index i, an element $x \in \Sigma^k$ and a set $S \subseteq \Sigma^k$, $|S| \leq p(k)$ and outputs an *attestation* $P_i(x, S) = P(i, x, S)$.
3. A *digest algorithm* D takes as input an index i, a set $S \subseteq \Sigma^k$, $|S| \leq p(k)$ and outputs a digest $D_i(S) = D(i, S)$.
4. A *verification algorithm* V takes as input an index i, a candidate element $x \in \Sigma^k$, a digest d and an attestation p and outputs
$$V_i(x, d, p) = V(i, x, d, p) \in \{\mathsf{Accept}, \mathsf{Reject}, \mathsf{Error}\} \ .$$
We require that for any $S \subseteq \Sigma^k$ with $|S| \leq p(k)$, and for any $x \in \Sigma^k$, $V_i(x, D_i(S), P_i(x, S))$ outputs Accept if $w \in S$ and Reject, otherwise. If $i \notin \Sigma^k \cap I$, $S \not\subseteq \Sigma^k$, $|S| > p(k)$ or $x \notin \Sigma^k$, then for any p, $V_i(x, D_i(S), p) = \mathsf{Error}$.

In practice, we want attesters to have "succinct" attestations and digests but also fast (average-case) update time. Informally, we say that an attester is *dynamic* if (average-case) time per insertion and deletion of elements is $O(k \log |S|)$. We say an attester is *succinct* if $|D_i(S)| = O(|i|)$ and $|P_i(x, S)| = O(|i| \cdot \log |S|)$. Note that by definition, any attester has $|D_i(S)| = |P_i(x, S)| = |i|^{O(1)}$.

Definition 2. Let $\mathcal{A} = (G, P, D, V)$ be an attester. Let

$$\mathsf{CRP}_{\mathcal{A},k}(A) := \Pr[i \leftarrow G(1^k), (x, S, p) \leftarrow A(1^k, i) : x \notin S \wedge V_i(x, D_i(S), p) = \mathsf{Accept}] \ ,$$

$$\mathsf{CRD}_{\mathcal{A},k}(A) := \Pr[i \leftarrow G(1^k), (x, S, \overline{p}) \leftarrow A(1^k, i) : x \in S \wedge V_i(x, D_i(S), \overline{p}) = \mathsf{Reject}]$$

and

$$\mathsf{UN}_{\mathcal{A},k}(A) := \Pr[i \leftarrow G(1^k), (x, d, p, \overline{p}) \leftarrow A(1^k, i) : V_i(x, d, p) = \mathsf{Accept} \wedge V_i(x, d, \overline{p}) = \mathsf{Reject}] \ .$$

Attester $\mathcal{A}$ is a *collision-resistant prover* (resp. *collision-resistant disprover*) if $\forall A \in \mathcal{EA}$, $\mathsf{CRP}_{\mathcal{A}}(A)$ (resp. $\mathsf{CRD}_{\mathcal{A}}(A)$) is negligible. $\mathcal{A}$ is a *collision-resistant attester* if for any $A \in \mathcal{EA}$, both $\mathsf{CRP}_{\mathcal{A}}(A)$ and $\mathsf{CRD}_{\mathcal{A}}(A)$ are negligible. $\mathcal{A} = (G, P, D, V)$ is *undeniable* if for any $A \in \mathcal{EA}$, $\mathsf{UN}_{\mathcal{A}}(A)$ is negligible.

4.3 Discussions

It is important to understand the (seemingly subtle but crucial in applications) difference between collision-resistant and undeniable attesters. Collision-resistant attesters assume that a verifier has access to the correctly calculated value $D_i(S)$. In practice, it means that she either has to rely on some trusted third party or has to have access to S herself. Both possibilities are undesirable in many security applications, including accountable certificate management. Undeniable attesters stay secure even in the presence of an adversary who forges the digest, and therefore potentially provide much higher level of confidence in the system.

Table 1 summarizes the properties of attesters described previously in Section 3, together with authenticated search tree attesters described later in Section 6. Note that the hash tree attester and the RSA attester are not succinct, since they have negative attestations of length $\Theta(|S| \cdot \log |S|)$. However, one can easily modify both attesters to be succinct, by defining $P_i(x, S)$ to be equal to some fixed constant for all $x \neq S$. Both the (modified) hash tree attester and the (modified) RSA attester are succinct collision-resistant provers. A similar trick does also work with the list attester and the one-time signature attester, but the resulting succinct constructs will only be attesters without satisfying any stronger security requirements.

As emphasized in Section 2, in accountable certificate management we are interested in undeniable attesters. However, as seen from the table, none of the previously known attesters is undeniable. The second main result of this paper—the first one being a model for accountable certificate management—is the description of authenticated search tree attester in Section 6 with accompanying proof that this attester is undeniable. To stay self-contained, we will now first describe the sorted hash tree attester and explain why it is not undeniable.

5. SORTED HASH TREE ATTESTER

In the following, we give a more precise description of the sorted hash tree attester, based on the hash tree attester (see Section 3) that is by itself a collision-resistant prover but not a collision-resistant

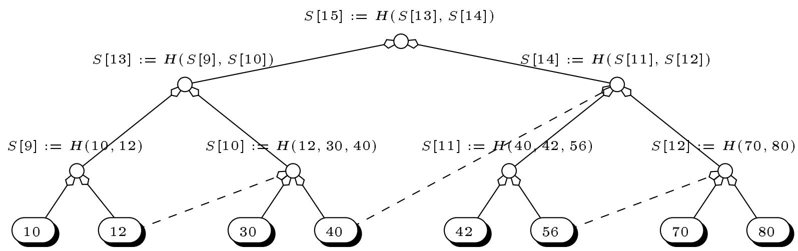

Figure 1: A toy example of the sorted hash tree attester. Dashed lines are present only in the improved sorted hash tree attester, described in Section 5.2. The values $S[v]$ are given for unimproved construction.

disprover. This holds since a candidate string x can be a label of any leaf, and therefore a negative attestation should incorporate all positive attestations. To understand it, think of searching from an unsorted database S. Showing that x belongs to S is accelerated by presenting an index j (an attestation) of x's occurrence, followed by checking that the jth element is equal to x. However, if x does not belong to the database, one has to verify for *each* j that the jth element is not equal to x. Sorting the data will make also this system more efficient. In the special case of hash trees, we assume that the values stored at the leaves are sorted from left to right. This assumption results in shorter lengths of negative attestations.

5.1 Construction

The next attester (G, P, D, V) is based on a fixed CRHF $\mathcal{H} = (G_{\mathcal{H}}, H)$. The only role of the generating function G in this attester is to choose a concrete hash function H_i from this family, according to the function $G_{\mathcal{H}}$. Therefore, for the sake of simplicity, we will describe attesters for a fixed $i \in \Sigma^k \cap I$ and for a fixed hash function $H = H_i$. The latter can in practice be instantiated with SHA-1 [21] or any other strong (keyed) hash function.

Next, suppose that $S = \{S[1], \dots, S[n]\}$ is a nonempty set of k-bit integers such that $S[j] < S[j+1]$ for any $1 \le j < n$. Let T be a (directed) binary tree with n leaves, with its jth leftmost leaf labeled by $S[j]$ (Figure 1). A non-leaf vertex $v \in T$ is labeled by an auxiliary hash value

$$S[v] = H(S[v_{\mathsf{L}}], S[v_{\mathsf{R}}]) \ ,$$

where v_{L} (v_{R}) denotes the left (right) child of v. The digest $d = D(S)$ of S is equal to the label of the root vertex v, or to Error, if the leaves were unsorted.

Let $p = (b; h_1, h_2, \dots, h_m)$, such that $h_j \in \Sigma^k$ and $b = b_1 \dots b_m$, $b_j \in \{0, 1\}$. The verification algorithm $V(x, d, p)$ returns Error if p does not have such form. Otherwise, V computes d_m by assigning $d_0 := x$ and then recursively, for every $j > 0$,

$$d_j := \begin{cases} H(d_{j-1}, h_j), & \text{if } b_j = 0 \ , \\ H(h_j, d_{j-1}), & \text{if } b_j = 1 \ . \end{cases}$$

Verification returns Accept, if $d_m = d$, and Error, otherwise. If $x \in S$, the proving algorithm P returns a p such that $V(x, d, p)$ accepts. Proving that $x \notin S$ is equivalent to finding a quadruple (x_1, p_1, x_2, p_2), such that

$$V(x_1, d, p_1) = V(x_2, d, p_2) = \mathsf{Accept} \ ,$$

$x_1 < x < x_2$, and x_1 and x_2 correspond to two neighboring leaves in the tree T. If x is smaller than the least element x_1 of S, we can define $P(x, S)$ to be equal to $P(x_1, S)$. The situation when x is bigger than the greatest element of S is dealt with analogously.

Looking at the tree depicted in Figure 1, $D(S) = S[15]$,

$$P(30, S) = (\mathtt{101}; 40, S[9], S[14])$$

and $P(35, S) = (P(30, S), P(40, S))$. On the other hand,

$$P(8, S) = P(10, S) = (\mathtt{111}; 12, S[10], S[14]) \ .$$

5.2 Further Efficiency Improvements

One can further shorten the negative attestations by inserting additional arcs to the underlying tree as follows (slightly different methods were also proposed in [15, 20]): If the parents of a leaf $v \neq 1$ and its left neighbor leaf w are different, then add an arc from w to v's parent, as in Figure 1. Build an attester upon the resulting graph, by modifying the algorithms P, D and V to account with the new arcs. Let the negative attestation of x be equal to the positive attestation of the smallest $x' > x$ in set S if such x' exists, or of the x, otherwise. As the result, both negative and positive attestations will have the same length.

5.3 Sorted Hash Tree Attester is not Undeniable

Sorted hash tree attester is succinct, dynamic (if built upon dynamic trees) and collision-resistant. However, it is not undeniable. We show this by the example depicted in Figure 2. There, the positive attestations of 10, 40 and 20 are respectively $p_1 = (\mathtt{11}; 40, S[6])$, $p_2 = (\mathtt{01}; 10, S[6])$ and $p_3 = (\mathtt{10}; 30, S[5])$. However, (p_1, p_2) is also a negative attestation of 20. Therefore, a verifier, given the digest $S[7]$ (root of the hash tree), accepts or rejects 20 depending on which attestation was earlier submitted to her.

Such "unsorting" attack is possible since there is no efficient way for the verifier to check whether the CA dutifully sorted the database. The only (obvious) possibility to prevent this attack, without involving another trusted third party, is to send all database elements of total size $|S| \cdot \log |S|$ to the verifier. The verifier would then recompute the hash tree, verifying that this database in the sorted order results in digest d, obtained by her beforehand from a reliable source. However, such solution is clearly impractical if $|S|$ is big, since the verifier has to do $|S| - 1$ hash computations per every verification. Moreover, such a solution is impossible if some elements in the database are inaccessible (if, to lessen the storage requirements, the old versions of the certificate database are not stored).

6. AUTHENTICATED SEARCH TREE ATTESTER

Next, we give a construction of what we call *authenticated search trees*. After that we show that the resulting attester (*authenticated search tree attester*) is an undeniable attester, and finish the section with some discussions. First, let us remember that a directed binary tree T is a *search tree* [14, Section 6.2.2] if every node $v \in T$ has a unique *search key* $K[v]$ associated to it, such that if w is the left (resp. right) child of v, then $K[w] < K[v]$ (resp. $K[w] > K[v]$).

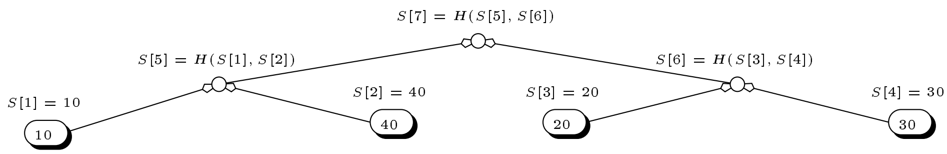

Figure 2: A toy example of improperly created sorted hash tree attester.

6.1 Construction

We give, as in Section 5, a construction for fixed k and for fixed $i \in \Sigma^k \cap I$. Let $S \subset \Sigma^k$ be a nonempty set and let T be a binary search tree with $|S|$ vertices. Each vertex v of T is labeled by a pair $(K[v], S[v])$. Here, the elements $K[v]$ belong to the set S and $K[v_1] \neq K[v_2]$, if $v_1 \neq v_2$. Moreover, the tree T together with keys $K[v]$ is a search tree. The value $S[v]$ is equal to

$$S[v] := H(S_{\mathsf{L}}, K[v], S_{\mathsf{R}}) \enspace ,$$

where S_{L} (resp. S_{R}) is equal to the label $S[\cdot]$ of the v's left (resp. right) child if the corresponding child exists, or to nil, otherwise. For example, if v is a leaf, then $S[v] = H(\mathsf{nil}, K[v], \mathsf{nil})$. Once again, the digest $D(S)$ is defined as $S[v]$, where v is the root vertex, or as Error, if T is not a proper search tree.

For a $x \in S$ (resp. $x \notin S$), the attestation $P(x, S)$ is defined as the least amount of data, necessary to verify that $K[v] = x$ for some v (resp. $K[v] \neq x$ for any v, given that T is a proper search tree). Intuitively, following an attestation of $x \in \Sigma^k$ is equivalent to searching x from a search tree, where the usage of hash functions in the vertices guarantees that the CA has to work with the same tree during each query. Moreover, the verification algorithm V returns Error if the tree is not found to be a proper search tree.

The rest of this subsection gives a more technical definition of the authenticated search tree attesters, including the necessary (local) verifications that T is a search tree. *It is necessary to perform these verifications for the authenticated search tree attester to be undeniable, and therefore to avoid any frauds.*

Let

$$p = (h_{\mathsf{L}}, k_0, h_{\mathsf{R}}; k_1, h_1; k_2, h_2; \dots ; k_m, h_m) \enspace ,$$

where all the elements are from Σ^k, and $m \geq 0$. The verification algorithm $V(x, d, p)$ returns Error if (1) $h_{\mathsf{L}} \neq \mathsf{nil}$ and $x < k_0$, or (2) $h_{\mathsf{R}} \neq \mathsf{nil}$ and $x > k_0$. Naturally, V also returns Error if the attestation p does not have the specified form. Otherwise, V assigns $d_0 := H(h_{\mathsf{L}}, k_0, h_{\mathsf{R}})$ and for all $0 < j < m$,

$$d_j := \begin{cases} H(d_{j-1}, k_j, h_j) & \text{if } x < k_j \enspace , \\ H(h_j, k_j, d_{j-1}) & \text{if } x > k_j \enspace . \end{cases}$$

After that, V outputs Error if

(ST1) $d_m \neq d$, or

(ST2) for some j, $x = k_j$ or $k_{j-1} = k_j$.

Otherwise, V returns Accept or Reject, depending on whether $k_0 = x$.

If $x \in S$, the algorithm $P(x, S)$ returns the unique list p such that $V(x, D(S), p)$ accepts. If $x \notin S$, $P(x, S)$ finds (1) An element x', such that x' is the greatest element $x' \leq x$ (the *predecessor* of x), if such exists, or the smallest element in S, otherwise; (2) An element x'', such that x'' is the smallest element $x'' \geq x$ (the *successor* of x), if such exists, or the greatest element in S, otherwise.

By the construction of search trees, either $x'' = K[v'']$ for some node v'' on the root path starting from the node with sorting key x', or vice versa. (Otherwise $P(x, S)$ returns Error) $P(x, S)$ returns the unique list p such that $V(x''', D(S), p)$ accepts, where $x''' = x'$ in the first case and $x''' = x''$ in the second case.

Clearly, $V(x, D(S), p)$ accepts if and only if $x \in S$. Note that the verification (ST2) returns Error only if the tree fragment, reconstructed from p, cannot be a part of a search tree.

A toy example with $S = \{10, 12, 30, 40, 42, 56, 70, 80\}$ is depicted in Figure 3. Here, $D(S) = S[4]$ and

$$\begin{aligned} P(41, S) &= P(42, S) = P(43, S) \\ &= (\mathsf{nil}, 42, \mathsf{nil}; 56, \mathsf{nil}; 70, S[8]; 40, S[2]) \enspace . \end{aligned}$$

This attestation contains the predecessor and the successor of 41 (40 and 42, resp.), 42 (42 and 42, resp.) and 43 (42 and 56, resp.).

6.2 Security

The next theorem states that the authenticated search tree attester is undeniable if $\mathcal{H}$ is a collision-resistant hash function family, where the reduction is security preserving (i.e., if an adversary breaks the proposed construction with success probability ε then there exists another adversary that breaks the underlying hash function family with the same probability in reasonable time).

THEOREM 1. *Let $A \in \mathcal{EA}$ be an algorithm, s.t. $\mathsf{UN}_{\mathcal{A}}(A) = \varepsilon$. Then there exists an adversary $M \in \mathcal{EA}$ with $\mathsf{CRH}_{\mathcal{H}}(M) = \varepsilon$.*

PROOF. The adversary M is defined as follows. Given an index i and the security parameter 1^k, M performs a query to $A(1^k, i)$. With probability ε, this query outputs a tuple $(x, d, p, \overline{p})$, such that $V_i(x, d, p) = \mathsf{Accept}$ and $V_i(x, d, \overline{p}) = \mathsf{Reject}$. Therefore,

$$p = (h_{\mathsf{L}}, k_0, h_{\mathsf{R}}; k_1, h_1, \dots , k_m, h_m)$$

and $\overline{p} = (\overline{h}_{\mathsf{L}}, \overline{k}_0, \overline{h}_{\mathsf{R}}; \overline{k}_1, \overline{h}_1; \dots ; \overline{k}_{\overline{m}}, \overline{h}_{\overline{m}})$ for some $m, \overline{m} \geq 0$. Analogously, we will overline the variables d_j that are calculated during the verification of $\overline{p}$.

The adversary processes p and $\overline{p}$ in parallel. From (ST1) $d_m = \overline{d}_{\overline{m}}$. Since $V(x, d, p) = \mathsf{Accept}$ and $V(x, d, \overline{p}) = \mathsf{Reject}$, then $k_0 = x \neq \overline{k}_0$. Therefore, using (ST2) we get that for some s and $\overline{s}$, $\overline{d}_{\overline{m}} = d_m, \overline{d}_{\overline{m}-1} = d_{m-1}, \dots , \overline{d}_{\overline{s}} = d_s$ but $\overline{d}_{\overline{s}-1} \neq d_{s-1}$. (Remember also that $\mathsf{nil} \notin \Sigma^k$).

Next, if $k_s \neq \overline{k}_{\overline{s}}$, then M has found a collision $H_i(\cdot, k_s, \cdot) = H_i(\cdot, \overline{k}_{\overline{s}}, \cdot)$. Otherwise let us assume, w.l.o.g., that $x < k_s = \overline{k}_{\overline{s}}$ and therefore $d_s = H_i(d_{s-1}, k_s, h_s)$ and $\overline{d}_{\overline{s}} = H_i(\overline{d}_{\overline{s}-1}, \overline{k}_{\overline{s}}, \overline{h}_{\overline{s}})$. Since $d_{s-1} \neq \overline{d}_{\overline{s}-1}$, M has found a collision $H_i(d_{s-1}, \cdot, \cdot) = H_i(\overline{d}_{\overline{s}-1}, \cdot, \cdot)$.

Therefore, the adversary M finds a collision to $\mathcal{H}$ with probability ε. Note that M works in time $\Theta(t \log |S|)$, where t is the working time of A. $\square$

As with any new cryptographic primitive—and undeniable attester *is* a new primitive—it is good to know how it relates to the previously known primitives. The next theorem establishes the relationships between undeniable attesters and CRHFs.

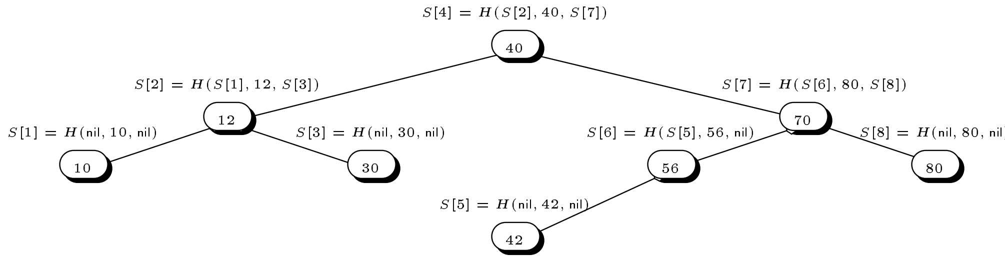

Figure 3: A toy example of authenticated search tree.

THEOREM 2. *1) Any undeniable attester is collision-resistant attester, but the opposite is not true. 2) Undeniable attesters exist if and only if CRHFs exist.*

PROOF. 1) Let $\mathcal{A} = (G, P, D, V)$, and let A be a machine, such that either $\mathsf{CRP}_{\mathcal{A}}(A) = \varepsilon$ or $\mathsf{CRD}_{\mathcal{A}}(A) = \varepsilon$. Next we construct an efficient machine M that has $\mathsf{UN}_{\mathcal{A}}(M) = \varepsilon$.

Let $i \leftarrow G(1^k)$. Adversary $M(1^k, i)$ lets $(x, S, p) \leftarrow A(1^k, i)$, $d \leftarrow D_i(S)$, $v \leftarrow V_i(x, d, p)$ and $\overline{p} \leftarrow P_i(x, S)$. M returns $(x, d, p, \overline{p})$, if $v = \mathsf{Accept}$, and $(x, d, \overline{p}, p)$, otherwise.

M queries once the algorithms $G(1^k)$, $A(1^k, i)$, P_i, D_i and V_i, and works otherwise in constant time. With probability ε_k, either (a) $x \notin S \subseteq \Sigma^k$, but $v = \mathsf{Accept}$, or (b) $x \in S \subseteq \Sigma^k$, but $v = \mathsf{Reject}$. Therefore, $\mathsf{UN}_{\mathcal{A}}(M) = \varepsilon$. As for the opposite, the construction in Section 5 showed that not each collision-resistant attester is undeniable.

2) Let $\mathcal{A} = (G, P, D, V)$ be an undeniable attester. By 1), $\mathcal{A}$ is also collision-resistant. Next, we show that if $\mathcal{A}$ is collision-resistant, then $\mathcal{D} = (G, D)$ is a CRHF on 2^{Σ^k} (i.e., on the subsets of Σ^k). Let $A \in \mathcal{EA}$ be an adversary, such that $\mathsf{CRH}_{\mathcal{D}}(A) = \varepsilon$.

Let M be the next machine. For $i \in G(1^k)$, M_i lets $(S_1, S_2) \leftarrow A(1^k, i)$. With the probability ε_k, $S_1 \neq S_2$ but

$$D_i(S_1) = D_i(S_2) =: d \ .$$

Since $|S_1|, |S_2| = k^{O(1)}$, we can efficiently find an element x in (w.l.o.g.) $S_1 \backslash S_2$. Let $\overline{p} := P_i(x, S_1)$. By the definition of attesters, $V_i(x, d, \overline{p}) = V_i(x, D_i(S_1), P_i(x, S_1)) = \mathsf{Accept}$. Thus, we have found a tuple $(x, S_2, \overline{p})$, such that $x \notin S_2$ but $V_i(x, D_i(S_2), \overline{p}) = \mathsf{Accept}$. A contradiction, and thus D_i is a CRHF on sets (i.e., on $2^{\Sigma^{|i|}}$, or alternatively, on concatenated strings $S[1]S[2] \cdots S[|S|]$, where $|S[j]| = |i|$ and for any $j < |S|$, $S[j] < S[j+1]$).

We finish the proof by constructing a CRHF $\mathcal{H} = (G, H)$ on the input domain Σ^* as follows. Let

$$S = S[1]S[2] \cdots S[n] \ ,$$

$n \leq p(k)$, be an arbitrary string, such that $|S[j]| = k - \log_2 n \leq k - \log_2 p(k)$ (it is sufficient to look at strings with length dividing $k - \log_2 p(k)$, due to the constructions presented in [9, 18]). Now define $H_i(S[1] \cdots S[n]) := D_i(\sigma[1] \cdots \sigma[n])$, where

$$\sigma[j] = \langle j \rangle_{\log_2 p(k)} S[j] \ ,$$

and $\langle i \rangle_k$ denotes a k-bit binary fixed representation of $i \in \mathbb{N}$. Clearly, if D is a CRHF on the domain 2^{Σ^k}, then $\mathcal{H}$ is a CRHF on domain Σ^*.

The opposite was proven by Theorem 1. □

6.3 Discussions

The construction of Section 6.1 generalizes to the case when the underlying tree is a multiway search tree [14, Section 6.2.4]. However, if we wish the attestations to have length $O(k \log |S|)$, we are restricted to the trees where the number of children of every node is upper-bounded with some constant that does not depend on k. As a result, we cannot base our construction on exponential search trees and other related data structures that have been lately extensively used in sub-logarithmic search algorithms [3].

Authenticated search trees can be made dynamic as in [20] by requiring that the CA stores the whole hash tree, and after each database update updates all the necessary hash values in the tree, including the value $D(S)$. Updating can be done in time $O(k \log |S|)$ by using appropriate dynamic search trees (say, AVL or 2-3 trees). Since our construction is just a slight reformulation of what is usually meant by search trees, and most of the "reasonable" data structures for searching can be seen as search trees, one can choose the data structure that is the most convenient in a concrete application.

There are many other possible constructions of undeniable attesters. For example, one could add a number of arcs to a binary tree as follows: For any non-leaf node v, add an arc (if it already does not exist) from its left child's rightmost descendant leaf to v. We emphasize that the main difference between the described constructions of collision-resistant and undeniable attesters is that in the first case the choice between the left and the right subtree is just done by an explicitly given bit b_i. In the latter case, there is instead an explicit search key $K[v]$, such that based on $K[v]$, the verifier can additionally check that the element returned in a query is in the correct location in this tree.

7. EFFICIENCY

7.1 Average-case Attestation Length

For a fixed size of S, authenticated search trees result in the shortest worst case attestation length if the underlying tree T is a complete binary tree. In this case, if we additionally assume that the search keys have length k—in practice, we store at leaves the hash values of certificates that are generally longer than k bits—then the worst case attestation length is $k \cdot (2 \log(n+1) + 1)$, where $n = 2^{d+1} - 1$ is the number of leaves (i.e., $n = |S|$). A simple calculation shows that the attestations $P_i(x, S)$ have in total $\frac{1}{k} \sum_{i=1}^{2^{d+1}-1} |P_i(x, S)| = 2^{d-1}(2d+2) - 3 + 2 \sum_{i=0}^{d-2} 2^i i = 2^{d+1}(d-1) + 1$ elements, which makes the average-case attestation length equal to

$$k \cdot \frac{2^{d+1}(d-1)+1}{2^d - 1} \approx k \cdot 2d \approx 2k \log n \ .$$

This is about twice as much as the attestation length in the complete binary tree based (improved) sorted hash tree attester. Also, in general, upon other types of trees, our construction has on average twice longer attestations than the optimal construction of collision-resistant attesters presented in Section 5.2. When using the dynamic AVL trees [14, Section 6.2.3], the worst case certificate length of the dynamic authenticated search tree attester is therefore $\approx 2.88 \cdot k \log n$.

7.2 Attestation Compression

Next, we describe a method for compressing the attestations. More often than not, compression algorithms are seen as consisting of two standard parts, modeling and coding [4]. An adaptive modeling algorithm estimates the source from the part of the data sequence seen so far, by outputting a probability distribution for the new symbol. After that, an encoder (say, the arithmetic encoder) uses this distribution to encode a new symbol by using as few bits as possible.

We can apply this general approach to the authenticated search trees. First, let T be a fixed search tree, and let k be the security parameter. We remind you that the elements of S are k bits long. During the modeling, we assign to every node v recursively a range (ℓ_v, u_v), as follows. As previously, let $\min S \geq 0$ be the least and let $\max S \leq |\Sigma^k| - 1$ be the greatest element in S. If v is the root vertex, then $(\ell_v, u_v) := (\min S, \max S)$. Now, let v be an arbitrary vertex. To the left child v_L (if existing) of v, we assign a range $(\ell_{v_\mathsf{L}}, u_{v_\mathsf{L}}) := (\ell_v, K[v]-1)$. Analogously, to the right child v_R (if existing) of v we assign the range $(\ell_{v_\mathsf{R}}, u_{v_\mathsf{R}}) := (K[v]+1, u_v)$. Next, every root path in T can be seen as a data sequence. For a node v in this sequence, the adaptive modeling algorithm returns the uniform distribution in (ℓ_v, u_v) to the encoder.

After that, the encoder encodes the value $K[v] - \ell_v$ as a binary number $K_c[v]$, using $\lceil \log_2(u_v - \ell_v) \rceil$ bits. The compressed attestation $P_c(x, S)$ is equal to the uncompressed attestation $P(x, S)$ with search keys $K[v]$ replaced with compressed keys $K_c[v]$. We additionally assume that the new digest $D_c(S)$ is equal to the triple $(D(S), \min S, \max S)$. Given that, the range (ℓ_v, u_v), and therefore also the search key $K[v]$, can be recalculated every time the compressed attestation $P_c(x, S)$ is used in verification. Not surprisingly, the fact that all intermediate values $K[v]$ can be unambiguously reconstructed from $P_c(x, S)$ is crucial for undeniability, and guided us during the choice of the encoder. Some more efficient encoders that we are aware of do not guarantee unambiguous reconstruction of all intermediate values, especially since the verifier has no previous knowledge about the tree T.

Assuming that T is a complete binary tree, the uncompressed attestations have the length $\leq k(2n+1)$, where $n = \log(|S|+1) \ll k$ is the height of T. On the other hand, the compressed attestations are never longer than $k(n+1) + \frac{n^2+n}{2}$. The worst case is obtained if $S = \{0, \dots, |\Sigma^k| - 2\}$. (We will not count in the short additional data necessary to encode the lengths of $K[v]$'s.) This provable gap between the worst case length of the compressed and uncompressed attestations is achieved thanks to the implicit structure hidden in the ordered data. However, the value $kn - \frac{n^2+n}{2} = \frac{2kn-n^2-n}{2}$ is a somewhat unexpected quantification of the amount of this structure.

As an example, let us look again at Figure 3. The root path from the root to the leaf with label $K[v] = 42$ has nodes with search keys $K[v_1] = 40$, $K[v_2] = 70$, $K[v_3] = 56$ and $K[v_4] = 42$. Computing the ranges, we find that $(\ell_{v_1}, u_{v_1}) = (10, 80)$, $(\ell_{v_2}, u_{v_2}) = (41, 80)$, $(\ell_{v_3}, u_{v_3}) = (41, 69)$ and $(\ell_{v_4}, u_{v_4}) = (41, 55)$. Therefore (as previously, we denote the n-bit binary encoding of m as $\langle m \rangle_n$), $K_c[v_1] = \langle 40 - 10 \rangle_7 = \mathsf{0011110}$, $K_c[v_2] = \langle 70 - 41 \rangle_5 = \mathsf{011101}$, $K_c[v_3] = \langle 56 - 41 \rangle_5 = \mathsf{01111}$ and $K_c[v_4] = \langle 42 - 41 \rangle_4 = \mathsf{0001}$. Hence,

$$P_c(42, S) = (\mathsf{nil}, \mathsf{0001}, \mathsf{nil}; \mathsf{01111}, \mathsf{nil}; \mathsf{011101}, S[8]; \mathsf{0011110}, S[2]) \enspace ,$$

and $|P_c(42, S)| = 5k + 21$, while $|P(42, S)| = 9k$. If $k = 160$, the compression gain is $\approx 1.754 \approx 9/5$. While this is an unrealistic example due to $\max S \approx \min S$ (remember that the elements of S are collision-resistant hashes of certificates!), it shows that this compression method can result in quite big savings. On the other hand, the attestations never shorten by a factor greater than two and therefore the authenticated search tree attester has longer attestations than the sorted hash tree one. However, the difference in space efficiency is negligible.

7.3 Optimality Questions

The classical *predecessor problem* requires one to maintain a set S so that the queries of the form "Is j an element of S and, if not, what element of S, if any, is just before it in sorted order?" may be answered efficiently. *Membership problem* only requires that the question "Is j an element of S?" may be answered efficiently.

There exist extremely efficient dynamic attesters if one does not require them to be collision-resistant. On the one hand, let $\mathcal{A}$ be an arbitrary attester, such that f_i, where $f \in \{G, P, D, V\}$, works in the worst-case time $t_{f,|i|}$. Straightforwardly, there exists a search algorithm working in time $t_D + t_P + t_V + O(1)$, which solves the membership problem.

On the other hand, according to the results of [10] for search algorithms solving the membership problem, there exists a dynamic attester, such that for any $S \subseteq \Sigma^k$ and for every $x \in S$, t_P, t_D, $t_V = O(1)$, $|P_i(x, S)| = 1$ and $|D_i(S)| = 0$. (Define $P_i(x, S) = 1$ if and only if $x \in S$, and fix $D_i(S)$ to be the empty string). However, both the sorted hash trees and our authenticated search trees do not solve only the membership but also the predecessor problem, since the attestation $P(x, S)$ always contains the predecessor of x, if it exists, or the smallest element in S. An interesting open problem is whether this is really necessary.

8. CONCLUSIONS

We proposed a model for long-term accountable certificate management and motivated the need for succinct undeniable attesters. We then described authenticated search tree attesters and proved that they are undeniable.

The resulting certificate management system has many desirable properties. It is accountable, since all disputes can be solved by the undeniable evidence present. This means in particular that all forgeries by the third parties can be explicitly proven and all false accusations explicitly disproven. It is efficient, since certificate validity can be verified, given only the certificate, a short digest of the certificate database and a short attestation.

Apart from the model of accountable certificate management system, the second main result of this paper is a construction of undeniable attesters. Undeniable attesters may become a very useful security primitive, since they make it possible for anyone to perform securely membership (and predecessor) queries without relying on the trusted third parties nor requiring an access to the whole database.

9. FURTHER WORK

Strict optimality of our constructions is left as an open question. For example, since it is easier to solve the membership problem [10] than the predecessor problem [3], it is interesting to know whether succinct undeniable attesters can be built upon the search algorithms solving the membership problem. Elaboration of exact protocols and duties of different participants in accountable certificate management is of utmost importance.

10. ACKNOWLEDGMENTS

The first author was supported by the Estonian Science Foundation, grant 3742. All authors have been affiliated with the Institute of Computer Science, University of Tartu, Estonia.

We would like to thank Andris Ambainis, Carl Ellison, Kobbi Nissim, Berry Schoenmakers and anonymous referees for useful comments that helped to improve the paper.

11. REFERENCES

[1] W. Aiello, S. Lodha, and R. Ostrovsky. Fast Digital Identity Revocation. In H. Krawczyk, editor, *Advances on Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 137–152, Santa Barbara, USA, 23–27 Aug. 1998. Springer-Verlag.

[2] N. Barić and B. Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In W. Fumy, editor, *Advances on Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494, Konstanz, Germany, May 1997. Springer-Verlag.

[3] P. Beame and F. Fich. Optimal Bounds for the Predecessor Problem. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 295–304, New York, 1–4 May 1999.

[4] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, 1990.

[5] J. Benaloh and M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures (Extended Abstract). In T. Helleseth, editor, *Advances in Cryptology—EUROCRYPT 93*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer-Verlag, 1994, 23–27 May 1993.

[6] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with Binary Linking Schemes. In H. Krawczyk, editor, *Advances on Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 486–501, Santa Barbara, USA, Aug. 1998. Springer-Verlag.

[7] A. Buldas, H. Lipmaa, and B. Schoenmakers. Optimally Efficient Accountable Time-stamping. In H. Imai and Y. Zheng, editors, *Public Key Cryptography '2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 293–305, Melbourne, Victoria, Australia, 18–20 Jan. 2000. Springer-Verlag.

[8] B. Crispo and M. Lomas. A Certification Scheme for Electronic Commerce. In *1996 Security Protocols International Workshop*, volume 1189 of *Lecture Notes in Computer Science*, pages 19–32, Cambridge, UK, 1996. Springer-Verlag.

[9] I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1990, 20–24 Aug. 1989.

[10] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. M. A. D. Heide, H. Rohnert, and R. E. Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. *SIAM Journal on Computing*, 23(4):738–761, Aug. 1994.

[11] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, Nov. 1976.

[12] I. Gassko, P. Gemmel, and P. MacKenzie. Efficient and Fresh Certification. In H. Imai and Y. Zheng, editors, *Public Key Cryptography '2000*, volume 1751 of *Lecture Notes in Computer Science*, pages 342–353, Melbourne, Victoria, Australia, 18–20 Jan. 2000. Springer-Verlag.

[13] S. A. Haber and W. S. Stornetta. How to Time-stamp a Digital Document. *Journal of Cryptology*, 3(2):99–111, 1991.

[14] D. E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching*. Addison-Wesley, 2 edition, 1998.

[15] P. Kocher. On Certificate Revocation and Validation. In R. Hirschfeld, editor, *Financial Cryptography — Second International Conference*, volume 1465 of *Lecture Notes in Computer Science*, pages 172–177, Anguilla, British West Indies, 23–25 Feb. 1998. Springer-Verlag.

[16] H. Lipmaa. *Secure and Efficient Time-stamping Systems*. PhD thesis, University of Tartu, June 1999.

[17] R. C. Merkle. Protocols for Public Key Cryptosystems. In IEEE, editor, *Proceedings of the 1980 Symposium on Security and Privacy, April 14–16, 1980 Oakland, California*, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1980. IEEE Computer Society Press.

[18] R. C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, 1990, 20–24 Aug. 1989.

[19] S. Micali. Efficient Certificate Revocation. Technical report, Massachusetts Institute of Technology, 22 Mar. 1996.

[20] M. Naor and K. Nissim. Certificate Revocation and Certificate Update. In *7th USENIX Security Symposium*, 1998.

[21] NIST. Announcement of Weakness in the Secure Hash Standard (SHS). FIPS 180-1, May 1994.

[22] K. Nyberg. Commutativity in Cryptography. In *Proceedings of the First International Workshop on Functional Analysis at Trier University*, pages 331–342, Berlin, 1996.

[23] K. Nyberg. Fast Accumulated Hashing. In D. Grollman, editor, *Fast Software Encryption: Third International Workshop*, volume 1039 of *Lecture Notes in Computer Science*, pages 83–87, Cambridge, UK, 21–23 Feb. 1996. Springer-Verlag.

[24] R. L. Rivest. Can We Eliminate Revocation Lists? In R. Hirschfeld, editor, *Financial Cryptography — Second International Conference*, volume 1465 of *Lecture Notes in Computer Science*, pages 178–183, Anguilla, British West Indies, 23–25 Feb. 1998. Springer-Verlag.

[25] T. Sander. Efficient Accumulators without Trapdoor. In V. Varadharajan and Y. Mu, editors, *The Second International Conference on Information and Communication Security*, volume 1726 of *Lecture Notes in Computer Science*, pages 252–262, Sydney, Australia, 9–11 Nov. 1999. Springer-Verlag. ISBN 3-540-66682-6.