Journal of Computer Security 10 (2002) 273–296
IOS Press

Eliminating counterevidence with applications to accountable certificate management [1]

Ahto Buldas [a], Peeter Laud [b] and Helger Lipmaa [c,*]

[a] *Tallinn Technical University/University of Tartu/Cybernetica AS, Akadeemia tee 21, 12618 Tallinn, Estonia*
E-mail: ahto.buldas@cyber.ee
[b] *FB 14 Informatik, Universität des Saarlandes, Im Stadtwald – Bau 45, Postfach 15 11 50, 66041 Saarbrücken, Germany*
E-mail: laud@cs.uni-sb.de
[c] *Laboratory for Theoretical Computer Science, Department of Computer Science and Engineering, Helsinki University of Technology, FIN-02015 HUT, Espoo, Finland*
E-mail: helger@tcs.hut.fi

This paper presents a method to increase the accountability of certificate management by making it intractable for the certification authority (CA) to create contradictory statements about the validity of a certificate. The core of the method is a new primitive, *undeniable attester*, that allows someone to commit to some set S of bitstrings by publishing a short digest of S and to give attestations for any x that it is or is not a member of S. Such an attestation can be verified by obtaining in authenticated way the published digest and applying a verification algorithm to the triple of the bitstring, the attestation and the digest. The most important feature of this primitive is intractability of creating two contradictory proofs for the same candidate element x and digest. We give an efficient construction for undeniable attesters based on authenticated search trees. We show that the construction also applies to sets of more structured elements. We also show that undeniable attesters exist iff collision-resistant hash functions exist.

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 [12]. 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 public-key cryptography can be successfully applied in practice is still that of key management. There has been a huge body of research on key management methods since [19] but key management involves still considerably more trust toward the third parties than rest of cryptography.

[1] A preliminary version of this paper [6] appeared in the *ACM CCS '2000 Conference*. The current version is the final submission to the *Journal of Computer Security*.

* Corresponding author.

We say a certificate management system is *accountable* when all forgeries by third parties can be explicitly proven and all false accusations explicitly disproven. 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 [16]. Recent work in time-stamping has also shown how to build efficient yet accountable time-stamping systems [7,8,20] 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 [10].

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 possession of the verifier. Under this intractability assumption, our certificate management system has several desirable properties. The most important property is that if physical visits to the Certification Authority (CA) are audited, every subsequent dispute in court can be solved by the present evidence. Moreover, one can verify 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 is focused on this assumption.

In Section 3, we give the 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 on input (x, d, p), depending on whether x belongs to a set with digest d. Here, p contains additional information about the status of $x \in^{?} S$.

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. In the context of certificate management, S is a database of identity certificate serial numbers.

In Section 4 we survey some attesters whose subsystems were considered previously in certificate management and public-key infrastructure. In particular, we

review attesters based on certificate revocation lists, hash trees [22], certificate revocation trees [18,25] and RSA accumulators [2,9]. 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 that the only previously 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 4.4. Sorted hash tree attesters are based on an efficient construction similar to the certificate revocation trees. We show in Section 4.4 that sorted hash tree attesters are not undeniable and therefore a sorted hash tree attester-based certificate management system makes it possible for the CA to cheat clients.

In Section 5, 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 sorted hash tree attesters and the proposed construction is that authenticated search tree attesters assign 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 key of v. (In sorted hash tree attesters, $S[v]$ does *not* authenticate the search key of v.) Moreover, authenticated search tree attesters are in several aspects more intuitive than sorted hash tree attesters: Being directly based on search trees as they are generally understood in computer science, they allow us to carry over to cryptography the research done in the area of algorithms and data structures [17]. As such, the proposed undeniable attester might have surprisingly wide applications in different security applications, and not only in certificate management.

After defining new attester, we will prove that it is undeniable. As with any new cryptographic primitive it is good to know how it relates to previously known primitives. A proof that undeniable attesters exist if and only if collision-resistant hash functions exist is presented in Section 5.

We show in Section 5.4 that our methods can be extended to multi-field records. First, we can think of every $x \in S$ as composed of two parts, a unique key and a body. Searching can be performed only by looking up the key; the authenticated search tree attester can be modified to enable detecting the case when the key is not unique. Further extensions are possible. These extensions provide also a better answer to natural question what *exactly* is a candidate string x in the case of certificate management. We purposefully have not yet specified this: x could be certificate serial number, but also a hash of whole certificate, whatever is more relevant in practical applications. However, the best solution might be to let the unique key to correspond to serial number, and the body to correspond to the hash of whole certificate.

In Section 6, we will provide efficiency analysis of the authenticated search tree attester. We show that attestations in the authenticated search tree can be compressed, under ideal conditions, 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. The attestation compressing method given in Section 6 might be of independent interest.

Terminology. We have intentionally chosen to use slightly nonstandard but self-consistent terminology. In particular, we have tried to avoid overloading already notoriously overloaded terms 'certificate' and 'proof' and relatives, by introducing the term *attestation*. We use the more concrete term *certificate management* instead of 'PKI'. (Certificate management denotes usually the process whereby certificates are managed and used, while the PKI refers to the entire framework established for certificate management.)

2. Motivations

Our research is motivated by the observation that for long-term authenticity and non-repudiation of digital documents, new methods are necessary for verifying whether identity certificates (bindings 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, it would be desirable to ensure that validity information of certificates cannot be forged by anybody, including the authorities.

We aim at construction of an accountable certificate management system, where all forgeries by third parties can be explicitly proven and all false accusations explicitly disproven [8]. More precisely, we would like the only part of certificate management (physical visit of a person to an authority) that clearly cannot be mathematically modeled also to be the only stage in the system that needs some non-cryptographic solution (i.e., involving physical presence of a client-chosen notary) to trust problems. If visits were 'correct', the system should need no auditing anywhere else. In particular, clients should be able to discover if the CA has maliciously issued new invalid certificates or removed valid certificates that are still valid.

Long-term certificate validity can be partially ensured by the methods of time-stamping [7,16], where absence of a proof that a certificate was issued is implicitly counted as the proof of its nonexistence. However, 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 a setting where the CA maintains a dynamic database S of valid certificates. See [15,28] for argumentation why a database of valid certificates is better than a database of revoked certificates. In our case, database of revoked certificates would just add unnecessary complexities to the system. The presence of a central authority lessens the communication complexity of the scheme

and simplifies tracking of the origins of frauds. Our model also includes the Publication Authority [8] and a (possibly huge) number of clients.

We assume that every client receives a positive (resp. negative) attestation from the CA if her certificate x belongs (resp., does not belong) to the database S of valid certificates. This assumption is not restricting, since some sort of attestation – or receipt – is returned to the client by the CA in every certificate management system. In our system, it is in 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 [7,20].

A digest of database S (denoted as $D(S)$) is published by the Publication Authority in some authenticated and widely available medium by using accountable publishing protocols [8]. Motivations behind this are the same as in time-stamping [7,8,16]. 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 [16]. 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 old versions of the dynamic database S: the system should still be accountable, if a 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 [7,16].

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. The proving algorithm P, given a candidate string x and a set S, outputs an attestation. The digest algorithm D, given a set S, outputs a short digest $d = D(S)$ of the database. Finally, the verification algorithm V takes as input a candidate element x, a digest d, and an attestation p, and accepts or rejects depending on whether x belongs to a set S such that $d = D(S)$.

In described model, the CA cannot cheat a client. (We assume that Denial of Service attacks by the CA, where the CA does not return an attestation to the client, can be prevented for example by letting a client-chosen notary to participate in handing over the attestation.) That is, if a client has a positive (resp., negative) attestation that some certificate belonged (resp., did not belong) 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 (resp. was) in this database at that time, 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 long-term authenticity undeniability is crucial – e.g., when the CA who issued a concrete certificate might have 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 a contradictory attestation. These properties will significantly increase the trustworthiness of the CAs.

2.1. Application in court

The relevance of undeniable attestations can probably be best exemplified by the next application. If a digital signature law is passed in some country, it becomes natural to expect that digital signatures would then be considered as legally valid as handwritten ones. In particular, one should be able to solve legal disputes based on the validity of a particular signature on a particular document. As is well known, one needs certificate management for that in order to determine the validity of the digital signature. However, one often overlooks the real process in court, where a person presents to the judge a piece of evidence, which in this case is a signed document together with a certificate. The judge cannot take an action, based on the evidence (that we call an attestation) alone, if it is possible for somebody else to create counterevidence (that we call the contradictory attestation). For example, certificate revocation list can be seen as counterevidence.

Currently, it is almost always possible, at least for the CA, to create counterevidence. Therefore, court's decisions often base on heuristics that involve trust in some authorities, eyewitnesses or other human beings. In the case of digital signatures, the definition of eyewitnesses is unclear, and one might not want to trust the authorities. Not only is it possible that the authorities might be corrupted, but a malicious client could also claim that a honest authority is guilty. In such cases the judge cannot take an action, since the authority might or might not be guilty. If it is intractable to create a counterevidence even for the authorities, the court can always authoratively decide a case, based on evidence. By doing this, the judge does not have to trust anybody, and everybody can also check that judge's actions are correct. The latter means that one cannot question ambiguous actions of court related to digital signatures, and hence there is no reason to appeal to a higher court if the actions of the the lower court were provably wrong.

2.2. 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 [10]. 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 separation between the authorities. For the same reason, we do not elaborate on the accountable publication protocols but rather refer the reader to [8] for necessary information.

3. Formal definitions

3.1. Preliminaries

Let $\Sigma = \{0, 1\}$. As usually, Σ^k denotes the set of k-bit words, $\Sigma^* := \bigcup_{k \geqslant 0} \Sigma^k$. For $\sigma \in \Sigma$, σ^k denotes string of k σ-s. From now on, k denotes the security parameter, relative to which 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 first input. A family $\mathsf{P} = (\mathsf{P}_k)$ of probabilities, $k \in \mathbb{N}$, is *negligible* if for all $\varepsilon > 0$ there exists a k_ε, such that $\mathsf{P}_k < k^{-\varepsilon}$, 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 $H \in \mathcal{EA}$, 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

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

Note that index i is only necessary when one requires H to be collision-resistant. Otherwise one can assume that $|I| = 1$.

3.2. Definition of attester

We have already given informal definitions of attesters. Next, we present the full formalism, followed by discussion.

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 f, $f(k) > k$, 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 $S \subseteq \Sigma^k$, $|S| \leqslant f(k)$ and outputs an *attestation* $P_i(x, S) = P(i, x, S)$.

3. A *digest algorithm* $D \in \mathcal{EA}$ takes as input an index i, a set $S \subseteq \Sigma^k$, $|S| \leqslant f(k)$ and outputs a digest $D_i(S) = D(i, S) \in \Sigma^{\leqslant f(k)} \cup \{\mathsf{Error}\}$.
4. A *verification algorithm* $V \in \mathcal{EA}$ 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 if $i \notin \Sigma^k \cap I$, $S \not\subseteq \Sigma^k$, $|S| > f(k)$ or $x \notin \Sigma^k$, then for any p, $V_i(x, D_i(S), p) = \mathsf{Error}$. (In practice, one should set $D_i(S) = \mathsf{Error}$ if $i \notin \Sigma^k \cap I$, $S \not\subseteq \Sigma^k$ or $|S| > f(k)$.) Otherwise, for any $S \subseteq \Sigma^k$ with $|S| \leqslant f(k)$, and for any $x \in \Sigma^k$, $V_i(x, D_i(S), P_i(x, S))$ outputs Accept if $x \in S$ and Reject if $x \notin S$.

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

3.3. Discussion

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-resistance and undeniability. Otherwise we can assume that $|I| = 1$. In informal treatment, one can omit I at all. However, in our constructions of undeniable attesters we have to assume that the used hash function is collision-resistant, which automatically introduces need for the index set I in the formal treatment.

It is important to understand the seemingly subtle but crucial in applications difference between collision-resistant attesters and undeniable attesters. Collision-resistant attesters assume that a verifier has access to the correctly computed

value $D_i(S)$. In practice, it means that she either has to rely on some trusted third party to provide a correct $D_i(S)$ 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 a much higher level of confidence in the system.

In practice, S is organized as a certain data structure. In many cases, attester is just a security add-on to this data structure. We will see this in Sections 4.4 and 4.3, where attesters will be based on corresponding types of trees. Therefore, we will often use the terminology of data structures in the context of attesters. For example, one would like an attester to have 'succinct' attestations and digests but also fast average-case update time of the data structure. Informally, we say that an attester is *dynamic* if average-case time per insertion and deletion of elements to the corresponding data structure is $O(|i| \log |S|)$ for any $i \in I$. We say an attester is *succinct* if for any $i \in I$, $|D_i(S)| = O(|i|)$ and $|P_i(x, S)| = O(|i| \cdot \log |S|)$. Note that since $P, D \in \mathcal{EA}$, any attester has $|D_i(S)| = |P_i(x, S)| = |i|^{O(1)}$.

4. Some known constructions

Next, we will give a short survey of some attesters based on previously proposed ideas. Table 1 summarizes the properties of attesters described in this section, together with authenticated search tree attesters described later in Section 5. 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, but the resulting succinct construct will only be attester without satisfying any stronger security requirements.

As emphasized in Section 2, in accountable certificate management we are interested in undeniable attesters. In the following we will briefly explain why already known attesters fail to satisfy our requirements. The main result of this paper is the provably secure authenticated search tree attester, described in Section 5.

4.1. List attester

List attester is the most trivial 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 k-bit hash $H(S)$ of S, where H is a collision-resistant hash function. (Remember that k is a 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.

Table 1

Some known succinct attesters, i.e., security is given only for the succinct versions (see Section 3.3). 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. (A = attester, P = prover, CR = collision resistant, U = undeniable.)

Primitive type	Name	Length in $\Theta(\cdot)$		
		Digest	Positive attestation	Negative attestation
A	List (§4.1)	k	$n \log n$	$n \log n$
CRP	RSA (§4.2)	k	k	$n \log n$
	Hash Tree (§4.3)	k	$k \log n$	$n \log n$
CRA	Sorted Hash Tree (§4.4)	k	$k \log n$	$k \log n$
UA	Authenticated Search Tree (§5)	k	$k \log n$	$k \log n$

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.

4.2. RSA attester

The RSA attester can be in a natural way built upon the RSA accumulator [2,9]. Here, the positive attestations have the form

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

for some $y_1, \ldots, y_m$, and therefore the attestation length is $\Theta(k)$, where k is again a security parameter. The digest has the same form and therefore also the same length. However, as first pointed out by Nyberg [26,27], the length of the attestations can be reduced by introducing built-in trapdoor information known to some coalition of participants, which should therefore be trusted. The best known method [29] of making the RSA accumulator trapdoorless introduces attestation lengths of order $\Theta(k^2)$. Since $k \geqslant 128 > \log |S|$, the trapdoorless RSA accumulator has longer attestations than the sorted hash tree attester, described below. Moreover, the negative attestations are equal to all of S.

4.3. Hash tree attester

Hash trees [22] 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 hash trees used 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. On the other hand, negative attestations must include every element of S an have hence length $\Theta(k|S|)$.

4.4. Sorted hash tree attester

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 [18,25]. (We assume that the values stored at the leaves are sorted from left to right.) 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. Since the observed weakness in sorted hash trees is a crucial motivation to our subsequent work, we will next give a detailed definition of sorted hash tree attesters together with a full explanation of their weak points.

4.4.1. 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 [24] or any other strong (keyed) hash function. Let f be an a priori fixed polynomial that does not depend on k.

Next, suppose that $S = \{S[1], \ldots, S[n]\}$ is a nonempty set of k-bit integers such that $S[j] < S[j+1]$ for any $1 \leqslant j < n$. Let T be a (directed) binary tree with n leaves, with its jth leftmost leaf labeled by $S[j]$ (Fig. 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, $|S| > f(k)$, or some leaf had a label $S[v] \notin \Sigma^k$.

Let $p = (b; h_1, h_2, \ldots, h_m)$, such that $h_j \in \Sigma^k$ and $b = b_1 \ldots b_m$, $b_j \in \{0, 1\}$ with 0 corresponding to the left and 1 corresponding to the right direction. The verifica-

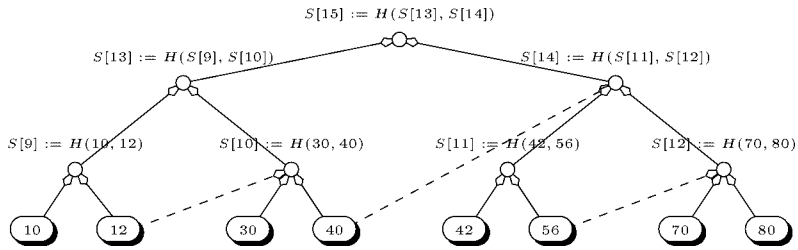

Fig. 1. A toy example of the sorted hash tree attester. Dashed lines are present only in the improved sorted hash tree attester, described in Section 4.4.2. The values $S[v]$ are given for the unimproved construction. For the ease of illustration we have chosen all vertices to be elements of $\mathbb{Z}$, although the concrete values of v's are not important in practice.

tion 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 $V(x_1, d, p_1) = V(x_2, d, p_2) =$ 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 Fig. 1, $D(S) = S[15]$,

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

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

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

4.4.2. 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 [18,25]): 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 Fig. 1. Build an attester upon the resulting graph, by modifying the algorithms P, D and V to account for 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.

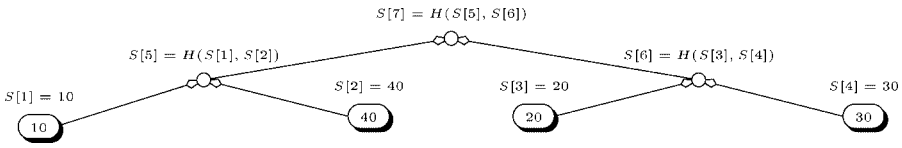

Fig. 2. A toy example of improperly created sorted hash tree attester.

4.4.3. 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 Fig. 2. There, the positive attestations of 10, 40 and 20 are respectively $p_1 = (\texttt{11}; 40, S[6])$, $p_2 = (\texttt{01}; 10, S[6])$ and $p_3 = (\texttt{10}; 30, S[5])$. 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. The same is also true for improved sorted hash tree attesters.

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 the entire database 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 large, 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).

Intuitively, the need to send the entire database is caused by the fact that 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. Therefore, 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.

5. 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* [17, 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. $K[w] > K[v]$).

5.1. Construction

Let f be some a priori fixed polynomial. We give, as in Section 4.4, 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

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

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, in particular, if $|S| > f(k)$ or for some leaf v, $S[v] \notin \Sigma^k$.

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 fraud.*

Let

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

where all the elements are from Σ^k, and $m \geqslant 0$. 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 $0 < j < m$,

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

After that, V outputs Error if

(ST1) $d_m \neq d$, or
(ST2) for some $j > 0$, $x = k_j$ or $k_{j-1} = k_j$.

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

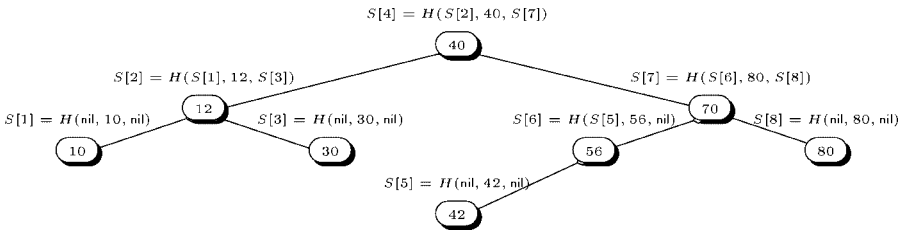

Fig. 3. A toy example of authenticated search tree. Note that vertices can be enumerated arbitrarily. For the ease of illustration we have chosen all vertices to be elements of $\mathbb{Z}$, although the concrete values of v's are not important in practice.

If $x \in S$, the algorithm $P(x, S)$ returns a list p such that $V(x, D(S), p)$ accepts. As we will see below, such a p is easy to find. If $x \notin S$, $P(x, S)$ finds (1) An element x', such that x' is the greatest element $x' \leqslant x$ (the *predecessor* of x), if such x exists, or the smallest element in S, otherwise; (2) An element x'', such that x'' is the smallest element $x'' \geqslant x$ (the *successor* of x), if such x 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 a 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 Fig. 3. Here, $D(S) = S[4]$ and

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

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.).

5.2. Security

Theorem 1. *The authenticated search tree attester is undeniable if $\mathcal{H}$ is a collision-resistant hash function family, where the reduction is security preserving in the next meaning. 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,

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

and $\overline{p} = (\overline{h}_{\mathsf{L}}, \overline{k}_0, \overline{h}_{\mathsf{R}}; \overline{k}_1, \overline{h}_1; \ldots; \overline{k}_{\overline{m}}, \overline{h}_{\overline{m}})$ for some $m, \overline{m} \geqslant 0$. Analogously, we will overline the variables d_j that are computed 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 $\overline{s}$, $\overline{d}_{\overline{m}} = d_m, \overline{d}_{\overline{m}-1} = d_{m-1}, \ldots, \overline{d}_{\overline{s}} = d_s$ 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, without loss of generality, 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. □

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. Next results establish the relationships between collision-resistant attesters, undeniable attesters and CRHFs.

Lemma 1. *Any undeniable attester is a collision-resistant attester.*

Proof. 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, $\mathsf{UN}_{\mathcal{A}}(M) = \varepsilon$. □

Note that the construction in Section 4.4 showed that not each collision-resistant attester is undeniable, and hence the opposite of this lemma is not true.

Theorem 2. *Undeniable attesters exist if and only if CRHFs exist.*

Proof. Let $\mathcal{A} = (G, P, D, V)$ be an undeniable attester. By Lemma 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.) $S_1 \setminus S_2$. 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 $S = S[1]S[2]\cdots S[n]$, $n \leqslant p(k)$, be an arbitrary string, such that $|S[j]| = k - \log_2 n \leqslant k - \log_2 p(k)$. (It is sufficient to look at strings with length dividing $k - \log_2 p(k)$, due to the constructions presented in [11,23]. 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. □

5.3. Discussion

The construction of Section 5.1 generalizes to the case when the underlying tree is a multiway search tree [17, 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 [5].

Authenticated search trees can be made dynamic as in [25] 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 but also skip lists). 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.

5.4. Extensions to multi-field records

Both our definition of the attesters and the construction of authenticated search trees were given for the case when the database S consists of some indivisible records x. However, quite often one works in a situation where x is composed of several fields. For example, in many cases the first field $\mathsf{key}(x)$ acts as a unique key to the entire record $x = \mathsf{key}(x)||\mathsf{body}(x)$, where $\mathsf{body}(x) \in \Sigma^t$ for some $t < k$. In such a case, there is a need for a set of algorithms (P, D, V) allowing the database maintainer to commit to the value of S in such a way, that

- The clients can perform a query $\mathsf{key}(x) \in^? S$, to which the response is either $\mathsf{body}(x)$ if the database S contains a unique record with the key $\mathsf{key}(x)$ or $\mathsf{Undefined}$ if there is none;
- It is intractable for the database maintainer to give different answers to queries for the same key $\mathsf{key}(x)$ (thus the response of the CA has to contain some kind of proof).

The authenticated search tree can be used to construct the necessary primitive. The database S is organized as an authenticated search tree; the algorithm D is left unchanged. If there exists a y for which $\mathsf{key}(x)||y \in S$ then let x' be the predecessor of $\mathsf{key}(x)||y$ and x'' be the successor of $\mathsf{key}(x)||y$ in S. Otherwise, let x' be the predecessor and x'' be the successor of $\mathsf{key}(x)||0^t$ in S. (If $\mathsf{key}(x)||0^t < \min S$ then $x' = \min S$. If $\mathsf{key}(x)||0^t > \max S$ then $x'' = \max S$.)

The attestation $P(\mathsf{key}(x), S)$ is equal to $P(x', S) \cup P(\mathsf{key}(x)||y, S) \cup P(x'', S)$ in some well-defined presentation. Given negative attestation, the verification procedure additionally checks that $\mathsf{key}(x') < \mathsf{key}(x) < \mathsf{key}(x'')$ and that there are no elements $\bar{x}$ with $\mathsf{key}(x') < \mathsf{key}(\bar{x}) < \mathsf{key}(x'')$ in the tree, and returns Error if this is not the case. Given positive attestation, the verification algorithm additionally checks that $\mathsf{key}(x') < \mathsf{key}(x) < \mathsf{key}(x'')$, that there are no elements $\bar{x}$ with $\mathsf{key}(x') < \mathsf{key}(\bar{x}) < \mathsf{key}(x)$ or $\mathsf{key}(x) \leqslant \mathsf{key}(\bar{x}) \leqslant \mathsf{key}(x'')$ in the tree, and that there is some $\mathsf{key}(x)||y \in P(\mathsf{key}(x), S)$. In both cases, (non-)existence of such $\bar{x}$ can be efficiently verified, because the positions of the vertices labeled by x', $\mathsf{key}(x)||y$ and x'' in the search tree can be determined from the attestation.

More generally, one could use *authenticated k-d trees* to perform multidimensional queries. Here, authenticated k-d trees are natural extensions of authenticated search trees to Bentley's k-d trees [3,4] for handling databases with k fields. As shown by Bentley, k-d trees can be balanced to have depth $\Theta(\log |S|)$, in which case it takes $O(|S|^{1-t/k})$ steps to find all records with t fields equal to specified values, and $O(t|S|^{1-1/k} + q)$ steps to find all elements in a t-dimensional subspace, when there is q such elements. (See [17, Section 6.5] for more information.)

In the context of certificate management, one could take, for example, $\mathsf{key}(x)$ to be equal to the unique certificate serial number, while $\mathsf{body}(x)$ would be the hash of the whole certificate.

6. Efficiency

6.1. 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 $n = |S| = 2^{d+1} - 1$. A simple calculation shows that the attestations $P_i(x, S)$ have in total $(1/k)\sum_{i=1}^{2^{d+1}-1} |P_i(x,S)| = 2^{d-1}(2d+2) - 3 + 2\sum_{i=0}^{d-2} 2^i i = 2^{d+1}(d-1)+1$ 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 4.4.2. When using the dynamic AVL trees [17, Section 6.2.3], the worst case certificate length of the dynamic authenticated search tree attester is therefore $\approx 2.88 \cdot k \log n$.

6.2. 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 [1]. 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 $\min S \geqslant 0$ be the least and let $\max S \leqslant |\Sigma^k| - 1$ 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 corresponding compressed keys $K_c[v]$. We additionally assume that the new digest $D_c(S)$ is equal to the triple $(D(S), \min S, \max S)$. Verification still needs the uncompressed attestation $P(x, S)$, which can be easily computed from $P_c(x, S)$. 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 $\leqslant k(2n+1)$, where $n = \log(|S|+1) \ll k$ is the height of T. The compressed attestations are never longer than $k(n+1) + (n^2+n)/2$. The worst case is obtained if $S = \{0, \ldots, |\Sigma^k| - 2\}$. (We will not count in the short additional data necessary to encode the lengths of $K[v]$'s. One could use, for example, Elias ω-codes [14] for the latter.) 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 $kn - (n^2+n)/2 = (2kn - n^2 - n)/2$ is a somewhat unexpected quantification of the amount of this structure.

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 factor can be arbitrarily close to 2 when $k \gg n$. As an example, let us look again at Fig. 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 $\langle m \rangle_n$), $K_c[v_1] = \langle 40 - 10 \rangle_7 = \texttt{0011110}$, $K_c[v_2] = \langle 70 - 41 \rangle_5 = \texttt{011101}$, $K_c[v_3] = \langle 56 - 41 \rangle_5 = \texttt{01111}$ and $K_c[v_4] = \langle 42 - 41 \rangle_4 = \texttt{0001}$. Hence,

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

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. When the hash function is modeled as a random function, $\max S \approx \min S$ holds with a negligible probability), it shows that this compression method can result in quite big savings already for small n-s. In real life situations an example with $k = 160$ and $|S| = 10^7$ (a database of that size might be typical in future certificate management systems in middle-sized countries), depicted by Table 2, is more appropriate.

Table 2

Comparison of the worst-case attestation lengths of collision-resistant and undeniable attesters in bytes, where $n = \lceil \log |S| \rceil$

Method	Attestation length in bits	$k = 160$, $\lvert S\rvert = 10^7$
Collision-Resistant Attesters		
Sorted hash tree	$k(n+1)$	500 B
Undeniable Attesters		
List	$k \cdot \lvert S\rvert$	2×10^8 B
Authenticated search tree	$2k(n+1)$	1000 B
AST (compressed)	$k(n+1) + \frac{n^2+n}{2}$	537.5 B

6.3. 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 [13] 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.

7. Conclusions

The approach to certificate management in this paper is straightforward: First we went back to the source (in our case, the legal system, and more precisely, the court), then we identified why certificate management is necessary there and how to minimize the number of frauds; in our case, this was achieved by making it infeasible to create counterevidences. We then gave a rigorous definition of necessary cryptographic primitives (undeniable attesters), and described and analyzed an efficient undeniable attester, the authenticated search tree attester. We showed that the latter is almost as efficient as less secure sorted hash tree attesters, by using a simple (but novel) compression technique.

The resulting certificate management system has many desirable properties. It is accountable, since all disputes can be solved by the present undeniable evidence. 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.

We stress that the current paper provides a model of certificate management, not a complete system, and that more work is needed for making the certificate management really accountable. More precisely, our system guarantees that it is intractable to create a certificate database S, such that if $x \in S$ (resp., $x \notin S$), the CA can create an attestation certifying that $x \notin S$ (resp., $x \in S$). In practice, S is dynamic and the CA may remove valid certificates or insert invalid certificates from S at his will. Such malpractice can be detected, for example, if interested parties ask attestations of $x \in^{?} S$ every time S is updated. One possibility would be to introduce another trusted party (or a set of volunteers), who would compare the changes between the subsequent versions of S with official documentation provided by the CA and notaries about the physical visits of clients. However, it would be desirable to find more efficient protocols for this purpose. (See [30] for some preliminary analysis.)

8. Further work

Strict optimality of our constructions is left as an open question. For example, since it is easier to solve the membership problem [13] than the predecessor problem [5], it would be 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.

Finally, what benefits would be gained by using authenticated search trees instead of sorted hash trees in other areas of data security? As an use example, results of the preliminary version of this paper [6] have already been used to build a secure key archival service in [21].

Acknowledgements

The authors were partially supported by the Estonian Science Foundation, grant 3742. The first author was partially supported by the Estonian Science Foundation,

grant 4760. A preliminary version of this paper was published as [6]. We would like to thank Andris Ambainis, Carl Ellison, Kobbi Nissim, Berry Schoenmakers, Stuart Stubblebine and anonymous referees for useful comments that helped to improve the paper.

References

[1] T.C. Bell, J.G. Cleary and I.H. Witten, *Text Compression*, Prentice Hall, ISBN: 0139119914, 1990.

[2] J. Benaloh and M. de Mare, One-way accumulators: a decentralized alternative to digital signatures, in: *Advances in Cryptology – EUROCRYPT '93*, Vol. 765 of Lecture Notes in Computer Science, T. Helleseth, ed., Lofthus, Norway, 1993, Springer-Verlag, ISBN 3-540-57600-2, 1994, pp. 274–285.

[3] J.L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM* **18**(9) (1975), 509–517.

[4] J.L. Bentley, Multidimensional binary search trees in database applications, *IEEE Transactions on Software Engineering* **5**(4) (1979), 333–340.

[5] P. Beame and F.E. Fich, Optimal bounds for the predecessor problem, in: *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, Atlanta, Georgia, USA, ACM Press, 1999, pp. 295–304.

[6] A. Buldas, P. Laud and H. Lipmaa, Accountable certificate management using undeniable attestations, in: *7th ACM Conference on Computer and Communications Security*, S. Jajodia and P. Samarati, eds, Athens, Greece, ACM Press, ACM ISBN 1-58113-203-4, 2000, pp. 9–18.

[7] A. Buldas, P. Laud, H. Lipmaa and J. Villemson, Time-stamping with binary linking schemes, in: *Advances on Cryptology – CRYPTO '98*, Vol. 1462 of *Lecture Notes in Computer Science*, H. Krawczyk, ed., Santa Barbara, USA, International Association for Cryptologic Research, Springer-Verlag, 1998, pp. 486–501.

[8] A. Buldas, H. Lipmaa and B. Schoenmakers, Optimally efficient accountable time-stamping, in: *Public Key Cryptography '2000*, Vol. 1751 of *Lecture Notes in Computer Science*, H. Imai and Y. Zheng, eds, Melbourne, Victoria, Australia, Springer-Verlag, ISBN 3-540-66967-1, 2000, pp. 293–305.

[9] N. Barić and B. Pfitzmann, Collision-free accumulators and fail-stop signature schemes without trees, in: *Advances on Cryptology – EUROCRYPT '97*, Vol. 1233 of *Lecture Notes in Computer Science*, W. Fumy, ed., Konstanz, Germany, Springer-Verlag, ISBN 3-540-62975-0, 1997, pp. 480–494.

[10] B. Crispo and M. Lomas, A certification scheme for electronic commerce, in: *1996 Security Protocols International Workshop*, Vol. 1189 of *Lecture Notes in Computer Science*, Cambridge, United Kingdom, Springer-Verlag, ISBN 3-540-62494-5, 1996, pp. 19–32.

[11] I. Damgård, A design principle for Hash functions, in: *Advances in Cryptology – CRYPTO '89*, Vol. 435 of *Lecture Notes in Computer Science*, G. Brassard, ed., Santa Barbara, California, USA, 1989, Springer-Verlag, 1990, pp. 416–427.

[12] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE Transactions Information Theory* **IT-22**(November) (1976), 644–654.

[13] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F.M.A. Der Heide, H. Rohnert and R.E. Tarjan, Dynamic perfect hashing: upper and lower bounds, *SIAM Journal on Computing* **23**(4) (1994), 738–761.

[14] P. Elias, Universal codeword sets and representations for the integers, *IEEE Transactions on Information Theory* **IT-21**(March) (1975), 194–203.

[15] I. Gassko, P. Gemmel and P. MacKenzie, Efficient and fresh certification, in: *Public Key Cryptography '2000*, Vol. 1751 of *Lecture Notes in Computer Science*, H. Imai and Y. Zheng, eds, Melbourne, Victoria, Australia, Springer-Verlag, ISBN 3-540-66967-1, 2000, pp. 342–353.

[16] S.A. Haber and W.S. Stornetta, How to time-stamp a digital document, *Journal of Cryptology* **3**(2) (1991), 99–111.

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

[18] P. Kocher, On certificate revocation and validation, in: *Financial Cryptography – Second International Conference*, Vol. 1465 of *Lecture Notes in Computer Science*, R. Hirschfeld, ed., Anguilla, British West Indies, Springer-Verlag, 1998, pp. 172–177.

[19] L.M. Kohnfelder, Toward a practical public-key cryptosystem, BSc thesis, MIT Department of Electrical Engineering, 1978.

[20] H. Lipmaa, Secure and efficient time-stamping systems, PhD thesis, University of Tartu, June, 1999.

[21] P. Maniatis and M. Baker, Enabling the archival storage of signed documents, in: *Conference on File and Storage Technologies*, Monterey, California, USA, 2002 (to appear).

[22] R.C. Merkle, Protocols for public key cryptosystems, in: *Proceedings of the 1980 Symposium on Security and Privacy*, Oakland, California, USA, IEEE Computer Society Press, 1980.

[23] R.C. Merkle, One way Hash functions and DES, in: *Advances in Cryptology – CRYPTO '89*, Vol. 435 of *Lecture Notes in Computer Science*, G. Brassard, ed., Santa Barbara, California, USA, 1989, Springer-Verlag, 1990, pp. 428–446.

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

[25] M. Naor and K. Nissim, Certificate revocation and certificate update, *IEEE Journal on Selected Areas in Communications* **18**(4) (2000), 561–570.

[26] K. Nyberg, Commutativity in cryptography, in: *Proceedings of the First International Workshop on Functional Analysis at Trier University*, S. Dierolf, S. Dineen and P. Domanski, eds, Berlin, Germany, 1994, Walter de Gruyter & Co, 1996, pp. 331–342.

[27] K. Nyberg, Fast accumulated Hashing, in: *Fast Software Encryption: Third International Workshop*, Vol. 1039 of *Lecture Notes in Computer Science*, D. Gollman, ed., Cambridge, UK, Springer-Verlag, 1996, pp. 83–87.

[28] R.L. Rivest, Can we eliminate revocation lists? in: *Financial Cryptography – Second International Conference*, Vol. 1465 of *Lecture Notes in Computer Science*, R. Hirschfeld, ed., Anguilla, British West Indies, Springer-Verlag, 1998, pp. 178–183.

[29] T. Sander, Efficient accumulators without trapdoor, in: *The Second International Conference on Information and Communication Security*, Vol. 1726 of *Lecture Notes in Computer Science*, V. Varadharajan and Y. Mu, eds, Sydney, Australia, Springer-Verlag, ISBN 3-540-66682-6, 1999, pp. 252–262.

[30] J. Särs, Analysis and application of accountable certificate management, in: *Helsinki University of Technology Seminar on Network Security. Mobile Security '2000*, H. Lipmaa and H. Pehu-Lehtonen, eds, Sjökulla, Helsinki University of Technology, 2000, Available from http://www.tml.hut.fi/Opinnot/Tik-110.501/2000/, as of February 2002.