Cryptographic Accumulators: Definitions, Constructions and Applications

Nelly Fazio Antonio Nicolosi

Courant Institute of Mathematical Sciences
New York University
New York, NY, USA
{fazio,nicolosi}@cs.nyu.edu

Abstract

After their first appearance in the cryptographic community ten years ago, cryptographic accumulators have received a discontinuous attention from the researchers of the field. Although occasionally studied, there has been no systematic effort to organize the knowledge of the subject, abstracting away from the unnecessary details of specific proposals, so as to provide a reliable starting point for further investigation.

The goal of this paper is to present a complete picture of the topic, starting from the issue of defining an adequate formalization for cryptographic accumulators, and then moving on to a description of known constructions. A quick tour of interesting applications is then presented; finally, some possible lines for future development are suggested.

1 Introduction

In 1993, Benaloh and de Mare [4] introduced the notion of accumulator schemes as a decentralized alternative to digital signatures in the design of secure distributed protocols and described the basic functionalities and security properties that such schemes should provide to eliminate the necessity

This paper was written as part of the class **G22.3033-010 — Topics in Cryptography** (Instructor: *Prof. Victor Shoup*, Term: *Fall 2002.*)

of a trusted authority in applications such as time-stamping and membership testing. Since their introduction, accumulators have been considered as interesting primitives, but overall they have received less attention than what they seem to be worthy.

Furthermore, even the few works that have investigated their properties, do not constitutes systematic and organized effort: many applications have been proposed during the last ten years (time-stamping [4], fail-stop signatures [2], anonymous credential system [7], group signatures [7]), but each time a new, different definition of accumulator scheme have been introduced to best suite the specific setting.

Basically, an *accumulator scheme* is an algorithm to combine a large set of values into one, short accumulator, such that there is a short *witness* that a given value was indeed incorporated into the accumulator.
Recently, Camenisch and Lysyanskaya [7] introduced the more challenging notion of *dynamic accumulators*, which allow to dynamically delete and add elements from/into the original set. The proposed variant is particularly interesting since it achieves such higher degree of flexibility with a work per deletion and addition independent of the number of accumulated values, and without requiring knowledge of any sensitive information to update old witnesses to be consistent with the new accumulator.

1.1 Organization of the Paper

The informal definition stated above gives an intuitive idea of what an accumulator scheme should be, but it is too vague to be of any use in the security proof of any concrete application: a proper formalization is required.

Section 2 compares the few definitions proposed so far, and combines them to provide a novel model for accumulator schemes that aims to be general enough to be appropriate for any reasonable application and, at the same time, formal enough to allow a precise security analysis.

Section 3 presents an overview of the constructions that have been proposed to date, trying to follow, as much as possible, the line of development by which subsequent proposals have built on previous results.

To provide a practical motivation for the study of cryptographic accumulators, some interesting applications will be described in Section 4.

Finally, Section 5 concludes, suggesting some open problems and possible directions for future research.

2 Definitions

2.1 One-Way Accumulators

Accumulator schemes are a relatively new gadget in Cryptography, and there is still no standard formal definition for them. When they were first proposed by Benaloh and de Mare [4], *one-way accumulators* were defined as a family of one-way hash functions with an additional special property, called *quasi-commutativeness.*

Definition 1 (One-Way Hash Functions, [4])
A family of one-way hash functions *is an infinite sequence of families of functions* $\{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$*, where* $\mathcal{H}_\lambda \doteq \{h_k : X_k \times Y_k \to Z_k\}$*, with the following properties:*

1. *For any integer* λ *and any* $h_k \in \mathcal{H}_\lambda$*,* $h_k(\cdot,\cdot)$ *is computable in time polynomial in* λ*.*
2. *For any probabilistic, polynomial-time algorithm* $\mathcal{A}$*:*

$$Pr[h_k \overset{R}{\leftarrow} \mathcal{H}_\lambda; x \overset{R}{\leftarrow} X_k; y, y' \overset{R}{\leftarrow} Y_k; x' \leftarrow \mathcal{A}(1^\lambda, x, y, y') : h_k(x,y) = h_k(x',y')] < \mathsf{negl}(\lambda)$$

where the probability is taken over the random choice of h_k, x, y, y' *and the random coins of* $\mathcal{A}$*.*

Definition 2 (Quasi-commutativeness, [4])
A function $f : X \times Y \to X$ *is said to be* quasi-commutative *if:*

$$(\forall x \in X)(\forall y_1, y_2 \in Y)[f(f(x,y_1),y_2) = f(f(x,y_2),y_1)]$$

Definition 3 (One-Way Accumulators, [4])
A family of one-way accumulators *is a family of one-way hash functions each of which is quasi-commutative.*

This definition, although elegant and simple, does not make clear the basic functionality of secure accumulators, which intuitively consists in the capability of accumulating a set L of values into a unique, small value z in such a way that only for elements $y \in L$ it is possible to provide a proof that y actually has been accumulated within z. Furthermore, as first noticed in [12], the one-way property imposed by the second requirement of Definition 1 is often too weak for applications where the adversary can choose some of the values to be accumulated. A more appropriate level of security can be obtained by strengthening such requirement as to enforce a *strongly one-wayness* property, as follows:

Definition 4 (Strongly One-Way Hash Functions, [12])
A family of strongly one-way hash functions *is an infinite sequence of families of functions* $\{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$*, where* $\mathcal{H}_\lambda \doteq \{h_k : X_k \times Y_k \to Z_k\}$*, having the following properties:*

1. *For any integer* λ *and any* $h_k \in \mathcal{H}_\lambda$*,* $h_k(\cdot,\cdot)$ *is computable in time polynomial in* λ*.*

2. *For any probabilistic, polynomial-time algorithm* $\mathcal{A}$*:*

$$Pr[h_k \overset{R}{\leftarrow} \mathcal{H}_\lambda; x \overset{R}{\leftarrow} X_k; y \overset{R}{\leftarrow} Y_k; (x', y') \leftarrow \mathcal{A}(1^\lambda, x, y) : \\ y' \neq y \wedge h_k(x,y) = h_k(x',y')] < \mathsf{negl}(\lambda)$$

where the probability is taken over the random choice of h_k, x, y *and the random coins of* $\mathcal{A}$*.*

2.2 Collision-free Accumulators

Still, the strongly one-way property does not completely solve the issue of guaranteeing security in a setting where the adversary $\mathcal{A}$ actively participates in the selection of the values $y \in L$ to be accumulated (i.e. x and y in the definition above are no longer randomly chosen, but rather carefully selected by the adversary). To fill this gap, Barić and Pfitzmann [2] proposed the notion of *collision-free accumulators*:

Definition 5 (Accumulator Scheme, [2])
An accumulator scheme *is a 4-tuple of polynomial time algorithms* (Gen, Eval, Wit, Ver)*, where:*

- Gen*, the* key generation algorithm*, is a probabilistic algorithm used to set up the parameters of the accumulator.* Gen *takes as input a security parameter* 1^λ *and an accumulation threshold* N *(i.e. an upper bound on the total number of values that can be securely accumulated) and returns an accumulator key* k *from an appropriate key space* $\mathcal{K}_{\lambda,N}$*;*

- Eval*, the* evaluation algorithm*, is a probabilistic algorithm used to accumulate a set* $L \doteq \{y_1, \ldots, y_{N'}\}$ *of* $N' \leq N$ *elements from an efficiently-samplable domain* Y_k*, where* k *is some accumulator key from* $\mathcal{K}_{\lambda,N}$*.* Eval *receives as input* $(k, y_1, \ldots, y_{N'})$ *and returns an accumulated value (or accumulator)* $z \in Z_k$ *and some auxiliary information* aux*, which will be used by other algorithms. Notice that every execution of* Eval *on the same input* $(k, y_1, \ldots, y_{N'})$ *must yield the same accumulated value* z*, whereas the auxiliary information* aux *can differ;*

- Wit*, the* witness extraction algorithm*, is a probabilistic algorithm that takes as input an accumulator key* $k \in \mathcal{K}_{\lambda,N}$*, a value* $y_i \in Y_k$ *and the auxiliary information* aux *previously output (along with the accumulator* z*) by* $\mathsf{Eval}(k, y_1, \ldots, y_{N'})$*, and returns either a witness* w_i *(from an efficiently-samplable witness space* $\mathcal{W}_k$*) that "proves" that* y_i *was accumulated within* z *if this is indeed the case, or the special symbol* $\perp$ *if* $y_i \notin \{y_1, \ldots, y_{N'}\}$*.*

- Ver*, the* verification algorithm*, is a deterministic algorithm that, on input* (k, y_i, w_i, z)*, returns a* Yes/No *answer according to whether the witness* w_i *constitutes a valid proof that* y_i *has been accumulated within* z *or not.*

Such definition doesn't require the quasi-commutative property, and is thus slightly more general than that of [4]. Following the terminology of [2], the class of accumulators as defined in [4] will be referred to as *elementary accumulators*. Elementary accumulators can be easily adapted to satisfy Definition 5: the details of the construction follows.

The key generation algorithm simply consists of randomly selecting a function h_k from the family $\mathcal{H}_\lambda$ for the appropriate value of the security parameter, along with a random $x \in X_k$. To accumulate a set of values $y_1, \ldots, y_N \in Y_k$ into a single, short accumulator $z \in Z_k$, it suffices to iteratively apply the function h_k on $x, y_1, \ldots, y_N$; formally:

```
$\mathsf{Eval}(k, y_1, \ldots, y_N)$ :
            $parse\ k\ as\ (h_k, x)$
            $z_0 \leftarrow x$
            $z_i \leftarrow h_k(z_{i-1}, y_i), \qquad i = 1, \ldots, N$
            $z \leftarrow z_N$
            $\mathrm{aux} \leftarrow (y_1, \ldots, y_N)$
  **Output:** $(z, \mathrm{aux})$
```

By the quasi-commutativeness of h_k, it is clear that the order in which the y_i's are hashed to obtain z in the above algorithm is irrelevant. Another advantage of the quasi-commutative property of h_k is that the Wit algorithm can be realized in terms of Eval:

```
$\mathsf{Wit}(k, y_i, \mathrm{aux})$ :
            $\textit{parse } k \textit{ as } (h_k, x)$
            $\textit{parse } \mathrm{aux} \textit{ as } (y_1, \ldots, y_N)$
            if $y_i \notin \{y_1, \ldots, y_N\}$ then fail
            $L' \leftarrow \{y_1, \ldots, y_N\} \setminus \{y_i\}$
            $w_i \leftarrow \mathsf{Eval}(k, L')$
    Output: $w_i$
```

To verify a witness w_i for an element y_i and accumulator z, the Ver algorithm just checks whether $h_k(w_i, y_i) \stackrel{?}{=} z$, accepting or rejecting accordingly. It is easy to see that the verification algorithm always succeeds on witnesses honestly created with the Wit algorithm. This shows that the resulting accumulator scheme is functionally correct, completing the description of the generic transformation.

Definition 6 (N-times Collision-Freeness, [2])
*An accumulator scheme is said to be N-*times collision-free *if for any integer λ and for any probabilistic, polynomial-time algorithm $\mathcal{A}$:*

$$\begin{aligned} Pr[k \leftarrow \mathsf{Gen}(1^\lambda, N); (y_1, \ldots, y_N, y', w') \leftarrow \mathcal{A}(1^\lambda, N, k);& \\ (z, \mathrm{aux}) \leftarrow \mathsf{Eval}(k, y_1, \ldots, y_N) :& \\ (y_1, \ldots, y_N \in Y_k) \wedge (y' \notin \{y_1, \ldots, y_N\}) \wedge (\mathsf{Ver}(z, y', w') = \mathrm{Yes})] < \mathsf{negl}(\lambda)& \end{aligned}$$

where the probability is taken over the random coins of Gen*,* Eval *and $\mathcal{A}$.*

Definition 7 (Collision-Freeness, [2])
An accumulator scheme is said to be collision-free *if it is N-times collision-free for any value of N polynomial in λ.*

2.3 Dynamic Accumulators

For many applications of cryptographic accumulators, the set of interest L evolves with the time, after the accumulator z has been computed. A naive way of handling such situations would be to recompute the accumulator from scratch each time the set L changes: however, this would be highly impractical, especially when the set L is rather large. For this reason, recently Camenisch and Lysyanskaya [7] introduced the notion of *dynamic* *accumulators*, where the cost of computing the new accumulator z', resulting after adding or deleting an element from L, is independent of the number of accumulated values.

Definition 8 (Dynamic Accumulator Scheme, [7])
A dynamic accumulator scheme *is a 7-tuple of polynomial time algorithms* (Gen*,* Eval*,* Wit*,* Ver*,* Add*,* Del*,* Upd)*, where:*

- Gen*,* Eval*,* Wit*,* Ver *are as in a regular accumulator scheme (see Definition 5);*
- Add*, the* element addition algorithm*, is a (usually deterministic) algorithm that given an accumulator key k, a value $z \in Z_k$ obtained as the accumulation of some set L of less than N elements of Y_k, and another element $y' \in Y_k$, returns a new accumulator z' corresponding to the set $L \cup \{y'\}$, along with a witness $w' \in \mathcal{W}_k$ for y' and some* update *information* $\mathrm{aux}_{\mathsf{Add}}$ *which will be used by the* Upd *algorithm;*
- Del*, the* element deletion algorithm*, is a (usually deterministic) algorithm that given an accumulator key k, a value $z \in Z_k$ obtained as the accumulation of some set L of elements of Y_k, and an element $y' \in L$, returns a new accumulator z' corresponding to the set $L \setminus \{y'\}$, along with some* update *information* $\mathrm{aux}_{\mathsf{Del}}$ *which will be used by the* Upd *algorithm;*
- Upd*, the* witness update algorithm*, is a deterministic algorithm used to update the witness $w \in \mathcal{W}_k$ for an element $y \in Y_k$ previously accumulated within an accumulator $z \in Z_k$, after the addition (or deletion) of an element $y' \in Y_k \setminus \{y\}$ in (or from) z.* Upd *takes as input* $(k, y, w, b, \mathrm{aux}_{\mathsf{op}})$ *(where* op *is either* Add *or* Del*), and returns an updated witness w' that "proves" the presence of y within the updated accumulator z'.*

The definition presented above is actually slightly different from that of [7]: a few changes were made to attain a formalization more adherent to the original motivation of [4], i.e. avoiding the need for a trusted central authority. In fact, to meet the efficiency requirement for the element deletion algorithm Del, in [7] Camenisch and Lysyanskaya considered schemes where the accumulator key generation algorithm Gen outputs, along with the accumulator key k, some secret information t_k that enables an efficient implementation of the Del algorithm, but at the same time opens a potential hole in the security of the scheme itself. Thus, the trapdoor t_k should

only be available to an "accumulator manager", who is trusted to use this knowledge exclusively for the purpose of updating the accumulator after the removal of some elements, and not for deriving fake witnesses for values which have not been accumulated.
Such schemes, in which the presence of a trapdoor information is essential for the proper functioning of the system, will be referred to as *trapdoor dynamic accumulator schemes*.

Once the notion of accumulator scheme has been augmented to allow dynamic updates of the accumulator, it is necessary to reconsider the security requirements that should be satisfied. In [7], Camenisch and Lysyanskaya suggested the following notion of security, defined in terms of an adaptive attack scenario:

Definition 9 (Security for Dynamic Accumulator Schemes, [7]) *Let $\mathcal{S}$ be a dynamic accumulator scheme. An accumulator manager runs* $\mathcal{S}.\mathsf{Gen}$ *and gives the key k to the adversary. The set L of values to be accumulated and the corresponding accumulator z are initially set to be empty. Then, the adversary adaptively modifies the set L, by asking the accumulator manager to add and/or remove values $y_i \in Y_k$ in/from the accumulator as she wishes, obtaining back the new accumulator z (along with the witness w_i for the newly inserted y_i, in the case of an* Add *operation) and the associated* $\mathrm{aux}_{\mathsf{op}}$ *information necessary to update the other witnesses (where* op *is either* Add *or* Del*). At the end, the adversary attempts to produce a witness $w' \in \mathcal{W}_k$ for a y' not belonging to the current set L such that* $\mathcal{S}.\mathsf{Ver}(y', w', z) = \mathrm{Yes}$*. $\mathcal{S}$ is secure if the adversary has only a negligible probability of succeeding, where the probability is taken over the random coins of the adversary and of the accumulator manager.*

As it turns out, the above definition is essentially equivalent to Definition 7, in the sense that a dynamic accumulator scheme $\mathcal{S}$ satisfies it if and only if the "restricted" accumulator scheme $\tilde{\mathcal{S}} \doteq (\mathcal{S}.\mathsf{Gen}, \mathcal{S}.\mathsf{Eval}, \mathcal{S}.\mathsf{Wit}, \mathcal{S}.\mathsf{Ver})$ is collision-free. This is because the Add, Del and Upd algorithms, which are available in $\mathcal{S}$ but not in $\tilde{\mathcal{S}}$, do not provide any new functionality, but only allow to achieve higher computational and communication efficiency. Indeed, all three of these extra algorithms can be simulated by $\tilde{\mathcal{S}}$ at the cost of a polynomial (although considerable) time overhead. Hence, if there were an attacker breaking $\mathcal{S}$ in the attack scenario of Definition 9, it would be possible to construct an attacker breaking the collision-freeness of $\tilde{\mathcal{S}}$, via a straightforward simulation of the Add, Del and Upd algorithms.

Things are slightly different for trapdoor dynamic accumulator schemes. Namely, the security of a trapdoor dynamic accumulator scheme $\mathcal{S}$ does

not immediately follow from the collision-freeness of the restricted accumulator scheme $\tilde{\mathcal{S}}$. But before arguing more about this, it is necessary to extend Definition 9 to the trapdoor case. This can be easily done by adding the requirement that, after executing the $\mathcal{S}.\mathsf{Gen}$ algorithm, the accumulator manager keeps secret the trapdoor t_k (corresponding to the accumulator key k), and only uses it to respond to Del queries.
Now, to see why the above equivalency argument does not apply "as is" to trapdoor dynamic accumulator schemes, observe that it could no longer be possible to simulate Del queries, since the Del algorithm of a trapdoor scheme requires knowledge of the trapdoor corresponding to the accumulator key, whereas there is no trapdoor information available in the restricted accumulator scheme $\tilde{\mathcal{S}}$.

Nevertheless, for specific schemes a simulation argument may still be possible: this is indeed the case for the trapdoor dynamic accumulator scheme constructed in [7], which will be presented in Section 3.

3 Constructions

The goal of the previous section was to formally define the functionalities and security properties that accumulator schemes should enjoy to be useful. For such definitional effort not to have been vain, another important question must be answered, namely whether cryptographic accumulators of any of the proposed kinds do exist at all, and if so, what computational or combinatoric assumptions are needed to construct such objects.

The current section aims to answer this question, by presenting an overview of the constructions that have been proposed to date in the literature. In brief, two main approaches have been pursued, quite different from each other: one based on (a variant of) a well-known number theoretic assumption, and the other based on families of functions with strong (pseudo-)random properties.

3.1 The Number-Theoretic Approach

In [4], Benaloh and de Mare presented a one-way elementary accumulator based on the exponentiation function modulo a composite number n. Such function is quasi-commutative, and for a suitable choice of the modulus n it is also believed to be one-way over an appropriate domain. To specify more precisely such conditions, a little bit of notation and terminology is needed.

Definition 10 (Safe Prime, [4])
A number p is said to be a safe prime *if $p = 2p' + 1$ and both p and p' are odd primes. For $\lambda \in \mathbb{N}$, let S-Prime$_\lambda$ be the set of safe primes of size λ.*

Definition 11 (Rigid Integer, [4])
A number n is said to be a rigid integer *if $n = pq$ where p and q are distinct safe primes such that $|p| = |q|$. For $\lambda \in \mathbb{N}$, let Rigid$_\lambda$ be the set of rigid integers of size λ.*

According to Definition 3, to fully specify an elementary accumulator, it suffices to describe the generic function $h_k \in \mathcal{H}_\lambda$, for an arbitrary $\lambda \in \mathbb{N}$. For the case of the elementary accumulator of Benaloh and de Mare, h_k is defined as follows. The key k is just a rigid integer n of size λ. Let $G \subset \mathbb{Z}_n^*$ be the group of quadratic residues modulo n. Then G is a cyclic group of order $n' \doteq \varphi(n)/4 = p'q'$. Hence, for any y relatively prime with p' and q', the y-power map is a permutation over G. Thus, a one-way, quasi-commutative function can be defined as:

$$h_k : G \times \mathbb{Z}_{n'}^* \to G$$
$$h_k : (x, y) \mapsto x^y \bmod n$$

Quasi-commutativeness of this function is clear. To show the one-way property, we need to prove that it is infeasible to compute a x' such that $h_k(x, y) = h_k(x', y')$ for random λ-bit rigid integer n and random $x \in G$, $y, y' \in \mathbb{Z}_{n'}^*$. Letting $z \doteq h_k(x, y) = x^y \bmod n$, this is equivalent to assert the computational hardness of computing a y'-th root of a random $z \in G$, even knowing a y-th root of z (i.e. x). Using a well-known algebraic manipulation due to Shamir [15], it is possible to show that knowledge of a y-th root of a random element of G doesn't help in finding a y'-th root of the same element, as long as y' doesn't divide y. Since in the current setting both y and y' are drawn independently at random from $\mathbb{Z}_{n'}^*$, the chance of y' being a divisor of y are negligible [11]: hence, the one-wayness of the proposed elementary accumulator is essentially equivalent to the difficulty of finding a root of random index of a random element of G, which is the well known RSA assumption.

There still is a technical problem with the above construction. The issue is that, for an accumulator to be useful, the domain from which the y's are drawn (which in this case is the set $\mathbb{Z}_{n'}^*$) should be efficiently sampleable, which means that n' should be publicly known. But since $n' \doteq \varphi(n)/4$, publishing n' would reveal $\varphi(n)$, thus allowing to efficiently recover of the factorization of n, which makes the RSA problem easily solvable.

Fortunately, there is an easy way out of this apparently circular situation: just consider the extension of the function h_k to $G \times \mathbb{Z}_{n/4}$. The domain for the y's is now efficiently sampleable; as for the one-wayness, the negligibility of the statistical distance between the uniform distribution over $\mathbb{Z}^*_{n'}$ and the uniform distribution over $\mathbb{Z}_{n/4}$ makes it possible to adjust the above argument so as to work for the new, broader domain.

As mentioned in Section 2, it was early recognized that one-wayness is often not enough for a family $\{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$ to be safely used to accumulate a list of values into a single, short accumulator. Indeed, if the adversary is allowed to attempt forgery of a witness w' for an element y' *of her choice* (rather than uniformly distributed over the domain Y_k), than the one-way property doesn't help. In particular, the argument outlined above to prove the one-wayness of the elementary accumulator of Benaloh and de Mare won't go through: the weak point is that in the strong one-way setting (see Definition 4) the probability of y' being a divisor of y is no longer negligible, as the adversary will have a good chance of finding a factor of a random element $y \in \mathbb{Z}_{n/4}$.

In [4], Benaloh and de Mare informally suggested to accommodate this problem by introducing a random oracle [3] in the construction. Indeed, given a function $H : \bar{Y}_k \rightarrow Y_k$ that behaves like a random oracle, it is possible to transform any one-way accumulator, with generic function $h_k : X_k \times Y_k \rightarrow X_k$, into a strongly one-way accumulator, whose generic function is $\bar{h}_k : X_k \times \bar{Y}_k \rightarrow X_k$, defined as $\bar{h}_k(x, \bar{y}) = h_k(x, H(\bar{y}))$. In this way, no matter how clever the adversary is in choosing the element $\bar{y}' \in \bar{Y}_k$ for which she wants to attempt a witness-forgery, she will still be facing the problem of inverting the one-way function h_k at the random point $y' \doteq H(\bar{y}')$, which is assumed to be hard.

However, the above solution relies on the Random Oracle Model, and thus provides just an heuristic argument [8], rather than a full proof of security, for concrete instantiations of the proposed construction. Alternatively, as proposed in [2] by Barić and Pfitzmann, it is possible to achieve strong one-wayness from the RSA-based construction of Benaloh and de Mare by restricting the domain Y_k to contain only prime numbers, and basing the security analysis on a stronger variant of the RSA assumption, in the standard model. More precisely, define *Primes* to be the set of all prime numbers, and let $Y_k \doteq \mathit{Primes} \cap \mathbb{Z}_{n/4}$. Consider the restriction of the modular exponentiation function to $G \times Y_k$, i.e.:

$$\hat{h}_k : G \times Y_k \to G$$
$$\hat{h}_k : (x, y) \mapsto x^y \bmod n$$

Now that the y's are subject to the additional constraint of being primes, the adversary can't take advantage of her knowledge of a root of random index y when trying to find another pair (x', y') such that $(x')^{y'} \bmod n = x^y \bmod n \doteq z$. However, the difficulty of such computation is not equivalent to the RSA assumption, since the adversary can pick her favorite index for the root extraction (whereas in the standard RSA assumption the index is chosen at random). What is needed here is a different computational assumption, first introduced by Barić and Pfitzmann in [2] under the name of *strong RSA assumption*:

Assumption 1 (Strong RSA Assumption, [2])
For any integer λ and for any probabilistic, polynomial-time algorithm $\mathcal{A}$:

$$Pr[n \overset{R}{\leftarrow} Rigid_\lambda; k \leftarrow n; z \overset{R}{\leftarrow} G; (x', y') \leftarrow \mathcal{A}(1^\lambda, n, z) : \\ y' \in Y_k \wedge (x')^{y'} = z \bmod n] < \mathsf{negl}(\lambda)$$

where the probability is taken over the random choice of n and z, and the random coins of $\mathcal{A}$.

In [2], Barić and Pfitzmann showed that the strong RSA assumption not only entails the strong one-wayness of the family of functions $\hat{\mathcal{H}}_\lambda \doteq \{\hat{h}_k \,|\, n \in Rigid_\lambda, k \leftarrow n\}$, but also suffices to prove that the accumulator scheme $\mathcal{S}^{\mathsf{BF}}$, obtained extending the elementary accumulator $\{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$ according to the generic transformation presented in Section 2.2, also satisfies the stronger notion of collision-freeness (for a proof, see Theorem 5 of [2]).

Adding an additional twist to the above construction, Camenisch and Lysyanskaya obtained a secure trapdoor dynamic accumulator scheme [7]. The crux of their result lies in the observation that the modular exponentiation function, which is the core of the collision-free accumulator discussed above, comes with an associated trapdoor information, namely the factorization of n as $p \cdot q$ (or, equivalently, the order $n' \doteq \varphi(n)/4$ of the group G). Although the presence of such auxiliary information was of no help for the construction of Barić and Pfitzmann (and indeed, some research has focused on the issue of defining accumulator schemes without trapdoor [12, 13], [14]), the availability of a trapdoor turned out to be a valuable tool for speeding up the Del algorithm, which is the whole point of a trapdoor dynamic accumulator scheme.

More in detail, the scheme of [7] is based on the exponentiation function modulo a rigid integer n (whose factorization is know only to a trusted accumulator manager), defined over the domain $G \times \tilde{Y}_k$, where $\tilde{Y}_k \doteq Primes \cap [A, B]$ and $2 < A < B < A^2 < n/4$:[1]

$$\tilde{h}_k : G \times \tilde{Y}_k \to G$$
$$\tilde{h}_k : (x, y) \mapsto x^y \bmod n$$

Notice that now the adversary cannot break the collision-freeness of the scheme even if she is allowed to attempt witness-forgery for a composite number $y' \in [A, B] \setminus \tilde{Y}_k$. In fact, the only way she could take advantage of such "out-of-domain" choice would be to let $y' \doteq y_1 \cdot y_2$, for some $y_1, y_2 \in \tilde{Y}_k$ that have already been accumulated. But this possibility is ruled out by the constraint $B < A^2$: indeed, if $y' \in [A, B]$, then $B \geq y' \doteq y_1 \cdot y_2 \geq A \cdot A = A^2 \Rightarrow B \geq A^2$, a contradiction.

To add an element y' into an already existing accumulator z, it suffices to apply the $\tilde{h}_k$ function once more:

```
$\mathsf{Add}(k, z, y')$ :
        $z' \leftarrow \tilde{h}_k(z, y')$
        $w' \leftarrow z$
        $\mathrm{aux}_{\mathsf{Add}} \leftarrow y'$
**Output:** $(z', w', \mathrm{aux}_{\mathsf{Add}})$
```

The trick to efficiently delete an element y' from an accumulator z given the trapdoor information n' is to compute the exponent $\tilde{y}$ that "undoes" the y'-power map, i.e. $\tilde{y} \doteq (y')^{-1} \bmod n'$. This can be efficiently done using the Extended Euclidean Algorithm $\mathsf{Ext\text{-}GCD}$. The rest is administrivia:

```
$\mathsf{Del}(k, n', z, y')$ :
        $\tilde{y} \leftarrow (y')^{-1} \bmod n'$
        $z' \leftarrow \tilde{h}_k(z, \tilde{y})$
        $\mathrm{aux}_{\mathsf{Del}} \leftarrow (y', z')$
**Output:** $(z', \mathrm{aux}_{\mathsf{Del}})$
```

[1] In this case, the accumulator key k is the pair (n, x).

Finally, the Upd algorithm operates as follows:

```
$\mathsf{Upd}(k, y, w, \mathsf{op}, \mathrm{aux}_{\mathsf{op}})$ :
        **if** $\mathsf{op} = \mathsf{Add}$ **then**
            $y_{\mathsf{Add}} \leftarrow \mathrm{aux}_{\mathsf{op}}$
            $w' \leftarrow \tilde{h}_k(w, y_{\mathsf{Add}})$
        **else**
            *parse* $\mathrm{aux}_{\mathsf{op}}$ *as* $(y_{\mathsf{Del}}, z')$
            $(d, a, b) \leftarrow \mathsf{Ext\text{-}GCD}(y, y_{\mathsf{Del}})$
            **if** $d \neq 1$ **then fail**
            $w' \leftarrow \tilde{h}_k(z', a) \cdot \tilde{h}_k(w, b)$
        **endif**
    **Output:** $w'$
```

The Upd algorithm is in fact comprised of two distinct parts, corresponding to the Add and Del operations. To update a witness w for y after the insertion of y_{Add} in z, it suffices to return as new witness the value $w' \doteq w^{y_{\mathsf{Add}}} \bmod n$; indeed, it holds that:

$$(w')^y \equiv (w^{y_{\mathsf{Add}}})^y \equiv (w^y)^{y_{\mathsf{Add}}} \equiv z^{y_{\mathsf{Add}}} \equiv z' \pmod n$$

where the last congruence holds by the definition of the Add algorithm presented above.
To update a witness w for y after the removal of y_{Del} from z (resulting in the new accumulator z'), one has to use the $\mathsf{Ext\text{-}GCD}$ algorithm to compute integers a, b such that $ay + by_{\mathsf{Del}} = 1$ (notice that this is possible only if $y \neq y_{\mathsf{Del}}$ since otherwise $\mathsf{GCD}(y, y_{\mathsf{Del}}) = y \neq 1$). Then, letting $w' \doteq (z')^a \cdot w^b \bmod n$ suffices, since:

$$(w')^y \equiv (z')^{ay} \cdot w^{by} \equiv (z')^{ay} \cdot z^b \overset{(\sharp)}{\equiv} (z')^{ay} \cdot (z')^{by_{\mathsf{Del}}} \equiv (z')^{ay+by_{\mathsf{Del}}} \equiv z' \pmod n$$

where the congruence $(\sharp)$ holds since $z = (z')^{y_{\mathsf{Del}}} \bmod n$ in virtue of the above implementation of the Del algorithm.

To complete the presentation of the Camenisch and Lysyanskaya's trapdoor dynamic accumulator scheme $\mathcal{S}^{\mathsf{CL}}$, it remains to be proved that it is indeed secure under Definition 9. A sketch of such proof is given below: for more details, see Theorem 2 of [7].

The crucial point of the security proof is that the trapdoor information n' is used in the Del algorithm just to speed up the computation. Indeed, access to the Add and Del oracles can be simulated by a stateful polynomial time algorithm that doesn't know the trapdoor. Such simulation would proceed as follows. At the beginning, create an initially empty set L that will be updated during the simulation to stay in sync with the values present, at any given moment, within the accumulator. When asked to insert an element y' into z, update L to reflect the insertion, and simply execute the Add algorithm above on input (k, y', z). To respond to a Del query, check whether $y' \in L$, and fail if not; then, update L accordingly and recompute the accumulator from scratch, executing $\mathsf{Eval}(k, L)$. Notice that this computation is much slower than that of Del, but is still polynomial time and produce the required result without the help of any trapdoor information.

To complete the argument, just notice that any probabilistic, polynomial-time adversary $\mathcal{A}$ attacking the security of the trapdoor dynamic accumulator scheme $\mathcal{S}^{\mathsf{CL}}$ can be turned, using the above simulation, into another probabilistic, polynomial-time adversary $\tilde{\mathcal{A}}$ attacking the collision-freeness of the restricted accumulator scheme $\tilde{\mathcal{S}}^{\mathsf{CL}} \doteq (\mathcal{S}^{\mathsf{CL}}.\mathsf{Gen}, \mathcal{S}^{\mathsf{CL}}.\mathsf{Eval}, \mathcal{S}^{\mathsf{CL}}.\mathsf{Wit}, \mathcal{S}^{\mathsf{CL}}.\mathsf{Ver})$. Now, it is easy to verify that $\tilde{\mathcal{S}}^{\mathsf{CL}}$ is just a syntactic variation of the accumulator scheme $\mathcal{S}^{\mathsf{BP}}$ of Barić and Pfitzmann, which was argued earlier to be collision-free under the strong RSA assumption. Hence, the trapdoor dynamic accumulator scheme $\mathcal{S}^{\mathsf{CL}}$ satisfies the requirements of Definition 9 under the same computational assumption.

3.2 The Combinatoric Approach

An alternative approach to the construction of cryptographic accumulators has been pursued by Nyberg in [12, 13]. These two papers are quite similar to each other, the second presenting a slightly improved solution. Thus, the following presentation will focus on their common features and on the specific details of the accumulator proposed in [13].

One of the motivations for Nyberg's work was to obtain an "absolute" accumulator, in the sense that the construction was to be provably secure and not based on any trapdoor information, possibly known to some third party — in line with the original intention of Benaloh and de Mare of providing a fully decentralized alternative to digital signatures. And in fact, both of the accumulators proposed by Nyberg are trapdoorless and, even better, their security does not depend on any computational assumption. However, the argument provided in [13] to prove the (strong) one-wayness of Nyberg's construction assumes the availability of a hash function producing long, truly random hash codes; in other words, the accumulator scheme is proven secure in the Random Oracle Model [3, 8].

Formally, the construction of [13] defines an elementary accumulator (see Definition 4) $\{\mathcal{H}_\lambda^{\mathsf{Nyb}}\}_{\lambda\in\mathbb{N}}$. Assume that $N = 2^d$ is an upper bound on the number of items to be accumulated within any single accumulator z. Define $\ell \doteq \frac{e}{\lg e}\lambda N \lg N$, where e is Neper's number and lg denotes base 2 logarithms, and let $H : \{0,1\}^* \to \{0,1\}^\ell$ be an hash function that in the security analysis will be treated as a random oracle. For simplicity, suppose that ℓ is an integer multiple of d, i.e. $\ell = rd$ for some integer r. The key k for the accumulator scheme will simply be a random bit string of length r: $k \overset{R}{\leftarrow} \{0,1\}^r$. Furthermore, define $X_k \doteq \{0,1\}^r$ and $Y_k \doteq \{0,1\}^*$.

The generic function $h_k^{\mathsf{Nyb}} \in \mathcal{H}_\lambda^{\mathsf{Nyb}}$ maps a pair $(x, y) \in X_k \times Y_k$ to an element $z \in X_k$ as follows. First, consider the (random) image $\bar{y}$ of y under the random oracle H, i.e. $\bar{y} \doteq H(y)$. Since H produces hash codes of length $\ell = rd$, $\bar{y}$ can be viewed as an r-digit number in base 2^d, i.e. $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_r)$, where $|\bar{y}_j| = d, j = 1, \ldots, r$. Define the bit string $\alpha_r(\bar{y}) \doteq b = (b_1, \ldots, b_r)$ as the result of replacing each $\bar{y}_j$ with a 0 if and only if $\bar{y}_j = 0$:

$$b_j \doteq \begin{cases} 0 & \textbf{if } \bar{y}_j = 0 \\ 1 & \textbf{if } \bar{y}_j \neq 0 \end{cases}$$

Under the assumption that H behaves as a random oracle, each b_j can be thought as an independent binary random variable, taking the value 0 with probability 2^{-d}.

In this way, $y \in Y_k$ has been mapped into a bit string b of length r, with much more 1's then 0's. Finally, compute the bitwise and (denoted with $\odot$) of x and b; summarizing:

$$h_k^{\mathsf{Nyb}} : (x, y) \mapsto x \odot \alpha_r(H(y))$$

The quasi-commutativeness of the function h_k^{Nyb} follows from the commutative and associative properties of the $\odot$ operator. Indeed, for any $x \in X_k, y_1, y_2 \in Y_k$, it holds that:

$$\begin{aligned} h_k^{Nyb}(h_k^{Nyb}(x, y_1), y_2) &= (x \odot \alpha_r(H(y_1))) \odot \alpha_r(H(y_2)) = \\ &= x \odot (\alpha_r(H(y_1)) \odot \alpha_r(H(y_2))) = \\ &= x \odot (\alpha_r(H(y_2)) \odot \alpha_r(H(y_1))) = \\ &= (x \odot \alpha_r(H(y_2))) \odot \alpha_r(H(y_1)) = \\ &= h_k^{Nyb}(h_k^{Nyb}(x, y_2), y_1) \end{aligned}$$

The strong one-way property of the elementary accumulator $\{\mathcal{H}_\lambda^{\mathsf{Nyb}}\}_{\lambda\in\mathbb{N}}$ hinges upon the following result of Nyberg (Theorem 1 of [13]):

Theorem 1 *Let $b_j^{(i)}$ and c_j be independent binary random variables such that $Pr[b_j^{(i)} = 0] = Pr[c_j = 0] = 2^{-d}$, for $i = 1, \ldots, m$, $j = 1, \ldots, r$ and $m \leq N$. Let $z = (z_1, \ldots, z_r)$ be the bitwise* and *of the r-bit strings $b^{(i)} \doteq (b_1^{(i)}, \ldots, b_r^{(i)}), i = 1, \ldots, m$, Then the probability that, for all $j = 1, \ldots, r$, it holds that $c_j = 0$ only if $z_j = 0$, is equal to $(1 - 2^{-d}(1 - 2^{-d})^m)^r$.*

Proof. For each $j = 1, \ldots, r$, the probability that $c_j = 0$ and $z_j = 1$ is $2^{-d}(1 - 2^{-d})^m$: the claimed result follows. ∎

The above theorem provides an information-theoretic bound on the probability of finding a forged element $c = (c_1, \ldots, c_r)$ consistent with an accumulator $z = (z_1, \ldots, z_r)$ resulting from the cumulative hash of the set of values $\{b^{(1)}, \ldots, b^{(m)}\}$. For the chosen length of the hash codes produced by the random oracle H, this bound is negligibly small:

$$(1 - 2^{-d}(1 - 2^{-d})^m)^r \leq \left(1 - \frac{1}{N}\left(1 - \frac{1}{N}\right)^N\right)^r \approx \left(1 - \frac{1}{eN}\right)^r \approx e^{-\frac{r}{eN}} = 2^{-\lambda}$$

where the last equality holds due to the careful choice of $\ell \doteq \frac{e}{\lg e}\lambda N \lg N$, and the relations $N = 2^d$ and $r = \ell/d$.

A few remarks are worth being mentioned regarding the combinatoric construction described above. First, all the operations involved in the computation of h_k^{Nyb} are simple bit manipulations, and could thus be implemented quite efficiently. Second, the verification algorithm doesn't even need a witness w to check whether an element y has been accumulated within the accumulator $z = (z_1, \ldots, z_r)$: just compute $\alpha_r(H(y)) \doteq b = (b_1, \ldots, b_r)$, and verify that $z_j = 0$ whenever $b_j = 0$. However, in order for "false positive" to happen with negligible probability $2^{-\lambda}$, the length of the hash codes produced by the random oracle H has to be prohibitively large, namely proportional to $\lambda N \lg N$,[2] even though the accumulator z itself will be shorter of a factor of $\lg N$.

Finally, observe that it is easy to augment the above accumulator scheme with an Add operation, since it suffices to apply the function h_k^{Nyb} one more time. This means that Nyberg's construction is suitable for applications where the set of accumulated values dynamically changes, but in a monotonically increasing fashion.

From a theoretical point of view, Nyberg's construction is also interesting because it shows that provably secure accumulators can be build without any

[2]Recall that N is an upper bound on the total number of elements to be cumulatively hashed within a single accumulator.

computational assumption. On the other hand, its reliance on the Random Oracle Model and its heavy space-inefficiency make the construction of little practical interest.

4 Applications

4.1 Time-Stamping and Membership Testing [4]

As noticed in Section 1, Benaloh and deMare's original motivation for the study of cryptographic accumulators was to provide a primitive for the design of space-efficient, distributed protocols not requiring a *trusted* party.

The first application considered by [4] is *Time-Stamping* [10], consisting of a protocol by which a "publication" date can be attached to any document, to provide an ordering criterion to determine the relative positions of any two documents. In the presence of a trusted central authority C, a simple solution can be obtained by having C signing, at discrete time instants called *rounds*, the set of all documents produced by all the m users of the system during the given round.

To reduce the amount of trust to be placed on the central authority, it is possible to augment the above scheme by requiring active cooperation from all those participants who contributed documents to be published. While this approach can eliminate the need for a trusted party, it can be shown that it still requires a storage per user per round logarithmic in the total number of participants. As it turns out, this space overhead can be made constant using an accumulator scheme. The protocol would proceed as follows.

At round t, a new accumulator key k_t is generated, and each of the m participants encodes the messages he/she wishes to publish as an element $y_{t,i}$ of the input domain Y_{k_t}, $i = 1, \ldots, m$. All the $y_{t,i}$'s values are then accumulated together, computing $(z_t, \mathrm{aux}_t) \doteq \mathsf{Eval}(k_t, y_{t,1}, \ldots, y_{t,m})$, and the participants store the resulting accumulator value z_t, along with the witness $w_{t,i} \doteq \mathsf{Wit}(k_t, w_{t,i}, \mathrm{aux}_t)$ for their own value $y_{t,i}$. In this way, to later show that a given document was time-stamped at round t corresponding to the accumulator value z_t, user i just needs to show that such document is encoded within the value $y_{t,i}$, and then provide the witness $w_{t,i}$ to prove that $y_{t,i}$ is indeed one of the values accumulated within z_t.

Membership testing can be obtained as a simple variation of the previous construction. Basically, each group corresponds to a round of the time-stamping protocol and the group members play the role of the time-stamped documents. Additionally, if the accumulated value z (which can be though of as a very compact representation of the membership list) is made available to users outside the group, then every member can prove to non-members that he/she belongs to the group without having to disclose the entire list of members.

4.2 Membership Revocation in ID Escrow Schemes [7]

Interestingly, dynamic accumulators can also help in improving the *time-efficiency* of complex distributed protocols. One such example can be found in [7], where Camenisch and Lysyanskaya showed how dynamic accumulators can be used to enable efficient membership revocation in the anonymous setting. In fact, the technique of [7] can be used to add a membership revocation capability to Identity Escrow schemes, Group Signature schemes and Anonymous Credential systems: to illustrate the basic ideas behind such technique, a brief description of the construction for the case of Identity Escrow schemes is included below.

An *Identity Escrow* (ID Escrow) scheme allows users to identify themselves as members of a group without disclosing their identity beyond the very fact that they do belong to the group in question. To discourage users from abusing their anonymity, the group manager is empowered with an "escrow capability". These two seemingly contradictory requirements are usually satisfied by having the identification process take the form of an interactive protocol between the group member and a (possibly external) verifier. To ensure user anonymity under normal circumstances, the transcript of such interaction should reveal no information about the identity of the group member to the eyes of any computationally-bounded party; the group manager, however, holds some trapdoor information, which enables him to "read between the lines" of the transcript and recover the identity of the user, should the need for such escrow operation arise.

It is clear that this escrow capability does not prevent a user from misusing her anonymity; furthermore, it only provides a way to uncover the identity of the abuser once a misusage has been detect by some other (maybe non-cryptographic) means. To be more effective, the escrow mechanism could be coupled with some form of membership revocation. Then, upon detection of a case of anonymity abuse, the guilty member could be traced and removed from the group.

Although satisfactory ID Escrow schemes have been proposed (e.g. [1]), none of them enjoyed efficient membership revocation. To fill this gap, Camenisch and Lysyanskaya presented in [7] an ID Escrow scheme which augments the proposal of [1] with the use of dynamic accumulators to keep track of the current members of the group: supporting membership revocation is then as simple as removing an element from the accumulated value.

Recall that in the ID Escrow scheme of [1], the group's public key consists essentially of four components: the description of a one-way function, the description of a commitment scheme, the public key for a signature scheme, and the public key for an encryption scheme. The group manager keeps for himself the secret keys corresponding to the signature and encryption schemes.

To join the group, a user u engages in an interactive Join protocol with the group manager. At the end of the protocol, user u obtains a unique membership certificate (x, v, e), where (v, e) is a signature (under the group manager's signing key) of a random message m_u, of which u only knows a preimage x under the system's one-way function.

To prevent malicious users from gaining extra information by pooling their certificates together, the Join protocol ensures that each message m_u associated to a user u is actually random by having the group manager contributing some randomness to the choice of the message's preimage x. This is done in a secure way that guarantees that the group manager doesn't learn anything about x (beyond the fact that its image under the one-way function is m_u). Were this not the case, the group manager would have been able to "frame" any user by first misusing the corresponding membership certificate, and then escrowing the underlying identity.

Once a user u has obtained a certificate from the group manager, she can prove her membership to the group via a Zero Knowledge Proof of Knowledge of both the signature (v, e) and the preimage x of the signed message. To enable the identity escrow, the proof of knowledge also includes (in a verifiable fashion) an encryption (under the group public encryption key) of the message m_u. In this way, if the verifier keeps the transcript of the identification protocol, the group manager will be able to uncover the identity of the member that participated in that specific interaction.

As mentioned above, the idea of Camenisch and Lysyanskaya [7] to add membership revocation to the ID Escrow scheme of [1] is to include in the group public key an accumulator z, whose value evolves dynamically as users join and leave the group. The protocols of the scheme are modified to make use of such accumulator z as follows.

During the Join protocol, the group manager also takes care of adding (part of) the user's membership certificate[3] to the current value of the accumulator z, and gives the user the corresponding witness w.

[3] In the specific construction of [7], only the prime e of a user's certificate (x, v, e) gets actually accumulated within z.

At the same time, the group manager makes available, in a publicly readable "group bulletin board", the auxiliary information aux_{Add} to allow existing members to update their own witnesses, using the Upd algorithm.

In a dynamic setting, where users can be removed from the group, proving knowledge of a membership certificate is no longer enough to prove (current) membership to the group: a group member u should also convince the verifier that her membership certificate has not been invalidated. To this aim, the verifier has to keep track of the current value of the accumulator z (which is the evolving part of the group public key); the user and the verifier then engage in a more involved Zero Knowledge Proof of Knowledge, by which the user u proves, beside her knowledge of both the signature (v, e) and of the preimage x of the signed message, that she also knows a witness w for the presence of e in the current accumulated value z. Clearly, if the group manager decides to remove a user from the group, and updates the accumulator z accordingly, such Zero Knowledge Proof of Knowledge would fail, and the verifier would (correctly) reject the user membership claim.

As a last remark, it is worth noticing that the idea of using an accumulator to keep track of user membership would work with any ID Escrow scheme; however, the accumulator scheme described in Section 3.1 fit particularly well with the ID Escrow scheme of [1], allowing for efficient and compact implementation of the scheme's protocols.

4.3 Other Applications [2, 9]

In [2], Barić and Pfitzmann showed how the "compression" property of accumulators can be exploited to reduce the space requirement of fail-stop signature to $O(1)$. More recently, Goodrich et al. proposed a *size-oblivious authenticated dictionary* [9] based on the space-efficiency property of cryptographic accumulators. The reader is referred to [2, 9] for the details of those constructions.

5 Conclusion and Open Problems

Accumulator schemes are a powerful primitive. We have mentioned some of the cases in which their cryptographic properties turned out to be valuable tools to improve the space- and time-efficiency of secure distributed protocols.

Somewhat surprisingly, they haven't been yet subject of thorough investigation. There still is no well-established formalization for them: we have compared the existing proposals, and tried to combine them into a single satisfactory definition.

We belive that accumulators schemes constitute a promising field for research, and that original contributions are possible in at least two orthogonal directions: providing new constructions and devising more applications. We conclude proposing a list of interesting questions regarding the design of alternative constructions.

1. Recently, the cryptographic properties of Gap Diffie-Hellman Groups [6] and Bilinear Maps [5] have been successfully employed to obtain a wide variety of primitives. Is it possible to design space-efficient accumulators based on these assumptions?

2. The dynamic accumulator scheme proposed by Camenisch and Lysyanskaya [7] features an update algorithm Upd that must be applied to a witness after each *single* addition/removal operation. Hence, the time to bring a witness up-to-date after a batch of N operations is proportional to N. Is it possible to construct dynamic accumulators in which the time to update a witness can be made independent from the number of changes to the accumulated value?

References

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik, *A practical and provably secure coalition-resistant group signature scheme*, Advances in Cryptology—Crypto'00, LNCS, vol. 1880, Springer-Verlag, 2000, pp. 255–270.

[2] N. Barić and B. Pfitzmann, *Collision-free accumulators and fail-stop signature schemes without trees*, Advances in Cryptology—Eurocrypt'97, LNCS, vol. 1233, Springer-Verlag, 1997, pp. 480–494.

[3] B. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, Proceedings of the 1st ACM conference on Computer and communications security, ACM Press, 1993, pp. 62–73.

[4] J. Benaloh and M. de Mare, *One-way accumulators: a decentralized alternative to digital signatures*, Advances in Cryptology—Eurocrypt'93, LNCS, vol. 765, Springer-Verlag, 1993, pp. 274–285.

[5] D. Boneh and M. Franklin, *An Efficient Public Key Traitor Tracing Scheme*, Manuscript, available at: `http://crypto.stanford.edu/~dabo/pubs.html`, 2001.

[6] D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the Weil pairing*, Advances in Cryptology—Asiacrypt'01, LNCS, vol. 2248, Springer-Verlag, 2001, pp. 514–532.

[7] J. Camenisch and A. Lysyanskaya, *Dynamic accumulators and applications to efficient revocation of anonymous credentials*, Advances in Cryptology—Crypto'02, LNCS, vol. 2442, Springer-Verlag, 2002, pp. 61–76.

[8] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM Press, 1998, pp. 209–218.

[9] M. Goodrich, R. Tamassia, and J. Hasić, *An efficient dynamic and distributed cryptographic accumulator*, Information Security (ISC'02), vol. 2433, 2002, pp. 372–388.

[10] S. Haber and W. Stornetta, *How to time-stamp a digital document*, Journal of Cryptology **3** (1991), no. 2, 99–111.

[11] D. Knuth and T. Pardo, *Analysis of a simple factorization algorithm*, Theoretical Computer Science **3** (1976), no. 3, 321–348.

[12] K. Nyberg, *Commutativity in cryptography*, Proc. of the 1st Int. Workshop on Functional Analysis at Trier University, de Gruiter, W., 1996, pp. 331–342.

[13] ______, *Fast accumulated hashing*, 3rd Fast Software Encryption Workshop, LNCS, vol. 1039, Springer-Verlag, 1996, pp. 83–87.

[14] T. Sander, *Efficient accumulators without trapdoor*, ICICS'99, LNCS, vol. 1726, Springer-Verlag, 1999, pp. 252–262.

[15] A. Shamir, *On the generation of cryptographically strong pseudorandom sequences*, ACM Transactions on Computer Systems (TOCS) **1** (1983), no. 1, 38–44.