On the Impossibility of Batch Update for Cryptographic Accumulators

Philippe Camacho and Alejandro Hevia

Dept. of Computer Science, University of Chile,
Blanco Encalada 2120, 3er piso, Santiago, Chile.
{pcamacho,ahevia}@dcc.uchile.cl

Abstract. A cryptographic accumulator is a scheme where a set of elements is represented by a single short value. This value, along with another value called witness, allows to prove membership into the set. If new values are added or existent values are deleted from the accumulator, then the accumulated value changes and the witnesses need to be updated. In their survey on accumulators [6], Fazio and Nicolosi noted that Camenisch and Lysyanskaya's construction [3] was such that the time to update a witness after m changes to the accumulated value was proportional to m. They posed the question whether *batch update* was possible, namely if a cryptographic accumulator where the time to update witnesses is independent from the number of changes in the accumulated set exists. Recently, Wang et al. answered positively by giving a construction for an accumulator with *batch update* [9, 10]. In this work, we show that the construction is not secure by exhibiting an attack. Moreover, we prove it cannot be fixed. If the accumulated value has been updated m times then the time to update a witness must be at least $\Omega(m)$ in the worst case.
Keywords: cryptographic accumulators, batch update.

1 Introduction

An accumulator is a scheme that hashes elements of a set X into a single, short size value called the *accumulated value*. Then it is possible to prove membership into X of an element x using a verification algorithm that takes as input the accumulated value, the element x, and some value called *witness*. The first construction of such scheme is due to Benaloh and De Mare [2]. Several improvements later followed, including the work of Camenisch and Lysyanskaya [3] who showed how to build dynamic accumulators (where the accumulated set can evolve), and how to use them as a tool to efficiently implement anonymous credentials. In their survey on accumulators [6], Fazio and Nicolosi pointed out that, in Camenisch and Lysyanskaya's construction, the time to recompute the witnesses once the accumulated set has been modified was proportional to m, the number of changes of the accumulated set. This raised a natural question: *"Is it possible to construct dynamic accumulators in which the update of several witnesses can be performed in constant (independently of m) time?"*

Wang et al. [9, 10] answered positively this question by showing a construction that allows *batch update*. Unfortunately, we show that this construction is not secure. Moreover, we prove that there is no way to fix Wang et al.'s accumulator by giving a lower bound of $\Omega(m)$ in the time required to update witnesses after m updates.

Related Work. The first construction for cryptographic accumulators with *batch update* was given in [9] and later revised in [10]. The existence of accumulators with *batch update* seems to have been taken for granted, and in fact assumed to exist in subsequent work. Damgärd and Triandopoulos [5] cite their availability as an example of an accumulator construction based on the Paillier cryptosystem. Camenisch et al. [4] also mentioned Wang et al.'s accumulator and claim to support *batch update*. If we consider strictly Fazio and Nicolosi's definition however, this is not the case, as the witness update algorithm in [4] performs a number of operations proportional to the combined size of the set of added elements and removed elements.

We remark that our impossibility result also apply to any batch update variant of the accumulator schemes proposed in [5, 7], which allow both membership and non-membership proofs.

Organization of the paper. First, in Section 2, we briefly recall the notion of a dynamic accumulator, and Section 3 we outline Wang et al.'s construction [9, 10]. Our attack is described next in Section 4. The general impossibility result is then presented in Section 5. Finally, we conclude in section 6.

2 Dynamic Accumulators

In this section, we briefly review the notion of dynamic accumulators.

Syntax. Accumulator schemes consider two types of participants: a *manager* who initializes the parameters, and computes the accumulated value as well as the witnesses; and *users*, whose role is to verify the membership of elements in the (accumulated) set and possibly ask the *manager* to insert/delete elements in the set.

Definition 1. *[3] Let $k \in \mathbb{N}$ be the security parameter. An accumulator scheme $\mathfrak{Acc}$ consists of the following algorithms.*

- $\mathsf{KeyGen}(1^k)$*: this probabilistic algorithm takes k in unary as input and returns a pair of public and private keys (PK, SK), and the initial accumulated value for the empty set $Acc_\emptyset$.*
- $\mathsf{AccVal}(X, Acc_\emptyset, PK, [SK])$*: given a finite set of elements X (of at most polynomial size in k), a public key, and the initial accumulated value $Acc_\emptyset$, this algorithm returns the accumulated value Acc_X corresponding to the set X. Depending on the implementation, the secret key SK may also be given as optional parameter, often to improve the efficiency*[1]*.*

[1] The secret key may also be an optional parameter in the algorithms WitGen, AddEle, DelEle, and UpdWitGen.

- $\mathsf{Verify}(x, w, Acc_X, PK)$*: given an element* x*, a witness* w*, an accumulated value* Acc_X*, and a public key* PK*, this deterministic algorithm returns* Yes *if the verification is successful, meaning that* $x \in X$*, or* $\perp$ *otherwise. This algorithm is run by a* user.
- $\mathsf{WitGen}(x, Acc_X, PK, [SK])$*: this algorithm returns a witness* w *associated to the element* x *of the set* X *represented by* Acc_X*.*
- $\mathsf{AddEle}(x, Acc_X, PK, [SK])$*: this algorithm computes the new accumulated value* $Acc_{X\cup\{x\}}$ *obtained after the insertion of* x *into set* X*.*
- $\mathsf{DelEle}(x, Acc_X, PK, [SK])$*: this algorithm computes the new accumulated value* $Acc_{X\setminus\{x\}}$ *obtained by removing the element* x *from the accumulated set* X*.*
- $\mathsf{UpdWitGen}(X, X', PK, [SK])$*: suppose the set* X *is transformed into the set* X' *after several updates (insertions/deletions). The algorithm* UpdWitGen *returns the information* $Upd_{X,X'}$ *required to update all the witnesses (using the algorithm* UpdWit*) of the elements of* X *that are still in* X'*. This algorithm is run by the* manager*.*
- $\mathsf{UpdWit}(w_x, Acc_X, Acc_{X'}, Upd_{X,X'}, PK)$*: this algorithm recomputes the witness* w_x *for some element* x *that remains in the set* X'*. It takes as parameters an existent witness* w_x *with respect to set* X *(represented by the accumulated value* Acc_X*), some update information* $Upd_{X,X'}$*, and the public key* PK*. It returns a new witness* w'_x *for the element* x *with respect to the new set* X' *represented by some accumulated value* $Acc_{X'}$*. This algorithm is run by the* user*.*

The above definition is slightly more general than the one proposed by Camenisch and Lysyanskaya [3] as it does not depend on how these algorithms are implemented, and it explicitly includes the update algorithms UpdWit and UpdWitGen in the syntax.

Correctness. The correctness property of an accumulator scheme simply says that if an element x belongs to the accumulated set X and if the corresponding witness w has been computed using WitGen or UpdWitGen, UpdWit then the verification process should pass. Although the notion of correctness for accumulators with *batch update* was informally used before, it appears it has not been formalized until now. If $\mathsf{Alg}(\cdot)$ is a possibly probabilistic algorithm, and a a parameter, let $\{\mathsf{Alg}(a)\}$ be the set of all possible values output by running algorithm Alg with argument a.

Definition 2. *(Correctness) Let* X, Y *be sets,* Acc_X, Acc_Y *their respective associated accumulated values,* PK *a public key,* SK *the corresponding private key, and* $y \in X \bigcap Y$*. Let* w_y *a value (witness) that satisfies either*

- $w_y \in \{\mathsf{WitGen}(y, Acc_Y, PK, SK)\}$*, or*
- $w_y \in \{\mathsf{UpdWit}(w'_y, Acc_X, Acc_Y, Upd_{X,Y}, PK)\}$ *where* w'_y *is witness of* y *with respect to* Acc_X*, and* $Upd_{X,Y} \in \{\mathsf{UpdWitGen}(X, Y, PK, SK)\}$*.*

We say that an accumulator scheme $\mathfrak{Acc}$ *is correct if and only if for every such* y, w_y, X*, and* Y*, it holds that* $\mathsf{Verify}(y, w_y, Acc_Y, PK) = \mathsf{Yes}$*.*

SECURITY. The security of an accumulator scheme is captured by an experiment where the adversary plays the role of a *user* and attempts to forge a witness (i.e. finding a valid witness for an element that does not belong to the set) while having access to an oracle that implements the operations relative to the *manager*. Such adversary must succeed with at most negligible probability on the security parameter. The definition proposed in [9] is built on the one of [3] and includes the new algorithms AddEle,DelEle (for sets) and UpdWit that are run by the oracle (*manager*).

Definition 3. *([3, 9]) Let $\mathfrak{Acc}$ be an accumulator scheme. We consider the notion of security denoted $\mathcal{UF}$-$\mathcal{ACC}$ described by the following experiment: on input the security parameter k, the adversary $\mathcal{A}$ has access to an oracle $\mathcal{O}(\cdot)$ that replies to queries by playing the role of the accumulator* manager*. Using the oracle, the adversary can insert and delete a polynomial number of elements of his choice. The oracle $\mathcal{O}(\cdot)$ replies with the new accumulated value. The adversary can also ask for witness computations or update information. Finally, the adversary is required to output a pair (x, w). The advantage of the adversary $\mathcal{A}$ is defined by:*

$$Adv_{\mathfrak{Acc}}^{\mathcal{UF}\text{-}\mathcal{ACC}}(\mathcal{A}) = \Pr\left[\,\mathsf{Verify}(x, w, Acc_X, PK) = \mathsf{Yes} \wedge x \notin X\,\right]$$

where PK is the public key generated by KeyGen*, and Acc_X is the accumulated value of the resulting accumulated set X. The scheme $\mathfrak{Acc}$ is said to be secure if for every probabilistic polynomial time adversary $\mathcal{A}$ we have*

$$Adv_{\mathfrak{Acc}}^{\mathcal{UF}\text{-}\mathcal{ACC}}(\mathcal{A}) = \mathsf{neg}(k)$$

where $\mathsf{neg} : \mathbb{N} \to \mathbb{R}^+$ is some negligible function.

BATCH UPDATE. As originally proposed [6], the *batch update* property for an accumulator scheme states that each *user* should be able to update each of its witness using the algorithm UpdWit in time independent from the number of changes (additions and deletions) to the accumulated set.

Definition 4. *(*Batch update for accumulator schemes*). Let $k \in \mathbb{N}$ be a security parameter and let $\mathfrak{Acc}$ be an accumulator scheme. Also, let X_i be a set of accumulated values at some time i, and $U_i \subset X_i$ be a set of elements for which some user $\mathcal{U}$ holds valid witnesses. Suppose that after $m > 0$ updates (insertions or deletions) to set X_i, the new accumulated set is X_{i+m} and the associated accumulated value is $Acc_{X_{i+m}}$. We say that $\mathfrak{Acc}$ has the* batch update property *if given some information $Upd_{X_i, X_{i+m}}$, user $\mathcal{U}$ running* UpdWit *can recompute a valid witness for any element in $U_i \bigcap X_{i+m}$ in constant time (with respect to m), or equivalently, time $\mathcal{O}(k)$.*

3 Wang et al.'s Construction

In this section, we briefly recall Wang et al.'s accumulator with *batch update* [9, 10]. The first version of their work [9] suffered from two correctness problems which were later fixed [10]. Below we review the improved version [10].

SYNTAX. The algorithms of Wang et al.'s scheme slightly deviate from the general syntax of section 2 as they allow to add and delete sets of more than one element to the accumulator. Moreover the algorithms AddEle and AccVal are randomized which means that the accumulated value does not only depend on the elements of the set, contrary to the definition of Fazio and Nicolosi [6]. The general idea however remains the same. The syntax from [9] can be found in the Appendix A.1.

CONSTRUCTION. Wang et al.'s accumulator relies on the Paillier cryptosystem [8] which we recall in Appendix A.2. In the following, λ will denote the value $lcm(p-1, q-1)$ where $n = pq$ is a product of large-enough safe primes p, q, and $F : u \rightarrow \frac{u-1}{n}$ is Paillier's L function [8].

- $\mathsf{KeyGen}(1^k)$: given the security parameter k in unary, compute a safe-prime product $n = pq$ that is k-bits long and create an empty set V. Let $\mathcal{C} = \mathbb{Z}^*_{n^2} \setminus \{1\}$ and $T' = \{3, ..., n^2\}$. Let $\beta \xleftarrow{R} \mathbb{Z}^*_{\varphi(n^2)}$ and $\sigma \xleftarrow{R} \mathbb{Z}^+$ be two random numbers. The public key PK is set to (n, β) and the private key SK to (σ, λ). The output is the parameter $\mathcal{P} = (PK, SK)$.
- $\mathsf{AccVal}(X, \mathcal{P})$: given a set $X = \{c_1, ..., c_m\}$ with $X \subset \mathcal{C}$, and the parameter $\mathcal{P}$, take $c_{m+1} \xleftarrow{R} \mathcal{C}$ and compute

$$\begin{aligned} x_i &= F(c_i^{\lambda} \bmod n^2) \bmod n \ \text{ (for } i = 1, ..., m+1) \\ Acc_X &= \sigma \textstyle\sum_{i=1}^{m+1} x_i \bmod n \\ y_i &= c_i^{\lambda\sigma\beta^{-1}} \bmod n^2 \ \text{ (for } i = 1, ..., m+1) \\ a_c &= \Pi_{i=1}^{m+1} y_i \bmod n^2 \end{aligned}$$

  Output the accumulated value Acc_X and the auxiliary information a_c.
- $\mathsf{WitGen}(a_c, X, \mathcal{P})$: given the auxiliary information a_c, a set $X = \{c_1, ..., c_m\}$, and the parameter $\mathcal{P}$, choose uniformly at random a set of m numbers $T = \{t_1, ..., t_m\} \subset T' \setminus \{\beta\}$ (for $i = 1, ..., m$) and compute

$$w_i = a_c c_i^{-t_i\beta^{-1}} \bmod n^2 \ \text{ (for } i = 1, ..., m)$$

  Output the witness $W_i = (w_i, t_i)$ for c_i (for $i = 1, ..., m$).
- $\mathsf{AddEle}(X^{\oplus}, a_c, Acc_X, \mathcal{P})$: given a set $X^{\oplus} = \{c_1^{\oplus}, ..., c_l^{\oplus}\} (X^{\oplus} \subseteq \mathcal{C} \setminus X)$, to be inserted, the auxiliary information a_c, the accumulated value Acc_X, and the parameter $\mathcal{P}$, choose $c_{l+1}^{\oplus} \xleftarrow{R} \mathcal{C}$ and a set of l numbers $T^{\oplus} = \{t_1^{\oplus}, ..., t_l^{\oplus}\} \xleftarrow{R} T' \setminus (T \cup \{\beta\})$, and compute

$$\begin{aligned} x_i^{\oplus} &= F((c_i^{\oplus})^{\lambda} \bmod n^2) \bmod n \ \text{ (for } i = 1, ..., l+1) \\ Acc_{X \cup X^{\oplus}} &= Acc_X + \sigma \textstyle\sum_{i=1}^{l+1} x_i^{\oplus} \bmod n \\ y_i^{\oplus} &= (c_i^{\oplus})^{\lambda\sigma\beta^{-1}} \bmod n^2 \ \text{ (for } i = 1, ..., l+1) \\ a_u &= \Pi_{i=1}^{l+1} y_i^{\oplus} \bmod n^2 \\ w_i^{\oplus} &= a_c a_u (c_i^{\oplus})^{-t_i^{\oplus}\beta^{-1}} \bmod n^2 \ \text{ (for } i = 1, ..., l) \end{aligned}$$

  Set $a_c = a_c a_u \bmod n^2, T = T \cup T^{\oplus}$, and $V = V \cup \{a_u\}$. Then output the new accumulated value $Acc_{X \cup X^{\oplus}}$ corresponding to the set $X \cup X^{\oplus}$, the witness

$W_i^{\oplus} = (w_i^{\oplus}, t_i^{\oplus})$ for the new added elements $c_i^{\oplus}$ (for $i = 1, ..., l$), and the auxiliary information a_u and a_c.

- $\mathsf{DelEle}(X^{\ominus}, a_c, Acc_X, \mathcal{P})$: given a set $X^{\ominus} = \{c_1^{\ominus}, ..., c_l^{\ominus}\}\,(X^{\ominus} \subset X)$ to be deleted, the auxiliary information a_c, the accumulated value Acc_X, and the parameter $\mathcal{P}$, choose $c_{l+1}^{\ominus} \stackrel{R}{\leftarrow} \mathcal{C}$ and compute

$$\begin{aligned} x_i^{\ominus} &= F((c_i^{\ominus})^{\lambda} \bmod n^2) \bmod n \ \text{ (for } i = 1, ..., l+1) \\ Acc_{X \setminus X^{\ominus}} &= Acc_X - \sigma \textstyle\sum_{i=1}^{l} x_i^{\ominus} + \sigma x_{l+1}^{\ominus} \bmod n \\ y_i^{\ominus} &= (c_i^{\ominus})^{\lambda\sigma\beta^{-1}} \bmod n^2 \ \text{ (for } i = 1, ..., l+1) \\ a_u &= y_{l+1}^{\ominus} \Pi_{j=1}^{l} (y_j^{\ominus})^{-1} \bmod n^2 \end{aligned}$$

  Set $a_c = a_c a_u \bmod n^2$ and $V = V \cup \{a_u\}$. Then output the new accumulated value $Acc_{X \setminus X^{\ominus}}$ corresponding to the set $X \setminus X^{\ominus}$ and the auxiliary information a_u and a_c.
- $\mathsf{Verify}(c, W, Acc_X, PK)$: given an element c, its witness $W = (w, t)$, the accumulated value Acc_X, and the public key PK, test whether $\{c, w\} \subset \mathcal{C}, t \in T'$ and $F(w^{\beta} c^t \bmod n^2) \equiv Acc_X (\bmod n)$. If so, output Yes, otherwise output $\perp$.
- $\mathsf{UpdWit}(W_i, a_u, PK)$: given the witness W_i, the auxiliary information a_u and the public key PK, compute $w_i' = w_i a_u \bmod n^2$ then output the new witness $W_i' = (w_i', t_i)$ for the element c_i.

In the following section we show that Wang et al.'s construction is not secure.

4 An Attack on the Accumulator with Batch Update of Wang et al.

4.1 Problems with the proof

A security proof for the scheme was presented in the original paper[2] by Wang et al. [9]. In this work, a security reduction is presented assuming the *Extended Strong RSA assumption* (or *es-RSA*), also proposed in [9] as an analogous to the *Strong RSA assumption* [1] but relative to modulus n^2 instead of n. Unfortunately, there are two main problems in the proof. First, it states that adversary $\mathcal{B}$ must "run the KeyGen algorithm" which means it knows the factorization of the product $n = pq$, or at least has the knowledge of $\varphi(n^2)$ and $\lambda = lcm(p-1, q-1)$ since $\beta = \sigma\lambda \bmod n^2$ (see [9]). Therefore, it is not clear how the reduction to break the *es-RSA* assumption can be achieved.

The second problem is that, to break the *es-RSA* assumption, $\mathcal{B}$ needs to find non trivial values (y, s) such that $y^s = x \ mod \ n^2$ where x is given as input to $\mathcal{B}$. This value x does not seem to be mentioned in the proof.

[2] As mentioned before, the subsequent paper [10] fixes two correctness flaws in [9] but does not give a new security proof. The attack we consider however is relative to the improved version [10].

4.2 Description of the attack

In order to show that the construction is not secure, i.e., the proof of security cannot be fixed, we present an attack. This attack considers the updated scheme [10]. The idea is simply to delete an element from the set, and then update the witness of this element with the update information obtained by the execution of the algorithm DelEle. We then observe that this new witness is a valid one for the deleted element, which of course should not happen. We start with the set $X = \{c_1\}$ for some c_1, and let $x_1 = F(c_1^\lambda \bmod n^2) \bmod n$. Then, a random element c_* is chosen and $x_* = F(c_*^\lambda \bmod n^2) \bmod n$ is computed. The accumulated value is set to $v = \sigma(x_1 + x_*) \bmod n$. The witness value $W_1 = (w_1, t_1)$ for c_1 is defined by $w_1 = a_c c_1^{-t_1\beta^{-1}} \bmod n^2$ where $a_c = y_* y_1 \bmod n^2$, $y_1 = c_1^{\lambda\sigma\beta^{-1}} \bmod n^2$, $y_* = c_*^{\lambda\sigma\beta^{-1}} \bmod n^2$, and t_1 is random.

Then the adversary asks the manager to delete element c_1. This means that the new accumulated value is $v' = v - \sigma x_1 + \sigma(x_{**}) \bmod n = \sigma(x_* + x_{**}) \bmod n$ where $x_{**} = F(c_{**}^\lambda \bmod n^2) \bmod n$ and c_{**} is random. The auxiliary value a_u used to update the witnesses is $a_u = y_{**} y_1^{-1} \bmod n^2$ where $y_{**} = c_{**}^{\lambda\sigma\beta^{-1}} \bmod n^2$. So, by updating the witness w_1 with a_u we obtain $w_1' = a_u w_1 \bmod n^2 = y_{**} y_1^{-1} y_* y_1 c_1^{-t_1\beta^{-1}} \bmod n^2 = y_{**} y_* c_1^{-t_1\beta^{-1}} \bmod n^2$. Then $w_1'^\beta c_1^{t_1} \equiv (y_{**} y_* c_1^{-t_1\beta^{-1}})^\beta c_1^{t_1} \bmod n^2 = (y_{**} y_*)^\beta \bmod n^2 = (c_{**} c_*)^{\lambda\sigma} \bmod n^2$. It follows that

$$\begin{aligned} F(w_1'^\beta c_1^{t_1} \bmod n^2) &\equiv F((c_{**} c_*)^{\lambda\sigma} \bmod n^2) \bmod n \\ &\equiv \sigma(F(c_{**}^\lambda \bmod n^2) + F(c_*^\lambda \bmod n^2)) \bmod n \\ &\equiv \sigma(x_* + x_{**}) \bmod n \\ &\equiv v' \bmod n \end{aligned}$$

This shows that (w_1', t_1) is a valid witness for the deleted element x_1. Therefore the scheme is not secure. Indeed the problem is simply that the information a_u allows to update *every* old witness including w_1 for which such an update should not be possible.

5 A Lower Bound for Updating the Witnesses

The attack of the last section is an indication that the proposed construction may have some design flaws. In this section, we show that the problem indeed is more fundamental and the *batch update* property is essentially unrealizable. We argue this by presenting a lower bound on the size of $Upd_{X,X'}$, the information needed to update the witnesses after m changes (more precisely deletions). Any deterministic[3] update algorithm UpdWit must at least read $Upd_{X,X'}$, and so it also bounds the running time of any such algorithm. In the following theorem log refers to the logarithm in base two function.

[3] For simplicity, in the proof we focus on deterministic update (UpdWit) and verification (Verify) algorithms. This is in fact the case for the construction of [10]. We believe the result can be extended to the probabilistic case but the proof becomes more involved.

Theorem 1. *Let* $\mathfrak{Acc}$ *be a secure accumulator scheme with deterministic* UpdWit *and* Verify *algorithms. For an update involving* m *delete operations in a set of* N *elements, the size of the information* $Upd_{X,X'}$ *required by the algorithm* UpdWit *is* $\Omega(m \log \frac{N}{m})$*. In particular if* $m = \frac{N}{2}$ *we have* $|Upd_{X,X'}| = \Omega(m) = \Omega(N)$*.*

Proof. The idea of the proof is that the update information must encode a minimum amount of information in order for the accumulator scheme to be correct and secure. We prove this by considering a theoretical game between the accumulator manager and some user U. In the game, starting from an accumulated set X, the accumulator manager updates the accumulator in a way that is not known to user U (namely, the manager deletes some elements in an arbitrary set $X_d \subset X$) while still providing the update information $Upd_{X,X'}$ to U, where $X' = X \setminus X_d$. We prove that, as long as the scheme is correct and secure, there is a simple strategy that allows the user U to recover the exact changes made by the manager, that is, the set of deleted elements X_d. We conclude that the information provided by the manager to the user must at least encode a description of set X_d. Details follow.

Consider the following game. The set accumulated in some point of the time is $X = \{x_1, x_2, ..., x_N\}$, and the corresponding accumulated value is Acc_X. We suppose the user possesses all the witnesses for each element in X and knows the accumulated value. Then m DelEle operations are performed, that is the new set obtained is $X' = X \setminus X_d$ where $X_d = \{x_{i_1}, x_{i_2}, ..., x_{i_m}\}$. The manager computes the new accumulated value $Acc_{X'}$ and sends it to the user along with the update information $Upd_{X,X'}$ required to update all the witnesses that are still in X'.

Armed with this update information $Upd_{X,X'}$ and the new accumulated value $Acc_{X'}$, user U is able to reconstruct the set X_d of deleted elements as follows: for each element in X, the user checks if the corresponding witness can be successfully updated using algorithm UpdWit with input $Upd_{X,X'}$. More specifically, the user computes $w'_x = \mathsf{UpdWit}(w_x, Upd_{X,X'}, PK)$ and checks whether or not w'_x is a valid witness. If not, that is, if $\mathsf{Verify}(x, w'_x, Acc_{X'}, PK) = \perp$, then the element x must have been deleted from X. Note that this condition is necessary otherwise it would contradict the scheme's correctness (there would be elements in X' for which an updated witness cannot be computed) or the scheme's security (it would be possible to update witnesses for deleted elements).

Hence, the user must be able to recompute the set of deleted elements X_d only from values $Upd_{X,X'}$ and $Acc_{X'}$. We therefore conclude that $(Upd_{X,X'}, Acc_{X'})$ must contain at least the information required to encode any subset with m elements of a set containing N elements. There are $\binom{N}{m}$ such subsets, so the minimum amount of information required is $\log \binom{N}{m}$ bits. Since $\log(\binom{N}{m}) \geq m \log \frac{N}{m}$ we obtain that $|(Acc_{X'}, Upd_{X,X'})| = \Omega(m \log \frac{N}{m})$. Given that $|Acc_{X'}|$ must be sufficiently short (say, at least $\mathcal{O}(\log(N))$, as otherwise the accumulated value is not longer a "short representation" of the set, and the scheme is not really useful), we conclude that $|Upd_{X,X'}| = \Omega(m \log \frac{N}{m})$. The result then follows.

Given a the security parameter k, the above theorem says that any update algorithm must take time at least $\mathcal{O}(m) = \mathcal{O}(N) = \mathcal{O}(p(k)) = \omega(k)$ for some

polynomial p, in order to *read* the input. This goes beyond $\mathcal{O}(k)$, the desired "constant time" in the number of changes. Interestingly, even if the number of *expensive operations* (say, $\mathcal{O}(k^3)$ modular exponentiations) of UpdWit were sublinear in the size of the update information $Upd_{X,X'}$, the overall complexity of the update operation would still be dominated by the time $\mathcal{O}(p(k))$ to read the update information (and the accumulated value).

Corollary 1. *Cryptographic accumulators with* batch update *(and deterministic update and verification) do not exist.*

6 Conclusion

This result shows that the *batch update* property as proposed in [6] essentially cannot be obtained, as the time to update all the witnesses cannot be linear in the security parameter k, i.e $\mathcal{O}(k)$, but it must be at least $\mathcal{O}(p(k)) = \omega(k)$ for some polynomial p. Notice that our lower bound is not tight since Camenisch and Lysyanskaya's accumulator requires $\mathcal{O}(p(k) \cdot k)$ time to update the witnesses after $\mathcal{O}(p(k))$ changes. Nonetheless, in principle, it leaves some (potential) room to improve their construction by at most a factor of k.

Finally, one may consider getting around this impossibility result by not allowing deletions in the set. Unfortunately, such an accumulator can be trivially implemented by signing the elements of the set, as in this case there is no replay-attack. The witness for every element consists in its signature under the manager's private key, and clearly needs not to be updated.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments and suggestions. Alejandro Hevia gratefully acknowledges the support of CONICYT via FONDECYT No. 1070332.

References

1. Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Proceedings of EUROCRYPT*, volume 1233 of *LNCS*, pages 480–494, 1997.
2. Josh Benaloh and Michael de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *Proceedings of EUROCRYPT*, volume 1440 of *LNCS*, pages 274–285, 1994.
3. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proceedings of CRYPTO*, volume 2442 of *LNCS*, pages 61–76, 2002.
4. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Proceedings of Public Key Cryptography - PKC*, volume 5443 of *LNCS*, pages 481–500, 2009.

5. I. Damgård and N. Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008.
6. Nelly Fazio and Antonio Nicolosi. Cryptographic accumulators: Definitions, constructions and applications. Technical report, 2002.
7. Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *Proceedings of Applied Cryptography and Network Security - ACNS*, volume 4521 of *LNCS*, pages 253–269. 2007.
8. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of EUROCRYPT*, volume 1592 of *LNCS*, pages 223–238. 1999.
9. Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. A new dynamic accumulator for batch updates. In *Proceedings of Information and Communications Security*, volume 4861, pages 98–112, 2007.
10. Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. Improvement of a dynamic accumulator at ICICS 07 and its application in multi-user keyword-based retrieval on encrypted data. In *Proceedings of IEEE Asia-Pacific Services Computing Conference - APSCC*, pages 1381–1386, 2008.

A Appendix

A.1 Syntax of Wang et al.'s Accumulator

Definition 5. *([9]) Let k be the security parameter. A dynamic accumulator with* batch update $\mathfrak{Acc}_{\mathfrak{BU}}$ *consists of the following algorithms:*

- $\mathsf{KeyGen}(1^k)$*: is a probabilistic algorithm that takes as input the security parameter k in unary and returns a parameter $\mathcal{P} = (PK, SK)$ where PK is the public key and SK is the private key*[4].
- $\mathsf{AccVal}(X, \mathcal{P})$*: is a probabilistic algorithm that computes an accumulated value. It takes as input the set $X = \{c_1, ..., c_m\}$ and the parameter $\mathcal{P}$ and returns an accumulated value Acc_X along with some auxiliary information a_c that will be used by other algorithms.*
- $\mathsf{Verify}(x, W, Acc_X, PK)$*: this deterministic algorithm checks whether an element x belongs to the set X represented by the accumulated value Acc_X using the witness W and the public key PK. It returns* Yes *whether the witness W for x is valid or $\perp$ otherwise.*
- $\mathsf{AddEle}(X^{\oplus}, a_c, Acc_X, \mathcal{P})$*: this probabilistic algorithm adds some new elements to the accumulated value. The input values are the set of new elements to add $X^{\oplus} = \{c_1^{\oplus}, ..., c_l^{\oplus}\}$, the auxiliary information a_c, the accumulated value Acc_X and the parameter $\mathcal{P}$. The returned values are $Acc_{X \cup X^{\oplus}}$ the accumulated value corresponding to the set $X \cup X^{\oplus}$, a witnesses $\{W_1^{\oplus}, ..., W_l^{\oplus}\}$ associated to the inserted elements $\{c_1^{\oplus}, ..., c_l^{\oplus}\}$, and the auxiliary information a_c, a_u, that will be used for future update operations.*

[4] In the original paper, the authors mention another parameter M which is an upper bound to the number of elements that can be accumulated. In order to simplify the notations, we omit it and recall that this upper bound must be a polynomial in k.

– $\mathsf{DelEle}(X^{\ominus}, a_c, Acc_X, \mathcal{P})$*: this probabilistic algorithm is analogous to* AddEle *and allows to delete a set of elements* $X^{\ominus}$*. The input values are the set of elements to delete* $X^{\ominus} = \{c_1^{\ominus}, ..., c_l^{\ominus}\}$*, the auxiliary information* a_c*, the accumulated value* Acc_X *and the parameter* $\mathcal{P}$*. The returned values are* $Acc_{X \setminus X^{\ominus}}$ *the accumulated value corresponding to the set* $X \setminus X^{\ominus}$*, and the auxiliary information* a_c*,* a_u*, that will be used for future update operations.*
– $\mathsf{WitGen}(a_c, X, \mathcal{P})$*: this probabilistic algorithm creates a witness for every element in the set* X*. It takes as input an auxiliary information* a_c*, the set* X *and the parameter* $\mathcal{P}$*.*
– $\mathsf{UpdWit}(W_i, a_u, PK)$ *: this deterministic algorithm updates witnesses for the elements accumulated* Acc_X *and that are still accumulated in* $Acc_{X'}$ *(the new set after update). The inputs are* W_i*, the witness to update, the auxiliary information* a_u *and the public key* PK*. It returns an updated witness* W_i' *that allows to prove that* c_i *is still accumulated in the new accumulated value* $Acc_{X'}$*.*

Note that in this definition $\mathsf{UpdWitGen}$ does not appear. The reason is that in Wang et al.'s construction, the update information required to recompute the witnesses is generated by algorithms AddEle and DelEle.

A.2 Paillier Cryptosystem

The Paillier cryptosystem [8] consists of the following three algorithms.

– KeyGen: let $n = pq$ be a RSA modulus, with p, q large prime integers. Let g an integer multiple of n modulo n^2. The public key is defined by $PK = (n, g)$ and the private key by $SK = \lambda = \lambda(n) = lcm(p-1, q-1)$.
– $\mathsf{Encrypt}$: let $M \in \mathbb{Z}_n$ be a plaintext message and r a random element in $\mathbb{Z}_n^*$, the encrypted message is $c = g^M r^n \bmod n^2$.
– $\mathsf{Decrypt}$: to recover M from ciphertext c, compute $M = \frac{F(c^\lambda \bmod n^2)}{F(g^\lambda \bmod n^2)} \bmod n$ where $F : u \to \frac{u-1}{n}$ takes as argument elements from the set $\mathcal{S}_n = \{u < n^2 | u = 1 \bmod n\}$.

The homomorphic property of the Paillier cryptosystem follows from the fact that $\forall x, y \in \mathcal{S}_n$ and $\sigma \in \mathbb{Z}^+$:

$$F((x.y)^\lambda \bmod n^2) \bmod n = F(x^\lambda \bmod n^2) + F(y^\lambda \bmod n^2) \bmod n$$
$$F(x^{\sigma\lambda} \bmod n^2) \bmod n = \sigma F(x^\lambda \bmod n^2) \bmod n$$