On the Impossibility of Batch Update for Cryptographic Accumulators

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

December 10, 2009

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 in a secure way (non)membership into the set. In their survey on accumulators [FN02], Fazzio and Nicolisi noted that in the Camenisch and Lysyanskaya's construction[CL02], the time to update a witness after m changes to the accumulated value is proportional to m.

A natural problem arises: *Is it possible to build an accumulator where the update of all the witnesses could be made through a constant size (relative to m) piece of information?* Such accumulators would have the *batch update* property.

An accumulator for batch update was proposed by Wang et. al in [WWP07] and its improvement [WWP08]. In this work we show that the construction is not secure and indeed cannot be fixed as we obtain the following lower bound: 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.

1 Introduction

An accumulator is a scheme that hashes elements of a set X into a single, constant size value called the *accumulated value*. Then it is possible to prove (non)membership into X of an element x using a verification algorithm with the accumulated value, x and another value called *witness*. The first construction of such scheme is due to Benaloh and De Mare [BdM94]. Several improvements have followed especially with the work of Camenisch and Lysanskaya [CL02] who showed that accumulators can be made dynamic (the set can evolve), and that they can be used as a tool to solve efficiently complex security protocols related to anonymous credentials. In their survey on accumulators [FN02], Fazio and Nicolisi pointed that in the Camenisch and Lysyanskaya's construction, the time to recompute the witnesses was proportional to m, the number of changes of the accumulated value, and then raised the natural question: *Is it possible to build an accumulator where the update of all the witnesses could be made through a constant size (relative to m) piece of information?*

In this work we answer by the negative: the time required to update the witness after m updates of the set is $\Omega(m)$. More precisely, the information required to update the witnesses is at least $\Omega(m)$ in the worst case.

Related Work. The problem of batch update has been addressed in a paper of Wang et al. [WWP07]. In this work the authors proposed a construction that allows batch update at unit cost. However we show that this construction and its improvement [WWP08] are not secure. Indeed we prove in this work that there is no way to fix Wang's et al. accumulator, as the lower bound to update witnesses after m updates[1] on the accumulated value is $\Omega(m)$.

2 Dynamic Accumulators

In this section we recall the main definitions related to dynamic accumulators. We do not review in details the concrete construction described in [CL02], but only the abstract operations of accumulators, and their security.

In accumulator schemes there are two kinds of participants: the *manager* who initializes the parameters, computes the accumulated value, and the witnesses, and the *user* which role is to verify the membership of elements in the set and possibly ask for elements insertion/deletion.

Definition 1. *(Adapted from [FN02]) Let k be the security parameter. An accumulator scheme $\mathfrak{Acc}$ consists in the following algorithms:*

- $\mathsf{Setup}(1^k)$ *is a probabilistic algorithm that takes 1^k as argument and returns a pair of private/public key (PK, SK) and the initial accumulated value for the empty set $Acc_\emptyset$.*
- $\mathsf{Eval}(X, Acc_\emptyset, PK, [SK])$*: given a set of elements X, the public key or (the private key), and the initial accumulated value $Acc_\emptyset$, this algorithm returns the accumulated value Acc_X, for the set X. Note that in some settings, it could be necessary to know the private key SK in order to get better performance.*
- $\mathsf{Check}(x, w, Acc, PK)$*: given an element x, a witness w, the accumulated value Acc and PK the public key, this deterministic algorithm returns OK if the verification is successful, meaning that $x \in X$ or $\perp$ otherwise.*
- $\mathsf{Witness}(x, Acc, PK, [SK])$*: this algorithm returns a witness w associated to the element x of the set represented by Acc. Note that in some settings, it could be necessary to know the private key SK in order to get better performance.*
- $\mathsf{Insert}(x, Acc_X, PK, [SK])$*: this algorithm computes the new accumulated $Acc_{X\cup\{x\}}$ value obtained after the insertion of x into X. Note that in some settings, it could be necessary to know the private key SK in order to get better performance.*
- $\mathsf{Delete}(x, Acc_X, PK, [SK])$*: the analogue algorithm for* Insert *that is used to delete elements from X. Note that in some settings, it could be necessary to know the private key SK in order to get better performance.*

[1] Delete operations

- $\mathsf{UpdWit}(x, w, Acc_X, Upd_{X,X'}, PK)$: *this algorithm is used to update the witnesses for each element x that are still in the set. It takes as parameters, the element for which the new witness must be computed, the old witness w that corresponds the set X represented by the accumulated value Acc_X, the update information, that allows to compute the new witness for the current set X' and the public key PK. This algorithm is run by the user.*

This definition is not exactly similar to the one of [CL02]: it is a bit more abstract as it does not depend on how these algorithms are implemented and we added explicitly the algorithm UpdWit in the syntax definition. However the meaning for each algorithm is the same as in Camenisch and Lysyanskaya's work. Note also that the fact that the algorithms Check, Witness, Eval, Insert, Delete, UpdWit are deterministic or probabilitic has no consequence on our result.

The correctness property of an accumulator scheme is quite straightforward: it states that if an element x belongs to the accumulated set X and if its witness w has been computed using Witness then the verification process should pass.

Definition 2. *(Correctness) Let X be a set, Acc_X its associated accumulated value and $x \in X$. An accumulator scheme is correct if:*

$$\mathsf{Check}(x, w, Acc_X, PK) = OK$$

where Acc_X is the accumulated value of the set X, $x \in X$, PK is the public key and w is the witness obtained by running Witness *on Acc_X, x and PK/SK.*

The following definition describes the security property for dynamic accumulators. The idea behind this definition is to consider the following experiment where the adversary takes the place of a user who tries to forge a witness (i.e. finds a witness for an element that does not belong to the set) and where an oracle implements the operations of the accumulator *manager*.

Definition 3. *(Security of a dynamic accumulator, adapted from[CL02]) Let $\mathfrak{Acc}$ be an accumulator scheme. We consider the notion of security denoted $\mathcal{UF} - \mathcal{ACC}$ described by the following experiment: there is an oracle $\mathcal{O}$ that takes the place of the accumulator manager. The adversary can insert and delete a polynomial number of elements[2] of its own election. The oracle replies with the new accumulated value. The adversary can also asks for witness computations or updates.*

Finally the adversary outputs a pair (x, w). The advantage of the adversary $\mathcal{A}$ is defined by:

$$\begin{aligned} Adv_{\mathfrak{Acc}}^{\mathcal{UF}-\mathcal{ACC}}(\mathcal{A}) &= \Pr\left[\,\mathsf{Check}(x, w, Acc, PK) = 1 \wedge x \notin X\,\right] \\ &= \Pr\left[\,Exp_{\mathfrak{Acc}}^{\mathcal{UF}-\mathcal{ACC}}(\mathcal{A}) = 1\,\right] \end{aligned}$$

where PK is the public key generated by Setup.

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

[2]With respect to the security parameter, that is $n = p(k)$ for some polynomial p.

3 An attack on the Accumulator For Batch Update of Wang et. al [WWP07, WWP08]

3.1 Why the proof does not work

There are two main problems in the security proof[3].

First the adversary $\mathcal{B}$ as to run the algorithm **KeyGen** requires the factorization or at least the knowledge of $\Phi(n^2)$ and $\lambda = lcm(p-1, q-1)$ where n is a safe integer of the form $n = pq$ with p, q primes because $\beta = \sigma\lambda mod(n^2)$. Without β computed in such a way, the correctness of the construction cannot hold anymore, and σ is also required. The first flaw is that the knowlege of the factorization of n or $\Phi(n^2)$ implies that the *es-RSA assumption* cannot be broken because this assumption requires the factorization of n to be kept secret.

The proof contains another inconsistency: $\mathcal{B}$ needs to find a non trivial pair (y, s) such that $y^s = x \ mod \ n^2$ where x is given to $\mathcal{B}$. This value x is not mentionned.

3.2 Description of the attack

As 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 changes made in [WWP08].

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 can then check easily that this new witness is a valid one for the deleted element, which of course should not happen.

So we start with the set $X = \{c_1\}$. We have $x_1 = F(c_1^{\lambda} \ mod \ n^2) \ mod \ n$. Then a random element c_* is choosen and $x_* = F(c_*^{\lambda} \ mod \ n^2) \ mod \ n$ is computed. The accumulated value is set to $v = \sigma(x_1 + x_*) \ mod \ n$. The witness value $W_1 = (w_1, t_1)$ for c_1 is defined by $w_1 = a_c c_1^{-t_1\beta^{-1}} \ mod \ n^2$ where $a_c = y_* y_1 \ mod \ n^2$, $y_1 = c_1^{\lambda\sigma\beta^{-1}} \ mod \ n^2$, $y_* = c_*^{\lambda\sigma\beta^{-1}} \ mod \ n^2$, and t_1 is random.

Then the adversary asks to delete the element c_1. This means that the new accumulated value is $v' = v - \sigma x_1 + \sigma(x_{**}) \ mod \ n = \sigma(x_* + x_{**}) \ mod \ n$ where $x_{**} = F(c_{**}^{\lambda} \ mod \ n^2) \ mod \ n$ and c_{**} is random. The value a_u that allows to update the witnesses is $a_u = y_{**} y_1^{-1} \ mod \ n^2$ where $y_{**} = c_{**}^{\lambda\sigma\beta^{-1}} \ mod \ n^2$.

So updating the witness w_1 with a_u we obtain $w_1' = a_u w_1 \ mod \ n^2 = y_{**} y_1^{-1} y_* y_1 c_1^{-t_1\beta^{-1}} \ mod \ n^2 = y_{**} y_* c_1^{-t_1\beta^{-1}} \ mod \ n^2$. Then $w_1'^{\beta} c_1^{t_1} \equiv (y_{**} y_* c_1^{-t_1\beta^{-1}})^{\beta} c_1^{t_1} \ mod \ n^2 = (y_{**} y_*)^{\beta} \ mod \ n^2 = (c_{**} c_*)^{\lambda\sigma}$.

It follows that

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

[3]The paper [WWP08] that fixes the correctness flaw does not mention another security proof, so we are considering the only proof avaible of the first work: [WWP07]. However the attack we consider works for the improved version [WWP08].

This shows that (w'_1, t_1) is a valid witness for the deleted element x_1. So the scheme is not secure. Indeed the problem is simply that the information a_u allows to update **all** the old witnesses including w_1 for which such an update must not be possible.

4 A lower bound for updating the witnesses

The attack of the last section gives us the idea to obtain a lower bound for the size of the information needed to update the witnesses after m changes, that are chosen to be deletions.

Theorem 1. *For an update involving m delete operations in a set of n elements, the size of the information* Upd *required by the algorithm* UpdWit *while keeping the dynamic accumulator secure is $\Omega(m \log \frac{n}{m})$. In particular if $m = \frac{n}{2}$ with n even, we have $|\mathsf{UpdWit}| = \Omega(m)$.*

Proof. We consider the following scenario. 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 Delete operations are performed, that is the new set obtained is $X' = X - X_d$ 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 required to update all the witnesses.

With this information Upd, the user is able to reconstruct the set X_d of deleted elements by checking for each element of x, if its corresponding witness can be successfully updated using Upd and the algorithm UpdWitness. That is if the result of $w' = \mathsf{UpdWitness}(w, Upd)$ is not a valid witness i.e. $\mathsf{Check}(x, w', X) = \perp$, then the element has been deleted. Note that this condition is necesarry as in the contrary the scheme would be incorrect (some existing elements in X' could not have their witness updated) or insecure (it could be possible to compute witnesses for deleted elements).

Hence, as the user is able to recompute the set of deleted elements X_d only using the value Upd, Upd must contain at least the information required to code a subset with m elements of a set with n elements. There are $\binom{n}{m}$ such subsets, so the minimum quantity of information required is $\log \binom{n}{m}$ bits. A standard lower bound for the binomial coefficient is $\binom{n}{m} \geq (\frac{n}{m})^m$. Then we have $\log(\binom{n}{m}) \geq m \log \frac{n}{m}$.

□

5 Conclusion

This result shows that the batch update property as proposed in [FN02] cannot be obtained, as the time to update all the witnesses cannot be only linear in the security parameter k, i.e $O(k)$, but is at least $O(m) = O(n) = O(p(k)) = \omega(k)$ where p is a polynomial. Our lower bound is not tight as Camenisch and Lysyanskaya's accumulator requires $O(p(k) \cdot k)$ time to update the witnesses after possible $O(p(k))$ changes, however, this bound can only be improved in a factor of k as our bound is $O(n) = O(p(k))$.

Finally considering an accumulator that would only allow addition of elements, can be implemented trivially by signing the elements of the set, as in this case there is no replay-attack

and the witness for every element consists in its signature under the manager's private key, and need not to be updated.

References

[BdM94] Josh Benaloh and Michael de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *Proceedings of EUROCRYPT*, volume 1440, pages 274–285, 1994.

[CL02] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proceedings of CRYPTO*, volume 2442, pages 61–76, 2002.

[FN02] Nelly Fazio and Antonio Nicolisi. Cryptographic accumulators: Definitions, constructions and applications. Technical report, 2002.

[WWP07] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. A new dynamic accumulator for batch updates. In *Information and Communications Security*, volume 4861, pages 98–112, 2007.

[WWP08] 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. *Asia-Pacific Conference on Services Computing. 2006 IEEE*, 0:1381–1386, 2008.