A New Dynamic Accumulator for Batch Updates

Peishun Wang[1], Huaxiong Wang[1,2], and Josef Pieprzyk[1]

[1] Centre for Advanced Computing – Algorithms and Cryptography
Department of Computing, Macquarie University, Australia
{pwang,hwang,josef}@ics.mq.edu.au
[2] Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
hxwang@ntu.edu.sg

Abstract. A dynamic accumulator is an algorithm, which gathers together a large set of elements into a constant-size value such that for a given element accumulated, there is a witness confirming that the element was indeed included into the value, with a property that accumulated elements can be dynamically added and deleted into/from the original set such that the cost of an addition or deletion operation is independent of the number of accumulated elements. Although the first accumulator was presented ten years ago, there is still no standard formal definition of accumulators. In this paper, we generalize formal definitions for accumulators, formulate a security game for dynamic accumulators so-called Chosen Element Attack (CEA), and propose a new dynamic accumulator for batch updates based on the Paillier cryptosystem. Our construction makes a batch of update operations at unit cost. We prove its security under the extended strong RSA (es-RSA) assumption.

Keywords: Dynamic accumulator, Paillier cryptosystem.

1 Introduction

An accumulator is an algorithm that merges a large set of elements into a constant-size value such that for a given element there is a witness confirming that the element was indeed accumulated into the value. It was originated by Benaloh and de Mare [3] as a decentralized alternative for digital signatures and was used in the design of secure distributed protocols. Baric and Pfitzmann [2] refined the concept of accumulators asking from them to be collision-free. The collision freeness requires that it is computationally hard to compute a witness for an element that is not accumulated. However, in many practical applications, the set of elements changes with the time. A naive way of handling such situations would be to re-run the accumulator. Obviously, this is highly impractical, especially when the element set is very large. To solve this problem, Camenisch and Lysyanskaya [4] developed more practical schemes – accumulators with dynamic addition and deletion of elements to or from the original set of accumulated elements. The cost of adding or deleting elements and updating individual witnesses is independent from the number of elements accumulated.

Accumulators are useful in a number of privacy-enhancing applications, such as time-stamping [3], fail-stop signatures [2], identity escrow and group signature schemes with membership revocation [4], authenticated dictionary [9], ad-hoc anonymous identification and ring signatures [6], and broadcast encryption [8]. However, they are still a relatively new tool in Cryptography, and there are only a handful of papers on the topic and they do not constitute a systematic and organized effort. In particular, there is still no standard formal definition for them, and the definitions used so far are driven by specific applications rather than a systematic study of underlying properties of the accumulator.

In the existing dynamic accumulators, usually the witnesses must be updated immediately whenever old elements leave or new elements join. In some applications, the updates of the witnesses for some elements cannot be done immediately after the changes. When the witnesses of elements need to be updated after a batch of N addition and deletion operations occurred, they have to be computed N times – one by one in the time sequence, that means, the time to bring a witness up-to-date after a batch of N operations is proportional to N. Clearly, these schemes are inefficient for batch update. In these circumstances, it is reasonable to cluster the updates into one single operation. In this paper we address an open question formulated by Fazio and Nicolosi in [7]. The question asks about how to design an efficient dynamic accumulator whose witnesses can be updated in one go independently from the number of changes. We answer this question by proposing a new dynamic accumulator that allows batch updates.

Related Work. In general, there exist two different types of accumulators, namely RSA-based [2,3,4,9,14,15] and combinatorial hash-based accumulators [11].

The basic RSA accumulator [3] is constructed as follows. Given a set of elements $X = \{x_1, \ldots, x_m\}$ that can be accumulated, the accumulator function is $y_i = f(y_{i-1}, x_i)$, where f is a one-way function defined as $f(u, x) = u^x \bmod n$ for suitably-chosen values of the seed u and RSA modulus n. The accumulated value is $v = u^{x_1 \cdots x_m} \bmod n$ and the witness for the element x_i is $w_i = u^{x_1 \cdots x_{i-1} x_{i+1} \cdots x_m} \bmod n$. This basic RSA accumulator has many different variants depending on the intended application. Baric and Pfitzmann [2] used the accumulator for elements that must be primes and they proved that it is collision-resistant provided factoring is intractable. Camenisch and Lysyanskaya studied in [4] dynamic RSA accumulators for which the domain of accumulated elements consists of primes in a particular range. The seed u is a random quadratic residue and the modulus is a safe number. Tsudik and Xu [15] relaxed the constraint and allowed the accumulated elements to be composite numbers that are products of two primes chosen from a specific interval. Goodrich *et al.* in [9] constructed dynamic RSA accumulators in which the seed u needs to be coprime to the modulus n only. This constraint is very easy to satisfy.

The collision resistance of RSA accumulators relies on the secrecy of factorization of the modulus n. Normally, the designers of accumulators are going to know the factors of n and therefore able to forge membership proofs and break the collision resistance. A solution to this problem has been provided by Sander [14] who constructed accumulators whose designers do not know the factors of the modulus. Security of all existing RSA accumulators is based on the strong RSA assumption.

In order to remove need for trapdoor information, Nyberg [11] came up with a combinatorial accumulator based on hashing. In the scheme, a one-way hash function is used to map bit strings of arbitrary length to bit strings of fixed length, and then each resulting output string is divided into small blocks of a fixed length. Next every block is replaced by 0 if it consists of all 0 bits, otherwise by 1. In such way, each accumulated element is mapped to a bit string of a fixed short length. Finally, the accumulated value is computed as a bitwise product of the bit strings of all elements. To verify whether an element has been accumulated, one first hashes the element, partitions the hash output into blocks and then converts the blocks into bits (as discussed above). The element is accumulated with a high probability if the 0 bits of the element coincide with the 0 bits of the accumulated value. The accumulator does not need witnesses and is provably secure in the random oracle model. However, it is not dynamic, and not space and time efficient.

Nguyen [10] constructed a dynamic accumulator from bilinear pairings, which, however, gives away the secret of the accumulator on a particular input. It has been proved that the scheme is not secure [16]. To make Nguyen scheme secure, Au *et al.* [1] constrained the number of accumulated elements and introduced a notion of bounded accumulator.

Our Contributions. Our technical contributions can be divided into the following:

Formal Definitions. We generalize formal definitions for dynamic accumulators, provide a definition of their security. Despite the fact that first accumulators were introduced more than 10 years ago, there is no standard definition of them. The definitions used so far are all application specific. In this paper we are going to rectify this by proposing generic definitions.

Security Game. We formulate a security game for dynamic accumulators that is based on the so-called Chosen Element Attack (CEA). The security game provides an environment for interaction between an accumulator algorithm and an adversary. The adversary can access the accumulator via an oracle, *i.e.* he provides inputs to the oracle and is able to collect outputs generated by the oracle. The game consists of two stages. In the first stage, the adversary chooses adaptively a set of elements L and then queries the accumulator oracle. Each time the oracle is queried, it provides the accumulated value and the witnesses to the adversary. Clearly, the next query depends on the reply obtained from the oracle. In the second stage, the adversary adaptively adds/deletes some elements and queries the oracle for the corresponding accumulated values and witnesses. In both stages, the adversary is allowed to issue a polynomial number of queries. The adversary wins the game if the adversary can forge another legitimate element and its witness such that the witness proves that the forged element is also included in the accumulated value corresponding to the set.

New Dynamic Accumulator. We construct a new dynamic accumulator from the Paillier cryptosystem, and prove its security under a new complexity assumption – extended strong RSA (es-RSA) assumption. Existing accumulators apply elements that are either primes or products of primes. We remove the restrictions on the selection of elements and allow them to be chosen from $Z^*_{n^2}$. Our scheme permits also for an efficient batch update of witnesses independently from the number of changes.

Affirmative Answer to an Open Problem. Existing accumulators have to update witnesses after each single addition/deletion operation. Fazio and Nicolosi posed the following question [7]: is it possible to construct dynamic accumulators in which the time to update a witness can be made independent from the number of changes that need to be done to the accumulated value? We answer this question in the affirmative, and have constructed a scheme in which, the time necessary to bring a witness up-to-date for a batch with an arbitrary number of additions and deletions is done at the unit cost.

Organization. Section 2 introduces notation and provides a cryptographic background necessary to understand our presentation. In Section 3 we give a few related definitions for accumulators. In Section 4 we construct a new dynamic accumulator. Section 5 describes a batch update. Section 6 shows the correctness of the construction. In Section 7 the security of the proposed scheme is proved. Finally Section 8 concludes our paper and discusses a possible future research.

2 Preliminaries

2.1 Notation

Throughout this paper, we use the following notation.

PPT denotes *probabilistic polynomial time.* For any positive integer m, $[m]$ denotes the set of integers $\{1, \ldots, m\}$. Let $a \xleftarrow{R} A$ denote that an element a is chosen uniformly at random from the set A. Let M be the upper bound on the number of elements that can be securely accumulated and $\mathcal{C}$ be an efficiently-samplable domain where all accumulated elements are coming from.

2.2 Complexity Assumption

Let us state without proof some basic number-theoretic facts, and make a complexity assumption that will be used in our work.

Fact 1: For a safe number $n = pq$, *i.e.* $p, q, \frac{p-1}{2}, \frac{q-1}{2}$ are prime, Euler's Totient function $\phi(n) = (p-1)(q-1)$, $\phi(n^2) = n\phi(n)$, and Carmichael's function $\lambda(n) = lcm(p-1, q-1)$, $\lambda(n^2) = lcm((p-1)p, (q-1)q)$.

Catalano *et al* in [5] introduced a variant of the RSA problem in $Z^*_{n^2}$ – Computational Small s-Roots (CSR) problem.

Definition 1 (CSR Problem). *Given a safe number n, an integer $s \in Z^*_{n^2} \setminus \{2\}$ and a random number $x \in Z^*_{n^2}$, the s-th roots problem is the task of finding $y \in Z^*_{n^2}$ such that $x = y^s \bmod n^2$.*

Catalano *et al* [5] gave evidence to that the CSR problem is intractable, even for every $s > 2$ such that $gcd(s, \lambda(n^2)) = 1$, when the factorization of n is unknown. However, they recommended $s = 2^{16} + 1$ for practical applications.

Fact 2: There exists a PPT algorithm that, given integers x, s, n, and the factorization of n, outputs x's s-th roots mod n^2.

We describe a variant of the Discrete Logarithm (DL) problem in $Z^*_{n^2}$, which we call the extended Discrete Logarithm (e-DL) problem.

Definition 2 (e-DL Problem). *Given a safe number n without integer factorization and two random numbers $x, y \in Z^*_{n^2}$, the discrete logarithm of x under the base y is an integer s such that $x = y^s \bmod n^2$.*

The general DL problem is defined as follows. Let g be a generator of a finite cyclic group $G = \langle g \rangle$ of order N. For a random number $x \in G$, we wish to find an integer s $(0 \le s < N)$ such that $g^s = x$. If N is composite, *i.e.*, $N = p_1^{k_1} p_2^{k_2} \cdots p_j^{k_j}$, one first computes $s \ mod \ p_i^{k_i}$ $(1 \le i \le j)$ in the subgroup of order $p_i^{k_i}$ for each prime power $p_i^{k_i}$, and then, one applies the Chinese Remainder Theorem to compute s. According to [13], calculating the discrete logarithm in the subgroup of order $p_i^{k_i}$ can be reduced to finding the discrete logarithm in the group of prime order p_i. Therefore, the e-DL problem is related to the DL problem in the subgroup of prime order $max(p_1, \cdots, p_j)$. If the DL problem is hard, then the e-DL problem is also intractable.

Now we introduce a new complexity assumption called the extended strong RSA (es-RSA) assumption, which is a variant of the strong RSA problem [2] in $Z^*_{n^2}$.

Assumption 1 (es-RSA). *There exists no PPT algorithm that, given a safe number n whose factors are secret and a number $x \in Z^*_{n^2}$, outputs a pair of integers (s, y) such that $x = y^s \bmod n^2$, $n^2 > s > 2$ and $y \in Z^*_{n^2}$.*

We are neither aware of any corroboration that it should be hard, nor can we break it. However, we can design an algorithm for solving the es-RSA problem if we know algorithms that solve the CSR or e-DL problems.

1. The first algorithm selects at random the exponent s and then calls, as a subroutine, the algorithm for solving the CSR problem.
2. The second algorithm chooses at random the value of y and then calls, as a subroutine, the algorithm for solving the e-DL problem.

This also means that the complexity of es-RSA has to be lower bounded by the complexities of CSR and e-DL problems. Note that the algorithm solving es-RSA is able to manipulate the pair (y, s) by using variants of the baby-step giant-step algorithms or the birthday paradox. Although the es-RSA assumption appears to be valid even for $s = 3$, we still recommend that one uses $s \ge 2^{16} + 1$ in practice.

2.3 Paillier Cryptosystem

We now briefly review the Paillier cryptosystem. For the detail, refer to [12]. Let n be a safe number. $\mathcal{B}_\alpha \subset Z^*_{n^2}$ denotes the set of elements of order $n\alpha$, and

$\mathcal{B}$ denotes their disjoint union for $\alpha = 1, \ldots, \lambda$, where λ is adopted instead of $\lambda(n)$ for visual comfort. Randomly select a base g from $\mathcal{B}$, and define a function $F(x) = \frac{x-1}{n}$.

The Paillier cryptosystem defines an integer-valued bijective function:

$$E_g \colon Z_n \times Z_n^* \to Z_{n^2}^*,\ (x, r) \to g^x \cdot r^n \bmod n^2,$$

where (n, g) are public parameters whilst the pair (p, q) (or equivalently λ) remains private. The encryption and decryption algorithms are as follows:

Encryption:

plaintext $x < n$, randomly select $r < n$, ciphertext $y = g^x \cdot r^n \bmod n^2$.

Decryption:

ciphertext $y < n^2$, plaintext $x = D(y) = \frac{F(y^\lambda \bmod n^2)}{F(g^\lambda \bmod n^2)} \bmod n$.

The cryptosystem has the following additive homomorphic properties:

$$\forall \sigma \in Z^+,\ \ D(y^\sigma \bmod n^2) = \sigma x \bmod n, \qquad \text{and}$$
$$\forall y_1, y_2 \in Z_{n^2},\ D(y_1 y_2 \bmod n^2) = x_1 + x_2 \bmod n$$

3 Accumulator and Security Definitions

In this section we give a few related definitions for accumulators. Firstly we generalize definitions of accumulators given in [2,3,4,7], and then we describe the security definition for dynamic accumulators and our security game.

Definition 3 (Accumulator)
An accumulator consists of the following four algorithms:

KeyGen(k, M): *is a probabilistic algorithm that is executed in order to instantiate the scheme. It takes as input a security parameter 1^k and the upper bound M on the number of accumulated elements, and returns an accumulator parameter $\mathcal{P} = (P_u, P_r)$ where P_u is a public key and P_r is a private key.*

AccVal($L, \mathcal{P}$): *is a probabilistic algorithm that computes an accumulated value. It takes as input a set of elements $L = \{c_1, \ldots, c_m\}$ $(1 < m \leq M)$ from a domain $\mathcal{C}$ and the parameter $\mathcal{P}$, and returns an accumulated value v, along with some auxiliary information a_c and A_l that will be used by other algorithms.*

WitGen($a_c, A_l, \mathcal{P}$): *is a probabilistic algorithm that creates the witness for every element. It takes as input the auxiliary information a_c and A_l, and the parameter $\mathcal{P}$, and returns a witness W_i for c_i $(i = 1, \ldots, m)$.*

Verify(c, W, v, P_u): *is a deterministic algorithm that checks if a given element is accumulated in the value v or is not. It takes as input an element c, its witness W, the accumulated value v and the public key P_u, and returns* Yes *if the witness W constitutes a valid proof that c has been accumulated in v, or* No *otherwise.*

Definition 4 (Dynamic Accumulator). *A dynamic accumulator consists of the following seven algorithms:*

KeyGen, AccVal, WitGen, Verify *are the same as in the Definition 3.*

AddEle($L^\oplus, a_c, v, \mathcal{P}$): *is a probabilistic algorithm that adds some new elements to the accumulated value. It takes as input a set of new elements $L^\oplus =$*

$\{c_1^{\oplus}, \ldots, c_k^{\oplus}\}$ ($L^{\oplus} \subset \mathcal{C}, 1 \leq k \leq M - m$) that are to be added, the auxiliary information a_c, the accumulated value v and the parameter $\mathcal{P}$, and returns a new accumulated value v' corresponding to the set $L^{\oplus} \cup L$, witnesses $\{W_1^{\oplus}, \ldots, W_k^{\oplus}\}$ for the newly inserted elements $\{c_1^{\oplus}, \ldots, c_k^{\oplus}\}$, along with new auxiliary information a_c and a_u that will be used for future update operations.

DelEle($L^{\ominus}, a_c, v, \mathcal{P}$): *is a probabilistic algorithm that deletes some elements from the accumulated value. It takes as input a set of elements $L^{\ominus} = \{c_1^{\ominus}, \ldots, c_k^{\ominus}\}$ ($L^{\ominus} \subset L, 1 \leq k < m$) that are to be deleted, the auxiliary information a_c, the accumulated value v and the parameter $\mathcal{P}$, and returns a new accumulated value v' corresponding to the set $L \setminus L^{\ominus}$, along with new auxiliary information a_c and a_u that will be used for future update operations.*

UpdWit(W_i, a_u, P_u): *is a deterministic algorithm that updates witnesses for the elements that have been accumulated in v and also are accumulated in v'. It takes as input the witness W_i, the auxiliary information a_u and the public key P_u, and returns an updated witness W_i' proving that the element c_i is accumulated in the new value v'.*

Definition 5 (Security for Dynamic Accumulator). *An dynamic accumulator is secure if the adversary has only a negligible probability of finding a set of elements $L = \{c_1, \ldots, c_m\} \subseteq \mathcal{C}$ ($1 < m \leq M$), an element $c' \in \mathcal{C} \setminus L$ and a witness w' which can prove that c' has been accumulated in the accumulated value corresponding to the set L, where the probability is taken over the random strings generated by the adversary and the accumulator.*

There exist other two weak security definitions used in the literature as follows.

Definition 6 (Security for Accumulator [3]). *Given a set of elements $L = \{c_1, \ldots, c_m\}$ ($1 < m \leq M$), their accumulated value v and an element $c' \in \mathcal{C} \setminus L$. Then an accumulator is weakly secure if an adversary has a negligible probability of finding a witness w' which can prove that c' has been accumulated in v, where the probability is taken over the random coins of the adversary and of the accumulator.*

Definition 7 (Security for Accumulator [2]). *Given a set of elements $L = \{c_1, \ldots, c_m\}$ ($1 < m \leq M$), their accumulated value v. Then an accumulator is weakly secure if an adversary has a negligible probability of finding an element $c' \in \mathcal{C} \setminus L$ and a witness w' which can prove that c' has been accumulated in v, where the probability is taken over the random coins of the adversary and of the accumulator.*

Note that the difference among these two security definitions and our definition is that, in the definition 6 the adversary tries to forge a witness for a given element c', in the definition 7 the adversary can choose a forged element c' himself, and in the definition 5 the adversary might be able to choose both a set of elements L that are to be accumulated and a forged element c' himself.

To capture the notion of security for a dynamic accumulator, we define a security model – Chosen Element Attack (CEA) against Dynamic Accumulator:

Definition 8 (CEA Game). *CEA is a security game between a PPT adversary $\mathcal{ADV}$ and a challenger $\mathcal{CHA}$:*

Setup. *$\mathcal{CHA}$ runs the* **KeyGen** *algorithm to set up the parameters of the accumulator. $\mathcal{ADV}$ chooses a family of sets $L^* \subset \mathcal{C}$ and returns them to $\mathcal{CHA}$. $\mathcal{CHA}$ runs the algorithm* **AccVal** *to compute their corresponding accumulated values and* **WitGen** *to make witnesses for the elements in each set, and sends them to $\mathcal{ADV}$.*

Queries. *$\mathcal{ADV}$ is allowed to adaptively modify the sets L^* and ask $\mathcal{CHA}$ to add a set of elements $L^{\oplus}$ ($L^{\oplus} \subset \mathcal{C}$) to some accumulated values and/or remove a set of elements $L^{\ominus}$ ($L^{\ominus} \subset L^*$) from some accumulated values as he wishes. $\mathcal{CHA}$ runs the* **AddEle** *and/or* **DelEle** *algorithm, and sends back the new accumulated values, the auxiliary information for updating witnesses, along with witnesses for the newly inserted elements in the case of an* **AddEle** *operation. Then $\mathcal{ADV}$ runs the* **UpdWit** *algorithm to update the witness for each element, which has been accumulated in the old accumulated value and is currently accumulated in the new value, in the corresponding set.*

Challenge. *After making a number of queries, $\mathcal{ADV}$ decides on challenge by picking a set of elements $L = \{c_1, \dots, c_m\}$ ($1 < m \leq M$) from $\mathcal{C}$ and sends L to $\mathcal{CHA}$ who invokes* **AccVal** *to obtain the corresponding accumulated value v and* **WitGen** *to make witnesses $\{W_1, \dots, W_m\}$ for the m elements and returns them to $\mathcal{ADV}$. On receiving v and $\{W_1, \dots, W_m\}$, $\mathcal{ADV}$ produces an element c' ($c' \in \mathcal{C} \setminus L$) with a witness W', and sends them to $\mathcal{CHA}$. Then $\mathcal{CHA}$ runs the* **Verify** *algorithm to test if c' with W' is accumulated in v.*

Response. *Eventually the* **Verify** *algorithm outputs a result. We say $\mathcal{ADV}$ wins in this game, if it, with non-negligible advantage, manages to produce the legitimate pair (W', c') such that the output of* **Verify** *is* Yes.

4 New Dynamic Accumulator

We construct a new dynamic accumulator as follows.

KeyGen(k, M): Given a security parameter 1^k and the upper bound M on the number of accumulated elements, generate a suitable safe modulus n that is k-bit long and an empty set V. Let $\mathcal{C} = Z^*_{n^2} \setminus \{1\}$ and $T' = \{3, \cdots, n^2\}$. Select adaptively a number $\sigma \in Z_{n^2}$ and compute $\beta = \sigma\lambda \bmod \phi(n^2)$ such that $\beta \in T'$. Choose $\gamma \xleftarrow{R} Z_{\phi(n^2)}$ such that $\gamma \notin \{\beta, \sigma\}$. Set the public key $P_u = (n, \beta)$ and the private key $P_r = (\sigma, \lambda, \gamma)$, then output the parameter $\mathcal{P} = (P_u, P_r)$.

AccVal($L, \mathcal{P}$): Given a set of m distinct elements $L = \{c_1, \dots, c_m\}$ ($L \subset \mathcal{C}$, $1 < m \leq M$) and the parameter $\mathcal{P}$, choose $c_{m+1} \xleftarrow{R} \mathcal{C}$, and compute

$$x_i = F(c_i^{\gamma\sigma^{-1}} \bmod n^2) \bmod n \ (i = 1, \dots, m+1),$$
$$v = \sigma \sum_{i=1}^{m+1} x_i \bmod n,$$

$$y_i = c_i^{\gamma\beta^{-1}} \bmod n^2 \ (i = 1, \dots, m+1), \text{ and}$$
$$a_c = \prod_{i=1}^{m+1} y_i \bmod n^2.$$

Then output the accumulated value v and the auxiliary information a_c and $A_l = \{y_1, \dots, y_m\}$.

WitGen($a_c, A_l, \mathcal{P}$): Given the auxiliary information a_c and A_l, and the parameter $\mathcal{P}$, choose randomly a set of m numbers $T = \{t_1, \dots, t_m\} \subset T' \setminus \{\beta, \gamma\}$ $(i = 1, \dots, m)$, and compute

$$w_i = a_c y_i^{\frac{-t_i}{\gamma}} \bmod n^2 \ (i = 1, \dots, m).$$

Then output the witness $W_i = (w_i, t_i)$ for c_i $(i = 1, \dots, m)$.

Verify(c, W, v, P_u): Given an element c, its witness $W = (w, t)$, the accumulated value v and the public key P_u, test whether $\{c, w\} \subset \mathcal{C}$, $t \in T'$ and $F(w^\beta c^t \bmod n^2) \equiv v \pmod n$. If so, output Yes; otherwise, output No.

AddEle($L^\oplus, a_c, v, \mathcal{P}$): Given a set of elements $L^\oplus = \{c_1^\oplus, \dots, c_k^\oplus\}$ $(L^\oplus \subset \mathcal{C} \setminus L, 1 \le k \le M - m)$ to be inserted, the auxiliary information a_c, the accumulated value v and the parameter $\mathcal{P}$, choose $c_{k+1}^\oplus \xleftarrow{R} \mathcal{C}$ and a set of k numbers $T^\oplus = \{t_1^\oplus, \dots, t_k^\oplus\} \xleftarrow{R} T' \setminus \{T \cup \{\beta, \gamma\}\}$, and compute

$$x_i^\oplus = F((c_i^\oplus)^{\gamma\sigma^{-1}} \bmod n^2) \bmod n \ (i = 1, \dots, k+1),$$
$$v' = v + \sigma \sum_{i=1}^{k+1} x_i^\oplus \bmod n,$$
$$y_i^\oplus = (c_i^\oplus)^{\gamma\beta^{-1}} \bmod n^2, \ (i = 1, \dots, k+1),$$
$$a_u = \prod_{i=1}^{k+1} y_i^\oplus \bmod n^2, \text{ and}$$
$$w_i^\oplus = a_c a_u (y_i^\oplus)^{\frac{-t_i^\oplus}{\gamma}} \bmod n^2 \ (i = 1, \dots, k).$$

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 v' corresponding to the set $L \cup L^\oplus$, the witnesses $W_i^\oplus = (w_i^\oplus, t_i^\oplus)$ for the new added elements $c_i^\oplus$ $(i = 1, \dots, k)$ and the auxiliary information a_u and a_c.

DelEle($L^\ominus, a_c, v, \mathcal{P}$): Given a set of elements $L^\ominus = \{c_1^\ominus, \dots, c_k^\ominus\}$ $(L^\ominus \subset L, 1 \le k < m)$ to be deleted, the auxiliary information a_c, the accumulated value v and the parameter $\mathcal{P}$, choose $c_{k+1}^\ominus \xleftarrow{R} \mathcal{C}$, and compute

$$x_i^\ominus = F((c_i^\ominus)^{\gamma\sigma^{-1}} \bmod n^2) \bmod n \ (i = 1, \dots, k+1),$$
$$v' = v - \sigma \sum_{i=1}^{k} x_i^\ominus + \sigma x_{k+1}^\ominus \bmod n,$$
$$y_i^\ominus = (c_i^\ominus)^{\gamma\beta^{-1}} \bmod n^2 \ (i = 1, \dots, k+1), \text{ and}$$

$$a_u = y_{k+1}^{\ominus} \prod_{j=1}^{k} (y_j^{\ominus})^{-1} \bmod n^2.$$

Set $a_c = a_c a_u \bmod n^2$ and $V = V \cup \{a_u\}$.

Then output the new accumulated value v' corresponding to the set $L \backslash L^{\ominus}$ and the auxiliary information a_u and a_c.

UpdWit(W_i, a_u, P_u)**:** Given the witness W_i, the auxiliary information a_u and the public key P_u, 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.

Notice that the second part t_i of the element c_i's witness W_i is generated in the first execution of the algorithm **WitGen** or **AddEle**, after that, no matter how many times the algorithms **AddEle** and **DelEle** are run, the value of t_i never changes, only the first part w_i does. So, t_i can also be treated as an alternative identifier of c_i in the accumulator. In addition, as we mentioned in Section 2.2, it is recommended for practical applications that $T' = \{2^{16} + 1, \cdots, n^2\}$.

5 Batch Update

Each element in the set V created by the algorithm **KeyGen** and updated by the algorithms **AddEle** and **DelEle** is related to a time when the element was added to V, and all element are arranged chronologically. When an element wants to use the accumulator after he missed N times update of witness, he can contact the accumulator and tell her the time of his last update, then the accumulator checks the set V, collects all data items $\{v_{i_1}, \ldots, v_{i_N}\} \subset V$ that the element did not use, computes the update information $a_u = v_{i_1} \ldots v_{i_N} \bmod n^2$, and returns a_u to the element. On receiving a_u, the element computes $w_i' = w_i a_u \bmod n^2$ to obtain the new witness $W' = (w_i', t_i)$.

Observe that, the element does not know the number of changes and the types of the changes, and makes batch update at unit cost (1 multiplication) without requiring knowledge of any sensitive information, and the accumulator takes N modular multiplications for the batch update. As mentioned above, the existing accumulators do not allow for batch updates so any collection of updates must be done sequentially. The sequential update is very inefficient if it is compared with our batch update. Consider the update operations for the dynamic accumulator from [4]. If one addition operation happens, then the witness w_i is updated as follows: $w_i' = (w_i)^{v_{i_j}} \bmod n$, where v_{i_j} ($j \in [N]$) is the update information for the addition. If one deletion operation happens, on receiving the update information v_{i_j} for the deletion, an element computes a pair (a, b) of integers such that $ac_i + bv_{i_j} = 1$, and then updates her witness $w_i' = (w_i)^b (v')^a \bmod n$, where v' is the new accumulated value. Therefore, to make a batch update, an element first needs to know the type of every change (addition or deletion), chooses different algorithms for different types of the change, and then, updates her witness one change by one change in the time sequence. In particular, for a deleting operation, an element must use the new accumulated value to update her witness.

To the best of our knowledge, our scheme is the first one to do batch update efficiently and give an answer in the affirmative to Fazio and Nicolosi's open problem [7].

6 Correctness

First we show the output of the algorithm **Verify** is correct for a regular accumulator. According to the properties of the Paillier cryptosystem, for ciphertexts $\{c_1^{\gamma\beta^{-1}}, \ldots, c_{m+1}^{\gamma\beta^{-1}}\}$ and their plaintexts $\{z_1, \ldots, z_{m+1}\}$, we have

$$\sigma(z_1 + \ldots + z_{m+1}) \equiv \frac{F((c_1^{\gamma\beta^{-1}} \ldots c_{m+1}^{\gamma\beta^{-1}})^{\sigma\lambda} \bmod n^2)}{F(g^\lambda \bmod n^2)} \pmod n.$$

It follows that,

$$\sigma \textstyle\sum_{j=1}^{m+1} z_j F(g^\lambda \bmod n^2) \equiv F(\prod_{j=1}^{m+1} (c_j^{\gamma\beta^{-1}})^\beta \bmod n^2) \pmod n.$$

Since $z_j F(g^\lambda \bmod n^2) \equiv F((c_j^{\gamma\beta^{-1}})^\lambda \bmod n^2) \pmod n$, i.e.,

$$z_j F(g^\lambda \bmod n^2) \equiv F(c_j^{\gamma\sigma^{-1}} \bmod n^2) \pmod n,$$

and $y_j = c_j^{\gamma\beta^{-1}} \bmod n^2$, i.e., $c_j^\gamma = (y_j^{\frac{\gamma - t_j}{\gamma}})^\beta (y_j^{\frac{t_j}{\gamma}})^\beta \bmod n^2$,
for any $i \in [m]$, we have

$$\sigma \textstyle\sum_{j=1}^{m+1} F(c_j^{\gamma\sigma^{-1}} \bmod n^2) \equiv F((y_i^{\frac{\gamma - t_i}{\gamma}} \prod_{j \in [m+1]\setminus\{i\}} y_j)^\beta (y_i^{\frac{t_i}{\gamma}})^\beta \bmod n^2) \pmod n.$$

From the construction of accumulator, we know that

$$x_j = F(c_j^{\gamma\sigma^{-1}} \bmod n^2) \bmod n,\ (y_i^{\frac{t_i}{\gamma}})^\beta = c_i^{t_i} \bmod n^2 \text{ and}$$
$$w_i = y_i^{\frac{\gamma - t_i}{\gamma}} \textstyle\prod_{j \in [m+1]\setminus\{i\}} y_j \bmod n^2.$$

Therefore,

$$\sigma \textstyle\sum_{j=1}^{m+1} x_j \equiv F(w_i^\beta c_i^{t_i} \bmod n^2) \pmod n.$$

That is,

$$F(w_i^\beta c_i^{t_i} \bmod n^2) \equiv v \pmod n.$$

The congruence shows that, in a regular accumulator, if an element c_i ($i \in [m]$) is accumulated in the value v, the witness $W_i = (w_i, t_i)$ can give c_i a valid proof.

When some elements are added, for the new added elements $c_i^\oplus$ ($i = 1, \ldots, k$), it is easy to verify the correctness in the same way; for the old elements (previously accumulated) c_i $(i = 1, \ldots, m)$ whose witness $W_i = (w_i, t_i)$ is updated to be $W_i' = (w_i', t_i)$, we show the correctness as follows.

$$\begin{aligned}
&F((w_i')^{\beta} c_i^{t_i} \bmod n^2)\\
&\equiv F((w_i a_u)^{\beta} c_i^{t_i} \bmod n^2) \pmod n\\
&\equiv F(((y_j^{\frac{\gamma - t_j}{\gamma}})^{\beta} \textstyle\prod_{j\in[m+1]\setminus\{i\}} y_j^{\beta})(\prod_{i\in[k+1]}(y_i^{\oplus})^{\beta}(y^{\frac{t_i}{\gamma}})^{\beta}) \bmod n^2) \pmod n\\
&\equiv F((\textstyle\prod_{j\in[m+1]\setminus\{i\}} c_i^{\gamma})c_i^{\gamma}(\prod_{i\in[k+1]}((c_i^{\oplus})^{\gamma\beta^{-1}})^{\beta}) \bmod n^2) \pmod n\\
&\equiv F((\textstyle\prod_{j\in[m+1]} c_i^{\gamma\beta^{-1}})^{\sigma\lambda}(\prod_{i\in[k+1]}((c_i^{\oplus})^{\gamma\beta^{-1}})^{\sigma\lambda}) \bmod n^2) \pmod n\\
&\equiv \sigma(\textstyle\sum_{i=1}^{m+1} F(c_i^{\gamma\sigma^{-1}} \bmod n^2) + \sum_{i=1}^{k+1} F((c_i^{\oplus})^{\gamma\sigma^{-1}} \bmod n^2)) \pmod n\\
&\equiv \sigma(\textstyle\sum_{i=1}^{m+1} x_i + \sum_{i=1}^{k+1} x_i^{\oplus}) \pmod n\\
&\equiv v + \sigma \textstyle\sum_{i=1}^{k+1} x_i^{\oplus} \pmod n\\
&\equiv v' \pmod n
\end{aligned}$$

When some elements are deleted, for the left elements (which has been accumulated and is still accumulated currently) c_i with updated witnesses $W_i' = (w_i', t_i)$, the correctness can be verified in the same way.

It is also easy to verify the correctness for batch update in the same way.

7 Security

The following theorem states that any PPT adversary can not find membership proofs for those elements that are not in the accumulated set.

Theorem 1. *If the es-RSA Assumption holds, then the proposed dynamic accumulator is secure under CEA.*

Proof. Suppose that there exists a PPT adversary $\mathcal{ADV}$ who wins in the CEA game with non-negligible advantage, that means, on input (n, β), $\mathcal{ADV}$ finds with non-negligible advantage m elements $\{c_1, \ldots, c_m\} \subset \mathcal{C}$ with their witnesses $\{W_1, \ldots, W_m\}$ and an element $c' \in \mathcal{C} \setminus \{c_1, \ldots, c_m\}$ with $W' = (w', t')$ such that $F(w'^{\beta} c'^{t'} \bmod n^2) \equiv v \pmod n$, where v is the accumulation value of $(c_1, \ldots, c_m)$. We construct an algorithm $\mathcal{B}$ that breaks es-RSA Assumption with non-negligible advantage. $\mathcal{B}$ simulates $\mathcal{CHA}$ as follows:

Setup. $\mathcal{B}$ runs the algorithm **KeyGen(**k, M**)** to get the element domain $\mathcal{C} = Z_{n^2}^* \setminus \{1\}$ and setup the system parameters. $\mathcal{ADV}$ requests the accumulated values and corresponding witnesses for a polynomial number of sets $L^* \subset \mathcal{C}$ from $\mathcal{B}$.

Queries. $\mathcal{ADV}$ adaptively modifies the sets L^* and asks $\mathcal{B}$ to add a set of elements $L^{\oplus}$ $(L^{\oplus} \subset \mathcal{C})$ to some accumulated values and/or remove a set of elements $L^{\ominus}$ $(L^{\ominus} \subset L^*)$ from some accumulated values as he wishes; $\mathcal{B}$ runs the algorithm **AddEle(**$L^{\oplus}, a_c, v, \mathcal{P}$**)** and/or **DelEle(**$L^{\ominus}, a_c, v, \mathcal{P}$**)** to reply $\mathcal{ADV}$. Then $\mathcal{ADV}$ runs the algorithm **UpdWit(**W_i, a_u, P_u**)** to update the witness for related elements.

Challenge. After making a polynomial number of queries, $\mathcal{ADV}$ decides on challenge by picking a set of elements $L = \{c_1, \ldots, c_m\}$ $(1 < m \leq M)$ from $\mathcal{C}$ and querying $\mathcal{B}$ for the corresponding accumulated values v and witnesses $\{W_1, \ldots, W_m\}$. On receiving them, $\mathcal{ADV}$ produces an element c' $(c' \in \mathcal{C} \setminus L)$ with a witness $W' = (w', t')$, and sends them to $\mathcal{B}$.

Response. $\mathcal{B}$ runs the **Verify** algorithm to test if c' is accumulated in v. If **Verify** outputs Yes with non-negligible advantage, then we say $\mathcal{B}$, without n's integer factorization, has non-negligible advantage to breaks the es-RSA assumption; otherwise, nothing.

Let's analyze that how $\mathcal{B}$ breaks the es-RSA assumption.

Because the scheme appends a random element to compute the accumulated value v every time in running the algorithm **AccVal**, **AddEle** and **DelEle**, and t_i is chosen at random, the probability distributions of v, w_i, t_i and a_u are uniform. Observing the outputs of queries for the keyword lists in the stage of **Queries** and their changes cannot help $\mathcal{ADV}$ to forge anything. So, let's only consider the challenge set $L = \{c_1, \ldots, c_m\}$.

With n's integer factorization, $\mathcal{B}$ computes the v and $\{W_1, \ldots, W_m\}$ for the set $L = \{c_1, \ldots, c_m\}$, so $\mathcal{B}$ has m congruences $F(w_i^{\beta} c_i^{t_i} \bmod n^2) \equiv v \pmod{n}$ $(i = 1, \ldots, m)$, which means that $\exists k \in Z$ such that

$$\frac{w_i^{\beta} c_i^{t_i} \bmod n^2 - 1}{n} = kn + v.$$

It follows that,

$$w_i^{\beta} c_i^{t_i} \equiv vn + 1 \pmod{n^2}.$$

This congruence can also be expressed in the following two ways:

$$w_i^{\beta} \equiv (vn + 1) c_i^{-t_i} \pmod{n^2}.$$

$$c_i^{t_i} \equiv (vn + 1) w_i^{-\beta} \pmod{n^2}.$$

Therefore, $\mathcal{B}$ has m triples (c_i, w_i, t_i) such that c_i, w_i and t_i are the t_i-th root of $(vn + 1)w_i^{-\beta} \pmod{n^2}$, the β-th root of $(vn + 1)c_i^{-t_i} \pmod{n^2}$ and the logarithmic value of $(vn + 1)w_i^{-\beta} \pmod{n^2}$ based on c_i, respectively.

If $\mathcal{ADV}$ wins in the CEA game with non-negligible advantage, the **Verify** outputs Yes with non-negligible advantage. That means, without n's integer factorization, $\mathcal{B}$ has non-negligible advantage to get a congruence

$$F(w'^{\beta} c'^{t'} \bmod n^2) \equiv v \pmod{n},$$

thus, $\mathcal{B}$ has non-negligible advantage to get a distinct triple (c', w', t') such that c', w' and t' are the t'-th root of $(vn + 1)w'^{-\beta} \pmod{n^2}$, the β-th root of $(vn+1)c'^{-t'} \pmod{n^2}$ and the logarithmic value of $(vn+1)w'^{-\beta} \pmod{n^2}$ based on c', respectively.

We know that $c' \notin L$, that means $c' \neq c_i$ for any $i \in 1, \ldots, m$, so let's identify four cases, depending on whether $w' = w_i$ or $t' = t_i$ (or not) for some $i \in [m]$.

Case 1: $w' \neq w_i$ for any $i \in [m]$.
We have $w'^{\beta} \neq w_i^{\beta} \pmod{n^2}$, so $(vn+1)w'^{-\beta} \neq (vn+1)w_i^{-\beta} \pmod{n^2}$. That means, $(vn+1)w'^{-\beta} \pmod{n^2}$ is a distinct number from $\{(vn+1)w_i^{-\beta} \pmod{n^2}\}_{i=1,\dots,m}$, and $\mathcal{B}$ has its t'-th root c' or the logarithmic value t' based on c'. Thus, $\mathcal{B}$ breaks the es-RSA assumption.
Case 2: $w' = w_i$ for some $i \in [m]$.
We have $c'^{t'} = (vn+1)w'^{-\beta} = (vn+1)w_i^{-\beta} = c_i^{t_i} \pmod{n^2}$, so $t' \neq t_i$. That means, $\mathcal{B}$ has c' as the t'-th root of $(vn+1)w'^{-\beta} \pmod{n^2}$ and t' as the logarithmic value of $(vn+1)w'^{-\beta} \pmod{n^2}$ based on c'. Thus, $\mathcal{B}$ breaks the es-RSA assumption.
Case 3: $t' = t_i$ for some $i \in [m]$.
We have $c'^{t'} \neq c_i^{t_i} \pmod{n^2}$. It is convenient to split this case into two subcases:
Case 3a: $c'^{t'} = c_j^{t_j} \pmod{n^2}$ for some $j \in [m] \setminus \{i\}$. This case is the same as in **Case 2**.
Case 3b: $c'^{t'} \neq c_j^{t_j} \pmod{n^2}$ for any $j \in [m] \setminus \{i\}$. So, $(vn+1)c'^{-t'} \pmod{n^2}$ is a distinct number from $\{(vn+1)c_i^{-t_i} \pmod{n^2}\}_{i=1,\dots,m}$, and $\mathcal{B}$ has its β-th root w'. This means, $\mathcal{B}$ breaks the es-RSA assumption.
Case 4: When $t' \neq t_i$ for any $i \in [m]$.
We also split this case into two subcases:
Case 4a: $c'^{t'} = c_j^{t_j} \pmod{n^2}$ for some $j \in [m]$. This case is the same as in **Case 2**.
Case 4b: $c'^{t'} \neq c_j^{t_j} \pmod{n^2}$ for any $j \in [m]$. This case is the same as in **Case 3b**.

Consequently, the theorem is proved.

8 Conclusions and Future Direction

We have considered the problem of design of the dynamic accumulators, and introduced formal generic definitions of accumulators and a new security model called CEA. We constructed a new dynamic accumulator that allows an efficient batch update. The scheme is based on the Paillier public-key cryptosystem, and is sound and secure under the es-RSA assumption.

The computation complexity of our approach is reasonable, but the length of witnesses is $4 \log n$ bits and the scheme needs to compute at least $(2^{16}+1)$-th roots. So designing more space-efficient and time-efficient accumulators remains a challenging open problem.

Acknowledgments

The work was in part supported by Australian Research Council Discovery grants DP0663452, DP0558773 and DP0665035. Huaxiong Wang's research was in part supported by Singapore Ministry of Education grant T206B2204.

References

1. Au, M.H., Wu, Q., Susilo, W., Mu, Y.: Compact E-Cash from Bounded Accumulator. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 178–195. Springer, Heidelberg (2006)
2. Baric, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)
3. Benaloh, J., Mare, M.: One-way accumulators: a decentralized alternative to digital signatures. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994)
4. Camenisch, J., Lysyanskaya, A.: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
5. Catalano, D., Gennaro, R., Howgrave-Graham, N., Nguyen, P.: Paillier's Cryptosystem Revisited. In: ACM CCS 2001, pp. 206–214 (2001)
6. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous Identification in Ad Hoc Groups. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 609–626. Springer, Heidelberg (2004)
7. Fazio, N., Nicolosi, A.: Cryptographic Accumulators: Definitions, Constructions and Applications. Available (2003), at `http://www.cs.nyu.edu/fazio/papers/`
8. Gentry, C., Ramzan, Z.: RSA Accumulator Based Broadcast Encryption. In: Zhang, K., Zheng, Y. (eds.) ISC 2004. LNCS, vol. 3225, pp. 73–86. Springer, Heidelberg (2004)
9. Goodrich, M., Tamassia, R., Hasić, J.: An efficient dynamic and distributed cryptographic accumulator. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 372–388. Springer, Heidelberg (2002)
10. Nguyen, L.: Accumulators from Bilinear Pairings and Applications. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 275–292. Springer, Heidelberg (2005)
11. Nyberg, K.: Fast accumulated hashing. In: Gollmann, D. (ed.) 3rd Fast Software Encryption Workshop. LNCS, vol. 1039, pp. 83–87. Springer, Heidelberg (1996)
12. Paillier, P.: Public-Key Cryptosystems based on Composite Degree Residue Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
13. Pohlig, S., Hellman, M.: An Improved Algorithm for Computing Logarithms in $GF(p)$ and its Cryptographic Significance. IEEE Trans. Inform. Theory IT-24, 106–111 (1978)
14. Sander, T.: Efficient accumulators without trapdoor. In: Varadharajan, V., Mu, Y. (eds.) Information and Communication Security. LNCS, vol. 1726, pp. 252–262. Springer, Heidelberg (1999)
15. Tsudik, G., Xu, S.: Accumulating Composites and Improved Group Signing. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 269–286. Springer, Heidelberg (2003)
16. Zhang, F., Che, X.: Cryptanalysis and improvement of an ID-based ad-hoc anonymous identification scheme at CT-RSA 2005, Cryptology ePrint Archive, Report 2005/103