Zero-Knowledge Accumulators and Set Operations

Esha Ghosh[1], Olga Ohrimenko[2], Dimitrios Papadopoulos[3], Roberto Tamassia[1] and Nikos Triandopoulos[4,3]

[1] Dept. of Computer Science, Brown University, Providence RI, USA
esha_ghosh@brown.edu, rt@cs.brown.edu
[2] Microsoft Research, Cambridge, UK
oohrim@microsoft.com
[3] Dept. of Computer Science, Boston University, Boston MA, USA
dipapado@bu.edu
[4] RSA Laboratories, Cambridge MA, USA
nikolaos.triandopoulos@rsa.com

Abstract. Accumulators provide a way to succinctly represent a set with elements drawn from a given domain, using an *accumulation value*. Subsequently, short proofs for the set-*membership* (or *non-membership*) of any element from the domain can be constructed and efficiently verified with respect to this accumulation value. Accumulators have been widely studied in the literature, primarily, as an *authentication* primitive: a malicious prover (e.g., an untrusted server) should not be able to provide convincing proofs on false statements (e.g., successfully prove membership for a value not in the set) to a verifier that issues membership queries (of course, having no access to set itself). In essence, in existing constructions the accumulation value acts as a (honestly generated) "commitment" to the set that allows selective "opening" as specified by membership queries—but with no "hiding" properties.

In this paper we revisit this primitive and propose a privacy-preserving enhancement. We define the notion of a *zero-knowledge accumulator* that provides the following very strong privacy notion: Accumulation values and proofs constructed during the protocol execution leak nothing about the set itself, or any subsequent updates to it (i.e., via element insertions/deletions). We formalize this property by a standard real/ideal execution game. An adversarial party that is allowed to choose the set and is given access to query and update oracles, cannot distinguish whether this interaction takes place with respect to the honestly executed algorithms of the scheme or with a simulator that is not given access to the set itself (and for updates, it does not even learn the type of update that occurred—let alone the inserted/deleted element). We compare our new privacy definition with other recently proposed similar notions showing that it is strictly stronger: We give a concrete example of the update-related information that can be leaked by previous definitions.

We provide a mapping of the relations between zero-knowledge accumulators and primitives that are either set in the same security model or solve the same problem. We formally show and discuss a number of implications among primitives, some of which are not immediately evident. We believe this contribution is interesting on its own, as the area has received considerable attention recently (e.g., with the works of [Naor et al., TCC 2015] and [Derler et al., CT-RSA 2015]).

We then construct the first dynamic universal zero-knowledge accumulator. Our scheme is perfect zero-knowledge and is secure under the q-Strong Bilinear Diffie-Hellman assumption.

Finally, building on our dynamic universal zero-knowledge accumulator, we define a *zero-knowledge authenticated set collection* to handle more elaborate set operations (beyond set-membership). In particular, this primitive allows one to outsource a collection of sets to an untrusted server that is subsequently responsible for answering union, intersection and set difference queries over these sets issued by multiple clients. Our scheme provides proofs that are succinct and efficiently verifiable and, at the same time, leak nothing beyond the query result. In particular, it offers verification time that is asymptotically optimal (namely, the same as simply reading the answer), and proof construction that is asymptotically as efficient as existing state-of-the-art constructions— that however, do not offer privacy.

Keywords: *zero-knowledge accumulators, cryptographic accumulators, secure set-operations, zero-knowledge authenticated set collection, secure data outsourcing*

1 Introduction

A cryptographic accumulator is a primitive that offers a way to succinctly represent a set $\mathcal{X}$ of elements by a single value acc referred to as the *accumulation value*. Moreover, it provides a method to efficiently and succinctly prove (to a party that only holds acc) that a candidate element x belongs to the set, by computing a constant-size proof w, referred to as *witness*. The interaction is in a three-party model, where the *owner* of the set runs the initial key generation and setup process to publish the accumulation value. Later an untrusted *server* handles queries regarding the set issued by a number of *clients*, providing membership answers with corresponding witnesses.

Accumulators were originally introduced by Benaloh and del Mare in [BdM94]. Since then, the relevant literature has grown significantly (see for example, [Nyb96a,BP97,BLL00,CL02,CHKO08,DT08,CKS09,ATSM09,Lip12,CPPT14,DHS15][1]) with constructions in various models. At the same time, accumulators have found numerous other applications in the context of public-key infrastructure, certificate management and revocation, time-stamping, authenticated dictionaries, set operations, anonymous credentials, and more.

Traditionally in the literature, the security property associated with accumulators was *soundness* (or *collision-freeness*), expressed as the inability to compute a witness for an element $x \notin \mathcal{X}$. Subsequently, accumulators were extended to *universal accumulators* [LLX07,DT08,ATSM09] that support non-membership proofs as well. Soundness for universal accumulators expresses the inability to forge a witness for an element, i.e., if $x \in \mathcal{X}$, it should be hard to prove $x \notin \mathcal{X}$ and vice-versa. No notion of privacy was considered, e.g., "what does an adversary that observes the client-server communication learn about the set $\mathcal{X}$" or "does the accumulation acc reveal anything about the elements of $\mathcal{X}$". It is clear to us that such a property would be attractive, if not—depending on the application—crucial. For example, in the context of securing the Domain Name System (DNS) protocol by accumulating the set of records in a zone, it is crucial to not leak any information about values in the accumulated set while responding to queries.[2] As an additional example, recently Miers et al. [MGGR13] developed a privacy enhancement for Bitcoin, that utilizes the accumulator of [CL02]. In such a context, it is very important to minimize what is leaked by accumulation values and witnesses in order to achieve anonymity (for individuals and transactions).

Quite recently, de Meer et al. [dMLPP12] and Derler et al. [DHS15] suggested the introduction of an *indistinguishability* property for cryptographic accumulators, in order to provide some notion of privacy. Unfortunately, the definition of the former was inherently flawed, as noted in [DHS15][3], whereas the later, while meant to serve cryptographic accumulators that support changes in the accumulated set (i.e., element insertion and deletion), did not protect the privacy of theses changes, as any adversary suspecting a particular modification can check the correctness of his guess.

In this work, we propose the notion of *zero-knowledge* for cryptographic accumulators. We define this property via an extensive real/ideal game, similar to that of standard zero-knowledge [GMR85]. In the real setting, an adversary is allowed to choose his challenge set and to receive the corresponding accumulation. He is then given oracle access to the querying algorithm as well as an update algorithm that allows him to request updates in the set (receiving the updated accumulation value every time). In the ideal setting, the adversary interacts with a simulator that does not know anything about the set or the nature of the updates, other than the fact that an update occurred. Zero-knowledge is then defined as the inability of the adversary to distinguish between the two settings. Our notion of zero-knowledge differs from the privacy notion of [DHS15], by protecting not only the originally accumulated set but also all subsequent updates. In fact, we formally prove that zero-knowledge is a strictly stronger property than indistinguishability in the context of cryptographic accumulators.

[1] We refer interested readers to [DHS15] for a comprehensive review of existing schemes.

[2] See for example, `https://tools.ietf.org/html/rfc5155`.

[3] Subsequently, the definition was strengthened in [SPB$^+$12], but it is still subsumed by that of [DHS15].

We provide the first zero-knowledge accumulator construction and prove its security. Our construction builds upon the *bilinear accumulator* of Nguyen [Ngu05] and achieves *perfect* zero-knowledge. Our scheme falls within the category of *dynamic universal* cryptographic accumulators: It allows to not only prove membership, but also non-membership statements (i.e., one can compute a witness for the fact that $x \notin \mathcal{X}$), and supports efficient changes in the accumulation value due to insertions and deletions in the set. It is secure under the q-Strong Bilinear Diffie-Hellman assumption, introduced in [BB04]. In order to provide non-membership witness computation in zero-knowledge, we had to deviate from existing non-membership proof techniques for the bilinear accumulator ([DT08,ATSM09]). We instead used the disjointness technique of [PTT11], appropriately enhanced for privacy. From an efficiency perspective, we show that the introduction of zero-knowledge to the bilinear accumulator comes at an insignificant cost: Asymptotically all computational overheads are either the same or within a poly-logarithmic factor of the construction of [Ngu05] that offers no privacy.

In general, a cryptographic accumulator can be viewed as special case of an *authenticated data structure* (ADS) [Mer80,Mer89,MTGS01,Tam03], where the supported data type is a set, and the queries are set membership/non-membership for elements of this set. As a result, our zero-knowledge accumulator has the same functionality as an authenticated set but with additional privacy property. Moreover, it falls within the general framework of *zero-knowledge authenticated data structures* (ZKADS) introduced recently in [GGOT15], where an underlying data structure supports queries such that the response to a query is verifiable and leaks nothing other than the answer itself.

Beyond set-membership One natural question is how to build a ZKADS with an expanded supported functionality that goes beyond set-membership. In particular, given multiple sets, we are interested in accommodating more elaborate set-operations (set union, intersection and difference).[4] We propose *zero-knowledge authenticated set collection* for the following setting. A party that owns a database of sets of elements outsources it to an untrusted server that is subsequently charged with handling queries, expressed as set operations among the database sets, issued by multiple clients. We provide the first scheme that provides not only integrity of set operations but also privacy with respect to clients (i.e., the provided proofs leak nothing beyond the answer). The fundamental building block for this construction is our zero-knowledge accumulator construction, together with a carefully deployed *accumulation tree* [PTT15]. We note that if we restrict the security properties only to soundness—as is the case in the traditional literature of authenticated data structures—there are existing schemes (specifically for set-operations) by Papamanthou et al. [PTT11] for the single-operation case, and by Canetti et al. [CPPT14] and Kosba et al. [KPP$^+$14] for the case of multiple (nested) operations. However, none of these constructions offer any notion of privacy, thus our construction offers a natural strengthening of their security guarantees.

Contributions. Our contributions can be summarized as follows:

– We define the property of zero-knowledge for cryptographic accumulators and show that it is strictly stronger than existing privacy notions for accumulators.
– We describe the complex relations between cryptographic primitives in the area. Specifically, we show that zero-knowledge sets can be used in a black-box manner to construct zero-knowledge accumulators (with or without trapdoors). We also show that zero-knowledge accumulators imply primary-secondary-resolver membership proof systems [NZ14].
– We provide the first construction of a zero-knowledge dynamic universal accumulator (with trapdoor). Our scheme is secure under the q-SBDH assumption and is perfect zero-knowledge.
– Using our zero-knowledge accumulator as a building block, we construct the first protocol for zero-knowledge outsourced set algebra operations. Our scheme offers secure and efficient intersection, union and set-difference operations under the q-SBDH assumption. We instantiate the set-difference operation

[4] We stress that these operations form a complete set-operations algebra.

in the random oracle model to achieve efficiency and discuss how it can be instantiated in the standard model with some efficiency overhead. Our construction (except for the update cost) is asymptotically as efficient as the previous state-of-the-art construction from [PTT11], that offered no privacy guarantees.

Other related work. Our privacy notion is reminiscent of that of zero-knowledge sets [MRK03,CHL$^+$05,CFM08,LY10] where set membership and non-membership queries can be answered without revealing anything else about the set. Zero-knowledge sets are a stronger primitive since they assume no trusted owner: Server and owner are the same (untrusted) entity. On the other hand, accumulators (typically) yield more lightweight constructions with faster verification and constant-size proofs, albeit in the three-party model[5].

Very recently, Naor et al. [NZ14] introduced primary-secondary-resolver membership proof systems, a primitive that is also a relaxation of zero-knowledge sets in the three-party model, and showed applications in network protocols in [GNP$^+$14]. Our definitions are quite similar, however since they define non-adaptive security our zero-knowledge accumulators imply their definition. Moreover, their privacy notion is functional zero-knowledge, i.e., they tolerate some function of the set to be leaked, e.g., its cardinality. Finally, they only cater for the static case and have to rely on external mechanisms (e.g., time-to-live cookies) to handle changes in the set. In contrast, our zero-knowledge definition also protects updates and our construction has built-in mechanisms to handle them.

In Section 4 we discuss more extensively the relation between these three primitives.

Existing works for cryptographic accumulators (e.g., [CL02,Ngu05,ATSM09,LLX07]) equip the primitive with zero-knowledge proof-of-knowledge protocols, such that a client that knows his value x is (or is not) in $\mathcal{X}$, can efficiently prove to a third-party arbitrator that indeed his value is (resp. is not) in the set, without revealing x. We stress that this privacy goal is very different from ours. Here we are ensuring that the entire protocol execution (as observed by a curious client or an external attacker) leaks nothing.

Regarding related work for set operations, the focus in the cryptographic literature has been on the privacy aspect with a very long line of works (see for example, [FNP04,KS05,BA12,HN12,HEK12]), some of which focus specifically on set-intersection (e.g., [JL09,DSMRY09,CT10,DCW13]). The above works fit in the secure two-party computation model and most are secure (or can be made with some loss in efficiency) also against malicious adversaries, thus guaranteeing the authenticity of the result. However, typically this approach requires multi-round interaction, and larger communication cost than our construction. On the other hand, here our two security properties are "one-sided": Only the server may cheat with respect to soundness and only the client with respect to privacy; in this setting we achieve non-interactive solutions with optimal proof-size. There also exist works that deal exclusively with the integrity of set operations, such as [MBKK04] that achieves linear verification and proof cost, and [ZX14] that only focuses on set-intersection but can be combined with an encryption scheme to achieve privacy versus the server.

Another work that is related to ours is that of Fauzi et al. [FLZ14] where the authors present an efficient non-interactive zero-knowledge argument for proving relations between committed sets. Conceptually, this work is close to zero-knowledge sets, allowing also for more general set operation queries. From a security viewpoint, this work is in the stronger two-party model hence it can accommodate our three-party setting as well. Also, from a functionality viewpoint, their construction works for (more general) multi-set operations. However, they rely on non-falsifiable knowledge-type assumptions to prove their scheme secure, and their scheme trivially leaks an upper-bound on the committed sets. Moreover, their construction cannot be efficiently generalized for operations on more than two sets at a time, and they do not explicitly consider efficient modifications in the sets.

We also note that recently other instantiations of zero-knowledge authenticated data structures have been proposed, including lists, trees and partially-ordered sets of bounded dimension [GOT14,GGOT15].

[5] See however the discussion of accumulators versus trapdoorless accumulators in Section 3.

2 Preliminaries

In this section we introduce notation and cryptographic tools that we will be using for the rest of the paper.

We denote with λ the security parameter and with $\nu(\lambda)$ a negligible function. A function $f(\lambda)$ is negligible if for each polynomial function $poly(\lambda)$ and all large enough values of λ, $f(\lambda) < 1/(poly(\lambda))$. We say that an event can occur with negligible probability if its occurrence probability can be upper bounded by a negligible function. Respectively, an event takes place with overwhelming probability if its complement takes place with negligible probability. The symbol $\xleftarrow{\$} \mathbb{X}$ denotes uniform sampling from domain $\mathbb{X}$. We denote the fact that a Turing machine Adv is probabilistic, polynomial-time by writing PPT Adv.

Bilinear pairings. Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p, generated by g. Let also $\mathbb{G}_T$ be a cyclic multiplicative group with the same order p and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear pairing with the following properties: (1) Bilinearity: $e(P^a, Q^b) = e(P,Q)^{ab}$ for all $P,Q \in \mathbb{G}$ and $a,b \in \mathbb{Z}_p$; (2) Non-degeneracy: $e(g,g) \neq 1_{\mathbb{G}_T}$; (3) Computability: There is an efficient algorithm to compute $e(P,Q)$ for all $P,Q \in \mathbb{G}$. We denote with $pub := (p, \mathbb{G}, \mathbb{G}_T, e, g)$ the bilinear pairings parameters, output by a randomized polynomial-time algorithm $\mathsf{GenParams}$ on input 1^λ. For clarity of presentation, we assume for the rest of the paper a symmetric (Type 1) pairing e. We note though that both our constructions can be securely implemented in the (more efficient) asymmetric pairing case, with straight-forward modifications (see [CM11] for a general discussion on pairings).

Our security proofs makes use of the q-Strong Bilinear Diffie-Hellman (q-SBDH) assumption over groups with bilinear pairings introduced by Boneh and Boyen in [BB04].

Assumption 1 (q-Strong Bilinear Diffie-Hellman) *For any PPT adversary* Adv *and for q being a parameter of size polynomial in λ, there exists negligible function $\nu(\lambda)$ such that the following holds:*

$$\Pr\left[\begin{array}{c} pub \leftarrow \mathsf{GenParams}(1^\lambda); s \leftarrow_R \mathbb{Z}_p^*; \\ (z,\gamma) \in \mathbb{Z}_p^* \times \mathbb{G}_T \leftarrow \mathsf{Adv}(pub, (g^s, \ldots, g^{s^q})) : \gamma = e(g,g)^{1/(z+s)} \end{array}\right] \leq \nu(\lambda)] .$$

Non Interactive Zero Knowledge proof of Knowledge Protocols (NIZKPoK). A non-interactive zero-knowledge proof of knowledge protocol (NIZKPoK) for any NP language membership L proof system operates in the public random string model, and consists of polynomial-time algorithms P and V that work as follows: The algorithm P takes the common reference string σ and values (x,w) such that $x \in L$ and w is a witness to this. P outputs a proof π. The algorithm V takes (σ, x, π) as input and outputs accept or reject. The security properties of a NIZKPoK are the following:

Competeness: For all $x \in L$, for all witnesses w for x, for all values of random string σ and for all outputs π of $P(\sigma,x,w)$, $V(\sigma,x,\pi) = \mathsf{accept}$.

Soundness: For all adversarial prover algorithms P^*, for a randomly chosen σ, the probability that P^* can produce (x,π) such that $x \notin L$ but $V(\sigma,x,\pi) = \mathsf{accept}$ is negligible.

Knowledge Soundness with error δ*:* A zero-knowledge proof of knowledge protocol for any NP language membership L has a stronger form of soundness that says that if a cheating prover P^* convinces the verifier that $x \in L$ with noticeable probability (i.e., more than the soundness error), then not only this means that $x \in L$ but it actually means that P^* "knows" a witness in the sense that it could obtain a witness by running some algorithm. To put more formally, for every possibly cheating prover P^*, and every x, if P^* produces π such that $Pr[V(\sigma,x,\pi) = \mathsf{accept}] > \delta + \rho$, ($\delta$ is the soundness error) then there's a algorithm E (called a knowledge extractor) with running time polynomial in $1/\rho$ and the running time of P^*, that on input x outputs a witness w for x with probability at least $1/2$.

Zero-Knowledge: There exist algorithms ZKSim-Setup and ZKSim-Prove, such that the following holds. ZKSim-Setup takes the security parameter as input and outputs (σ, s). For all x, ZKSim-Prove takes (σ, s, x) as input and outputs simulated proof π^S. Even for a sequence of adaptively and adversarially picked $(x_i,\dots,x_m)$ (where m is polynomial in the security parameter), if $x_i \in L$ for $i \in [1,m]$, then the simulated proofs $\pi_1{}^S,\dots\pi_m{}^S$ are distributed indistinguishably from proofs $\pi_1,\dots,\pi_m$ that are computed by running $P(\sigma,x_i,w_i)$ where w_i is some witness that $x_i \in L$.

NIZKPoK protocol for Discrete Log (DL). Here we describe a NIZKPoK protocol based on DL assumption, following the style of Schnorr protocols [Sch89]. The construction is non-interactive and uses the Fiat-Shamir transformation [FS87] for efficiency and its security is provable in the random-oracle (RO) model [BR93]. Informally, in the following protocol, the prover P proves to the verifier V that it knows the discrete log of a given value in zero-knowledge. We succinctly represent this protocol as $\mathsf{PK} = \{(h,x) : h = g'^x\}$. Let us denote the proof units sent by the prover P to the verifier V as $\mathsf{PKproof}$. We describe the protocol in the RO model in Figure 1, where $\mathcal{H}$ is a cryptographic hash function viewed as a RO.

The protocol proceeds as follows:
- P picks a random $u \in \mathbb{Z}_p^*$, computes $b \leftarrow g'^u$.
- Then P computes $c \leftarrow \mathcal{H}(b)$.
- P computes $r \leftarrow u + cx$ and sets $\mathsf{PKproof} := (b,c,r)$.
- Finally P sends $\mathsf{PKproof}$ to the verifier.

The verification proceeds as follows:
- Parse $\mathsf{PKproof}$ as (b,c,r).
- Verify if $c = \mathcal{H}(b)$. If not, **return** reject. Else proceed to next step.
- Verify if $g'^r = bh^c$. If the verification fails, **return** reject. Else **return** accept.

Fig. 1: $\mathsf{PK} = \{(h,x) : h = g'^x\}$

Characteristic Polynomial. A set $\mathcal{X} = \{x_1,\dots,x_n\}$ with elements $x_i \in \mathbb{Z}_p$ can be represented by a polynomial following an idea introduced in [FNP04]. The polynomial $\mathsf{Ch}_{\mathcal{X}}(z) = \prod_{i=1}^n (x_i + z)$ from $\mathbb{Z}_p[z]$, where z is a formal variable, is called the *characteristic polynomial* of $\mathcal{X}$. In what follows, we will denote this polynomial simply by $\mathsf{Ch}_{\mathcal{X}}$ and its evaluation at a point y as $\mathsf{Ch}_{\mathcal{X}}(y)$.

The following lemma characterizes the efficiency of computing the characteristic polynomial of a set and of Extended Euclidian algorithm.

Lemma 1 ([PSL76]) *Given a set $\mathcal{X} = x_1,\dots,x_n \in \mathbb{Z}_p^n$, its characteristic polynomial $\mathsf{Ch}_{\mathcal{X}} := \sum_{i=0}^n c_i z^i \in \mathbb{Z}_p[z]$ can be computed with $O(n \log n)$ operations by FFT interpolation.*

We will use the following Lemma while proving correctness of coefficients of a polynomial:

Lemma 2 (Schwartz–Zippel) *Let $p[z], q[z]$ be two d-degree polynomials from $\mathbb{Z}_p[z]$. Then for $w \xleftarrow{\$} \mathbb{Z}_p$, the probability that $p(w) = z(w)$ is at most d/p, and the equality can be tested in time $O(d)$.*

If $p \in O(2^\lambda)$, it follows that the above probability is negligible, if d is $poly(\lambda)$.

Complexity Model. To explicitly measure complexity with respect to the number of primitive cryptographic operations, without considering the dependency on the security parameter, we adopt the complexity model used in [PTT11]. The *access complexity* of an algorithm is defined as the number of memory accesses this algorithm performs on the authenticated data structure stored in an indexed memory of n cells, in order for the algorithm to complete its execution. We require that each memory cell can store up to $O(poly(\log n))$ bits. The *group complexity* of data collection is defined as the number of elementary data objects contained in that object. Whenever it is clear from the context, we omit the terms “access" and “group".

Given a collection of sets $\mathcal{X}_{i_1}, \ldots \mathcal{X}_{i_k}$ and their characteristic polynomial representation, we summarize a characterization of the intersection of the sets in the following lemma.

Lemma 3 ([PTT11]) *Set* answer *is the intersection of the sets* $\mathcal{X}_{i_1}, \ldots \mathcal{X}_{i_k}$ *if and only if there exists polynomials* $q_1[z], \ldots q_k[z]$ *such that* $\sum_{j \in [i_1, i_k]} q_1[z] P_1[z] = 1$ *where* $P_j[z] = \mathsf{Ch}_{\mathcal{X}_j \setminus \mathsf{answer}}[z]$. *Moreover, computing polynomials* $q_j[z]$ *where* $j \in [i_1, i_k]$ *has* $O(N \log^2 N \log \log N)$ *complexity where* $N = \sum_{j \in [i_1, i_k]} n_j$ *and* $n_j = |\mathcal{X}_j|$.

Accumulation tree: Given a collection of sets $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, let $\mathsf{acc}(\mathcal{X}_i)$ be a succinct representation (constant size) of $\mathcal{X}_i$ using its characteristic polynomial. We describe an authentication mechanism that does the following. A trusted party computes m hash values $h_i := h(\mathsf{acc}(\mathcal{X}_i))$ (using collision resistant cryptographic hash function) of the m sets of $\mathbb{S}$. Then given a short public digest information of the current set collection $\mathbb{S}$, the authentication mechanism provides publicly verifiable proofs of the form "h_i is the hash of the i^{th} set of the current set collection $\mathbb{S}$".

A popular authentication mechanism for proofs of this form are Merkle hash trees that based on a single value digest can provide logarithmic size proofs and support updates. An alternative authentication mechanism to Merkle trees, (specifically in the bilinear group setting) are *accumulation trees* [PTT15]. Intuitively, an accumulation tree can be seen as a "flat" version of Merkle trees.

An accumulation tree (AT) is a tree with $\lceil \frac{1}{\varepsilon} \rceil$ levels, where $0 < \varepsilon < 1$ is a parameter chosen upon setup, and m leaves. Each internal node of T has degree $O(m^{\varepsilon})$ and T has constant height for a fixed ε. Each leaf node contains the h_i and each internal node contains the hash of the values of its children. We will describe the setup, query, update and verification of AT in the following algorithms. For simplicity, we skip an explicit key generation phase and describe the keys needed for each algorithm as its input. We have 3 kinds of keys for an AT: the secret key (sk), an evaluation key (ek) derivable from the secret key, which is used by the $\mathsf{ATQuery}$ algorithm for generating authentication paths, and finally a verification key (vk) corresponding to the secret key, which is used for verification of an authentication path. An accumulation tree scheme AT is defined as a tuple of 4 PPT algorithms: $\mathsf{AT} = (\mathsf{ATSetup}, \mathsf{ATQuery}, \mathsf{ATUpdate}, \mathsf{ATVerify})$. We describe the input and output of each of the algorithms here. To capture the notion of the most recent set collection, we use subscript t, i.e., $\mathbb{S}_t$ denotes the set collection at time t. Though this subscript is not necessary to describe these algorithms by themselves, we would need this index when using AT as a subroutine for our zero-knowledge authenticated set collection scheme (ZKASC). A detailed construction from [PTT11] can be found in Appendix A.

Setup: $(\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0) \leftarrow \mathsf{ATSetup}(\mathsf{sk}, (\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m)))$ $\mathsf{ATSetup}$ takes a secret key (sk) and a set of accumulation values $(\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m))$ for a set collection ($\mathbb{S}_0$) and builds an AT on top of it. This algorithm returns the authentication information for the set collection ($\mathsf{auth}(\mathbb{S}_0)$) and the root of the AT as the digest (digest_0).

Query: $(\Pi_i, \alpha_i) \leftarrow \mathsf{ATQuery}(\mathsf{ek}_t, i, \mathsf{auth}(\mathbb{S}_t))$ $\mathsf{ATQuery}$ takes the evaluation key ek_t, authentication information for the set collection $\mathsf{auth}(\mathbb{S}_t)$ and a particular index i of the set collection and returns the authentication path for that set $\mathcal{X}_i$, denoted as Π_i and the accumulation value of that set as α_i.

Update: $(\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}_i) \leftarrow \mathsf{ATUpdate}(\mathsf{sk}, i, \mathsf{acc}'(\mathcal{X}_i), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$ $\mathsf{ATUpdate}$ is the update algorithm that updates the accumulation value for a particular set $\mathcal{X}_i$ to $\mathsf{acc}'(\mathcal{X}_i)$. This algorithm takes the old the authentication information for the set collection $\mathsf{auth}(\mathbb{S}_t)$, and the root digest digest_t as input along with $\mathsf{acc}'(\mathcal{X}_i)$. It outputs the updated authentication information for the set collection auth', the updated digest digest' and the update authentication information (in $\mathsf{updinfo}_i$).

For our construction, we will use a variant of the $\mathsf{ATUpdate}$ algorithm that allows for batch updates. The batch update algorithm $\mathsf{ATUpdateBatch}$ takes a series of updated accumulation values $(i_1, \mathsf{acc}'(\mathcal{X}_{i_1}), \ldots, i_k, \mathsf{acc}'(\mathcal{X}_{i_k}))$ instead of one, along with the old authentication information $\mathsf{auth}(\mathbb{S}_t)$ and root digest digest_t as input. It outputs auth', digest' and a series of update authentication information (in $(\mathsf{updinfo}_{i_1}, \ldots, \mathsf{updinfo}_{i_k})$). We describe a construction for $\mathsf{ATUpdateBatch}$ from $\mathsf{ATUpdate}$ in Appendix A.

Verify: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, i, \Pi_i, \alpha_i)$ ATVerify is the verification algorithm that takes the verification key of the scheme vk, digest of the set collection digest_t and a particular set index i along with its authentication path (Π_i) and accumulation value (α_i) as input and returns accept if the α_i is indeed the accumulation value of the i^{th} set of the collection. It returns reject otherwise.

The following lemma summarizes its security and efficiency.

Lemma 4 *[PTT11] Under the q-SBDH assumption, for any adversarially chosen authentication path Π_i for y against an honestly generated digest,* digest*, such that* $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}, i, \Pi_i, y)$ *returns* accept*, it must be that y is the i^{th} element of the tree except for negligible probability. Algorithm* ATQuery *takes $O(m^{\varepsilon} \log m)$ and outputs a proof of $O(1)$ group elements and algorithm* ATVerify *has access complexity $O(1)$ and algorithm* ATUpdate *has access complexity $O(1)$.*

3 Zero-Knowledge Universal Accumulators

A cryptographic accumulator is a primitive that allows one to succinctly represent a set $\mathcal{X}$ of elements from a domain $\mathbb{X}$, by a single value acc from a (possibly different) domain $\mathbb{A}$, known as the *accumulation value*. Moreover, it provides a way to efficiently and succinctly prove that a candidate element x belongs to the set, (to a party that only holds acc) by computing a constant-size proof w, referred to as *witness*.

Accumulators were introduced by Benaloh and del Mare in [BdM94] as an alternative to cryptographic signatures for timestamping purposes. The authors provided the first construction in the RSA setting. Later, Baric and Pfitzmann [BP97] refined this construction by strengthening the security notion and Camenisch and Lysyanskaya [CL02] added the property of efficiently updating the accumulation value. More recently, Nguyen [Ngu05] and Camenisch et al. [CCs08] proposed constructions in the prime-order bilinear group setting. Li et al. [LLX07], and Damgård and Triandopoulos [DT08] extended the RSA and bilinear accumulator respectively, adding the property to prove non-membership as well. Accumulators that achieve this are referred to as *universal*.

Trusted/untrusted setup. All the above constructions –and the one we provide here– are in the trusted-setup model, i.e., the party that generates the scheme parameters originally, holds some trapdoor information that is not revealed to the adversary. E.g., for the RSA-based constructions, any adversary that knows the factorization of the modulo can trivially cheat. An alternative body of constructions aims to build trapdoorless accumulators (also referred to as *strong* accumulators) [Nyb96a,Nyb96b,San99,BLL00,CHKO08,Lip12], where the owner is entirely untrusted (effectively the owner and the server are the same entity). Unfortunately, the earlier of these works are quite inefficient for all practical purposes, while the more recent ones either yield witnesses that grow logarithmically with the size of $\mathcal{X}$ or rely on algebraic groups, the use of which is not yet common in cryptography. Alternatively, trapdoorless accumulators can be trivially constructed from zero-knowledge sets [MRK03], a much stronger primitive. While a scheme without the need for a trusted setup is clearly more attractive in terms of security, it is safe to say that we do not yet have a practical scheme with constant-size proofs, based on standard security assumptions.

In this work we aim to construct a dynamic, universal accumulator that also achieves a very strong privacy property. We formalize the required privacy property by defining *Zero-Knowledge Universal Accumulators* (ZKUA). Informally, the zero-knowledge property ensures that an adversarial party that sees the accumulation value as well as all membership and non-membership witnesses exchanged during the protocol execution learns nothing about the set, *not even its size*. This property even holds for adversarial clients that issue queries hoping to learn additional information about the set. Zero-knowledge guarantees that nothing can be learned from the protocol except for the answer to a query itself. In other words, explicitly querying for an element is the only way to learn whether an element appears in the set or not.

Moreover, privacy should also hold with respect to updates (insertions or deletions) in the set. That is, an adversary that observes the accumulation and witnesses both before and after an update should learn nothing about the occurred changes, other than the fact that an update occurred –not even whether it was an insertion or deletion.

In the rest of the section, we provide the definition for ZKUA and the corresponding security properties.

Note on definitional style. Traditionally in the literature, a cryptographic accumulator is defined as an ensemble of families of functions, referring to the function f that takes as input a set and a seed accumulation value and outputs a new accumulation value. At a high level, the requirements included the existence of an efficient algorithm to sample from the ensemble and that f can be efficiently evaluated. Moreover, f had to be quasi-commutative (i.e., after a series of invocations the final output is order-independent) which yields an efficient way to produce witnesses. Informally, a witness for $x \in \mathcal{X}$ consists of the evaluation of f on all the other elements of the set (corresponding to the accumulation value of $\mathcal{X} \setminus x$). Since f is quasi-commutative, a client can evaluate it using the witness and x and check that it is equal to the set accumulation value produced by the trusted owner. However, in this work we follow the more general definitional style introduced in [FN02], and more recently by [DHS15]. The cryptographic accumulator is described as a tuple of algorithms which is more meaningful in the context of the applications discussed here. The quasi-commutativity property (while satisfied by our construction) is purposely omitted, as there exist more recent constructions of cryptographic accumulators that do not satisfy it but have alternative ways for witness generation (e.g., hash-tree based construction).

A *universal accumulator* (UA) consists of four algorithms $(\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Witness}, \mathsf{Verify})$ and it supports queries of the following form: "is $x \in \mathcal{X}$?" for elements from a domain $\mathbb{X}$. The response is of the form (b, w) where b is a boolean value indicating if the element is in the set, i.e., $b = 1$ if $x \in \mathcal{X}$, $b = 0$ if $x \notin \mathcal{X}$ and w is the corresponding membership/non-membership witness for x.

Definition 1 ***(Universal Accumulator)*** *A universal accumulator is a tuple of four PPT algorithms* $(\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Witness}, \mathsf{Verify})$ *defined as follows:*

$(sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda)$

This probabilistic algorithm takes as input the security parameter and outputs a (public) verification key vk that will be used by the client to verify query responses and a secret key sk that is kept by the owner.

$(\mathsf{acc}, ek, \mathsf{aux}) \leftarrow \mathsf{Setup}(sk, \mathcal{X})$

This probabilistic algorithm is run by the owner. It takes as input the source set $\mathcal{X}$ and produces the accumulation value acc *that will be published to both server and client, and an evaluation key ek as well as auxiliary information* aux *that will be sent only to the server in order to facilitate proof construction.*

$(b, \mathsf{w}) \leftarrow \mathsf{Witness}(\mathsf{acc}, \mathcal{X}, x, ek, \mathsf{aux})$

This algorithm is run by the server. It takes as input the evaluation key and the accumulation value ek, acc *generated by the owner, the source set $\mathcal{X}$, a queried element x, as input. It outputs a boolean value b indicating whether the element is in the set and a witness* w *for the answer.*

$(\mathsf{accept/reject}) \leftarrow \mathsf{Verify}(\mathsf{acc}, x, b, \mathsf{w}, vk)$

This algorithm is run by the client. It takes as input the accumulation value acc *and the public key vk computed by the owner, a queried element x, a bit b, the witness* w *and it outputs* $\mathsf{accept/reject}$.

The above definition captures what is known in the literature as *static* accumulator, where Setup has to be executed again whenever change in $\mathcal{X}$ occurs. The following defines dynamic accumulators, by introducing an algorithm Update that takes the current accumulation value and the description of an update (e.g., "insert x" or "remove x") and outputs the appropriately modified accumulation value, together with a possibly modified ek' and aux'. Of course, this can always be achieved by re-running Setup, but we also require that the execution of Update is faster than that of Setup. Moreover, there must also exist a $\mathsf{WitUpdate}$ algorithm that takes the accumulation value and witness before an update, together with the new accumulation value after the update and produces a corresponding new witness.

Definition 2 ***(Dynamic Universal Accumulator)*** *A dynamic universal accumulator is a tuple of five PPT algorithms,* $\mathsf{DUA} = (\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Witness}, \mathsf{Verify}, \mathsf{Update})$ *defined as follows:*

$(\mathsf{GenKey}, \mathsf{Setup}, \mathsf{Witness}, \mathsf{Verify})$ *as in Definition 1.*

$(\mathsf{acc}', ek', \mathsf{aux}') \leftarrow \mathsf{Update}(\mathsf{acc}, \mathcal{X}, x, sk, \mathsf{aux}, \mathsf{upd})$

This algorithm takes as input the current set with its accumulation value and auxiliary information, as well as an element x to be inserted to $\mathcal{X}$ if $\mathsf{upd} = 1$ *or removed from $\mathcal{X}$ if* $\mathsf{upd} = 0$*. If* $\mathsf{upd} = 1$ *and $x \in \mathcal{X}$, (likewise if* $\mathsf{upd} = 1$ *and $x \notin \mathcal{X}$) the algorithm outputs $\perp$ and halts, indicating an invalid update. Otherwise, it outputs* $(\mathsf{acc}', ek', aux')$ *where* acc' *is the new accumulation value corresponding to set $\mathcal{X} \cup \{x\}$ or $\mathcal{X} \setminus \{x\}$ (to be published), ek' is the (possibly) modified evaluation key, and* aux' *is respective auxiliary information (both to be sent only to the server).*

$(\mathsf{upd}, \mathsf{w}') \leftarrow \mathsf{WitUpdate}(\mathsf{acc}, \mathsf{acc}', x, \mathsf{w}, y, ek', \mathsf{aux}, \mathsf{aux}', \mathsf{upd})$

This algorithm is to be run after an invocation of Update*. It take as input the old and the new accumulation values and auxiliary informations, the evaluation key ek' output by* Update*, as well as the element x that was inserted or removed from the set, according to the binary value* upd *(the same as in the execution of* Update*). It also takes a different element y and its existing witness* w *(that may be a membership or non-membership witness). It outputs a new witness* w' *for y, with respect to the new set $\mathcal{X}'$ as computed after performing the update. The output must be the* same *as the one computable by running* $\mathsf{Witness}(\mathsf{acc}', \mathcal{X}', y, ek', \mathsf{aux}')$.

In terms of efficiency, the focus is on the owner's ability to quickly compute the new accumulation value after a modification in the set (typically with most existing constructions, the presence of the trapdoor information makes this achievable with a constant number of group operations). Slightly more formally, the runtime of Update must be asymptotically smaller than that of Setup on the updated set. An even more attractive property is the ability to update existing witnesses efficiently (i.e., not recomputing them from scratch) after an update occurs, with $\mathsf{WitUpdate}$. As a last remark, we point out that the ability to do this for positive witnesses is inherently more important than that of non-membership witnesses. The former corresponds to the (polynomially many) values in the set whereas the latter will be exponentially many (or infinite). A server that wants to cache witness values and update them efficiently can thus benefit more from storing pre-computed positive witnesses than negative ones (that are less likely to be used again).

Early cryptographic accumulator constructions had deterministic algorithms; once the trapdoor was chosen each set had a uniquely defined accumulation value. As discussed at the end of this section, the consequent introduction of privacy requirements led to constructions where Setup and Update are randomized, which, in turn, introduced the natural distinction of accumulators into deterministic and randomized. In fact, any deterministic accumulator trivially fails to meet even weak privacy notions: An adversary that suspects that set $\mathcal{X}$ is the pre-image of a given accumulation value can test this himself. In order to achieve the wanted zero-knowledge property, our definition also refers to randomized schemes but, to simplify notation, we omit randomness from the input (unless otherwise noted).

Another important point is which parties can run each algorithm. The way we formulated our definition, Setup and Update require knowledge of sk to execute, $\mathsf{Witness}$ requires ek and Verify takes only vk. From a practical point of view, the owner is the party that is responsible for maintaining the accumulation value at all times (e.g., signing it and posting it to a public log); all changes in $\mathcal{X}$ should, in a sense, be validated by him first. On the other had, in most popular existing schemes (e.g., the RSA construction of [CL02] and the bilinear accumulator of [Ngu05]) setup and update processes can also be executed by the server (who does not know the trapdoor sk) and the only distinction is that the owner can achieve the same result much faster (utilizing sk). The same is true for our construction here, but in the following security definitions we adopt the more general framework where the adversary is given oracle access to these algorithms, that captures both cases.

3.1 Security Properties

The first property we require from a cryptographic accumulator is completeness, i.e., a witness output by any sequence of invocations of the scheme algorithms, for a valid statement (corresponding to the state of the set at the time of the witness generation) is verified correctly with all but negligible probability.

Definition 3 (Completeness) *Let $\mathcal{X}_i$ denote the set constructed after i invocations of the* Update *algorithm (starting from a set $\mathcal{X}_0$) and likewise for ek_i, aux_i. A dynamic universal accumulator is secure if, for all sets $\mathcal{X}_0$ where $|\mathcal{X}_0|$ and and $l \geq 0$ polynomial in λ and all $x_i \in \mathbb{X}$, for $0 = 1, \dots, l$, there exists a negligible function $\nu(\lambda)$ such that:*

$$\Pr\left[\begin{array}{c}(sk,vk) \leftarrow \mathsf{GenKey}(1^\lambda); (ek_0, \mathsf{acc}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(sk, \mathcal{X}_0);\\ \{(\mathsf{acc}_{i+1}, ek_{i+1}, \mathsf{aux}_{i+1}) \leftarrow \mathsf{Update}(\mathsf{acc}_i, \mathcal{X}_i, x_i, sk, \mathsf{aux}_i, \mathsf{upd}_i)\}_{0 \leq i \leq l}\\ (b, \mathsf{w}) \leftarrow \mathsf{Witness}(\mathsf{acc}_l, \mathcal{X}_l, x, ek_l, \mathsf{aux}_l) : \mathsf{Verify}(\mathsf{acc}_l, x, b, \mathsf{w}, vk) = \mathsf{accept}\end{array}\right] \geq 1 - \nu(\lambda)$$

where the probability is taken over the randomness of the algorithms.

In the above we purposely omitted the $\mathsf{WitUpdate}$ algorithm that was introduced purely for efficiency gains at the server. In fact, recall that we restricted it to return the exact same output as Update (run for the corresponding set and element) hence the value w in the above definition might as well have been computed during an earlier update and subsequently updated by (one or more) calls of $\mathsf{WitUpdate}$.

The second property is soundness which captures that fact that adversarial servers cannot provide accepting witnesses for incorrect statements. It is formulated as the inability of Adv to win a game during which he is given oracle access to all the algorithms of the scheme (except for those he can run on his own using ek, aux –see discussion on private versus public setup and updates above) and is required to output such a statement and a corresponding witness.

Definition 4 (Soundness) *For all PPT adversaries* Adv *running on input 1^λ and all l polynomial in λ, the probability of winning the following game, taken over the randomness of the algorithms and the coins of* Adv *is negligible:*

Setup *The challenger runs $(sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda)$ and forwards vk to* Adv. *The latter responds with a set $\mathcal{X}_0$. The challenger runs $(ek_0, \mathsf{acc}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(sk, \mathcal{X}_0)$ and sends the output to the adversary.*

Updates *The challenger initiates a list $\mathcal{L}$ and inserts the tuple $(acc_0, \mathcal{X}_0)$. Following this, the adversary issues update x_i and receives the output of $\mathsf{Update}(\mathsf{acc}_i, \mathcal{X}_i, x_i, sk, \mathsf{aux}_i, \mathsf{upd}_i)$ from the challenger, for $i = 0, \dots, l$. After each invocation of* Update, *if the output is not $\perp$, the challenger appends the returned $(\mathsf{acc}_{i+1}, \mathcal{X}_{i+1})$ to $\mathcal{L}$. Otherwise, he appends $(\mathsf{acc}_i, \mathcal{X}_i)$.*

Challenge *The adversary outputs an index j, and a triplet (x^*, b^*, w^*). Let $\mathcal{L}[j]$ be $(\mathsf{acc}_j, \mathcal{X}_j)$. The adversary wins the game if:*

$$\mathsf{Verify}(\mathsf{acc}_j, x^*, b^*, \mathsf{w}^*, vk) = \mathsf{accept} \wedge ((x^* \in \mathcal{X}_j \wedge b^* = 0) \vee (x^* \notin \mathcal{X}_j \wedge b^* = 1))$$

discussion on the winning conditions of the game is due here. This property (also referred to as collision-freeness) was introduced in this format in [LLX07] and was more recently adapted in [DHS15] with slight modifications. In particular, Adv outputs set $\mathcal{X}^*$ and accumulation value acc^* as well as the randomness used (possibly) to compute the latter (to cater for randomized accumulators). It is trivial to show that the two versions of the property are equivalent.

An alternative, more demanding, way to formulate the game is to require that the adversary wins if he outputs two accepting witnesses for the same element and with respect to the same accumulation value (without revealing the pre-image set): a membership and a non-membership one. This property, introduced in the context of accumulators in [BLL00], is known as *undeniability* and is the same as the privacy property of zero-knowledge sets. Recently, Derler et al. [DHS15] showed that undeniability is a stronger property than soundness. However, existing constructions for undeniable accumulators are in the trapdoor-less setting (with the limitations discussed above); since our construction is the trusted setup setting, we restrict our attention to soundness. This should come as no surprise, as undeniability allows an adversary to provide a candidate accumulation value, without explicitly giving a corresponding set. In a three-party setting (with trusted setup) the accumulation value is always maintained by the trusted owner; there is no need to question whether it was honestly computed (e.g., whether he knows a set pre-image or even whether there exists one) hence undeniability in this model is an "overkill" in terms of security (see also the related discussion in Section 4.3).

The last property is zero-knowledge. As we already explained, this notion captures that even a malicious client cannot learn anything about the set beyond what he has queried for. We formalize this in a way that is very similar to zero-knowledge sets (e.g, see the definition of [CHL+05]) appropriately extended to handle not only queries but also updates issued by the adversary. We require that there exists a simulator such that no adversarial client can distinguish whether he is interacting with the algorithms of the scheme or with the simulator that has no knowledge of the set or the element updates that occur, other than whether a queried element is in the set and whether requested updates are valid.

Definition 5 (Zero-Knowledge) *Let D be a binary function for checking the validity of queries and updates on a set. For queries, $D(\mathsf{query}, x, \mathcal{X})) = 1$ iff $x \in \mathcal{X}$. For updates $D(\mathsf{update}, x, c, \mathcal{X})) = 1$ iff $(c = 1 \wedge x \notin \mathcal{X})$ or $(c = 0 \wedge x \in \mathcal{X})$. Let $\mathsf{Real}_{\mathsf{Adv}}(1^\lambda), \mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$ be games between a challenger, an adversary Adv and a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$, defined as follows:*

$\mathsf{Real}_{\mathsf{Adv}}(1^\lambda)$**:**

Setup *The challenger runs $(sk, vk) \leftarrow \mathsf{GenKey}(1^\lambda)$ and forwards vk to Adv. The latter chooses a set $\mathcal{X}_0$ with $|\mathcal{X}_0| \in poly(k)$ and sends it to the challenger who in turn responds with acc_0 output by running the algorithm $\mathsf{Setup}(sk, \mathcal{X}_0)$. Finally, the challenger sets $(\mathcal{X}, \mathsf{acc}, \mathsf{aux}) \leftarrow (\mathcal{X}_0, \mathsf{acc}_0, \mathsf{aux}_0)$.*

Query *For $i = 1, \ldots, l$, where $l \in poly(\lambda)$, Adv outputs (op, x_i, c_i) where $\mathsf{op} \in \{\mathsf{query}, \mathsf{update}\}$ and $c_i \in \{0, 1\}$:*

If $\mathsf{op} = \mathsf{query}$**:** *The challenger runs $(b, \mathsf{w}_i) \leftarrow \mathsf{Witness}(\mathsf{acc}, \mathcal{X}, x_i, ek, \mathsf{aux})$ and returns the output to Adv.*

If $\mathsf{op} = \mathsf{update}$**:** *The challenger runs $\mathsf{Update}(\mathsf{acc}, \mathcal{X}, x_i, sk, \mathsf{aux}, c_i)$. If the output is not $\perp$ he updates the set accordingly to get $\mathcal{X}_i$, sets $(\mathcal{X}, \mathsf{acc}, ek, \mathsf{aux}) \leftarrow (\mathcal{X}_i, \mathsf{acc}_i, ek_i, \mathsf{aux}_i)$ and forwards acc to Adv. Else, he responds with $\perp$.*

Response *The adversary outputs a bit d.*

$\mathsf{Ideal}_{\mathsf{Adv}}(1^\lambda)$**:**

Setup *The simulator Sim_1, on input 1^λ, forwards vk to Adv. The adversary chooses a set $\mathcal{X}_0$ with $|\mathcal{X}_0| \in poly(\lambda)$. Sim_1 (without seeing $\mathcal{X}_0$) responds with acc_0 and maintains state $\mathsf{state}_{\mathcal{S}}$. Finally, let $(\mathcal{X}, \mathsf{acc}) \leftarrow (\mathcal{X}_0, \mathsf{acc}_0)$.*

Query *For $i = 1, \ldots, l$ Adv outputs (op, x_i, c_i) where $\mathsf{op} \in \{\mathsf{query}, \mathsf{update}\}$ and $c_i \in \{0, 1\}$:*

If $\mathsf{op} = \mathsf{query}$**:** *The simulator runs $(b, \mathsf{w}_i) \leftarrow \mathsf{Sim}_2(\mathsf{acc}, x_i, \mathsf{state}_{\mathcal{S}}, D(\mathsf{query}, x_i, \mathcal{X}))$ and returns the output to Adv.*

If $\mathsf{op} = \mathsf{update}$**:** *The simulator runs $\mathsf{Sim}_2(\mathsf{acc}, \mathsf{state}_{\mathcal{S}}, D(\mathsf{update}, x_i, c_i, \mathcal{X}))$. If the output of $D(\mathsf{update}, x_i, c_i, \mathcal{X})$ is 1, let $\mathcal{X} \leftarrow \mathcal{X}_i \cup x_i$ in the case $c_1 = 1$ and $\mathcal{X} \leftarrow \mathcal{X}_i \setminus x_i$ in the case $c_1 = 0$ –i.e., $\mathcal{X}$ is a placeholder variable for the latest set version at all times according to valid updates,* *that is however never observed by the simulator. The simulator responds to* Adv *with* acc'. *If the response* acc' *is not* $\perp$ *then* $\mathsf{acc} \leftarrow \mathsf{acc}'$.

Response *The adversary outputs a bit* d.

A dynamic universal accumulator is zero-knowledge if there exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that for all adversaries* Adv *there exists negligible function* ν *such that:*

$$|\Pr[\mathsf{Real}_{\mathsf{Adv}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathsf{Adv}}(1^\lambda) = 1]| \leq \nu(\lambda).$$

If the above probabilities are equivalent, then the accumulator is perfect zero-knowledge. If the inequality only holds for PPT Adv*, then the accumulator is computational zero-knowledge.*

Observe that, even though Adv may be unbounded (in the case of statistical or perfect zero-knowledge) the size of the set is always polynomial in the security parameter; in fact it is upper bounded by $O(|\mathcal{X}_0| + l)$. This ensures that we will have polynomial-time simulation, matching the real-world execution where all parties run in polynomial-time. Having computationally unbounded adversaries is still meaningful; such a party may, after having requested polynomially many updates, spend unlimited computational resources trying to distinguish the two settings.

While trying to provide a formal notion of privacy for cryptographic accumulators, the fact that the accumulation value computation must be randomized becomes evident. If Setup is a deterministic algorithm, then each set has a uniquely defined accumulation value (subject to particular sk) that can be easily reproduced by any adversary even with oracle access to the algorithm. This observation has already been made in [dMLPP12,dMPPS14,DHS15].

4 Relation to Other Definitions

In this section, we discuss the relation of zero-knowledge accumulators with the existing notion of indistinguishable accumulations as well as with other cryptographic primitives.

4.1 Zero-knowledge implies indistinguishability (for accumulators)

The notion of zero-knowledge defined here is a strengthening of the indistinguishability property introduced in [DHS15]. There the authors introduce a notion similar to ours that also requires the accumulation value produced by Setup to be randomized. If we restrict our attention to static accumulators, the effect of both notions is the same, i.e., the clients see a randomized accumulation value and corresponding "blinded" witnesses.

However, while the indistiguishability game entails updates, it inherently does not offer any privacy for the elements inserted to or removed from the set, as the Update algorithm is deterministic. At a high level, that notion only protects the original accumulated set and not subsequent updates. We believe this is an important omission for a meaningful privacy definition for accumulators, as highlighted by the following example. Consider, for example, a third-party adversary that observes the protocol's execution before and after an insertion (or deletion) update. If the adversary has reasons to suspect that the inserted (or deleted) value may be y, he can always test that. A very realistic example of this behavior is a setting where the accumulator is used to implement a revocation list. In that case an adversary may want to know if his fake certificate (value y in the above case) has been "caught" yet.

We provide the following result[6]:

[6] In [DHS15] the indistinguishability definition assumes that the adversary is also given access to the Setup algorithm arbitrarily many times. This makes sense in their model, since they explicitly require that Setup is randomized whereas Update is deter-

Theorem 1 *Every zero-knowledge dynamic universal accumulator is also indistinguishable under the definition of [DHS15], while the opposite is not always true.*

Proof. We first show that every scheme that is zero-knowledge is also indistinguishable. Then we show that the construction of [DHS15] is not zero-knowledge.

ZK $\Rightarrow$ IND: We prove this direction by contradiction. Assume there exists an accumulator that is zero-knowledge but not indistinguishable. Then, there exists PPT adversary Adv that wins the indistinguishability game. Adv gives two sets $\mathcal{X}_0, \mathcal{X}_1$ to a challenger who flips a coin b and provides oracle access to Adv for the algorithms with respect to $\mathcal{X}_b$. By assumption, Adv can output a bit b' correctly guessing b with non-negligible advantage ε over $1/2$. The (natural) constraint is that Adv cannot issue a query (or update request) that is trivially revealing the chosen set (e.g., if $x \in \mathcal{X}_0$ and $x \notin \mathcal{X}_1$, Adv is not allowed to query for x). We defer interested readers to [DHS15] for a formal definition of the indistinguishability game.

We will now construct PPT adversary Adv' that breaks the zero-knowledge property of the scheme as follows. Adv' on input $1^\lambda, vk$ runs Adv with the same input and receives sets $\mathcal{X}_0, \mathcal{X}_1$. He then forwards $\mathcal{X}_1$ as the challenge for the zero-knowledge game and receives accumulation value acc_0, which he forwards to Adv. Consequently, he responds to all messages of Adv (queries and updates) with calls to the zero-knowledge game interface and forwards all responses back to Adv. Finally, he outputs the output bit b' of Adv.

Firstly, observe that Adv' is clearly PPT, since Adv is PPT. Now let us argue about his success probability in distinguishing between real and ideal interaction. Observe that, if Adv' is interacting with the algorithms of the scheme (i.e., is playing the real game), the interface he is providing to Adv is a perfect simulation of the indistinguishability game for $b = 1$. On the other hand, if he is interacting with Sim, the view of the latter during this interaction is exactly the same independently of whether the set chosen by Adv' is $\mathcal{X}_0$ or $\mathcal{X}_1$. Hence, the view offered to Adv is the same in both cases, and therefore $\Pr[b' = 1] = \Pr[b' = 0] = 1/2$. Let E be the event that the Adv' is playing the real game (and likewise for the complement E^c). From the above analysis (recall that Adv' outputs the bit b' returned by Adv), it holds that $\Pr[b' = 1|E] > 1/2 + \varepsilon$ and $\Pr[b' = 1|E^c] = 1/2$. This implies that Adv' can distinguish between the two executions with non-negligible probability, breaking the zero-knowledge property of the scheme. The claim follows by contradiction.

IND $\not\Rightarrow$ ZK: In the construction of [DHS15], given the accumulation acc of set $\mathcal{X}$, the accumulation value acc' of set $\mathcal{X} \cup x$, for $x \notin \mathcal{X}$, is computed via a deterministic Update algorithm, that is executable in polynomial time even without access to the trapdoor (and similarly for a deletion).

Assume now an adversary Adv that simply observes query execution and the accumulation value throughout the protocol, and wants to deduce whether value x is added to the set after a given update. Adv proceeds as follows (assuming acc is the accumulation state pre-update and acc' the one published afterwards). After each update, run Update on input x, acc to receive acc''. Check whether $\mathsf{acc}'' = \mathsf{acc}'$. If so, deduce that x was inserted, otherwise not. This clearly violates zero-knowledge. Recall that in the ideal game the simulator does not get access to the inserted (or deleted) element. Since the update algorithm is deterministic, given acc and x there exists a unique output accumulation value, and the simulator thus has negligible probability to emulate the real interaction.

This concludes our proof. ∎

ministic. Here this requirement is redundant since both processes may be randomized; any setup response can be emulated by a series of update calls that shape the required set. To simplify the process, we assume that the indistinguishability adversary only makes Update and $\mathsf{Witness}$ calls. We stress that this is not a limitation of the reduction. We could alternatively have chosen to define our zero-knowledge game giving the adversary access to Setup and the result would still hold.

The indistinguishability property of [DHS15] is a strengthening of a notion introduced in [dMLPP12]. The latter was the first work to formally define a privacy property for cryptographic accumulators, however their definition had inherent problems, e.g., it was easy to prove that deterministic accumulators –that clearly were not private– satisfied it. Another technique for providing privacy to cryptographic accumulators was proposed earlier in [LLX07], without a formalization. The idea is to simply produce a randomized accumulation value for a set $\mathcal{X}$ by choosing at random an element x from the elements universe during Setup and outputting the accumulation of set $\mathcal{X} \cup \{x\}$. This generic mechanism will work for any static accumulator, but will also not protect updates. Moreover it weakens soundness as an adversary could potentially produce a membership witness for the element $x \notin \mathcal{X}$. Out approach does not suffer from such issues as there is no additional element accumulated and the randomness r used to blind the accumulation value during Setup is explicitly given to the server without compromising soundness.

Contrary to [DHS15], our zero-knowledge property provably protects not only the original set but also all subsequent updates. In fact, the only thing that an adversary (client or third-party) learns is that an update happened; not even whether the update was an insertion or deletion! Liskov in [Lis05] achieved a weaker notion of privacy for updates (called update transparency), in the model of zero-knowledge databases, that relies on assigning a pseudonym pattern $N(x)$ to each element x and leaks not only the fact that an update occurred but also an associated pseudonym. The author conjectures that the use of pseudonyms is unavoidable in order to achieve non-membership witnesses; here we show that this is not necessarily true, albeit in the more restricted three-party setting of cryptographic accumulators.

Finally, Theorem 1 implies that our construction from Section 5, is also the only known algebraic construction of a universal indistinguishable accumulator. The two schemes of [DHS15] are a black-box reduction from the stronger primitive of zero-knowledge sets, and a construction similar to ours that only offers membership witnesses.

4.2 Relation to other primitives

Next we turn our attention to how zero-knowledge accumulators compare against other similar cryptographic primitives. We present a mapping of the research literature for the construction of cryptographic proofs for set-membership and non-membership, which has attracted significant attention lately. This is far from a complete presentation of results in the area; we focus on the relation between those primitives that are most closely related to the problem, avoiding general approaches (e.g., general-purpose zero-knowledge protocols) or related models that address similar problems (such as group signatures, e.g., [ACJT00]).

The overall picture for the static case (i.e., without assuming changes in the set) can be seen in Figure 2. Arrows denote implication, e.g., an arrow from A to B translates to “B can be built in black-box manner from A”. Likewise, double-sided arrows denote equivalence of definitions, i.e., both can be constructed in a black-box manner from each other.

Zero-knowledge sets are a stronger primitive than accumulators; they satisfy the same soundness property with trapdoorless accumulators but they additionally offer privacy. Hence they are a starting point for our mapping, since they can be used to build the other primitives.

In Section 3, we already discussed trapdoorless (or strong) accumulators. If a scheme is a trapdoorless accumulator it is secure with an untrusted setup execution, therefore (and quite trivially) it is also secure with a trusted setup, hence it is a also an accumulator.

As a mental exercise, let us now try to define the privacy-preserving counterparts of strong accumulators, i.e., *trapdoorless zero-knowledge accumulators*. Quite informally, the completeness and zero-knowledge definitions remain the same but the soundness property is replaced by the, strictly stronger, property of undeniability (see, e.g., [Lip12] for a concrete definition), which is the same as the soundness property of zero-knowledge sets: By “merging” the existing soundness guarantee of trapdoorless accumulators with our zero-knowledge property (which, for the static case, is identical to that of zero-knowledge sets) we –quite

$$\begin{tikzcd}
 & \text{T-ACC} \arrow[r] & \text{ACC} & \\
\text{ZKS} \arrow[ur] \arrow[dr, leftrightarrow] & & & \\
 & \text{T-ZKACC} \arrow[uu] \arrow[r] & \text{ZKACC} \arrow[uu] \arrow[r] & \text{PSR}
\end{tikzcd}$$

Fig. 2: Relations among cryptographic primitives for proof of membership and non-membership (static case). ZKS: zero-knowledge sets, T-ACC: trapdoorless accumulators, ACC: accumulators, T-ZKACC: trapdoorless zero-knowledge accumulators, ZKACC: our zero-knowledge accumulators (circled), PSR: primary-secondary-resolver membership proof systems.

unsurprisingly– ended up with zero-knowledge sets. We stress that latter exist in the common reference string model (or the trusted parameters model) hence this *must* also be true for trapdoorless zero-knowledge accumulators (e.g., a trusted authority runs the key-generation algorithm and publishes the result as a common reference string). On the contrary, this is not necessary for trapdoorless accumulators (without privacy) since the security game there is one-sided; the client can perform key-generation himself. As a final note, we point out, that zero knowledge (trapdoorless) accumulators imply (trapdoorless) accumulators since the former satisfy a strict superset of the security properties of the latter.

This equivalence of zero-knowledge sets and trapdoorless zero-knowledge accumulators can be useful in two ways: (i) more efficient (e.g., with smaller proof sizes) zero-knowledge sets may be achievable with techniques borrowed from the accumulators literature, and (ii) an impossibility result in one of the two models is translatable to the other. This holds, for example, in the case of the batch-update impossibility for accumulators of [CH10] and the lower bound for online public-key operation of [GNP$^+$14]. We want to stress that our construction in Section 5 is not trapdoorless; to the best of our knowledge, the best known way to construct trapdoorless zero-knowledge accumulators is via a black-box reduction from zero-knowledge sets.

Another related primitive are primary-secondary-resolver membership proof systems (PSR) introduced in [NZ14]. Their privacy notion a relaxation defined as functional zero-knowledge, i.e., the simulator is allowed to learn some function of the set (typically its size). Also, the games in the PSR definition are non-adaptive in the following sense: Adv needs to declare its cheating set before he even receives the corresponding keys (ek, vk for soundness and only vk for zero-knowledge –using our terminology)[7]. For the above reasons, while it is trivial that zero-knowledge accumulators imply PSR (where the leaked function is void), the other direction is generally not true. We stress that the above distinction between adaptive and selective security does not hold in the dynamic setting. There an adversary may declare a cheating set originally, receive the keys, and then modify his choice via a series of update calls (see, however, our discussion for this setting in the next paragraph).

Our results here are complementary to the relations proven in [NZ14]. There, the authors prove that PSR systems exist, if and only if, one-way functions exist, which in turn implies that zero-knowledge sets cannot be built in a black-box manner from PSR.

[7] One could possibly modify the PSR model –and the security games– significantly to make them adaptive, by separating the key generation and setup algorithms. Indeed, to the best of our knowledge, the PSR construction of [GNP$^+$14] would probably satisfy such a modified definition, assuming it was instantiated with an adaptively-secure signature scheme and an adaptively-secure verifiable random function.

Dynamic setting. Once we move to the dynamic setting, where there exist efficient algorithms for modifications in the set, the relations are largely the same as in Figure 2, but some clarifications are in order.

Firstly, the only work addressing updatable zero-knowledge sets is [Lis05], where two notions of privacy are introduced: opacity and transparency. The relations between definitions hold with respect to opacity. We defer the interested reader to that work for an in-detail discussion of the two properties. Here, we will only mention that an (efficient) construction for opaque zero-knowledge sets remains an open problem. On the other hand, when restricted to the three-party model (i.e., with trusted setup), it can be shown that our construction from Section 5 (with minor modifications) satisfies the opacity property.

Regarding the relation between zero-knowledge accumulators and PSR, matters are also straight-forward as the latter are explicitly defined only for the static case. In [NZ14], the authors recommend the usage of techniques from certificate-revocation lists [NN00], as an additional external mechanism to accommodate updates. Contrary to this, our definitional approach is to make update-handling mechanisms explicitly part of the scheme. In this sense, zero-knowledge accumulators are a natural definitional extension of PSR in the dynamic setting. That said, we explicitly require that clients can at all times access the latest accumulation value, which would not be the case following the revocation scheme approach. We stress however that this does not necessitate authenticated channels between owner and clients; in practice it is achievable with a "timestamp-sign-and-publish" from the owner.

4.3 Relation to zero-knowledge authenticated data structures

Zero-knowledge accumulators can be seen as a straight-forward relaxation of zero-knowledge sets in the three-party model, i.e., in a honest-committer setting. The set is at all times maintained by a trusted party (the owner) that oversees insertions and deletions checking their validity. This is strongly reflected in the security property: Soundness in *zero-knowledge sets* (ZKS) does not require that the prover produces a commitment pre-image; indeed the prover may not even know such a set. On the other hand, soundness for zero-knowledge accumulators (trapdoorless) is defined as the inability of an adversary to produce a particular set and an element and a satisfying witness for a false statement. This stems from the fact that the trusted owner "authenticates" that the accumulation value known to clients (corresponding to the commitment in the case of ZKS) is indeed honestly computed, i.e., it corresponds to executing setup (and possibly update) on a known set. With that observation in mind, we can say that zero-knowledge accumulators are, in a sense, zero-knowledge authenticated sets. We note that a similar observation was made by [GOT14] who addressed the problem of order queries on a list in both two-party and three-party settings. Their three-party model (*privacy-preserving authenticated list* (PPAL)) is also a similar relaxation of their two-party model (*zero-knowledge list* (ZKL)) .

This in turn, highlights the relation of zero-knowledge accumulators with the framework of zero-knowledge authenticated data structures (ZK-ADS), recently introduced in [GGOT15].[8] ZK-ADS extend the well-known primitive of authenticated data structures (ADS) adding an additional zero-knowledge property. The setting is the standard three-party model but now the supported type may be any kind of data structure. The choice of data structure defines the kind of data stored and the type of supported queries. In [GGOT15], the authors provided constructions for various types of data structures, in particular for a zero-knowledge authenticated list (i.e., a data structure that supports "insert-after", "delete" operations, as well as "order" queries), a tree, and a partially-ordered set (poset) of bounded dimension. Consequently, a zero-knowledge accumulator (or zero-knowledge authenticate set, as discussed above) is a type of ZK-ADS where the data structure is a set of elements supporting –unordered– insertions and deletions, and membership/non-membership queries.

The above constructions are the only ZK-ADS instantiations in the literature so far. One natural way to extend zero-knowledge authenticated sets to accommodate more elaborate query types is by allowing

[8] Though [GGOT15] uses the term Privacy-Preserving Authenticated Data Structures, we use ZK-ADS to fit our notation.

for set-operations beyond (non-)membership. In particular, consider a data structure, called set collection, that consists of a collection of sets and accommodates operations among (a subset of) them. We stress that a construction that accommodates set unions, intersection and differences, allows for a complete set-operation algebra (building any possible "circuit" of set-operations[9]). In Section 6 we provide a definition of *zero-knowledge authenticated set collection*, in the style of [GGOT15], and in Section 7 we provide the corresponding construction (which naturally uses our zero-knowledge authenticated set construction from Section 5 as a building block).

5 A Zero-knowledge accumulator from the q-Strong Bilinear Diffie-Hellman Assumption

In this section we present our construction for a zero-knowledge dynamic universal accumulator. It builds upon the bilinear accumulator of Nguyen [Ngu05], adopting some of the techniques of [DHS15] that we further expand to achieve zero-knowledge. It supports sets with elements from $\mathbb{Z}_p \setminus \{s\}$ where p is prime and $p \in O(2^\lambda)$ and s is the scheme trapdoor. Note that, the fact that the elements must be of $\log p$ bits each, is not a strong limitation of the scheme; one can always apply a collision-resistant hash function that maps arbitrarily long strings to $\mathbb{Z}_p$. The description of the scheme can be seen in Figure 3.

Observe that the key vk published from the key-generating algorithm, reveals nothing for the set itself. The accumulation value produced by Setup is the standard bilinear accumulation value of [Ngu05] which is now blinded by a random value r, also revealed to the server. Witness generation for both cases utilizes this randomness r. For membership queries, the process is the same as in [Ngu05,DT08] with one additional exponentiation with r for privacy purposes.

The major deviation occurs in the non-membership case. As previously discussed, there are existing works [DT08,ATSM09] that enhance the bilinear accumulator to provide non-membership witnesses. Their technique is a complement of the one used for the membership case. At a high level, it entails proving that the degree-one polynomial $x+z$ does not divide $\mathsf{Ch}_{\mathcal{X}}[z]$, by revealing the scalar (i.e., zero-degree polynomial) remainder of their long division. Unfortunately, using this approach here entirely breaks the zero-knowledge property: It essentially reveals r (multiplied by an easily computable query-specific value) to any client. Instead, we adopt an entirely different approach. Our scheme uses the set-disjointness test, first proposed by Papamanthou et al. [PTT11], in order to prove non-membership of a queried element x. In order to prove that $x \notin \mathcal{X}$, the server proves the equivalent statement $\mathcal{X} \cap \{x\} = \emptyset$. The different nature of the proved statement allows us to use fresh query-specific randomness γ together with r to prove non-membership in zero-knowledge.

Verification is also different in the two cases, but always very efficient. Finally, the way updates are handled is especially important as it is another strong point of divergence from previous schemes that seek to provide privacy. After each update, a fresh randomness r' is used to blind the new accumulation value. Indeed this re-randomization technique that perfectly hides the nature of the change in $\mathcal{X}$ is what allows us to achieve our strong notion of zero-knowledge. Observe that, at all times, the owner maintains a variable N which is the maximum set-cardinality observed up to that point (through the original setup and subsequent insertions). When an insertion occurs that increases N (by one), then the owner provides to the server one additional ek component, that is necessary to the latter for subsequent witness generation. This is a slight deviation from our notation in Section 3 where the new key produced from Update replaces the previous ek. Instead the new evaluation key must be set to $ek \cup ek'$. This difference has no meaningful impact in the security of our scheme; we could always have Update output the entire old key together with the additional element. From an efficiency perspective though, that overly naive approach would require Update to run in time linear to N.

[9] In the computationally-bounded setting, a negation operation is infeasible unless the element domain is of polynomial size in the security parameter. In that case, a negation can be instantiated as a set difference from the set that contains the entire domain.

Notation: The notation $q[z]$ denotes polynomial q over undefined variable z and $q(s)$ is the evaluation of the polynomial at point s. All arithmetic operations are performed $\bmod p$. N is a variable maintained by the owner.

Key Generation $(sk, vk) \leftarrow \mathsf{GenKey}(1^{\lambda})$

Run $\mathsf{GenParams}(1^k)$ to receive bilinear parameters $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g)$. Choose $s \xleftarrow{\$} \mathbb{Z}_p^*$. Return $sk = s$ and $vk = (g^s, pub)$.

Setup $(\mathsf{acc}, ek, \mathsf{aux}) \leftarrow \mathsf{Setup}(sk, \mathcal{X})$

Choose $r \xleftarrow{\$} \mathbb{Z}_p^*$. Set value $N = |\mathcal{X}|$. Return $\mathsf{acc} = g^{r \cdot \mathsf{Ch}_{\mathcal{X}}(s)}$, $ek = (g, g^s, g^{s^2}, \ldots, g^{s^N})$ and $\mathsf{aux} = (r, N)$.

Witness Generation $(b, \mathsf{w}) \leftarrow \mathsf{Witness}(\mathsf{acc}, \mathcal{X}, x, ek, \mathsf{aux})$

If $x \in \mathcal{X}$ compute $\mathsf{w} = (\mathsf{acc})^{\frac{1}{s+x}} = g^{r \cdot \mathsf{Ch}_{\mathcal{X} \setminus \{x\}}(s)}$ and return $(1, \mathsf{w})$.

Else, proceed as follows:

- Using the Extended Euclidean algorithm, compute polynomials $q_1[z], q_2[z]$ such that $q_1[z]\mathsf{Ch}_{\mathcal{X}}[z] + q_2[z]\mathsf{Ch}_{\{x\}}[z] = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q_1'[z] = q_1[z] + \gamma \cdot \mathsf{Ch}_{\{x\}}[z]$ and $q_2'[z] = q_2[z] - \gamma \cdot \mathsf{Ch}_{\mathcal{X}}[z]$.
- Set $W_1 := g^{q_1'(s) r^{-1}}, W_2 = g^{q_2'(s)}$ and $\mathsf{w} := (W_1, W_2)$. Return $(0, \mathsf{w})$.

Verification $(\mathsf{ACCEPT}/\mathsf{REJECT}) \leftarrow \mathsf{Verify}(\mathsf{acc}, x, b, \mathsf{w}, vk)$

If $b = 1$ return ACCEPT if $e(\mathsf{acc}, g) = e(\mathsf{w}, g^x \cdot g^s)$, REJECT otherwise. If $b = 0$ do the following:

- Parse w as (W_1, W_2).
- Return ACCEPT if $e(W_1, \mathsf{acc})e(W_2, g^x \cdot g^s) = e(g, g)$, REJECT otherwise.

Update $(\mathsf{acc}', ek', \mathsf{aux}') \leftarrow \mathsf{Update}(\mathsf{acc}, \mathcal{X}, x, sk, \mathsf{aux}, \mathsf{upd})$

Parse aux as (r, N). If $(\mathsf{upd} = 1 \wedge x \in \mathcal{X})$ or $(\mathsf{upd} = 0 \wedge x \notin \mathcal{X})$ output $\perp$ and halt. Choose $r' \xleftarrow{\$} \mathbb{Z}_p^*$. If $\mathsf{upd} = 1$:

- Compute $\mathsf{acc}' = \mathsf{acc}^{(s+x)r'}$.
- If $|\mathcal{X}| + 1 > N$, set $N = |\mathcal{X}| + 1$ and compute $\mathsf{ek}' = g^{s^N}$.

Else, compute $\mathsf{acc}' = \mathsf{acc}^{\frac{r'}{s+x}}$ and $ek' = \emptyset$. In both cases, set $\mathsf{aux}' := (r \cdot r', N)$ and return $(\mathsf{acc}', ek', \mathsf{aux}')$.

Witness Update $(\mathsf{upd}, \mathsf{w}') \leftarrow \mathsf{WitUpdate}(\mathsf{acc}, \mathsf{acc}', x, \mathsf{w}, y, ek', \mathsf{aux}, \mathsf{aux}', \mathsf{upd})$

Parse $\mathsf{aux}, \mathsf{aux}'$ to get r, r'.

- If w is a membership witness:

  If $\mathsf{upd} = 1$ output $(1, \mathsf{w}' = (\mathsf{acc} \cdot \mathsf{w}^{x-y})^{r'})$. Else, output $(0, \mathsf{w}' = (\mathsf{acc}'^{-1} \cdot \mathsf{w})^{\frac{r'}{(x-y)}})$.
- If w is a non-membership witness:

  Let $\mathcal{X}'$ be the set produced after the execution of Update for element x (i.e., the currently accumulated set). Run $\mathsf{Witness}(\mathsf{acc}', \mathcal{X}', y, ek', \mathsf{aux}')$ and return its output.

Fig. 3: Zero-knowledge Dynamic Universal Accumulator Construction

Regarding witness updates, observe that for the (more meaningful, as discussed in Section 3) case of membership witnesses there indeed exists a fast method. On the other hand, for non-membership witness updates, our scheme resorts to re-computation from scratch.

Efficiency. We discuss the asymptotic complexity of the algorithms of our construction. Let n be the cardinality of the set at any given time. GenKey runs in time $poly(\lambda)$. The algorithm Setup runs has access complexity $O(n)$ since a degree n polynomial can be evaluated in that many operations, and ek takes n consecutive exponentiations. From Lemma 1, computing the characteristic polynomial of a set of size n takes $O(n \log n)$, hence this is the overall complexity for membership witness generation. Computing a non-membership witness takes $O(n \log^2 n \log\log n)$, due to the complexity of the Extended Euclidean algorithm execution. The witnesses consist of $O(1)$ elements from $\mathbb{G}$ and verification requires can be done with $O(1)$ operations. Finally, Update has access complexity $O(1)$, membership witness updates can also be achieved in $O(1)$, and non-membership witness updates take as long as fresh witness generations, i.e., $O(n \log^2 n \log\log n)$.

One alternative way to run the scheme is to have the owner pre-compute all the membership witnesses at no additional asymptotic overhead during Setup. In that case, the server can just use the cached member witnesses and serve membership queries with $O(1)$ lookups. Whenever an update occurs, the server can take $O(n)$ independent witness updates to update all his cached positive witnesses. However, he would still need to compute the non-membership witnesses on the fly; pre-computation is impossible since there are exponentially many non-members of $\mathcal{X}$ and updating any cached ones in not faster than generating them.

In terms of storage requirements, the client only needs to store vk and acc, i.e., a constant number of bilinear group elements. The owner stores the set $\mathcal{X}$, hence $O(n)$ group complexity is trivially necessary. From the description of $\mathsf{Setup}, \mathsf{Update}$, $|ek| = n$ at all times. Hence, the server's storage is also $O(n)$.

Overall, a comparison with the bilinear accumulator of [Ngu05] which achieves the exact same soundness property (under the same assumption), reveals that zero-knowledge is achieved by our construction with no asymptotic overhead at all for membership queries and a very small additional cost for non-membership queries. The above shows that the very strong privacy notion of zero-knowledge is achievable with minimal additional overhead.

Proving (non-)membership in batch. Another important property of our construction is that it allows the server to efficiently prove statements in batch. Assume a client holds an entire set $\mathcal{Y} = (y_1, \ldots, y_m)$ and wants to issue a query for each y_i. One way to achieve this would be to provide a separate membership/non-membership witness separately. This approach yields a proof that consists of $O(m)$ group elements.

However, with our construction the server can produce a single membership witness for all $y_i \in \mathcal{X}$ and a single non-membership witness for those $\notin \mathcal{X}$. The detailed construction for this case (as well as its practical benefits) is presented in Section 7. We can now present our main result:

Theorem 1. *The algorithms* $\{\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Witness}, \mathsf{Verify}, \mathsf{Update}, \mathsf{WitUpdate}\}$ *constitute a zero-knowledge dynamic universal accumulator that: (i) has perfect completeness, (ii) is perfect zero-knowledge, (iii) is secure under the N-SBDH assumption, where N is the maximum set-size observed during the soundness game. Let n be the cardinality of the set. Then, the runtime of* GenKey *is $O(poly(\lambda))$ where λ is the security parameter, the access complexity of* Setup *is $O(n)$, that of* $\mathsf{Witness}$ *is $O(n \log n)$ for membership witnesses and $O(n \log^2 n \log\log n)$ for non-membership witnesses, that of* Verify *is $O(1)$, that of* Update *is $O(1)$, and that of* $\mathsf{WitUpdate}$ *is $O(1)$ for membership witnesses and $O(n \log^2 n \log\log n)$ for non-membership witnesses. Finally, witnesses consist of $O(1)$ bilinear group elements.*

Completeness follows by close inspection of the algorithms' execution. We proceed to prove soundness and zero-knowledge.

Proof of Soundness. Assume for contradiction that there exists PPT adversary Adv that on input 1^λ breaks the soundness of our scheme with non-negligible probability. We will construct a PPT adversary Adv' that breaks the N-SBDH assumption. Adv' runs as follows:

1. On input $(pub, (g^s, \ldots, g^{s^N}))$, run Adv on input $(g^s, pub, 1^\lambda)$.
2. Upon receiving set $\mathcal{X}_0$, choose $r_0 \xleftarrow{\$} \mathbb{Z}_p^*$. Use r_0 and $(g^s, \ldots, g^{s^N})$ to compute $\mathsf{acc}_0 = g^{r_0 \cdot \mathsf{Ch}_{\mathcal{X}_0}(s)} = g^{(\mathsf{Ch}_{\mathcal{X}_0}(s))^{r_0}}$ and respond with $(ek_0 = (g, g^s, \ldots, g^{s^{|\mathcal{X}_0|}}), \mathsf{acc}_0, r_0)$. Initiate list $\mathcal{L}$ and insert triplet $(\mathsf{acc}_0, \mathcal{X}_0, r_0)$ as $\mathcal{L}[0]$ (i.e., the first element of the list). The notation $\mathcal{L}[i]_j$ denotes the first part of the i-th element of the list (e.g., $\mathcal{L}[0]_0 = \mathsf{acc}_0$). Also set $n = |\mathcal{X}_0|$.
3. Initiate update counter $i = 0$. While $i \leq l$ proceed as follows. Upon receiving update upd_i, x_i, check whether this is a valid update for $\mathcal{X}_i = \mathcal{L}[i]_1$. If it is not, respond with $\perp$ and re-append $acc_i = \mathcal{L}[i]_0, \mathcal{X}_i, r_i$ to $\mathcal{L}$. Otherwise, pick $r' \xleftarrow{\$} \mathbb{Z}_p^*$ and set $r_{i+1} = r_i \cdot r'$. Update $\mathcal{X}_i$ according to upd_i, x_i to get $\mathcal{X}_{i+1}$. If $|\mathcal{X}_{i+1}| > n$, set $n = |\mathcal{X}_{i+1}|$ and $ek_{i+1} = g^n$. Else, $ek_{i+1} = \emptyset$. Use r_{i+1} and $(g^s, \ldots, g^{s^N})$ to compute $\mathsf{acc}_{i+1} = g^{r_{i+1} \cdot \mathsf{Ch}_{\mathcal{X}_{i+1}}(s)} = g^{(\mathsf{Ch}_{\mathcal{X}_{i+1}}(s))^{r_{i+1}}}$ and respond with $(ek_{i+1}, \mathsf{acc}_{i+1}, r_{i+1})$. Append triplet $(\mathsf{acc}_{i+1}, \mathcal{X}_{i+1}, r_{i+1})$ to $\mathcal{L}$. In both cases, increase i by 1.
4. Upon receiving challenge index j and challenge triplet (x^*, b^*, w^*) proceed as follows:
   - If $b^* = 1$, then $x^* \notin \mathcal{X}_j$ yet $\mathsf{Verify}(\mathsf{acc}_j, x^*, 1, vk)$ accepts. Compute polynomial $q[z]$ and scalar c such that $\mathsf{Ch}_{\mathcal{X}_j}[z] = (x^* + z)q[z] + c$. Output $[x^*, (e(\mathsf{w}^*, g^{(x^*+s)})^{r_j^{-1}} e(g, g^{-q(s)}))^{c^{-1}}]$.

– If $b^* = 0$, then $x^* \in \mathcal{X}_j$ yet $\mathsf{Verify}(\mathsf{acc}_j, x^*, 0, vk)$ accepts. Parse w^* as (W_1^*, W_2^*). Compute polynomial $q[z]$ such that $\mathsf{Ch}_{\mathcal{X}_j}[z] = (x^* + z)q[z]$. Output $[x^*, (e(W_1^*, g^{r_j \cdot q(s)})e(W_2^*, g))]$.

First of all observe that Adv' perfectly emulates the challenger for the DUA security game to Adv. This holds since all accumulation values and witness are computable without access to trapdoor sk in polynomial time. All the necessary polynomial arithmetic can be also run efficiently hence Adv' is PPT.

Regarding its success probability, we argue for the two cases separately as follows:

$\mathbf{b}^* = \mathbf{1}$ Since $x^* \notin \mathcal{X}_j$, it follows that $(x^* + z) \not| \mathsf{Ch}_{\mathcal{X}_j}[z]$ which guarantees the existence of $q[z], c$. Also observe that c is a scalar (zero-degree polynomial) since it is the remainder of the polynomial division and it must have degree less than that of $(x^* + z)$. Since verify accepts we can write:

$$\begin{aligned} e(\mathsf{w}^*, g^{x^*} \cdot g^s) = e(\mathsf{w}^*, g)^{(x^*+s)} &= e(\mathsf{acc}_j, g) \\ &= e(g^{r_j \cdot \mathsf{Ch}_{\mathcal{X}_j}(s)}, g) \\ &= e(g,g)^{r_j((x^*+s)q(s)+c)}, \end{aligned}$$

from which it follows that:

$$\begin{aligned} e(\mathsf{w}^*, g)^{r^{-1}(x^*+s)} &= e(g,g)^{(x^*+s)q(s)+c} \\ e(\mathsf{w}^*, g)^{r^{-1}} &= e(g,g)^{q(s)+c/(x^*+s)} \\ e(\mathsf{w}^*, g)^{r^{-1}} e(g,g)^{-q(s)} &= e(g,g)^{c/(x^*+s)} \\ [e(\mathsf{w}^*, g)^{r^{-1}} e(g,g)^{-q(s)}]^{c^{-1}} &= e(g,g)^{1/(x^*+s)}. \end{aligned}$$

$\mathbf{b}^* = \mathbf{0}$ Since $x^* \in \mathcal{X}_j$, it follows that $(x^* + z) | \mathsf{Ch}_{\mathcal{X}_j}[z]$ which guarantees the existence of $q[z]$. Since verify accepts we can write:

$$\begin{aligned} e(W_1^*, \mathsf{acc}_j)e(W_2^*, g^{x^*} \cdot g^s) &= e(g,g) \\ e(W_1^*, g^{r_j \cdot \mathsf{Ch}_{\mathcal{X}_j}(s)})e(W_2^*, g^{(x^*+s)}) &= e(g,g) \\ e(W_1^*, g^{r_j (x^*+s) q(s)})e(W_2^*, g^{(x^*+s)}) &= e(g,g) \\ [e(W_1^*, g^{r_j \cdot q(s)})e(W_2^*, g)]^{(x^*+s)} &= e(g,g) \\ [e(W_1^*, g^{r_j \cdot q(s)})e(W_2^*, g)] &= e(g,g)^{1/(x^*+s)} \end{aligned}$$

Observe that in both cases the left hand of the above equations is efficiently computable with access to $pub, (g^s, \ldots, g^{s^N}), r_j, \mathcal{X}_j, x^*, \mathsf{w}^*$. Hence, whenever Adv' succeeds in breaking the soundness of our scheme, Adv' outputs a pair breaking the N-SBDH assumption. By assumption the latter can happen only with negligible probability, and our claim that our scheme has soundness follows by contradiction. ∎

Proof of Zero-Knowledge. We define simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ as follows. At all times, we assume $\mathsf{state}_\mathcal{S}$ contains all variables seen by the simulator this far.

– Sim_1 runs $\mathsf{GenParams}$ to receive pub. He then picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and sends g, g^s, pub to Adv. After Adv has output his set choice $\mathcal{X}$, Sim_1 picks $r \xleftarrow{\$} \mathbb{Z}_p^*$ and responds with $\mathsf{acc} = g^r$. Finally, he stores r initiates empty list $\mathcal{C}$.
– For $i = 1, \ldots, l$ upon input (op, x_i, c_i):
  - If $\mathsf{op} = \mathsf{query}$, the simulator checks if $x_i \in \mathcal{C}$. If not, then if $D(\mathsf{query}, x_i, \mathcal{X}) = 1$, he computes $\kappa = r \cdot (x_i + s)^{-1}$ and responds with $(b = 1, \mathsf{w} = g^\kappa)$. Else, if $D(\mathsf{query}, x_i, c_i, \mathcal{X}) = 1$ he computes q_1, q_2 such that $q_1 \cdot r + q_2 \cdot (x_i + s) = 1$, picks $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and responds with $(b = 1, \mathsf{w} = (W_1 = g^{q_1 + \gamma(x_i+s)}, W_2 = g^{q_2 - \gamma r}))$. In both cases, the simulator appends (x_i, b, w) to $\mathcal{C}$. Finally, if $x_i \in \mathcal{C}$ he responds with the corresponding entries b, w.

- If $\mathsf{op} = \mathsf{update}$ then the simulator proceeds as follows. If $D(\mathsf{update}, x_i, c_i, \mathcal{X}) = 0$ then he responds with $\perp$. Else, he picks $r' \xleftarrow{\$} \mathbb{Z}_p^*$ and responds with $\mathsf{acc} = g^{r'}$. Finally he sets $r \leftarrow r$ and $\mathcal{C} \leftarrow \emptyset$.

The simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ produces a view that is identically distributed to that produced by the challenger during $\mathsf{Real}_{\mathsf{Adv}}$. Observe that random values r are chosen independently after each update (and initial setup) in both cases. Once s, r are fixed then for any possible choice of $\mathcal{X}$ there exists unique $r^* \in \mathbb{Z}_p^*$ such that $g^r = g^{r^* \cdot \mathsf{Ch}_{\mathcal{X}}(s)}$. It follows that the accumulation values in $\mathsf{Real}_{\mathsf{Adv}}$ are indistinguishable from the (truly random) ones produced by Sim. For fixed s, r, given a set-element combination $(\mathcal{X}, x_i)$ with $x_i \in \mathcal{X}$, in each game there exists a unique membership witness w that satisfies the verifying equation. For negative witness $\mathsf{w} = (W_1, W_2)$, given a set-element combination $(\mathcal{X}, x_i)$ with $x_i \notin \mathcal{X}$, for each possible independently chosen value of γ, in both games there exists only one distinct corresponding pair W_1, W_2 that satisfies the verifying equation.

It follows that the probabilities in Definition 5 are equivalent and our scheme is perfect zero-knowledge.
■

6 Zero-Knowledge Authenticated Set Collection (ZKASC)

Zero-knowledge accumulators presented so far provide a succinct representation for maintaining a dynamic set and replying (non-)member queries in zero-knowledge. As we described in Section 4.3, zero-knowledge accumulators (without trapdoor) can also be viewed as zero-knowledge authenticated sets (ZK-AS) where authenticated zero-knowledge membership/non-membership queries are supported on an outsourced set. In this section, we generalize this problem to a collection of sets and study verification of outsourced set algebra operations in zero-knowledge, which we refer to as *zero-knowledge authenticated set collection* (ZKASC). In particular, we consider a dynamic collection $\mathbb{S}$ of m sets $\mathcal{X}_1, \ldots, \mathcal{X}_m$ that is remotely stored on an untrusted server. We then develop mechanisms to answer primitive queries on these sets (is-subset, intersection, union and set difference) such that the answers to these queries can be verified publicly and in zero-knowledge. That is, the proofs of the queries should reveal nothing beyond the query answer. In addition, we require the verification of any set operation to be operation-sensitive, i.e., the required complexity depends only on the (description and outcome of the) operation, and not on the sizes of the involved sets.

The *sets collection* data structure $\mathbb{S}$, consists of m sets, denoted with $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, each containing elements from a universe $\mathbb{X}$. A set does not contain duplicate elements, however an element can appear in more than one set. The *abstract data type* for set collection is defined as $\mathbb{S}$ with two types of operations defined on it: immutable operations $Q()$ and mutable operations $U()$. $Q(\mathbb{S}, q)$ takes a set algebra query q as input and returns an answer and a proof and it does not alter $\mathbb{S}$. The queries are defined with respect to the indices of a collection of sets $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$. $U(\mathbb{S}, u)$ takes as input an update request and changes $\mathbb{S}$ accordingly. It then outputs the modified set collection $\mathbb{S}'$. An update $u = (x, \mathsf{upd}, i)$ is either an insertion (if $\mathsf{upd} = 1$) of an element x into a set $\mathcal{X}_i$ or a deletion (if $\mathsf{upd} = 0$) of x from $\mathcal{X}_i$. The following queries are supported on $\mathbb{S}$:

Subset The query q takes a set of elements Δ and a set index i as input and returns answer where $\mathsf{answer} = 1$ if $\Delta \subseteq \mathcal{X}_i \in \mathbb{S}$, and $\mathsf{answer} = 0$, otherwise.

Set Difference The query q takes two set indices i_1 and i_2 and returns $\mathsf{answer} = \mathcal{X}_{i_1} \setminus \mathcal{X}_{i_2}$.

Intersection The query q takes a set of indices $(i_1, \ldots, i_k)$ as input and returns $\mathsf{answer} = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2} \cap \ldots \cap \mathcal{X}_{i_k}$.

Union The query q takes a set of indices $(i_1, \ldots, i_k)$ as input and returns $\mathsf{answer} = \mathcal{X}_{i_1} \cup \mathcal{X}_{i_2} \cup \ldots \cup \mathcal{X}_{i_k}$.

6.1 Model

ZKASC can be seen as the traditional *authenticated data structure* (ADS) model with the added requirement of privacy (zero-knowledge). Indeed, ZKASC follows the model of zero-knowledge authenticated data structure [GGOT15] instantiated for a set collection and set algebra queries. In particular, ZKASC is a tuple of six probabilistic polynomial time algorithms $\mathsf{ZKASC} = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Update}, \mathsf{UpdateServer}, \mathsf{Query}, \mathsf{Verify})$. We note that zero-knowledge authenticated set also follows the zero-knowledge authenticated data structure [GGOT15] model where the data structure consists of a single set and the supported queries are membership and non-membership queries.

We first describe how the algorithms of ZKASC are used between the three parties of our model and then give their API. The owner uses KeyGen to generate the necessary keys. He then runs Setup to prepare $\mathbb{S}_0$ for outsourcing it to the server and to compute digest for the client and necessary auxiliary information for the server. The owner can update his set collection and make corresponding changes to digest using Update. Since the set collection and the information of the server need to be updated on the server as well, the owner generates an update string that is enough for the server to make the update herself using UpdateServer. The client can query the data structure by sending queries to the server. For a query, the server runs Query and generates answer. Using the auxiliary information, she also prepares a proof of the answer. The client then uses Verify to verify the query answer against proof and the digest he has received from the owner after the last update.

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ where 1^λ is the security parameter. KeyGen outputs a secret key (for the owner) and the corresponding verification key vk.

$(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk}, \mathsf{vk}, \mathbb{S}_0)$ where $\mathbb{S}_0$ is the initial set collection and sk, vk are the keys generated by KeyGen. Setup computes the authentication information $\mathsf{auth}(\mathbb{S}_0)$ for $\mathbb{S}_0$, a short digest digest_0 for $\mathbb{S}_0$, an evaluation key ek_0 and auxiliary information aux_0. digest_0 is public, while $\mathsf{auth}(\mathbb{S}_0)$, ek_0 and aux_0 are sent to the server. These units lets the server compute proofs of query answer of the clients.

$(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t) \leftarrow \mathsf{Update}(\mathsf{sk}, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, \mathsf{ek}_t, \mathsf{SID}_t, u_t)$ where $u_t = (x, \mathsf{upd}, i)$ is an update operation to be performed on $\mathbb{S}_t$. SID_t is set to the output of a function f on the queries invoked since the last update (Setup for the 0^{th} update). Update returns the updated set collection, $\mathbb{S}_{t+1} = U(\mathbb{S}_t, u)$, the corresponding the authentication information $\mathsf{auth}(\mathbb{S}_{t+1})$, the evaluation key ek_{t+1} and auxiliary information aux_{t+1}, the updated public digest digest_{t+1}, and an update string $\mathsf{updinfo}_t$ that is used by the server to update her information. Note that the evaluation key is a part of the information the server requires to compute proofs of answers to the client queries. The secret key and the verification key do not change throughout the scheme.

$(\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{ek}_t, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, u_t, \mathsf{updinfo}_t)$ where $\mathsf{updinfo}_t$ is used to update $\mathsf{auth}(\mathbb{S}_t)$, digest_t, aux_t and ek_t to $\mathsf{auth}(\mathbb{S}_{t+1})$, digest_{t+1}, aux_{t+1} and ek_{t+1} respectively. u_t is used to update $\mathbb{S}_t$ to $\mathbb{S}_{t+1}$.

$(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, q)$ where q depends on the exact set algebra operation. In particular, for subset query $q = \Delta, i$ where Δ denotes a set of elements and i denotes the set index of $\mathbb{S}_t$. For intersection and union queries $q = i_1, \ldots, i_k$ where $i_1, \ldots, i_k$ are set indices of $\mathbb{S}_t$ and for set difference $q = i_1, i_2$. where i_1, i_2 are indices of $\mathbb{S}_t$. The algorithm outputs the query answer answer, is its proof proof.

$(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{Verify}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$ where input arguments are as defined above. The output is accept if $\mathsf{answer} = Q(\mathbb{S}_t, q)$, and reject, otherwise.

We leave function f to be defined by a particular instantiation. An example could be making f return the cardinality of its input. Once defined, f remains fixed for the instantiation. Since the function is public, anybody, who has access to the (authentic) queries since the last update, can compute it.

6.2 Security Properties

A zero-knowledge authenticated data structure [GGOT15] has three security properties: completeness, soundness and zero-knowledge. Our ZKASC adapts these properties as follows.

Completeness dictates that if all three parties are honest, then for a set collection, the client will always accept an answer to his query from the server. Here honest behavior implies that whenever the owner updates the set collection and its public digest, the server updates the set collection and her authentication and auxiliary information accordingly and replies client's queries faithfully w.r.t. the latest set collection and digest.

Definition 6 (Completeness) *For an ZKASC* $(\mathbb{S}_0, Q, U)$*, any sequence of updates* $u_0, u_1, \ldots, u_L$ *on the set collection* $\mathbb{S}_0$*, and for all queries* q *on* $\mathbb{S}_L$*:*

$$\begin{aligned}
&\Pr[(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda); (\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{vk},\mathbb{S}_0);\\
&\quad \big\{(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t) \leftarrow\\
&\qquad\quad \mathsf{Update}(\mathsf{sk}, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, \mathsf{ek}_t, \mathsf{SID}_t, u_t);\\
&\quad (\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{ek}_t, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, u_t, \mathsf{updinfo}_t);\big\}_{0 \le t \le L}\\
&\quad (\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{Query}(\mathsf{ek}_L, \mathsf{aux}_L, \mathsf{auth}(\mathbb{S}_L), \mathbb{S}_L, q) :\\
&\qquad\quad \mathsf{Verify}(\mathsf{vk}, \mathsf{digest}_L, \mathsf{answer}, \mathsf{proof}) = \mathsf{accept} \wedge \mathsf{answer} = Q(\mathbb{S}_L, q)] = 1.
\end{aligned}$$

where the probability is taken over the randomness of the algorithms.

Soundness protects the client against a malicious server. This property ensures that if the server forges the answer to a client's query, then the client will accept the answer with at most negligible probability. The definition considers adversarial server that picks the set collection and adaptively requests updates. After seeing all the replies from the owner, she can pick any point of time (w.r.t. updates) to create a forgery.

Since, given the authentication and auxiliary information to the server, the server can compute answers to queries herself, it is superfluous to give Adv explicit access to Query algorithm. Therefore, we set input of f to empty and SID to $\perp$, as a consequence, in algorithm Update.

Definition 7 (Soundness) *For all PPT adversaries,* Adv *and for all possible valid queries* q *on the set collection* $\mathbb{S}_j$ *of an abstract data type* $(\mathbb{S}_j, Q, U)$*, there exists a negligible function* $\nu(.)$ *such that, the probability of winning the following game is negligible, where the probability is taken over the randomness of the algorithms and the coins of* Adv*:*

Setup Adv *receives* vk *where* $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*. Given* vk*,* Adv *picks* $(\mathbb{S}_0, Q, U)$ *and receives* $\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0$ *for* $\mathbb{S}_0$*, where* $(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk},\mathsf{vk},\mathbb{S}_0)$.

Query Adv *requests a series of updates* $u_1, u_2, \ldots, u_L$*, where* $L = \mathsf{poly}(\lambda)$*, of its choice. For every update request* Adv *receives an update string. Let* $\mathbb{S}_{i+1}$ *denote the state of the set collection after the* i^{th} *update* u_i *and* $\mathsf{updinfo}_i$ *be the update string corresponding to* u_i *received by the adversary, i.e.,* $(\mathbb{S}_{i+1}, \mathsf{auth}(\mathbb{S}_{i+1}), \mathsf{digest}_{i+1}, \mathsf{aux}_{i+1}, \mathsf{ek}_{i+1}, \mathsf{updinfo}_i) \leftarrow \mathsf{Update}(\mathsf{sk}, \mathbb{S}_i, \mathsf{auth}(\mathbb{S}_i), \mathsf{digest}_i, \mathsf{aux}_i, \mathsf{ek}_i, \mathsf{SID}_i, u_i)$ *where* $\mathsf{SID}_i = \perp$.

Response *Finally,* Adv *outputs* $(\mathbb{S}_j, q, \mathsf{answer}, \mathsf{proof})$*,* $0 \le j \le L$*, and wins the game if the following holds:*

$$\mathsf{answer} \neq Q(\mathbb{S}_j, q) \wedge \mathsf{Verify}(\mathsf{vk}, \mathsf{digest}_j, \mathsf{answer}, \mathsf{proof}) = \mathsf{accept}.$$

Zero-knowledge captures privacy guarantees about the set collection against a malicious client. Recall that the client receives a proof for every query answer. Periodically he also receives an updated digest, due to the owner making changes to the set collection. Informally, (1) the proofs should reveal nothing beyond the query answer, and (2) an updated digest should reveal nothing about update operations performed on the set collection. This security property guarantees that the client does not learn which elements were updated, unless he queries for an updated element (deleted or replaced), before and after the update.

The definition of the zero-knowledge property captures the adversarial client's (Adv) view in two games. Let $\mathcal{E}$ be a ZKASC scheme. In the $\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda)$ game (λ is the security parameter), Adv interacts with the honest owner and the honest server (jointly called challenger), whereas in the $\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$ game, it interacts with a simulator, who mimics the behavior of the challenger with oracle access to the source set collection, i.e., it is allowed to query the set collection only with client's queries and does not know anything else about the set collection (except its cardinality, which is a public information) or the updates.

Adv picks $\mathbb{S}_0$ and adaptively asks for queries and updates on it. Its goal is to determine if it is talking to the real challenger or to the simulator, with non-negligible advantage over a random guess.

We note that here SID need not be used explicitly in the definition, since the challenger and the simulator know all the queries and can compute f themselves.

Let D be a binary function for checking the validity of queries and updates on a set collection. For queries, $D(\mathbb{S}_t, q) = 1$ iff the query indices in q are valid set indices of $\mathbb{S}_t$. For updates, $D(\mathbb{S}_t, u = (x, \mathsf{upd}, i)) = 1$ iff i is a valid set index of $\mathbb{S}_t$ and u is a valid update on $\mathcal{X}_i \in \mathbb{S}_t$.

Definition 8 (Zero-Knowledge) *Let $\mathcal{E}$ be a* ZKASC *scheme.* $\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}$ *and* $\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}$ *be defined as follows where, the simulator always checks the validity of an update or query using oracle access to the function $D()$ as defined above.*

$\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda)$**:**

Setup *The challenger runs* $\mathsf{KeyGen}(1^\lambda)$ *to generate* sk, vk *and sends* vk *to* Adv_1. *Given* vk, Adv_1 *picks* $(\mathbb{S}_0, Q, U)$ *of its choice and receives* digest^0 *corresponding to* $\mathbb{S}_0$ *from the real challenger* $\mathcal{C}$ *who runs* $\mathsf{Setup}(\mathsf{sk}, \mathsf{vk}, \mathbb{S}_0)$ *to generate it.* Adv_1 *saves its state information in* state_A.

Query Adv_2 *has access to* state_A *and requests a series of queries* $\{\mathsf{op}_1, \mathsf{op}_2, \ldots, \mathsf{op}_M\}$, *for* $M = \mathsf{poly}(\lambda)$.

If $\mathsf{op} = u$ **is an update request:** $\mathcal{C}$ *runs function $D()$ to check if the update request is valid. If not it returns* $\perp$. *Else,* $\mathcal{C}$ *runs* Update *algorithm. Let* $\mathbb{S}_t$ *be the most recent set collection and* digest_t *be the public digest on it generated by the* Update *algorithm.*
$\mathcal{C}$ *returns* digest_t *to* Adv_2.

If $\mathsf{op} = q_i$ **is a query:** $\mathcal{C}$ *runs function $D()$ to check if the query is valid. If not it returns* $\perp$. *Else,* $\mathcal{C}$ *runs* Query *algorithm for the query with the most recent set collection and the corresponding digest as its parameter.*
$\mathcal{C}$ *returns* answer *and* proof *to* Adv_2.

Response Adv_2 *outputs a bit b.*

$\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$**:**

Setup *Initially* Sim_1 *generates a public key, i.e.,* $(\mathsf{vk}, \mathsf{state}_S) \leftarrow \mathsf{Sim}_1(1^\lambda)$ *and sends it to* Adv_1. *Given* vk, Adv_1 *picks* $(\mathbb{S}_0, Q, U)$ *of its choice and receives* digest_0 *from the simulator* Sim_1, *i.e.,* $(\mathsf{digest}_0, \mathsf{state}_S) \leftarrow \mathsf{Sim}_1(\mathsf{state}_S)$. Adv_1 *saves its state information in* state_A.

Query Adv_2, *who has access to* state_A, *requests a series of queries* $\{\mathsf{op}_1, \mathsf{op}_2, \ldots, \mathsf{op}_M\}$, *for* $M = \mathsf{poly}(\lambda)$.
Sim_2 *is given oracle access to the the most recent set collection and is allowed to query the set collection oracle only for queries that are queried by* Adv. *Let* $\mathbb{S}_{t-1}$ *denote the state of the data structure at the time of* op. *The simulator runs* $(\mathsf{state}_S, a) \leftarrow \mathsf{Sim}_2^{D, \mathbb{S}_{t-1}}(1^\lambda, \mathsf{state}_S, D(\mathbb{S}_{t-1}, \mathsf{op}_i))$ *and returns answer a to* Adv_2 *where:*

If $D(\mathbb{S}_{t-1}, \mathsf{op}_i) = 1$ **and** op_i **is an update request:** $a = \mathsf{digest}_t$, *the updated digest.*

If $D(\mathbb{S}_{t-1}, \mathsf{op}_i) = 1$ **and** op_i **is a query:** $a = (\mathsf{answer}, \mathsf{proof})$ *corresponding to the query* q_i.

Response Adv_2 *outputs a bit b.*

A ZKASC $\mathcal{E}$ is zero-knowledge if there exists a PPT algorithm $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ *s.t. for all malicious stateful adversaries* $\mathsf{Adv} = (\mathsf{Adv}_1, \mathsf{Adv}_2)$ *there exists a negligible function* $\nu(.)$ *s.t.*

$$|\Pr[\mathsf{Real}_{\mathcal{E},\mathsf{Adv}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{E},\mathsf{Adv},\mathsf{Sim}}(1^\lambda) = 1]| \le \nu(\lambda).$$

7 Zero-Knowledge Authenticated Set Collection Construction

In this section, we give an efficient construction for ZKASC. Our construction relies on two basic primitives: *zero-knowledge dynamic universal accumulator* introduced in Section 3 and *accumulation tree* described in Section 2. The sets collection data structure consists of m sets, denoted with $\mathbb{S} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_m\}$, each containing elements from a universe $\mathbb{X}$[10]. We use our zero-knowledge accumulator construction from Section 5 to represent every set in the set collection succinctly using $\mathsf{acc}(\mathcal{X}_i)$. Our construction, similar to [PTT11], relies on accumulation trees to verify that $\mathsf{acc}(\mathcal{X}_i)$ is indeed a representation of the i^{th} set of the current set collection. We note that accumulation tree presents an optimal choice when considering several parameters, including digest size, proof size, verification time, setup and update cost, and storage size.

Given the above representation of a set collection, we develop techniques for efficiently and authentically proving zero-knowledge set algebraic queries and performing updates on $\mathbb{S}$. We note that the authors of [PTT11] also consider authenticated set algebra. Unfortunately, since privacy was not an objective of [PTT11], their techniques for authentic set queries cannot be trivially extended to support our strong privacy notion (zero-knowledge).

We organize the description of our construction as follows. The setup and maintenance of the dynamic set collection $\mathbb{S}$ is given in the next section. These phases instantiate and update zero-knowledge accumulators and authentication tree used to store and update the sets in the collection. We then develop algorithms for verifiable zero-knowledge query and verification algorithms. The query and verify algorithms invoke a subroutine for each individual operation, namely, is-subset, intersection, union and set difference. We describe two algorithms for each set operation: an algorithm executed by the server to construct a proof of an answer on $\mathbb{S}$ and a verification algorithm executed by the client to verify the answer and the proof. For each algorithm we analyze its efficiency as we go. We then compare the asymptotic performance of the ZKASC algorithms with that of [PTT11], which offers no privacy guarantee, in Figure 4. Finally, we prove that the construction described in this section meets the security properties of ZKASC.

7.1 Setup and Update Algorithms

Here we describe and give pseudocode for the algorithms GenKey, Setup, Update, and $\mathsf{UpdateServer}$. We recall that the first two algorithms are used by the owner to prepare keys, collection $\mathbb{S}$ and relevant data structures before outsourcing them to the untrusted server. The owner runs Update to make efficient updates to $\mathbb{S}$, while the server executes $\mathsf{UpdateServer}$ to propagate these updates. Each algorithm is described below.

The GenKey algorithm (Algorithm 1) takes the security parameter as input and generates the secret key sk and the corresponding verification key vk for the scheme. As part of the algorithm, it also initializes our zero-knowledge accumulator from Section 3. For simplicity, we refer to it as DUA.

The Setup algorithm (Algorithm 2) takes the secret key, the verification key and the set collection as input and generates the authentication information for the set collection, a short public digest, the evaluation key and some auxiliary information for the set collection. Among the output objects, only the digest is public and is accessible by the client. The rest of the output are only sent to the server. The server uses them to generate proofs for client queries.

[10] Without loss of generality we assume that our universe $\mathbb{X}$ is the set of nonnegative integers in the interval $[m+1, p-1]$ as in [PTT11].

Algorithm 1 $(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{GenKey}(1^\lambda)$

```
1: Run $\mathsf{GenParams}(1^\lambda)$ to receive bilinear parameters $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g)$.
2: Pick a cryptographic hash function $\mathcal{H}$ which will be viewed as a Random Oracle and append $pub$ with $\mathcal{H}$.
   %$\mathcal{H}$ will be viewed as a Random Oracle for set difference only.
3: Invoke $\mathsf{DUA.GenKey}(1^\lambda)$. Let $\mathsf{DUA.GenKey}(1^\lambda) = (s, g^s)$.
4: return $(s, (g^s, pub))$.
```

Algorithm 2 $(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0) \leftarrow \mathsf{Setup}(\mathsf{sk}, \mathsf{vk}, \mathbb{S}_0)$

```
1: Let $\mathbb{S}_0 = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$. Invoke $\mathsf{DUA.Setup}(s, \mathcal{X}_i)$ for each $\mathcal{X}_i \in \mathbb{S}_0$. Let $\mathsf{DUA.Setup}(s, \mathcal{X}_i) = (\mathsf{acc}(\mathcal{X}_i), \mathsf{ek}_i, \mathsf{aux}_i)$.
2: Let $|\mathcal{X}_i| = n_i$. Set $\mathsf{ek}_0 := (g, g^s, \ldots, g^{s^N})$ where $N = \sum_{i \in [1,m]} n_i$
3: Parse $\mathsf{aux}_i$ corresponding to $\mathcal{X}_i$ as $(r_i, n_i)$ for all $i \in [1, m]$.
4: Set $\mathsf{aux}_0 = ((r_i \mid \mathcal{X}_i \in \mathbb{S}_0), N)$.
5: Invoke $\mathsf{AT.ATSetup}(\mathsf{sk}, (\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m)))$ and let $\mathsf{AT.ATSetup}(\mathsf{sk}, (\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m))) = (\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0)$.
6: return $(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0)$
```

Update (Algorithm 3) algorithm takes as input an update string $u_t = (x, \mathsf{upd}, i)$ where x denotes the element, upd, as in DUA, is a boolean indication if an insert or delete is to be preformed and i denotes the index of the set in which the update is to be performed. This algorithm updates the corresponding set in the set collection and accordingly updates the authentication information, the auxiliary information and the public digest.

As described so far, the accumulation value of the set updated due to u_t is regenerated with fresh randomness (in DUA.Update), which causes a subsequent change in the digest for $\mathbb{S}$. However, this is not enough to achieve zero-knowledge. If a client queries wrt some set $j \neq i$ before and after u_t was performed, and sees that $\mathsf{acc}(\mathcal{X}_j)$ has not changed, then he learns that $\mathcal{X}_j$ is not affected by the update. This will also imply that the proofs that the client holds wrt $\mathcal{X}_j$ between updates are still valid, i.e, they are not ephemeral.

Zero-knowledge property implies that even an adversarial client does not learn anything beyond the fact that an update has occurred (unless his query explicitly returns some updated element) and none of the proofs that a client holds should not be valid after an update. We set f to be a function that takes the client queries since the last update and returns the indices of the sets accessed by them; therefore SID_t has the indices of the sets touched by queries since the $(t-1)^{th}$ update u_{t-1} (we consider setup as the 0^{th} update). To achieve zero-knowledge property, Update needs to use the subroutine refresh, which picks fresh randomness to refresh accumulation values of all the set indices in SID_t. Update algorithm also produces a string $\mathsf{updinfo}_t$ that the server can use to update its authentication and auxiliary information. Note that, SID_t used for efficiency. The non-efficient way of achieving zero-knowledge would be to refresh accumulation values of *all* the sets in the collection.

Next we describe UpdateServer algorithm which is run by the server to propagate the update on the set collection and its authentication information using $\mathsf{updinfo}_t$ generated by the owner corresponding to the t^{th} update. Informally, this algorithm updates the relevant set and updates all the authentication paths corresponding to the sets whose accumulation value has been changed or refreshed by the owner. The algorithm also updates its evaluation key and the auxiliary information.

Efficiency We analyze the access complexity of each algorithm. Let $M = \sum_{i \in m} |\mathcal{X}_i|$ where $\mathbb{S} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ and let $L = |\mathsf{SID}_t|$ for update u_t.

GenKey**:** The complexity of this algorithm is $O(poly(\lambda))$.

Setup**:** For the Setup algorithm, DUA.Setup is called for all m sets. Therefore, the complexity of this step is $O(M)$. Setting up the AT has complexity $O(m)$. Therefore, the total complexity of Setup is $O(M+m)$.

Algorithm 3 $(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t) \leftarrow \mathsf{Update}(\mathsf{sk}, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, \mathsf{ek}_t, \mathsf{SID}_t, u_t)$ where $u_t = (x, \mathsf{upd}, i)$

```
1: Initialize set $W := \perp$.
2: $u_t = (x, \mathsf{upd}, i)$ to be done on set $\mathcal{X}_i$.
3: If $(\mathsf{upd} = 1 \wedge x \in \mathcal{X}_1)$ or $(\mathsf{upd} = 0 \wedge x \notin \mathcal{X}_i)$ **return** $\perp$. Else proceed to next step.
4: Parse $\mathsf{aux}_t$ as $(\{r_j \mid \mathcal{X}_j \in \mathbb{S}_t\}, N)$ and set $\mathsf{aux}_j = (r_j, N)$ for all $j \in [1, m]$.
5: Call $\mathsf{DUA.Update}(\mathsf{acc}(\mathcal{X}_i), \mathcal{X}_i, x, s, \mathsf{aux}_i, \mathsf{upd})$. Let $\mathsf{DUA.Update}(\mathsf{acc}(\mathcal{X}_i), \mathcal{X}_i, x, s, \mathsf{aux}_i, \mathsf{upd}) = (\mathsf{acc}'(\mathcal{X}_i), \mathsf{ek}'_i, \mathsf{aux}'_i)$
6: Replace old $\mathcal{X}_i$ with the updated $\mathcal{X}_i$ to obtain $\mathbb{S}_{t+1}$.
7: Let $M = \sum_{\mathcal{X}_j \in \mathbb{S}_{t+1}} |\mathcal{X}_j|$. Set $N := M$.
8: **If** $M > N$ do the following:
9:     Set $N := M$.
10:    Update $N$ in $\mathsf{aux}'_i$.
11:    Set $\mathsf{ek}_{t+1} := \mathsf{ek}_t \cup g^{s^N}$ and $\mathsf{ekupdate} := g^{s^N}$.
12: **Else**
13:    Set $\mathsf{ek}_{t+1} := \mathsf{ek}_t$ and $\mathsf{ekupdate} := \phi$
14: Update $W \leftarrow W \cup i$.
15: Call $\mathsf{refresh}(s, \mathsf{SID}_t, (\forall j \in \mathsf{SID}_t : (\mathsf{acc}(\mathcal{X}_j), \mathsf{aux}_j))$. Let $\mathsf{refresh}(s, \mathsf{SID}_t, (\forall j \in \mathsf{SID}_t : (\mathsf{acc}(\mathcal{X}_j), \mathsf{aux}_j)) = (\forall j \in \mathsf{SID}_t : (\mathsf{acc}'(\mathcal{X}_j), \mathsf{aux}'_j))$, where $\mathsf{aux}_j = (r_j, N)$
16: Update $W \leftarrow W \cup j$ for all $j \in \mathsf{SID}_t$.
17: Set $\mathsf{aux}_{t+1} := ((r'_j \cdot r_j \mid j \in W), N)$
18: Call $\mathsf{ATUpdateBatch}$ to update the accumulation tree for all leaves corresponding to $\mathcal{X}_j$ for $j \in W$.
19: Let $\mathsf{auth}'$ be the updated authentication information, $\mathsf{digest}'$ be the updated root and a list of $\mathsf{updinfo}_j$ containing the updated authentication paths for each set $\mathcal{X}_j$.
20: Set $\mathsf{auth}(\mathbb{S}_{t+1}) := \mathsf{auth}'$ and $\mathsf{digest}_{t+1} := \mathsf{digest}'$
21: Set $\mathsf{updinfo}_t := (W, (j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{aux}'_j, \mathsf{updinfo}_j) \mid j \in W, \mathsf{ekupdate})$
22: **return** $(\mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}, \mathsf{ek}_{t+1}, \mathsf{updinfo}_t)$.
```

Algorithm 4 $(\forall i \in \mathsf{SID}_t : (\mathsf{acc}'(\mathcal{X}_i), \mathsf{aux}'_i)) \leftarrow \mathsf{refresh}(\mathsf{sk}, \mathsf{SID}_t, (\forall i \in \mathsf{SID}_t : (\mathsf{acc}(\mathcal{X}_i), \mathsf{aux}_i))$

```
1: **For every** $i \in \mathsf{SID}_t$:
2:     Pick $r'_i \leftarrow \mathbb{Z}_p^*$.
3:     Compute $\mathsf{acc}'(\mathcal{X}_i) \leftarrow \mathsf{acc}(\mathcal{X}_i)^{r_i}$.
4:     Parse $\mathsf{aux}_i$ as $(r_i, N)$ and compute $\mathsf{aux}'_i \leftarrow (r_i \cdot r'_i, N)$.
5: **return** $(\forall i \in \mathsf{SID}_t : (\mathsf{acc}'(\mathcal{X}_i), \mathsf{aux}'_i))$
```

Algorithm 5 $(\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1}) \leftarrow \mathsf{UpdateServer}(\mathsf{ek}_t, \mathbb{S}_t, \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t, \mathsf{aux}_t, u_t, \mathsf{updinfo}_t)$ where $u_t = (x, \mathsf{upd}, i)$

```
1: Update $\mathcal{X}_i$ with $u_t$.
2: Replace old $\mathcal{X}_i$ in $\mathbb{S}_t$ with the updated $\mathcal{X}_i$ to obtain $\mathbb{S}_{t+1}$.
3: Parse $\mathsf{updinfo}_t$ as $(W, (j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{aux}'_j, \mathsf{updinfo}_j) \mid j \in W, \mathsf{ekupdate})$.
4: **For all** $j \in W$ do the following:
5:     Parse $\mathsf{aux}'_j$ as $(\tilde{r}_j, N)$.
6:     Replace $r_j$ with $\tilde{r}_j$ and update $N$ in $\mathsf{aux}_t$ to generate $\mathsf{aux}_{t+1}$.
7:     Using $\mathsf{updinfo}_j$ update the authentication path of $\mathsf{acc}(\mathcal{X}_j)$ in the accumulation tree.
8:     Update the authentication path of $\mathsf{acc}(\mathcal{X}_j)$ in $\mathsf{auth}(\mathbb{S}_t)$ to obtain $\mathsf{auth}(\mathbb{S}_{t+1})$
9: Set the updated root of the accumulation tree as $\mathsf{digest}_{t+1}$
10: **If** $\mathsf{ekupdate} = \phi$, then set $\mathsf{ek}_{t+1} := \mathsf{ek}_t$. **Else** set $\mathsf{ek}_{t+1} := \mathsf{ek}_t \cup \mathsf{ekupdate}$.
11: **return** $(\mathsf{ek}_{t+1}, \mathbb{S}_{t+1}, \mathsf{auth}(\mathbb{S}_{t+1}), \mathsf{digest}_{t+1}, \mathsf{aux}_{t+1})$.
```

Update**:** This algorithm takes constant time for the update and constant time for refreshing each accumulator value for the sets whose indices are in SID_t. From Lemma 4, updating each authentication path has complexity $O(1)$. Hence the total complexity of this algorithm is $O(L)$.

UpdateServer**:** This algorithm has access complexity similar to Update, i.e., $O(L)$.

7.2 Set Algebra Query and Verify Algorithms

We proceed with verifiable zero-knowledge set algebra operations that use data structures described in the previous section. Query and Verify algorithms let the server construct a proof of a response to a query and the client verify it, respectively. Since ZKASC supports several set operations, we describe each algorithm in terms of modular subroutines as follows. Algorithm Query takes the query arguments along with an indicator flag indicator that denotes the type of the input query (is-subset, intersection, union, difference) and invokes the appropriate subroutine that we describe subsequently. In particular, we annotate query q with indicator. If indicator $= S$, Query invokes subsetQ (Section 7.3), if indicator $= I$, Query invokes intersectionQ (Section 7.4), if indicator $= U$, Query invokes unionQ (Section 7.5) and if indicator $= D$, Query invokes differenceQ (Section 7.6). Similarly, we annotate the input argument of Verify with the flag indicating which verification algorithm should be used.

Query and Verify algorithms for each set algebra operation are given in the next four sections along with the analysis of their efficiency.

7.3 Subset Query

A subset query is parametrized by a set of elements Δ and an index i of a set collection, i.e., $q = (\Delta, i)$. Given q, the subset query returns a boolean as the answer. In particular, answer $= 1$ if $\Delta \subseteq \mathcal{X}_i$ and answer $= 0$ if $\Delta \nsubseteq \mathcal{X}_i$. This query can be seen as a generalization of Witness algorithm for DUA where membership/non-membership query is supported for a batch of elements instead of a single element. The proof technique is similar to the membership and non-membership proof generation for a single element using Witness algorithm. We give the pseudocode for subset query in Algorithm 6 and verification Algorithm 7.

Algorithm 6 *Subset*: $(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{SubsetQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, q = (\Delta, i))$

```
1: Let $\Delta = \{x_1, \ldots, x_k\}$ and $[a_0, \ldots, a_k]$ be the coefficients of the polynomial $\mathsf{Ch}_\Delta[z]$.
2: If $\Delta \subseteq \mathcal{X}_i$:
3:     Set answer = 1
4:     Compute $W_{\Delta,\mathcal{X}_i} \leftarrow g^{r_i \mathsf{Ch}_{\mathcal{X}_i - \Delta}(s)}$.
5: Else when $\Delta \nsubseteq \mathcal{X}_i$:
6:     Set answer = 0
7:     Using Extended Euclidian Algorithm, compute polynomials $q_1[z]$ and $q_2[z]$ such that $q_1[z]\mathsf{Ch}_\Delta[z] + q_2[z]\mathsf{Ch}_{\mathcal{X}_i}[z] = 1$.
8:     Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q'_1[z] = q_1[z] + \gamma\mathsf{Ch}_{\mathcal{X}_i}[z]$ and $q'_2[z] = (r_i^{-1})(q_2[z] - \gamma\mathsf{Ch}_\Delta[z])$.
9:     Compute $F_1 \leftarrow g^{q'_1(s)}$ and $F_2 \leftarrow g^{q'_2(s)}$.
10:    Set $W_{\Delta,\mathcal{X}_i} := (F_1, F_2)$.
11: Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, i, \mathsf{auth}(\mathbb{S}_t))$. Let $(\Pi_i, \alpha_i) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, i, \mathsf{auth}(\mathbb{S}_t))$.
12: Set proof $= ([a_0, \ldots, a_k], \{g^s, \ldots, g^{s^k}\}, W_{\Delta,\mathcal{X}_i}, \Pi_i, \alpha_i)$.
13: return (answer, proof)
```

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below. Note that $|\Delta| = k$ in Algorithm 6 and 7.

Query: Let us analyze the complexity of each of the steps. Step 1 has complexity $O(k \log k)$ by Lemma 1. Step 4 has complexity $O((|\mathcal{X}_i| - k)\log(|\mathcal{X}_i| - k))$ by the same lemma. In case of non-membership, in Step 7, by Lemma 3, this step has complexity $O(N \log^2 N \log\log N)$ where $N = |\mathcal{X}_i| + k$. The complexity of Step 9 is $O(N \log N)$. The complexity of Step 11 is $O(m^\varepsilon \log m)$ by Lemma 4, where $m = |\mathbb{S}|$. Hence the overall complexity of the query algorithm is $O(k \log k + (|\mathcal{X}_i| - k)\log(|\mathcal{X}_i| - k) + N \log^2 N \log\log N + N \log N + m^\varepsilon \log m) = O(N \log^2 N \log\log N + m^\varepsilon \log m)$.

Algorithm 7 *Subset Verification*: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{SubsetV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

```
1: Parse proof as $([a_0, \ldots, a_k], \{g^s, \ldots, g^{s^k}\}, W_{\Delta, \mathcal{X}_i}, \Pi_i, \alpha_i)$.
2: Certify the validity of $[a_0, \ldots, a_k]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i \in [0,\Delta]} a_i w^i = \prod_{i \in [0,\Delta]} (x_i + w)$ where $\Delta = \{x_1, \ldots, x_k\}$.
   If the verification fails, **return** reject. Else proceed to next step.
3: Invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, i, \Pi_i, \alpha_i)$ If verification fails **return** reject. Else proceed to next step.
4: **If** answer = 1:
5:         Check if $e(g^{\mathsf{Ch}_\Delta(s)}, W_{(\Delta, \mathcal{X}_i)}) = e(\mathsf{acc}(\mathcal{X}_i), g)$. If verification fails **return** reject. Else **return** accept.
6: **Else if** answer = 0:
7:         Parse $W_{(\Delta, \mathcal{X}_i)}$ as $(F_1, F_2)$
8:         Check if $e(F_1, g^{\mathsf{Ch}_\Delta(s)}) e(F_2, \mathsf{acc}(\mathcal{X}_i)) = e(g, g)$ If verification fails **return** reject. Else **return** accept.
```

Verification: Verification in Step 2 takes time $O(k)$ by Lemma 2. By Lemma 4, Step 3 has complexity $O(1)$. Computing $g^{\mathsf{Ch}_\Delta(s)}$ takes time $O(k)$. Therefore Step 5 has complexity $O(k)$. Finally Step 9 involves requires 5 bilinear map computations and one multiplication and hence has complexity $O(1)$. Therefore, the overall complexity of verification is $O(k)$.

Proof size: The proof size is $O(k)$.

7.4 Set Intersection Query

Set intersection query q is parameterized by a set of indices of the set collection, $q = (i_1, \ldots, i_k)$. The answer to an intersection query is a set of elements which we denote as answer and a simulatable proof of the correctness of the answer. If the intersection is computed correctly then $\mathsf{answer} = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2} \cap \ldots \cap \mathcal{X}_{i_k}$. We express the correctness of intersection with the two following conditions as in [PTT11]:

Subset condition: $\mathsf{answer} \subseteq \mathcal{X}_{i_1} \wedge \mathsf{answer} \subseteq \mathcal{X}_{i_2} \wedge \ldots \wedge \mathsf{answer} \subseteq \mathcal{X}_{i_k}$. This condition ensures that the returned answer is a subset of all the queried set indices, i.e., every element of answer belongs to each of the sets $\mathcal{X}_j$ for $j \in [i_1, i_k]$;

Completeness condition: $(\mathcal{X}_{i_1} - \mathsf{answer}) \cap (\mathcal{X}_{i_2} - \mathsf{answer}) \cap \ldots \cap (\mathcal{X}_{i_k} - \mathsf{answer}) = \emptyset$. This is the completeness condition, that ensures that answer indeed contains *all* the common elements of $\mathcal{X}_{i_1}, \ldots, \mathcal{X}_{i_k}$, i.e., none of the elements have been omitted from answer.

To prove the first condition, we will use subset query as a subroutine. Proving the second condition is more tricky; it relies on the fact that the characteristic polynomials for the sets $\mathcal{X}_j - \mathsf{answer}$, for all $j \in [i_1, i_k]$, do not have common factors. In other words, these polynomials should be co-prime and their GCD should be 1 (Lemma 3). Since the proof units should be simulatable, we cannot directly use the technique as in [PTT11]. To this end, we randomize the proof units by generalizing the randomization technique in Section 5 used to prove non-membership in a single set. We present the pseudocode for the set intersection query in Algorithm 8 and its verification in Algorithm 9.

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below.

Query: Let us analyze the complexity of each of the steps. Let $m = |\mathbb{S}|$, $N = \sum_{j \in [i_1, i_k]} n_j$, $n_j = |\mathcal{X}_j|$ and k is the number of indices in the query. Note that N is an upper bound on ρ. Let us denote $\rho = \rho$ Step 3 has complexity $O(\rho \log \rho)$ by Lemma 1. By Lemma 3, Step 5 has complexity $O(N \log^2 N \log\log N)$. Step 7 has complexity $O(N \log N)$ (by Lemma 1). Finally, Step 9 has complexity $O(km^\varepsilon \log m)$ by Lemma 4. Therefore, the overall complexity is $O(\rho \log \rho + N \log^2 N \log\log N + N \log N + km^\varepsilon \log m) = O(N \log^2 N \log\log N + km^\varepsilon logm)$

Verification: For the verification algorithm, let us look at each of the verification steps individually. Step 2 has complexity $O(\rho)$ by Lemma 2. Step 3takes has complexity $O(k)$ by Lemma 4. In Step 4, computing $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ has complexity $O(\rho)$ and then there are $2k$ bilinear map computations. Therefor this step has

Algorithm 8 *Set Intersection*: $(\mathsf{answer},\mathsf{proof}) \leftarrow \mathsf{IntersectionQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, q = (i_1, \ldots, i_k))$

```
1: Let $\mathsf{answer} = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2} \cap \ldots \cap \mathcal{X}_{i_k}$ and WLOG let us assume $k$ is even.
2: Let $\mathsf{Ch}_{\mathsf{answer}}$ be the charactersitic polynomial for answer.
3: Let $[a_\rho, \ldots, a_0]$ be the coefficients of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
4: Let $P_j[z] = \mathsf{Ch}_{\mathcal{X}_j \setminus \mathsf{answer}}$ for all $j \in [i_1, i_k]$ where $r_i$ is the random number used as the blinding factor for $\mathcal{X}_j$.
5: Using Extended Euclidian Algorithm, compute polynomials $q_j[z]$ such that $\sum_{j \in [i_1, i_k]} q_j[z] P_j[z] = 1$.
6: Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q'_{i_1}[z] := (r_{i_1}^{-1})(q_{i_1}[z] + \gamma P_{i_2}[z])$ and $q'_{i_2}[z] := q_{i_2}[z] - \gamma P_{i_1}[z]$, $q'_{i_3}[z] := (r_{i_3}^{-1})(q_{i_3}[z] + \gamma P_{i_4}[z])$ and $q'_{i_4}[z] := q_{i_4}[z] - \gamma P_{i_3}[z], \ldots, q'_{i_{k-1}}[z] := (r_{i_{k-1}}^{-1})(q_{i_{k-1}}[z] + \gamma P_{i_k}[z])$ and $q'_{i_k}[z] := q_{i_k}[z] - \gamma P_{i_{k-1}}[z]$.
7: Compute $F_j = g^{q'_j(s)}$ for all $j \in [i_1, i_k]$.
8: Invoke Subset query with $\mathsf{subsetQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, \rho, j)$ for all $j \in [i_1, i_k]$. Let $\mathsf{subsetQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, \rho, j) = (1, W_{\mathsf{answer}, \mathcal{X}_j})$
9: Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $(\Pi_j, \alpha_j) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$.
10: Set $\mathsf{proof} = ([a_\rho, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{F_j, W_{(\mathsf{answer}, \mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$, where $|\mathsf{answer}| = \rho$.
11: return $(\mathsf{answer}, \mathsf{proof})$
```

Algorithm 9 *Set Intersection Verification*: $(\mathsf{accept/reject}) \leftarrow \mathsf{IntersectionV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

```
1: Parse proof as $([a_\rho, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{F_j, W_{(\mathsf{answer}, \mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$ where $|\mathsf{answer}| = \rho$.
2: Certify the validity of $[a_\rho, \ldots, a_0]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i \in [0, \rho]} a_i w^i = \prod_{i \in [0, \rho]} (x_i + w)$ where $\mathsf{answer} = \{x_0, \ldots, x_\rho\}$. If the verification fails, return reject. Else proceed to next step.
3: For all $j \in [i_1, i_k]$ invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, j, \Pi_j, \alpha_j)$ If verification fails for at least one $\alpha_j$, return reject. Else proceed to next step.
4: For each $j \in [i_1, i_k]$ verify that: $e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, W_{(\mathsf{answer}, \mathcal{X}_j)}) = e(\mathsf{acc}(\mathsf{state}_j), g)$. If verification fails for at least one $\alpha_j$, return reject. Else proceed to next step.
5: Check the completeness condition by checking if $\prod_{j \in [i_1, i_k]} e(F_j, W_{(\mathsf{answer}, \mathcal{X}_j)}) = e(g, g)$. If verification fails return reject. Else
6: return accept
```

complexity $O(\rho + k)$. Finally, Step 4 requires $k+1$ bilinear map computations, $k-1$ products and hence has complexity $O(k)$. Therefore, the overall complexity of verification is $O(\rho + k)$.

Proof size: The proof size is $O(\rho + k)$.

7.5 Set Union

Set union query, like intersection, is parameterized by a set of indices of the set collection, $q = (i_1, \ldots i_k)$. The answer to an union query contains a set of elements, denoted as $\mathsf{answer} = \mathcal{X}_{i_1} \cup \mathcal{X}_{i_2} \cup \ldots \cup \mathcal{X}_{i_k}$, and a simulatable proof of the correctness of the answer.

We introduce a technique for checking correctness of union operation based on the following conditions:

Superset condition: $\mathcal{X}_{i_1} \subseteq \mathsf{answer} \wedge \mathcal{X}_{i_2} \subseteq \mathsf{answer} \wedge \ldots \wedge \mathcal{X}_{i_k} \subseteq \mathsf{answer}$. This condition ensures that no element has been excluded from the returned answer.

Membership condition: $\mathsf{answer} \subseteq \tilde{U}$ where $\tilde{U} = \mathcal{X}_{i_1} \uplus \mathcal{X}_{i_2} \uplus \ldots \uplus \mathcal{X}_{i_k}$. $\uplus$ denotes multiset union of the queries sets, i.e., $\uplus$ preserves the multiplicity of every element in the union. This condition ensures that every element of answer belongs to *at least* one of the sets $\mathcal{X}_{i_1}, \ldots, \mathcal{X}_{i_k}$.

The query algorithm generates the query answer along with proofs for both the conditions mentioned here. The first condition can be checked by using the subset proof developed in Section 7.3. The second condition should be proved carefully and not reveal (1) whether an element belongs to more than one of the sets in the query, and (2) which set an element in the union comes from. For example, returning $\tilde{U}$ in the clear trivially reveals the multiplicity of every element in answer. Instead, the server returns $\mathsf{acc}(\tilde{U})$ which equals $g^{\mathsf{Ch}_{\tilde{U}}(s)}$ blinded with randomness in the exponent. In order to prove that the server computed $\mathsf{acc}(\tilde{U})$ correctly, we introduce a *union tree*.

A union tree (UT) is a binary tree computed as follows. Corresponding to the k queried indices, $\mathsf{acc}(\mathcal{X}_{i_1}), \ldots, \mathsf{acc}(\mathcal{X}_{i_k})$ are the leaves of UT. The leaves are computed bottom up. Every internal node is computed as follows: For each internal node v, let v_1 and v_2 be its two children. The (multi)set associated with v is the multiset $M = M_1 \uplus M_2$ where M_1 and M_2 are (multi)sets for v_1 and v_2 respectively. Let r_1 and r_2 be the corresponding blinding factors. Then the node v stores value $g^{r_1 r_2 \mathsf{Ch}_M(s)}$. Finally, the server constructs proof of subset for answer in $\tilde{U}$. We give the pseudocode for the set union query in Algorithm 10 and its verification in Algorithm 11.

Algorithm 10 *Set Union*: $(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{unionQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, (i_1, \ldots, i_k))$

```
1: Let answer $= \mathcal{X}_{i_1} \cup \mathcal{X}_{i_2} \cup \ldots \cup \mathcal{X}_{i_k}$.
2: Let $[a_\rho, \ldots, a_0]$ be the coefficients of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
3: Let $\tilde{U} = \mathcal{X}_1 \uplus \mathcal{X}_2 \uplus \ldots \uplus \mathcal{X}_k$ where $\uplus$ denotes multiset union, i.e., $\uplus$ preserves the multiplicity of every element in the union.
4: For each index $j \in [i_1, i_k]$ compute $W_{(\mathcal{X}_j, \mathsf{answer})} \leftarrow g^{\frac{\mathsf{Ch}_{\mathsf{answer}}(s)}{r_j \mathsf{Ch}_{\mathcal{X}_j}(s)}}$.
5: Build a binary tree on $k$ leaves (representing the $k$ indices in the query) bottom-up as follows: For a leaf node, let $a(v) = \mathsf{acc}(\mathcal{X}_j) = g^{r_j \mathsf{Ch}_{\mathcal{X}_j}(s)}$. For each internal node $v$, let $v_1$ and $v_2$ be its two children. The (multi)set associated with $v$ is the multiset $M = M_1 \uplus M_2$ where $M_1$ and $M_2$ are (multi)sets for $v_1$ and $v_2$ respectively. Let $r_1$ and $r_2$ be the corresponding blinding factors. Then $a(v) = g^{r_1 r_2 \mathsf{Ch}_M(s)}$.
6: Compute $W_{(\mathsf{answer}, \tilde{U})} \leftarrow g^{\frac{(\prod_{j \in [i_1, i_k]} r_j) \mathsf{Ch}_{\tilde{U}}(s)}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$
7: Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $(\Pi_j, \alpha_j) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$.
8: proof := $([a_\rho, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{a(v)\}_{v \in V(\mathsf{UT})}, W_{(\mathsf{answer}, \tilde{U})}, \{W_{(\mathcal{X}_j, \mathsf{answer})}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$, where $|\mathsf{answer}| = \rho$.
9: return (answer, proof)
```

Algorithm 11 *Set Union*: $(\mathsf{accept}/\mathsf{reject}) \leftarrow \mathsf{unionV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

```
1: Check that answer has no repeated elements. If not, return reject.
2: Parse proof as $([a_\rho, \ldots, a_0], \{g^s, \ldots, g^{s^\rho}\}, \{a(v)\}_{v \in V(\mathsf{UT})}, W_{(\mathsf{answer}, \tilde{U})}, \{W_{(\mathcal{X}_j, \mathsf{answer})}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$ where $|\mathsf{answer}| = \rho$.
3: Certify the validity of $[a_\rho, \ldots, a_0]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i \in [0,\rho]} a_i w^i = \prod_{i \in [0,\rho]} (x_i + w)$ where answer $= \{x_1, \ldots, x_\rho\}$. If the verification fails, return reject. Else proceed to next step.
4: For all $j \in [i_1, i_k]$ invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, j, \Pi_j, \alpha_j)$ If verification fails for at least one $\alpha_j$, return reject. Else proceed to next step.
5: For each $j \in [i_1, i_k]$ verify that: $e(\alpha_j, W_{(\mathcal{X}_j, \mathsf{answer})}) = e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, g)$. If verification fails for at least one $\alpha_j$, return reject. Else proceed to next step.
6: For each given node $v$ in UT, verify that $e(a(v), g) = e(a(v_1), a(v_2))$, where $v_1, v_2$ are children of $v$. If verification fails for at least one node, return reject. Else proceed to next step.
7: Let root of UT be denoted as $\mathsf{acc}(\tilde{U})$. Verify that $e(W_{(\mathsf{answer}, \tilde{U})}, g^{\mathsf{Ch}_{\mathsf{answer}}(s)}) = e(\mathsf{acc}(\tilde{U}), g)$. If the verification fails, return reject. Else
8: return accept.
```

Note: Previously developed techniques for verifiable set union cannot be used for the following reasons. The proof of membership condition in [PTT11] does not satisfy zero-knowledge property since it reveals which particular set an element in answer comes from. Moreover, it is inefficient as pointed out in [CPPT14]. On the other hand, the union proof in [CPPT14] relies on non-falsifiable knowledge assumption and it is not zero-knowledge as it requires set intersection of [PTT11] as a subroutine.

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below.

Query: Let $m = |\mathbb{S}|$, $N = \sum_{j \in [i_1, i_k]} n_j$, $n_j = |\mathcal{X}_j|$ and k is the number of indices in the query. Note that $N = |\tilde{U}|$ and N is an upper bound on ρ. We will analyze the complexity in each step. By Lemma 1, Step 2 has complexity $O(\rho\log\rho)$. By the same lemma, Step 4 has complexity $O(k\rho\log\rho)$. Now let us analyze the complexity of computing the union tree in Step 5. Computing all the nodes at a level has complexity $O(N\log N)$ (by Lemma 1) and the tree has $\log k$ levels. So the overall complexity in computing the union tree is $O(N\log N\log k)$. In Step 6, computing the member witness takes time $O((N-\rho)\log(N-\rho))$ which is bounded by $O(N\log N)$. Finally, Step 7 takes time $O(km^{\varepsilon}\log m)$. So the overall complexity of unionQ is $O(\rho\log\rho + k\rho\log\rho + N\log N\log k + N\log N + km^{\varepsilon}\log m) = O(k\rho\log\rho + N\log N\log k + km^{\varepsilon}\log m)$.

Verification: For the verification algorithm, let us look at each of the verification steps individually. Step 1 takes time $O(\rho)$. Step 3 takes time $O(\rho)$ by Lemma 2. By Lemma 4, Step 4takes total time $O(k)$. In Step 5, computing $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ takes time $O(\rho)$ and then there are $2k$ bilinear map computations. Therefor this step runs in time $O(\rho+k)$. Verifying the correctness of the union tree in Step 6 requires 2 bilinear map computation for each internal node of the tree and therefore takes time $O(k)$, where k is the number of nodes in the union tree. Finally, Step 5 takes $O(1)$ time since $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ has already been computed. Therefore, the total time at verification is $O(\rho+2k) = O(\rho+k)$.

Proof size: The proof size is $O(\rho+k) = O(\rho+k)$.

7.6 Set Difference Query

Set difference query q is parameterized by two set indices of the set collection $q = (i_1, i_2)$. The answer to a set difference query is $\mathsf{answer} = \mathcal{X}_{i_1} - \mathcal{X}_{i_2}$ and a proof of correctness of the answer. We express the correctness of the answer using the following condition $(\mathsf{answer} = \mathcal{X}_{i_1} - \mathcal{X}_{i_2}) \iff \mathcal{X}_{i_1} - (\mathcal{X}_{i_1} - \mathsf{answer}) = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$. This condition ensures that (1) all the elements of answer indeed belongs to $\mathcal{X}_{i_1}$ and (2) *all* the elements of $\mathcal{X}_{i_1}$ that are not in $\mathcal{X}_{i_2}$ are contained in answer. In other words, the union of answer and the intersection $I = \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$ equals $\mathcal{X}_{i_1}$.

The condition is tricky to prove for the following reasons. The server cannot reveal neither $\mathcal{X}_{i_1} - \mathsf{answer}$ nor $\mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$ to the client, since this reveals more than the set difference answer the client requested for (hence, breaking our zero-knowledge property).[11] Hence, we are required to provide blinded accumulators corresponding to these sets. Unfortunately, the blinded version of $\mathcal{X}_{i-1} - (\mathcal{X}_{i_1} - \mathsf{answer})$ and $\mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$, even if the server computed the answer correctly, would be different. This is caused by different blinding factors used for these accumulators, even though the exponent that corresponds to the elements of the sets is the same. We use this fact and let the server prove that the non-blinded exponents are the same.

Our solution relies on an NIZKPoK protocol as described in Section 2. We use the NIZK version of the Σ-protocol in the random oracle model (standard Fiat Shamir transformation) as described in Figure 1. We would like to note that there are alternative methods [GMY03,MY04] to make Σ-protocols NIZK that do not rely on the random oracle model. Unfortunately, these methods come with some performance penalty. We give the pseudocode for the set difference query in Algorithm 12 and its verification in Algorithm 13.

Efficiency We analyze the run time of the query and the verification algorithms and the size of the proof below.

Query Let us analyze the complexity of each of the steps. Let $m = |\mathbb{S}|$, $N = \sum_{j\in[i_1,i_2]} n_j$, $n_j = |\mathcal{X}_j|$ and k is the number of indices in the query. Note that N is an upper bound on ρ. Step 2 has complexity $O(\rho\log\rho)$ by Lemma 1. By Lemma 1, Step 3 has complexity $O((\mathcal{X}_{i_1}-\rho)\log(\mathcal{X}_{i_1}-\rho))$. By the same lemma, Step 5 has complexity $O(|I|\log|I|)$. By the analysis for set intersection, we know that Steps 7-11 has complexity $O(N\log^2 N\log\log N)$. From Figure 1, we see, computing PKw takes time $O(1)$ in Step 12. Finally, Step 13 has complexity $O(m^{\varepsilon}\log m)$ by Lemma 4. Therefore, the overall complexity is $O(\rho\log\rho + (\mathcal{X}_{i_1}-\rho)\log(\mathcal{X}_{i_1}-\rho) + |I|\log|I| + N\log^2 N\log\log N + m^{\varepsilon}\log m) = O(N\log^2 N\log\log N + m^{\varepsilon}\log m)$.

[11] We note that the sets are revealed to the client in [PTT11] where privacy is not a concern.

Algorithm 12 *Set Difference*: $(\mathsf{answer}, \mathsf{proof}) \leftarrow \mathsf{differenceQ}(\mathsf{ek}_t, \mathsf{aux}_t, \mathsf{auth}(\mathbb{S}_t), \mathbb{S}_t, (i_1, i_2))$
]

```
1: Let $\mathsf{answer} = \mathcal{X}_{i_1} - \mathcal{X}_{i_2} = \{x_1, \dots, x_\rho\}$.
2: Let $[a_\rho, \dots, a_0]$ be the coefficients of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
3: Compute membership witness for answer in $\mathcal{X}_{i_1}$ using the subset algorithm. Let $W_{(\mathsf{answer},\mathcal{X}_{i_1})} = g^{r_i \mathsf{Ch}_{\mathcal{X}_{i_1} - \mathsf{answer}}(s)}$.
4: Compute $I \leftarrow \mathcal{X}_{i_1} \cap \mathcal{X}_{i_2}$
5: Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $\mathsf{acc}(I) \leftarrow g^{r_{i_1} r_{i_2} \gamma \mathsf{Ch}_I}$.
6: Let $P_j[z] = \mathsf{Ch}_{\mathcal{X}_j - I}$ for all $j \in [i_1, i_2]$.
7: Compute $W_{I,\mathcal{X}i_1} \leftarrow g^{\frac{P_{i_1}(s)}{r_{i_2}\gamma}}$
8: Compute $W_{I,\mathcal{X}i_2} \leftarrow g^{\frac{P_{i_2}(s)}{r_{i_1}\gamma}}$
9: Using Extended Euclidian Algorithm, compute polynomials $q_j[z]$ such that $\sum_{j \in [i_1,i_2]} q_j[z] P_j[z] = 1$.
10: Pick a random $\beta \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q'_{i_1}[z] = (r_{i_2}\gamma)(q_{i_1}[z] + \beta P_{i_2}[z])$ and $q'_{i_2}[z] = (r_{i_2}\gamma)(q_{i_2}[z] - \beta P_{i_1}[z])$.
11: Compute $F_j = g^{q'_j(s)}$ for all $j \in [i_1, i_k]$.
12: Compute PKproof by invoking PK with $h = acc(I)$, $g' = W_{\mathsf{answer},\mathcal{X}_{i_1}}$ and $x = r_{i_2}\gamma$.
13: Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$. Let $(\Pi_j, \alpha_j) \leftarrow \mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for $j \in [i_1, i_2]$.
14: Set $\mathsf{proof} := ([a_\rho, \dots, a_0], \{g^s, \dots, g^{s^\rho}\}, W_{(\mathsf{answer},\mathcal{X}_{i_1})}, \mathsf{acc}(I), \{F_j, W_{(I,\mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1,i_2]}, \mathsf{PKproof})$ where where $|\mathsf{answer}| = \rho$.
15: return $(\mathsf{answer}, \mathsf{proof})$
```

Algorithm 13 *Set Difference Verification*: $(\mathsf{accept/reject}) \leftarrow \mathsf{differenceV}(\mathsf{vk}, \mathsf{digest}_t, \mathsf{answer}, \mathsf{proof})$

```
1: Parse $\mathsf{proof} := ([a_\rho, \dots, a_0], \{g^s, \dots, g^{s^\rho}\}, W_{(\mathsf{answer},\mathcal{X}_{i_1})}, \mathsf{acc}(I), \{F_j, W_{(I,\mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1,i_2]}, \mathsf{PKproof})$ where $|\mathsf{answer}| = \rho$.
2: Certify the validity of $[a_\rho, \dots, a_0]$ by picking $w \xleftarrow{\$} \mathbb{Z}_p^*$ and verifying that $\sum_{i \in [0,\rho]} a_i w^i = \prod_{i \in [0,\rho]} (x_i + w)$ where $\mathsf{answer} = \{x_1, \dots, x_\rho\}$. If the verification fails, return reject. Else proceed to next step.
3: For all $j \in [i_1, i_k]$ invoke $\mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, j, \Pi_j, \alpha_j)$ If verification fails for at least one $\alpha_j$, return reject. Else proceed to next step.
4: Verify $W_{(\mathsf{answer},\mathcal{X}_{i_1})}$ by checking $e(W_{(\mathsf{answer},\mathcal{X}_{i_1})}, g^{\mathsf{Ch}_{\mathsf{answer}}(s)}) = e(\alpha_i, g)$. If the verification fails return reject. Else proceed to next step.
5: Parse PKproof as $(b, c, r)$ and run the verification steps as in Figure 1 with $h = acc(I)$, $g' = W_{\mathsf{answer},\mathcal{X}_{i_1}}$. If the verification fails return reject. Else proceed to next step.
6: Verify if $e(W_{(I,\mathcal{X}_j)}, \mathsf{acc}(I)) = e(\alpha_j, g)$ for $j \in [i_1, i_2]$. If verification fails for at least one $\alpha_j$, return reject. Else proceed to next step.
7: Check if $\prod_{j \in [i_1,i_2]} e(F_j, W_{(I,\mathcal{X}_j)}) = e(g, g)$. If verification fails return reject. Else
8: return accept
```

Verification: For the verification algorithm, let us look at each of the verification steps individually. Step 2 takes time $O(\rho)$. By Lemma 2. Step 3 has complexity $O(1)$ by Lemma 4. In Step 4 computing $g^{\mathsf{Ch}_{\mathsf{answer}}(s)}$ has complexity $O(\rho)$. Step 6, 5 and 7 has complexity $O(1)$ each. Therefore, the overall complexity of verification is $O(\rho)$.

Proof size: The proof size is $O(\rho)$.

7.7 Efficiency Comparison with the Scheme of [PTT11]

We compare the asymptotic complexity of the algorithms of our ZKASC scheme with that of [PTT11] in Figure 4. Recall that our ZKASC scheme satisfies both authenticity and privacy (in particular, the strong notion of zero-knowledge property) for set algebra queries, while [PTT11] provides only authenticity and trivially reveals information about the set collection $\mathbb{S} = \{\mathcal{X}_1, \dots, \mathcal{X}_m\}$ beyond query answers. We show that our setup, query and verify algorithms have the exact same asymptotic performance as those of [PTT11]. The update algorithms are more expensive compared to that of [PTT11]. At a high level, the extra cost is due to zero-knowledge property we achieve in return. We require all the proofs to be ephemeral, i.e., proofs should not hold good between updates. Hence, every set for which the client has received a proof between the updates has to be refreshed. Finally, we note that the construction of [CPPT14] offers faster union queries (roughly by a multiplicative $k/\log N$ factor) but its security relies on non-falsifiable type assumptions.

		[PTT11] \ This paper
Setup		$M+m$
Update	Owner	$1 \setminus L$
	Server	$1 \setminus L$
Subset	Query	$N\log^2 N\log\log N + m^{\varepsilon}\log m$
	Verify/Proof size	k
Instersection	Query	$N\log^2 N\log\log N + km^{\varepsilon}\log m$
	Verify/Proof size	$\rho+k$
Union	Query	$kN\log N + km^{\varepsilon}\log m$
	Verify/Proof size	$\rho+k$
Difference	Query	$N\log^2 N\log\log N + m^{\varepsilon}\log m$
	Verify/Proof size	ρ

Fig. 4: This table compares the access complexity of each operation with that of [PTT11]. When only one value appears in the last column, it applies to both constructions. We note that the access complexity of Union Query was originally mistakenly reported as $O(N\log N)$ in [PTT11].
Notation: $m = |\mathbb{S}|$, $M = \sum_{i\in m}|\mathcal{X}_i|$, $n_j = |\mathcal{X}_j|$, $N = \sum_{j\in[i_1,i_k]} n_j$, k is the number of group elements in the query input (for the subset query it is the cardinality of a queried subset Δ and for the rest of the queries it is the number of set indices), ρ denotes the size of a query answer, and L is the number of sets touched by queries between updates u_{t-1} and u_t, and $0<\varepsilon<1$ is a constant chosen during setup. The reported complexities are asymptotic.

7.8 Security Proofs

In this section, we will prove that our ZKASC construction satisfies the security properties of soundness (Definition 7) and zero-knowledge (Definition 8). Completeness (Definition 6) directly follows from the constructions as described with the respective algorithms. We first prove a short lemma that we will use in our soundness proof.

Lemma 5 *For any adversarially chosen proof* Π_i*, if algorithm* $\mathsf{ATVerify}(\mathsf{vk},\mathsf{digest}_t,i,\Pi_i,\alpha_i)$ *accepts, then* $\alpha_i = \mathsf{acc}(\mathcal{X}_i)$ *with probability* $\Omega(1-\nu(\lambda))$

Proof The lemma is identical to Lemma 5 in [PTT11]. The only difference is that now each set is represented using a randomized accumulator instead of a general accumulator. We will show a reduction from Lemma 5 in [PTT11] to this lemma. For the sake of contradiction, let us assume that there exists an adversary Adv that outputs Π_i such that $\mathsf{ATVerify}(\mathsf{vk},\mathsf{digest}_t,i,\Pi_i,\alpha_i) = \mathsf{accept}$ and $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i)$. Then there exists an adversary $\mathcal{A}$ that does the following:

- Initially $\mathcal{A}$ sends the public key vk to Adv.
- Adv comes up with a set collection $\mathbb{S}_0 = \{\mathcal{X}_1,\ldots,\mathcal{X}_m\}$.
- $\mathcal{A}$ picks random elements $r_1,\ldots,r_m \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $\mathbb{S}'_0 = \{\mathcal{X}_1 \cup r_1,\ldots,\mathcal{X}_m \cup r_m\}$., where $r_i \notin \mathcal{X}_i$. Otherwise $\mathcal{A}$ picks a fresh randomness for that set.

- $\mathcal{A}$ forwards all the authentication units to Adv. Note that, from Adv's perspective, the blinding factor used for $\mathcal{X}_i$ is $s + r_i$.
- $\mathcal{A}$ forwards all the update requests to its challenger and forwards the challenger's responses to Adv. Note that, if Adv requests to insert element r_i is $\mathcal{X}_i$, $\mathcal{A}$ aborts. But this happens only with probability $\frac{1}{|\mathbb{Z}_p^*|}$ which is negligible in the security parameter λ.
- Finally $\mathcal{A}$ outputs Adv's forgery Π_i and inherits Adv's success probability.
  Therefore, $\mathcal{A}$ breaks Lemma 5 in [PTT11]. ∎

Proof of Soundness: In this section we prove that the ZKASC construction satisfies the soundness property. Formally, we prove Lemma 6.

Lemma 6 *Under the q-SBDH assumption, the ZKASC scheme is sound as per Definition 7.*

Proof Let q be some polynomial in the security parameter of the scheme such that q is an upper bound on the maximum size the adversarial set collection can grow to. Since the adversary has to come with a set collection that is $poly(\lambda)$, and is allowed to ask a polynomial (in λ) number of queries, this upper bound exists. We will give a reduction to q-SBDH assumption. Let $\mathcal{A}$ be the reduction which receives a q-SBDH tuple $(g, g^s, \ldots, g^{s^q})$ as input. The reduction does the following:

Recall the soundness game where the adversary Adv sees the vk, then comes up with a set collection $\mathbb{S}_0 = \mathcal{X}_1, \ldots, \mathcal{X}_k$. and then sends it to $\mathcal{A}$. $\mathcal{A}$ runs all the steps of Algorithm 2 where it computes the proof units using $(g, g^s, \ldots, g^{s^q})$ instead of the secret key s and sends $(\mathbb{S}_0, \mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0, \mathsf{ek}_0, \mathsf{aux}_0)$. Note that all the steps can be executed using the tuple $(g, g^s, \ldots, g^{s^q})$. For an update query, $\mathcal{A}$ runs the steps of Algorithm 3 and uses $(g, g^s, \ldots, g^{s^q})$ instead of s as before.

If Adv outputs a succesful forgery, it has to forge proof for at least one of the following queries: subset, set intersection, set union or set difference. Let us define the following events:
Event A_1: The adversary outputs a forged subset proof.
Event A_2: The adversary outputs a forged set intersection proof.
Event A_3: The adversary outputs a forged set union proof.
Event A_4: The adversary outputs a forged set difference proof.

The probability that Adv outputs a successful forgery is $Pr[A_1 \cup A_2 \cup A_3 \cup A_4] \leq Pr[A_1] + Pr[A_2] + Pr[A_3] + Pr[A_4]$. We will individually bound the probability of success of each of the events individually.

Lemma 7 *Under the q-SBDH assumption, there exists a negligible function* $\nu(\lambda)$ *such that* $Pr[A_1] \leq \nu(\lambda)$

Proof If Adv outputs a succesful forgery, it has to forge on at least one of the following steps: Step 2, 3, and 5 or 8 in Algorithm subsetQ. Let us denote the corresponding events as E_i for $i \in [1,4]$ respectively and bound the probability of each event. By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

By Lemma 5 $Pr[E_2] \leq \nu_2(\lambda)$ for some negligible function ν_2.

Next we analyze the probability of forging a membership proof, i.e., $Pr[E_3]$.
- The equation $e(g^{\mathsf{Ch}_\Delta(s)}, W_{(\Delta,\mathcal{X}_i)}) = e(\mathsf{acc}(\mathcal{X}_i), g)$ verified to be true but $\Delta \nsubseteq \mathcal{X}_i$. This implies there exists at least one element $y \in \Delta$ such that $y \notin \mathcal{X}_i$.
- Therefore there exists polynomial $P[z]$and a scalar c such that $\mathsf{Ch}_{\mathcal{X}_i}[z] = (y+z)P[z] + c$.
- $\mathcal{A}$ computes polynomial $P[z]$ and scalar c using polynomial long division.
- $\mathcal{A}$ outputs $[y, (e(W_{(\Delta,\mathcal{X}_i)}, g^{\mathsf{Ch}_{\{\Delta-y\}}(s)})^{r_i^{-1}} e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\Delta,\mathcal{X}_i)}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_3] \leq \nu_3(\lambda)$ for some negligible function ν_3.

Therefore, the probability of successfully forging a membership proof is $Pr[E_1 \wedge E_2 \wedge E_3] \leq Pr[E_1] + Pr[E_2] + Pr[E_3] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'.

Now let us look at the probability of forging a non-membership proof, i.e., $Pr[E_4]$.

- E_4 is the event when $e(F_1, g^{\mathsf{Ch}_\Delta(s)})e(F_2, \mathsf{acc}(\mathcal{X}_i)) = e(g,g)$ but $\Delta \subseteq \mathcal{X}_i$. This implies for all $y \in \Delta$, $y \in \mathcal{X}_i$.
- Therefore for any $y \in \Delta$ there exists a polynomials $P[z]$ such that $\mathsf{Ch}_{\mathcal{X}_i}[z] = (y+z)P[z]$.
- Compute polynomial $P[z]$ using polynomial long division.
- Compute $e(F_1, g^{\mathsf{Ch}_{\Delta-y}(s)})e(F_2, g^{r_i P(s)})$ which equals $e(g,g)^{\frac{1}{y+s}}$
- Output $[y, e(F_1, g^{\mathsf{Ch}_{\Delta-y}(s)})e(F_2, g^{r_i P(s)})]$

Therefore using F_j's, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_4] \leq \nu_4(\lambda)$ for some negligible function ν_4.

Therefore, the probability of successfully forging a membership proof is $Pr[E_1 \wedge E_2 \wedge E_3] \leq Pr[E_1] + Pr[E_2] + Pr[E_4] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_4(\lambda) \leq \nu''(\lambda)$ for some negligible function ν''. ∎

Lemma 8 *Under the q-SBDH assumption, there exists a negligible function* $\nu(\lambda)$ *such that* $Pr[A_2] \leq \nu(\lambda)$

Proof If Adv outputs a succesful forgery, it has to forge on at least one of the following steps: Step 2, 3, 4 and 5 in Algorithm insertQ. Let us denote the corresponding events as E_i for $i \in [1,4]$ respectively and bound the probability of each event.

By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

Next we bound $Pr[E_2]$. By Lemma 5, the probability that for some set $\mathcal{X}_i$, $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i) \leq \nu^i(\lambda)$ for some negligible function ν^i. Let $\nu(\lambda)$ be the maximum of the negligible functions $\nu^i(\lambda)$ for $i \in [1,k]$. Note that all the ν^i functions are independent of each other by By Lemma 4. Therefore, by union bound, $Pr[E_2] \leq k\nu(\lambda) = \nu_2(\lambda)$ for some negligible function ν_2 since $k = poly(\lambda)$. Therefore, we have, $Pr[E_2] \leq \nu_2(\lambda)$.

Now let us bound $Pr[E_3]$.

- At least for one j, it must be the case that $e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, W_{(\mathsf{answer},\mathcal{X}_j)}) = e(\mathsf{acc}(\mathsf{state}_j), g)$ but $\mathsf{answer} \not\subseteq \mathcal{X}_j$.
- This in turn implies, there exists at least one element $y \in \mathsf{answer}$ such that $y \notin \mathcal{X}_j$.
- This implies $(y+z)$ does not divide $\mathsf{Ch}_{\mathcal{X}_j}[z]$. Therefore there exists a polynomial $q[z]$ and a scalar c such that $\mathsf{Ch}_{\mathcal{X}_j}[z] = (y+z)q[z] + c$.
- Use polynomial long division to compute $q[z]$ and c.
- Output $[y, (e(W_{(\mathsf{answer},\mathcal{X}_j)}, g^{\mathsf{Ch}_{\{\mathsf{answer}-y\}}(s)})^{r_j^{-1}} e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\mathsf{answer},\mathcal{X}_j)}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_3] \leq \nu_3(\lambda)$ for some negligible function ν_3.

Now we look at $Pr[E_4]$.

- E_4 is the event when $\prod_{j\in[i_1,i_k]} e(F_j, W_{(\mathsf{answer},\mathcal{X}_j)}) = e(g,g)$ but the intersection $(\mathcal{X}_{i_1} - \mathsf{answer}) \cap (\mathcal{X}_{i_2} - \mathsf{answer}) \cap \ldots \cap (\mathcal{X}_{i_k} - \mathsf{answer}) \neq \phi$, i.e., there exists at least one element y that belongs to all the sets $(\mathcal{X}_j - \mathsf{answer})$ for $j \in [i_1, i_k]$.
- Therefore, there exists polynomials $\tilde{P}_j[z]$ such that $\mathsf{Ch}_{\mathcal{X}_j-\mathsf{answer}}[z] = (y+z)\tilde{P}_j[z]$ for $j \in [i_1, i_k]$.
- Compute polynomial $\tilde{P}_j[z]$ using polynomial long division.
- Compute $\prod_{j\in[i_1,i_k]} e(F_j, g^{r_j \tilde{P}_j(s)})$ which equals $e(g,g)^{\frac{1}{y+s}}$
- Output $[y, \prod_{j\in[i_1,i_k]} e(F_j, g^{r_j \tilde{P}_j(s)})]$

Therefore using F_j's, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_4] \leq \nu_4(\lambda)$ for some negligible function ν_4.

Therefore, we have $Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4] \leq Pr[E_1] + Pr[E_2] + Pr[E_3] + Pr[E_4] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'. ∎.

Lemma 9 *Under the q-SBDH assumption, there exists a negligible function* $\nu(\lambda)$ *such that* $Pr[A_3] \leq \nu(\lambda)$

Proof If Adv outputs a succesful forgery, it has to forge on at least one of the following steps: Step 3, 4, 5, 6 and 8 in Algorithm unionQ. Let us denote the corresponding events as E_i for $i \in [1,5]$ respectively and bound the probability of each event.

By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

Now let us look at $Pr[E_2]$. By Lemma 5, we have that the probability that for some set $\mathcal{X}_i$, $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i) \leq \nu^i(\lambda)$ for some negligible function ν^i. Let $\nu(\lambda)$ be the maximum of the negligible functions $\nu^i(\lambda)$ for $i \in [1,k]$. Note that all the ν^i functions are independent of each other by By Lemma 4. Therefore, by union bound, $Pr[E_2] \leq k\nu(\lambda) = \nu_2(\lambda)$ for some negligible function ν_2 since $k = poly(\lambda)$. Therefore, we have, $Pr[E_2] \leq \nu_2(\lambda)$.

Now let us bound $Pr[E_3]$.

- At least for one j, it must be the case that $e(\alpha_j, W_{(\mathcal{X}_j,\mathsf{answer})}) = e(g^{\mathsf{Ch}_{\mathsf{answer}}(s)}, g)$ but $\mathcal{X}_j \nsubseteq \mathsf{answer}$.
- This in turn implies, there exists at least one element $y \in \mathcal{X}_j$ such that $y \notin \mathsf{answer}$.
- This implies $(y+z)$ does not divide $\mathsf{Ch}_{\mathsf{answer}}[z]$. Therefore there exists a polynomial $q[z]$ and a scalar c such that $\mathsf{Ch}_{\mathsf{answer}}[z] = (y+z)q[z] + c$.
- Use polynomial long division to compute $q[z]$ and c.
- Output $[y, (e(W_{(\mathcal{X}_j,\mathsf{answer})}, g^{\mathsf{Ch}_{\{\mathcal{X}_j - y\}}(s)})^{r_j} e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\mathcal{X}_j,\mathsf{answer})}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_3] \leq \nu_3(\lambda)$ for some negligible function ν_3.

Now we look at $Pr[E_4]$.

Forgery for some node v in the tree with children v_1, v_2 implies $e(a(v), g) = e(a(v_1), a(v_2))$ but $a(v_1)$ is not the correct accumulation value at v_1 or $a(v_2)$ is not the correct accumulation value at v_2 (or both). Since the verification happens bottom up, and by definition, $a(v) = g^{r_1 r_2 \mathsf{Ch}_M(s)}$ therefore, it must be the case that for some leaf node v_l, $a(v_l)$ was not the correct accumulation for the corresponding set (say, $\mathcal{X}_i$), i.e., $a(v_l) \neq \mathsf{acc}(\mathcal{X}_i)$. But by Lemma 4, this probability is negligibly small. Therefore, following the same argument as for E_2, we have $Pr[E_4] \leq \nu_4(\lambda)$ for some negligible function ν_4.

Now let us estimate the probability of $Pr[E_5]$. We will show that if $e(W_{(\mathsf{answer},\tilde{U})}, g^{\mathsf{Ch}_{\mathsf{answer}}(s)}) = e(\mathsf{acc}(\tilde{U}), g)$, but $\mathsf{answer} \nsubseteq \tilde{U}$, then, $\mathcal{A}$ can break the q-SBDH assumption with non-negligible probability.

- Since $\mathsf{answer} \nsubseteq \tilde{U}$ it must be the case that there exists at least one element $y \in \mathsf{answer}$ such that $y \notin \tilde{U}$.
- This implies $(y+z)$ does not divide $\mathsf{Ch}_{\tilde{U}}[z]$. Therefore there exists a polynomial $q[z]$ and a scalar c such that $\mathsf{Ch}_{\tilde{U}}[z] = (y+z)q[z] + c$.
- Use polynomial long division to compute $q[z]$ and c.
- Output $[y, (e(W_{(\mathsf{answer},\tilde{U})}, g^{\mathsf{Ch}_{\{\mathsf{answer}-y\}}(s)})^{(\prod_{j \in [i_1, i_k]} r_j^{-1})} e(g, g^{-q(s)}))^{c^{-1}}]$

Therefore, using a forged witness $W_{(\mathsf{answer},\tilde{U})}$, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability. Hence it must be the case that $Pr[E_5] \leq \nu_5(\lambda)$ for some negligible function ν_5.

Therefore, we have $Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4 \wedge E_5] \leq Pr[E_1] + Pr[E_2] + Pr[E_3] + Pr[E_4] + Pr[E_5] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) + \nu_5(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'. ▮.

Lemma 10 *Under the q-SBDH assumption, there exists a negligible function* $\nu(\lambda)$ *such that* $Pr[A_4] \leq \nu(\lambda)$

Proof If Adv outputs a successful forgery, it has to forge on at least one of the following steps: Step 2, 3, 4, 5, 6 and 7 in Algorithm $\mathsf{differenceQ}$. Let us denote the corresponding events as E_i for $i \in [1,6]$ respectively and bound the probability of each event.

By Lemma 2 we know that there exists a negligible function ν_1 such that $Pr[E_1] \leq \nu_1(\lambda)$.

Now let us look at $Pr[E_2]$. By Lemma 5, we have that the probability that for some set $\mathcal{X}_i$, $\alpha_i \neq \mathsf{acc}(\mathcal{X}_i) \leq \nu^i(\lambda)$ for some negligible function ν^i. Let $\nu(\lambda)$ be the maximum of the negligible functions $\nu^i(\lambda)$ for $i \in [1,k]$. Note that all the ν^i functions are independent of each other by By Lemma 4. Therefore, by union bound, $Pr[E_2] \leq k\nu(\lambda) = \nu_2(\lambda)$ for some negligible function ν_2 since $k = poly(\lambda)$. Therefore, we have, $Pr[E_2] \leq \nu_2(\lambda)$.

By the soundness of subset query, there exists negligible functions ν_3, ν_4 such that $Pr[E_3] \leq \nu_3(\lambda)$ and $Pr[E_4] \leq \nu_4(\lambda)$.

Therefore, by the soundness of the zero knowledge protocol PK there exists negligible functions ν_4 such that $Pr[E_4] \leq \nu_4(\lambda)$.

In Step 4, the soundness of $W_{\mathsf{answer},\mathcal{X}_{i_1}}$ has been verified and in Step 5, it has been verified that $\mathsf{acc}(I)$ is $W_{\mathsf{answer},\mathcal{X}_{i_1}}$ raised to some exponent that the prover has knowledge of. By the knowledge soundness of PK, there exists an efficient extractor that can extract the exponent. Let us call this exponent x and let $\gamma = x/r_{i_2}$. $\mathcal{A}$ can us the extractor and compute $\mathsf{acc}(I)^{x^{-1}r_{i_1}{}^{-1}}$. $\mathsf{acc}(I)^{x^{-1}r_{i_1}{}^{-1}}$ can be though of as $g^{\mathsf{Ch}_{\mathcal{X}_{i_1}-\mathsf{answer}}(s)}$. Now $\mathcal{A}$ can compute polynomial $Q[z]$ and scalar c such that $\mathsf{Ch}_{\mathcal{X}_j} = (y+z)Q[z]+c$ and output
$[y,(e(W_{I,\mathcal{X}_j}, g^{\mathsf{Ch}_{\{\mathcal{X}_{i_1}-\mathsf{answer}-y\}}(s)})^{r_{j'}\gamma} e(g,g^{-Q(s)}))^{c^{-1}}] = [y, e(g,g)^{\frac{1}{y+s}}]$ (where j' is $(i_1,i_2)\setminus j$). This breaks the q-SBDH assumption. Hence we conclude that $Pr[E_5] \leq \nu_5(\lambda)$ for some negligible function ν_5.

Finally, we $Pr[E_6]$. Using γ as above, $\mathcal{A}$ can use forged F_j, $j \in [i_1,i_2]$ as in the proof of set intersection to come up with $[y', e(g,g)^{\frac{1}{y'+s}}]$ which breaks the q-SBDH assumption. Therefore using F_j's, $\mathcal{A}$ can break the q-SBDH assumption with the same success probability as Adv. Hence it must be the case that $Pr[E_6] \leq \nu_6(\lambda)$ for some negligible function ν_6.

Therefore, we have $Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4 \wedge E_5 \wedge E_6] \leq Pr[E_1] + Pr[E_2] + Pr[E_3] + Pr[E_4] + Pr[E_5] + Pr[E_6] \leq \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) + \nu_5(\lambda) + \nu_6(\lambda) \leq \nu'(\lambda)$ for some negligible function ν'. ■.

Therefore, by Lemma 7, Lemma 8, Lemma 9 and Lemma 10 we have $Pr[A_1 \cup A_2 \cup A_3 \cup A_4] \leq \nu(\lambda)$ for some negligible function ν. ■

Proof of Zero-Knowledge: We prove that the ZKASC construction satisfies the zero-knowledge property. Formally, we prove Lemma 11.

Lemma 11 *The ZKASC scheme satisfies zero-knowledge property as per Definition 8.*

Proof We write a simulator Sim that acts as follows:
- It first runs $\mathsf{GenParams}(1^\lambda)$ to receive bilinear parameters $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g)$
- Then it picks $s \xleftarrow{\$} \mathbb{Z}_p^*$ and sends $\mathsf{vk} := g^s, pub$ to Adv and saves s as its secret key in its state information $\mathsf{state}_\mathcal{S}$.
- Given vk, Adv comes up with a set collection $\mathbb{S}_0$ of its choice. Sim is given $|\mathbb{S}_0| = m$ which is the public information of the scheme.
- Sim picks m random numbers $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ for $i \in [1,m]$ and computes $\mathsf{acc}(\mathcal{X}_i) \xleftarrow{\$} g^{r_i}$ and saves all the r_i's in its state information.
- Then Sim invokes $\mathsf{AT.ATSetup}(\mathsf{sk}, (\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m)))$. Let $\mathsf{AT.ATSetup}(\mathsf{sk}, (\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m))) = (\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0)$.
- Sim saves $\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0$ in $\mathsf{state}_\mathcal{S}$ and sends digest_0 to Adv.

After this, when Adv adaptively asks for op. Sim maintains a list of previously asked queries and uses the proof units used by the Sim. If a query is repeated by Adv between two updates, then Sim uses the same proof units. Otherwise Sim does the following:

If $\mathsf{op} = \mathsf{update}$: Sim makes oracle call to $D()$ to check the validity of update. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, for all indices $i \in [1,m]$, Sim looks up its state information to see if any query touched index i since the last Update. Let $[i_1, \ldots, i_k]$ be the set of indices. Let $\mathbb{S}_t$ is the most recent set collection
- For all the indices $j \in [i_1, \ldots, i_k]$, Sim picks fresh random elements $r'_j \xleftarrow{\$} \mathbb{Z}_p^*$.
- It updates sets $\mathsf{acc}(\mathcal{X}_j) \leftarrow g^{r_j r'_j}$ and updates r_j with $r_j r'_j$ in its state information.
- Then it calls $\mathsf{AT.ATUpdate}(\mathsf{sk}, j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$. Let $\mathsf{AT.ATUpdate}(\mathsf{sk}, j, \mathsf{acc}'(\mathcal{X}_j), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t) = (\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}_t)$.

Return digest_{t+1} to Adv.

If $\mathsf{op} = \mathsf{subsetQ}$ **for** (Δ, i) Sim makes oracle call to $D()$ to check the validity of subsetQ. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, it makes oracle call to $\mathbb{S}_t$ (where $\mathbb{S}_t$ is the most recent set collection) and gets answer. Let answer $= 1$. Then Sim computes $W_{\Delta,\mathsf{state}_i} \leftarrow g^{\frac{r_i}{\mathsf{Ch}_\Delta(s)}}$ Else if answer $= 0$, Sim does the following:

- let $\Delta = \{x_1, \ldots, x_k\}$. Compute coefficients $[a_0, \ldots, a_k]$ of the polynomial Ch_ρ.
- Now let $x = GCD(r_i, \mathsf{Ch}_\Delta[z])$.
- Using Extended Euclidian algorithm, compute q_1, q_2 such that $q_1(\mathsf{Ch}_\Delta[z]/x) + q_2(r_i/x) = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$.
- Set $q_1'[z] := \frac{q_1[z]+\gamma r_i}{x}$ and $q_2'[z] := \frac{q_2[z]-\gamma \mathsf{Ch}_\Delta[z]}{x}$.
- Compute $F_1 \leftarrow g^{q_1'(s)}, F_2 \leftarrow g^{q_2'(s)}$.
- Set $W_{\Delta,\mathcal{X}_i} := (F_1, F_2)$

Set proof $= ([a_0, \ldots, a_k], W_{\Delta,\mathcal{X}_i}, \Pi_i, \alpha_i)$ Return (answer, proof) to Adv.

If $\mathsf{op} = \mathsf{IntersectionQ}$ **for indices** $i_1, \ldots, i_k$**:** Sim makes oracle call to $D()$ to check the validity of IntersectionQ. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, it makes oracle call to $\mathbb{S}_t$ (where $\mathbb{S}_t$ is the most recent set collection) and gets answer. Let answer $= \{x_1, \ldots, x_\rho\}$. Now Sim does the following:

- Compute coefficients $[a_\rho, \ldots, a_0]$ of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
- For each index $j \in [i_1, i_k]$ compute $W_{(\mathsf{answer},\mathcal{X}_j)} \leftarrow g^{\frac{r_j}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$.
- Now let $x = GCD(r_j)$ for $j \in [i_1, i_k]$ and let us denote as $\tilde{r_j} = r_j/x$.
- Note that $\tilde{r_j}$'s are co-prime. Using Extended Euclidian algorithm, compute q_j's such that $\sum_{j \in [i_1, i_k]} q_j \tilde{r_j} = 1$.
- Pick a random $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $q_j'[z] := (q_j + (\gamma \frac{r_{j+1}}{\mathsf{Ch}_{\mathsf{answer}}[z]})) \frac{\mathsf{Ch}_{\mathsf{answer}}[z]}{x}$.
- Compute $F_j \leftarrow g^{q_j'(s)}$.
- Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t)) = (\Pi_j, \alpha_j)$.
- Set Set proof $= ([a_\rho, \ldots, a_0], \{F_j, W_{(\mathsf{answer},\mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$.

Return (answer, proof) to Adv.

If $\mathsf{op} = \mathsf{unionQ}$ **for indices** $i_1, \ldots, i_k$**:** Sim makes oracle call to $D()$ to check the validity of unionQ. If $D()$ returns 0, Sim outputs $\perp$. Otherwise, it makes oracle call to $\mathbb{S}_t$ (where $\mathbb{S}_t$ is the most recent set collection) and gets answer. Let answer $= \{x_1, \ldots, x_\rho\}$. Now Sim does the following:

- Compute coefficients $[a_\rho, \ldots, a_0]$ of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
- For each index $j \in [i_1, i_k]$ compute $W_{(\mathcal{X}_j,\mathsf{answer})} \leftarrow g^{\frac{\mathsf{Ch}_{\mathsf{answer}}(s)}{r_j}}$.
- Build a binary tree on k leaves (representing the k indices in the query) as follows: For a leaf node, let $a(v) = \mathsf{acc}(\mathcal{X}_j) = g^{r_j}$. For each internal node, $a(v) = g^{r_1 r_2}$ where r_1 and r_2 are the corresponding blinding factors for v_1 and v_2 respectively.
- Compute $W_{(\mathsf{answer},\tilde{U})} \leftarrow g^{\frac{(\Pi_{j \in [i_1, i_k]} r_j)}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$
- Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_k]$. Let $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t)) = (\Pi_j, \alpha_j)$.
- proof $:= ([a_\rho, \ldots, a_0], \{a(v)\}_{v \in V(\mathsf{UT})}, W_{(\mathsf{answer},\tilde{U})}, \{W_{(\mathcal{X}_j,\mathsf{answer})}, \Pi_j, \alpha_j\}_{j \in [i_1, i_k]})$

Sim returns (answer, proof) to Adv.

If $\mathsf{op} = \mathsf{differenceQ}$ **for indices** i_1, i_2**:** Sim makes oracle call to $D()$ to check the validity of differenceQ. If $D()$ returns 0, Let answer $= \{x_1, \ldots, x_\rho\}$. Now Sim does the following:

- Compute coefficients $[a_\rho, \ldots, a_0]$ of the polynomial $\mathsf{Ch}_{\mathsf{answer}}$.
- Compute $W_{\mathsf{answer},\mathcal{X}_{i_1}} \leftarrow g^{\frac{r_{i_1}}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$.
- Pick $\gamma \xleftarrow{\$} \mathbb{Z}_p^*$ and set $\mathsf{acc}(I) := g^{\frac{r_{i_1} r_{i_2} \gamma}{\mathsf{Ch}_{\mathsf{answer}}(s)}}$.

- Compute $W_{I,\mathcal{X}i_1} \leftarrow g^{\frac{1}{r_{i_2}\gamma}}$
- Compute $W_{I,\mathcal{X}i_2} \leftarrow g^{\frac{1}{r_{i_1}\gamma}}$
- Now let $x = GCD(r_{i_1}, r_{i_2})$ and let us denote as $\tilde{r_j} = r_j/x$ for $j \in [i_1, i_2]$.
- Note that $\tilde{r_j}$'s are co-prime. Using Extended Euclidian algorithm, compute q_j's such that $\sum_{j\in[i_1,i_2]} q_j\tilde{r_j} = 1$.
- Pick a random $\beta \xleftarrow{\$} \mathbb{Z}_p^*$.
- Set $q'_{i_1}[z] := (q_{i_1}[z] + \beta r_{i_2})\frac{r_{i_1}r_{i_2}\gamma}{x}$.
- Set $q'_{i_2}[z] := (q_{i_2}[z] - \beta r_{i_1})\frac{r_{i_1}r_{i_2}\gamma}{x}$.
- Compute $F_j \leftarrow g^{q'_j(s)}$.
- Compute $\mathsf{PKproof}$ by invoking PK with $h = acc(I)$, $g' = W_{\mathsf{answer},\mathcal{X}_{i_1}}$ and the exponent as $r_{i_2}\gamma$.
- Invoke $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t))$ for all $j \in [i_1, i_2]$. Let $\mathsf{AT.ATQuery}(\mathsf{ek}_t, j, \mathsf{auth}(\mathbb{S}_t)) = (\Pi_j, \alpha_j)$.
- Set $\mathsf{proof} := ([a_\rho, \ldots, a_0], W_{(\mathsf{answer},\mathcal{X}_{i_1})}, \mathsf{acc}(I), \{F_j, W_{(I,\mathcal{X}_j)}, \Pi_j, \alpha_j\}_{j\in[i_1,i_2]}, \mathsf{PKproof})$.

Return $(\mathsf{answer}, \mathsf{proof})$ to Adv

It is easy to see that all the verification steps are satisfies. Observe that random values r are chosen independently after each update (and initial setup) in both cases and all the units of proof and digest have a one random blinding factor in the exponent. Hence they are distributed identically to random elements. This follows from a simple argument. Let $x, y, z \in \mathbb{Z}_p^*$ where x is a fixed element and $z = x + y$ or $z = xy$. Then z is identically distributed to y in $\mathbb{Z}_p^*$ in both cases. In other words, if y is picked with probability γ, then so is z. The same argument holds for elements in G and G_T.

Thus simulator Sim produces a view that is identically distributed to that produced by the challenger during $\mathbf{Real}_{\mathsf{Adv},k}$ and the simulation is perfect. ▮

We summarize the efficiency and the security properties of the ZKASC scheme in Theorem 2.

Theorem 2 *The* $\mathsf{ZKASC} = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Update}, \mathsf{UpdateServer}, \mathsf{Query}, \mathsf{Verify})$ *scheme satisfies the properties of completeness (Definition 6), soundness (Definition 7), and zero-knowledge (Definition 8). Let* $\mathbb{S} = \{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$ *be the set original set collection. Define* $M = \sum_{i\in m} |\mathcal{X}_i|$, $n_j = |\mathcal{X}_j|$, *and* $N = \sum_{j\in[i_1,i_k]} n_j$. *Let* k *be the number of group elements in the query input (for the subset query, it is the cardinality of a queried subset, and for the rest of the queries it is the number of set indices). Let* ρ *be the size of a query answer,* L *be the number of sets touched by the queries between updates* u_{t-1} *and* u_t, *and* $0 < \varepsilon < 1$ *be a constant chosen at the time of setup. We have:*

- KeyGen *has access complexity* $O(1)$;
- Setup *has complexity* $O(M + m)$;
- Update *and* $\mathsf{UpdateServer}$ *have complexity* $O(L)$;
- Query *and* Verify *have the following access complexity:*
  - *For is-subset, the access complexity is* $O(N\log^2 N \log\log N + m^\varepsilon \log m)$. *The proof size is* $O(k)$ *and the verification has complexity* $O(k)$.
  - *For set intersection, the access complexity is* $O(N\log^2 N \log\log N + km^\varepsilon \log m)$. *The proof size is* $O(\rho + k)$ *and the verification has complexity* $O(\rho + k)$.
  - *For set union, the access complexity is* $O(k\rho\log\rho + N\log N\log k + km^\varepsilon \log m)$. *The proof size is* $O(\rho + k)$ *and the verification has complexity* $O(\rho + k)$.
  - *For set difference, the access complexity is* $O(N\log^2 N \log\log N + m^\varepsilon \log m)$. *The proof size is* $O(\rho)$ *and the verification has complexity* $O(\rho)$.

8 Conclusion

In this work, we have introduced zero-knowledge as a privacy notion for cryptographic accumulators. Zero-knowledge is a very strong security property that requires that witnesses and accumulation values leak nothing about the accumulated set at any given point in the protocol execution, even after insertions and deletions. We have shown that zero-knowledge accumulators are located between zero-knowledge sets and the recently introduced notion of primary-secondary-resolver membership proof systems, as the they can be constructed (in a black-box manner) from the former and they can be used to construct (in a black-box manner) the latter.

We have then provided the first construction of a zero-knowledge accumulator that achieves computational soundness and perfect zero-knowledge. Using this construction as a building block, we have designed a zero-knowledge authenticated set collection scheme that handles set-related queries that go beyond set (non-)membership. In particular, our scheme supports set unions, intersections, and differences, thus offering a complete set algebra.

There are plenty of future research directions in the area that, we believe, can use this work as a stepping stone. For example, our construction is secure under a parametrized (i.e., q-type) assumption. It would be interesting to develop an alternative construction from a constant-size assumption (such as RSA). Another interesting research direction would be to come up with an alternative protocol for the set-difference operation, since the one we present here utilizes a Σ-protocol and, in order to make it non-interactive zero-knowledge, without compromising on efficiency, we must rely on the Fiat-Shamir heuristic. Finally, the relations between primitives presented here complement the related results of [NZ14], but are far from being complete; we believe there exist plenty of other directions to be explored.

Acknowledgments

We would like to thank Leonid Reyzin and Asaf Ziv for helpful discussions and comments. Research supported in part by the U.S. National Science Foundation under CNS grants 1012798, 1012910, and 1228485.

References

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 255–270, London, UK, UK, 2000. Springer-Verlag.

[ATSM09] Man Ho Au, Patrick P. Tsang, Willy Susilo, and Yi Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, pages 295–308, 2009.

[BA12] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 40–41, New York, NY, USA, 2012. ACM.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 56–73, 2004.

[BdM94] Josh Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, EUROCRYPT '93, pages 274–285, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[BLL00] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 9–17, New York, NY, USA, 2000. ACM.

[BP97] Niko Baric and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'97, pages 480–494, Berlin, Heidelberg, 1997. Springer-Verlag.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 234–252, Berlin, Heidelberg, 2008. Springer-Verlag.

[CFM08] Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, EUROCRYPT'08, pages 433–450, Berlin, Heidelberg, 2008. Springer-Verlag.

[CH10] Philippe Camacho and Alejandro Hevia. On the impossibility of batch update for cryptographic accumulators. In Michel Abdalla and PauloS.L.M. Barreto, editors, *Progress in Cryptology, LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 178–188. Springer Berlin Heidelberg, 2010.

[CHKO08] Philippe Camacho, Alejandro Hevia, Marcos Kiwi, and Roberto Opazo. Strong accumulators from collision-resistant hashing. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *Information Security*, volume 5222 of *Lecture Notes in Computer Science*, pages 471–486. Springer Berlin Heidelberg, 2008.

[CHL$^+$05] Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In *EUROCRYPT*, pages 422–439, 2005.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In StanisÅĆaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 481–500. Springer Berlin Heidelberg, 2009.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 61–76, 2002.

[CM11] Sanjit Chatterjee and Alfred Menezes. On cryptographic protocols employing asymmetric pairings - the role of revisited. *Discrete Applied Mathematics*, 159(13):1311–1322, 2011.

[CPPT14] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable set operations over outsourced databases. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 113–130, 2014.

[CT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers*, pages 143–159, 2010.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM Conference on Computer and Communications security*, CCS '13, pages 789–800, New York, NY, USA, 2013. ACM.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. Cryptology ePrint Archive, Report 2015/087, 2015.

[dMLPP12] Hermann de Meer, Manuel Liedel, Henrich C. Pöhls, and Joachim Posegga. Indistinguishability of one-way accumulators. In *Technical Report MIP-1210, Faculty of Computer Science and Mathematics (FIM), University of Passau*, 2012.

[dMPPS14] Hermann de Meer, Henrich C. Pöhls, Joachim Posegga, and Kai Samelin. Redactable signature schemes for trees with signer-controlled non-leaf-redactions. In Mohammad S. Obaidat and Joaquim Filipe, editors, *E-Business and Telecommunications*, volume 455 of *Communications in Computer and Information Science*, pages 155–171. Springer Berlin Heidelberg, 2014.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, ACNS '09, pages 125–142, Berlin, Heidelberg, 2009. Springer-Verlag.

[DT08] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008.

[FLZ14] Prastudy Fauzi, Helger Lipmaa, and Bingsheng Zhang. Efficient non-interactive zero knowledge arguments for set operations. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 216–233, 2014.

[FN02] Nelly Fazio and Antonio Nicolosi. Cryptographic accumulators: Definitions, constructions and applications. In *Technical Report. Courant Institute of Mathematical Sciences, New York University*, 2002.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 1–19, 2004.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194, London, UK, UK, 1987. Springer-Verlag.

[GGOT15] Esha Ghosh, Michael T. Goodrich, Olga Ohrimenko, and Roberto Tamassia. Fully-dynamic verifiable zero-knowledge order queries for network data. ePrint Report 2015/283, 2015.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304, 1985.

[GMY03] Juan A. Garay, Philip MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'03, pages 177–194, Berlin, Heidelberg, 2003. Springer-Verlag.

[GNP$^+$14] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably preventing DNSSEC zone enumeration. Cryptology ePrint Archive, Report 2014/582, 2014.

[GOT14] Esha Ghosh, Olga Ohrimenko, and Roberto Tamassia. Verifiable order queries and order statistics on a list in zero-knowledge. ePrint Report 2014/632, 2014. To appear in ACNS 2015.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[HN12] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *J. Cryptology*, 25(3):383–433, 2012.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pages 577–594, 2009.

[KPP$^+$14] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: faster verifiable set computations. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 765–780, 2014.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 241–257, 2005.

[Lip12] Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In *Proceedings of the 10th International Conference on Applied Cryptography and Network Security*, ACNS'12, pages 224–240, Berlin, Heidelberg, 2012. Springer-Verlag.

[Lis05] Moses Liskov. Updatable zero-knowledge databases. In *Proceedings of the 11th International Conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'05, pages 174–198, Berlin, Heidelberg, 2005. Springer-Verlag.

[LLX07] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, pages 253–269, 2007.

[LY10] Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In *Proceedings of the 7th International Conference on Theory of Cryptography*, TCC'10, pages 499–517, Berlin, Heidelberg, 2010. Springer-Verlag.

[MBKK04] Ruggero Morselli, Samrat Bhattacharjee, Jonathan Katz, and Peter J. Keleher. Trust-preserving set operations. In *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004*, 2004.

[Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, pages 218–238, 1989.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 397–411, 2013.

[MRK03] Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 80–91, 2003.

[MTGS01] Roberto Tamassia Michael T. Goodrich and Andrew Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. *DARPA Information Survivability Conference and Exposition II*, pages 68 – 82, 2001.

[MY04] Philip MacKenzie and Ke Yang. On simulation-sound trapdoor commitments. In Christian Cachin and JanL. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 382–400. Springer Berlin Heidelberg, 2004.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Proceedings of the 2005 international conference on Topics in Cryptology*, CT-RSA'05, pages 275–292, Berlin, Heidelberg, 2005. Springer-Verlag.

[NN00] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000.

[Nyb96a] Kaisa Nyberg. Commutativity in cryptography. In *1st International Trier Conference in Functional Analysis*. Walter Gruyter & Co, 1996.

[Nyb96b] Kaisa Nyberg. Fast accumulated hashing. In *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, pages 83–87, 1996.

[NZ14] Moni Naor and Asaf Ziv. Primary-secondary-resolver membership proof systems. Cryptology ePrint Archive, Report 2014/905, 2014.

[PSL76] F.P. Preparata, D.V. Sarwate, and ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB. *Computational Complexity of Fourier Transforms Over Finite Fields*. DTIC, 1976.

[PTT11] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 91–110, 2011.

[PTT15] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Authenticated hash tables based on cryptographic accumulators. *Algorithmica*, pages 1–49, 2015.

[San99] Tomas Sander. Efficient accumulators without trapdoor. In *In Second International Conference on Information and Communication Security ICICS'99*, pages 252–262. Springer, 1999.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[SPB$^+$12] Kai Samelin, Henrich C. Poehls, Arne Bilzhause, Joachim Posegga, and Hermann De Meer. Redactable signatures for independent removal of structure and content. In *Proc. Int. Conf. on Information Security Practice and Experience (ISPEC)*, volume 7232 of *LNCS*. Springer, 2012.

[Tam03] Roberto Tamassia. Authenticated data structures. In *Proc. European Symp. on Algorithms (ESA)*, volume 2832 of *LNCS*, pages 2–5. Springer, 2003.

[ZX14] Qingji Zheng and Shouhuai Xu. Verifiable delegated set intersection operations on outsourced encrypted data. *IACR Cryptology ePrint Archive*, 2014:178, 2014.

A The Accumulation Tree of [PTT11] with Extension to Batch Updates

Here we provide the detailed construction of the accumulation tree of Papamanthou et al. [PTT11]. Additionally, we discuss how to extend the update algorithm to support a batch of updates efficiently. These algorithms are called as subroutines during the execution of our authenticated set collections construction of Section 7.

Setup: $(\mathsf{auth}(\mathbb{S}_0), \mathsf{digest}_0) \leftarrow \mathsf{ATSetup}(\mathsf{sk}, (\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m)))$ Recall thatATSetup takes a secret key (sk) and a set of accumulation values $(\mathsf{acc}(\mathcal{X}_1), \ldots, \mathsf{acc}(\mathcal{X}_m))$ for a set collection $(\mathbb{S}_t)$ and builds an AT on top of it. This algorithm returns the authentication information for the set collection $(\mathsf{auth}(\mathbb{S}_t))$ and the root of the AT as the digest digest_t. The algorithm works as follows:

1. Pick a constant $0 < \varepsilon < 1$ and build an *accumulation tree* T as follows:
2. If v is a leaf corresponding to set $\mathcal{X}_i$, then set $d(v) = \mathsf{acc}(\mathcal{X}_i)^{\mathsf{sk}+i}$. Otherwise, set $d(v) = g^{\prod_{w \in C(v)}(\mathsf{sk}+h(d(w))}$ where $C(v)$ denotes the children of node v.
3. Set $\mathsf{auth}(\mathbb{S}_t) = \{d(v) | v \in V(T)\}$
4. Let root be the root of T. Set $\mathsf{digest}_t := d(\mathsf{root})$
5. Return $(\mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$

Query: $(\Pi_i, \alpha_i) \leftarrow \mathsf{ATQuery}(\mathsf{ek}_t, i, \mathsf{auth}(\mathbb{S}_t))$ Recall that ATQuery takes the evaluation key ek_t, authentication information for the set collection $\mathsf{auth}(\mathbb{S}_t)$ and a particular index of the set collection and returns the authentication path for that set, denoted as Π_i and the accumulation value of that set as α_i. The algorithm works as follows:

1. Let leaf $v_0 = \mathsf{acc}(\mathcal{X}_i)$ and $v_0, \ldots, v_l$ be the leaf to root path in T.
2. Let $\mathsf{w}_{j-1,j} = g^{\prod_{w \in \{C(v_j) - v_{j-1}\}}(\mathsf{sk}+h(d(w))}$ for $j \in [1, l]$.
3. Let $\pi_j = (d(v_{j-1}), \mathsf{w}_{j-1,j})$
4. Set $\Pi_i := \{\pi_j\}_{j \in [1,l]}$
5. Set $\alpha_i := \mathsf{acc}(\mathcal{X}_i)$.
6. (Π_i, α_i)

Update: $(\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}_i) \leftarrow \mathsf{ATUpdate}(\mathsf{sk}, i, \mathsf{acc}'(\mathcal{X}_i), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$ Recall that ATUpdate is the update algorithm that updates the accumulation value for a particular set $\mathcal{X}_i$, the authentication information for the set collection $\mathsf{auth}(\mathbb{S})$, and the root digest digest_t. This algorithm outputs the updated authentication information auth', the updated digest digest' and the update authentication information (in updinfo). The algorithm works as follows:

1. Let v_0 be the node corresponding to $\mathcal{X}_i$ and let $v_0, v_1, \ldots, v_l$ be the leaf to root path in T. Set $d'(v_0) := \mathsf{acc}(\mathcal{X}_i)$
2. For each node v_j on the path (i.e., $j = 1, \ldots, l$), set $d'(v_j) = g^{(\mathsf{sk}+d(v_{j-1}))^{-1}(\mathsf{sk}+d'(v_{j-1}))}$ where $d(v_{j-1})$ is the old value and $d'(v_{j-1})$ is the updated value.
3. Set $\mathsf{updinfo}_i = \{d'(v_{j-1})\}_{j \in [1,l]}$
4. Set $\mathsf{digest}' := d'(v_l)$.
5. Set $\mathsf{auth}' := \mathsf{auth}(\mathbb{S}_t) - \{d(v_{j-1})\}_{j \in [1,l]} + \{d'(v_{j-1})\}_{j \in [1,l]}$.
6. $(\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}_i)$.

Batch Update: $(\mathsf{auth}', \mathsf{digest}', \mathsf{updinfo}) \leftarrow \mathsf{ATUpdateBatch}(\mathsf{sk}, i_1, \mathsf{acc}'(\mathcal{X}_{i_1}), \ldots, i_k, \mathsf{acc}'(\mathcal{X}_{i_k}), \mathsf{auth}(\mathbb{S}_t), \mathsf{digest}_t)$ This algorithm is an extension of ATUpdate to support batch updates efficiently. Note this extension is our contribution.

1. The accumulation tree needs to be updated bottom-up starting at the leaf level. All the updates on a level have to be completed before proceeding to the next level. The update procedure is the same as ATUpdate with the difference that instead of updating one leaf, here all the leaves are updated together.
2. At the leaf level, update all the leaves with $\mathsf{acc}'(\mathcal{X}_j)$ where $\mathsf{acc}'(\mathcal{X}_j)$ is the new accumulated value for $\mathcal{X}_j$, for $j \in [i_1, i_k]$.
3. Proceed to update the levels bottom up as described in ATUpdate.
4. At the end of update, return the updated authentication information auth', the updated root digest' and updinfo that contains the updated authentication path for each updated leaf $\mathcal{X}_j$.

Verify: $(\mathsf{accept/reject}) \leftarrow \mathsf{ATVerify}(\mathsf{vk}, \mathsf{digest}_t, i, \Pi_i, \alpha_i)$ Recall that ATVerify is the verification algorithm that takes the verification key of the scheme vk, digest of the set collection digest_t and a particular set index i along with its authentication path (Π_i) and accumulation value (α_i) as input and returns accept if the α_i is indeed the accumulation value of the i^{th} set of the collection. It returns reject otherwise. The algorithm works as follows:

1. Parse Π_j as $\{\pi_1, \ldots, \pi_l\}$.
2. Parse as π_j as (β_j, γ_j).
3. reject if any of the following is true:
   (a) $e(\beta_1, g) \neq e(\alpha_i, (\mathsf{vk})g^i)$
   (b) For some $j \in [2, l]$: $(\beta_j, g) \neq e(\gamma_{j-1}, (\mathsf{vk})g^{h(\beta_{j-1})})$
   (c) $e(\mathsf{digest}_t, g) \neq e(\gamma_l, (\mathsf{vk})g^{h(\beta_l)})$
4. accept otherwise.