Batching, Aggregation, and Zero-Knowledge Proofs in Bilinear Accumulators

Shravan Srinivasan
University of Maryland
College Park, USA

Ioanna Karantaidou
George Mason University
Fairfax, USA

Foteini Baldimtsi
George Mason University
Fairfax, USA

Charalampos Papamanthou
Yale University
New Haven, USA

ABSTRACT

An accumulator is a cryptographic primitive that allows a prover to succinctly commit to a set of values while being able to provide proofs of (non-)membership. A batch proof is an accumulator proof that can be used to prove (non-)membership of multiple values simultaneously.

In this work, we present a zero-knowledge batch proof with constant proof size and constant verification in the Bilinear Pairings (BP) setting. Our scheme is 16× to 42× faster than state-of-the-art SNARK-based zero-knowledge batch proofs in the RSA setting. Additionally, we propose protocols that allow a prover to aggregate multiple individual non-membership proofs, in the BP setting, into a single batch proof of constant size. Our construction for aggregation satisfies a strong soundness definition—one where the accumulator value can be chosen arbitrarily.

We evaluate our techniques and systematically compare them with RSA-based alternatives. Our evaluation results showcase several scenarios for which BP accumulators are clearly preferable and can serve as a guideline when choosing between the two types of accumulators.

CCS CONCEPTS

• **Security and privacy → Cryptography**; • **Theory of computation → Cryptographic primitives**; **Cryptographic protocols**.

KEYWORDS

Accumulators, bilinear pairings, proof aggregation, zero-knowledge

ACM Reference Format:
Shravan Srinivasan, Ioanna Karantaidou, Foteini Baldimtsi, and Charalampos Papamanthou. 2022. Batching, Aggregation, and Zero-Knowledge Proofs in Bilinear Accumulators. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560676

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560676

1 INTRODUCTION

An *accumulator* is an authenticated data structure for a set of elements. It allows a *prover* to provide a succinct binding digest to a set of elements and to generate a short proof of membership or non-membership for any element in the accumulator domain. A *verifier* can efficiently check the proof of (non-)membership using the digest without requiring access to the entire set. Accumulators have found numerous applications including timestamping [8], fail-stop signature schemes [6], anonymous credentials [2, 4, 15, 16], cloud storage [48, 57] and more recently, stateless and privacy-preserving cryptocurrencies [10, 21, 35].

Batching and aggregation. In traditional applications, accumulators have been used in a setting where the prover had to provide (non-)membership proofs for a single element at a time. However, in emerging applications, such as cryptocurrencies, a prover must simultaneously prove (non-)membership of multiple elements. Naively, the prover could include individual proofs for each element, but this imposes high bandwidth usage and computational cost on the verifier. A better approach is *a batch proof*, that is, a succinct proof for multiple elements, which can be used to efficiently and simultaneously prove (non-)membership of multiple elements. For example, in UTXO-based stateless blockchains [10, 21], all transactions are accompanied by a proof of membership in the UTXO set. If a block proposer naively includes all individual proofs for validation (instead of a single batch proof), the size of the blocks transmitted across the network increases along with the computational overhead on the verifiers.

Let X be the set of elements in an accumulator. A batch membership proof for a set of elements $I \subseteq X$ can be computed: (1) by using the *trapdoor* (e.g., factors of the modulus in the RSA setting), or (2) *from scratch* using *all* the elements in X, or (3) by *aggregating* previously computed individual proofs of elements in I. Unfortunately, computing the batch proof using the trapdoor is impractical as it requires a trusted accumulator manager to hold the trapdoor. Furthermore, if the trapdoor is compromised, an adversary can forge proofs at-will. Computing the batch proof using the entire set is also impractical in a *distributed setting*, as nodes may not have access to the entire set or the trapdoor or a trusted accumulator manager. Moreover, updates to the accumulator arrive in batches (e.g., batch of transactions in a block). In comparison with other approaches, computing the batch proof by *aggregating* individual proofs is useful and relevant in the distributed setting as it does not require the trapdoor or the entire set. This brings us to the first question: *Is it possible to efficiently aggregate individual (non-)membership proofs?*

Zero-knowledge proofs. Proofs of (non-)membership for accumulators are often required in applications where privacy is critical. As an example, consider the application of anonymous credentials for authentication where valid (i.e., non-revoked) credentials are stored in an accumulator. When users wish to prove something about their credential embedded attributes, they also need to prove that their credential is valid via a proof of membership. Such proofs are usually done in a zero-knowledge (ZK) fashion in order to guarantee unlinkability between proving sessions and specific user credentials. More concretely, let x be a user credential accumulated in A. The user will compute a commitment $c = Commit(x)$, and prove, in zero-knowledge, membership of the committed value x. However, being able to simultaneously prove (non-)membership of a set of elements, I, is an important property in scenarios where a user/organization controls multiple credentials. Additionally, a prover may also need to argue that this set I is of at least some size d while hiding I. For instance an organization holding $|I| \geq d$ valid credentials might want to prove in ZK that it can cast up to d votes. Such questions become even more relevant in recently developed decentralized identity systems [1, 30]. This brings us to a second question: *Is it possible to efficiently prove knowledge of a set I that is a subset of/disjoint from X, without revealing I? And reveal a lower bound of $|I|$, if needed?*

Batching, aggregation, and ZK in RSA accumulators. One of the most popular accumulator instantiations is the *RSA accumulator* [8, 16, 32]. Given a set X of prime numbers $(x_1, \ldots, x_n)$, one can define the accumulator A_X as the RSA group element $g^{\prod x_i}$, and a membership proof w of $y \in X$ as the accumulator of $X \setminus \{y\}$, which can be verified by checking whether w^y equals A_X. Boneh et al. [10] defined batch proofs for the RSA accumulator and also provided aggregation algorithms for both membership and non-membership RSA proofs. The resulting batch proofs are non-interactive and of constant size. They also present a Proof-of-Exponentiation protocol (PoE) to concretely speed up batch verification by reducing the number of group operations from $O(|I|)$ to a constant. Without PoE, the verifier would need to perform a large exponentiation that grows with the number of elements in the proof. On the ZK front, Ozdemir et al., introduced *improved* SNARK-friendly techniques to batch prove (non-)membership in the RSA setting [39]. Subsequently, Campanelli et al. adopted a "hybrid" approach where they prove the batch membership without SNARKs and prove that elements of the batch are from the prime domain using SNARKs [18]. However, their accumulator does not support both membership and non-membership simultaneously.

While RSA accumulators have been used in many applications, they do present some crucial limitations:

- First, RSA group elements are large and this affects verification time and proof sizes.
- Second, they only support accumulation of elements that reside in a prime domain. Thus, they cannot be directly used for the accumulation of arbitrary elements. [1]

These problems are fundamental and limit the use of RSA accumulators in the batch setting regardless of the privacy concerns.

[1]While there exist techniques to map arbitrary elements to primes (i.e., hash to primes [10]), such mappings can harm the soundness of the accumulator if not carefully implemented during verification, especially if the mappings are not 1-1.

The case for bilinear accumulators. As opposed to RSA accumulators, *bilinear-pairing accumulators* (BP) [38, 57] have much smaller proofs and faster exponentiations. Moreover, BP accumulators can support the accumulation of arbitrary elements, making them directly applicable to a broader set of applications. Given a set X of arbitrary numbers $(x_1, \ldots, x_n)$, the accumulator A is: $g^{\prod(s+x_i)}$, where g is a prime order group generator and s is a secret *trapdoor*. The membership proof w of $y \in X$ is the accumulator of $X \setminus \{y\}$, which can be verified by using the A_X and the pairing operator. As highlighted above, we are interested in accumulators for the distributed setting, i.e., where nobody can have access to the secret trapdoor s or the accumulated set X. Existing batching approaches in bilinear setting are only secure under a weak soundness definition [19, 26, 44] (with [44] additionally missing security proof), or require large public parameters [57]. Moreover, to the best of our knowledge, no prior work allows for efficient aggregation of non-membership proofs or efficient zero-knowledge (non-)membership proofs in the batch setting for *trapdoorless* accumulators (no accumulator operation except the setup requires the trapdoor, and the trapdoor is destroyed after setup if used). This is the main focus of our work.

Contributions. In the distributed setting, we make the following contributions to trapdoorless BP accumulators:

(1) We formally prove soundness of batch membership and non-membership proofs for the Nguyen [38] accumulator under the Uber assumption [7] (§4). Our proof holds for a *stronger* definition of soundness than those considered in prior works, since we *do not* assume that the accumulator is well-formed and we allow the adversary to pick the accumulator value (Definition 3.2). We remark that this strong definition is *crucial* for many modern applications of accumulators, especially in the distributed/blockchain setting, as it is not always realistic to assume that the accumulator value is well-formed.
(2) We design the first efficient ZK scheme that can prove batch (non-)membership of BP accumulator using the knowledge-of-exponent assumption (§7). Moreover, we show how to additionally reveal a lower bound on the size of the batch witness without revealing the elements of the batch proof. Asymptotically, our proof size is constant, the verification cost is constant, and the prover is $O(d)$, where d is the size of the batched subset.
(3) We propose a new algorithm to aggregate individual non-membership proofs into a single constant sized proof (§5) and provide a constant-time algorithm to update the extended Euclidean based individual non-membership proofs (§6).
(4) We perform an experimental evaluation and comparison of the RSA and BP accumulators in both ZK and non-ZK setting (§9). Concretely, we observe that in the ZK setting, both the prover and the verifier is at least 16× and 7× faster than RSA based baseline, respectively.
In the non-ZK setting, we benchmark the batching and aggregation of accumulator proofs. For the first time, we explicitly take the mapping cost (from arbitrary domain to accumulator domain) into account. We observe that membership and non-membership proofs in BP accumulators are 2.5× to 5×

smaller and 3.5× smaller than the RSA accumulators, respectively. Moreover, verification of aggregated BP accumulator proofs is on an average 4× faster than the RSA accumulators.

(5) Finally, in §8, we propose a PoE protocol in Elliptic Curve (EC) groups that help us speed up batch verification. Our PoE can prove exponentiation of an arbitrary group element and it concretely saves one exponentiation for the prover when compared to concurrent works [23, 44]. We use our PoE to verify exponentiation of a group element by ℓ-degree polynomial without having to perform $O(\ell)$ group exponentiations, only $O(\ell)$ operations in the field.

Implications of our results and evaluations. As indicated above, batch *membership* proofs in BP accumulators, clearly outperform their RSA counterparts in the size of the proofs (2.5× to 5×). This makes our protocols very appealing to applications where communication cost is critical.

An example of such an application is the extension of ℓ-bit Byzantine agreement (BA) and broadcast protocols (BB) [37] where bilinear-accumulators are used to obtain state-of-the-art communication costs for ℓ-bit BA/BB. Our batching techniques can improve their so called: "distribute phase" by batching the proofs sent between the participants which would result in a constant size proof and significantly reduce their communication costs.

In the ZK setting, the advantages of BP accumulators are apparent in both prover and verification costs. In decentralized identity systems with privacy, issuers sign the users attributes/identity and these signatures (aka the credential) are kept in an accumulator. Later, users should be able to perform batch proofs of (non-)membership for their stored credentials. Typically, the underlying signature schemes output integers (dlog, RSA-based schemes) or group elements that can be trivially mapped to integers (ECC schemes). Thus, one can immediately utilize a BP accumulator. If an RSA accumulator is used to hold the credentials, a $\mathsf{H}_{\mathsf{prime}}$ function (maps to prime) needs to be applied to each element individually and a ZK proof of primality is required. Ozdemir et al. [39] show how the use of Pocklington certificates reduce the cost of proving primality in ZK, but it only reduces the number of Miller-Rabin primality tests. This still results in non-constant verification time. This hashing operation, along with a primality test, needs to also be included in the ZK proof. This makes the computation of the ZK proof much more complex (e.g., if one uses zk-SNARKs or other specialized ZK proofs for non-algebraic statements [3]).

Limitations. While both BP and RSA accumulators require a trusted setup phase to compute the public parameters, the size of BP parameters is significant: 18 MiB for 2^{17} elements (although not necessarily needed for verification). These parameters grow even more (3×) in our ZK setting due to the use of the knowledge-of-exponent assumption. Finally, it should be noted that in BP accumulators (as opposed to RSA), it is unknown how to add or generate proofs of (non-)membership without the knowledge of the entire accumulated set X.

2 RELATED WORK

Based on the use of *trapdoors* for accumulator operations, accumulators can be: *trapdoor-based* or *trapdoorless*. In a *trapdoor-based* accumulator, a trusted entity, called the *accumulator manager*, holds some secret trapdoor information, which allows the entity to efficiently perform accumulator operations. A *trapdoorless* accumulator on the other hand, operates without the trapdoor and if a trapdoor is used during setup, it is later destroyed.

We classify prior works into three broad categories: (1) accumulators based on hash functions, (2) accumulators in hidden-order groups, and (3) accumulators in known-order groups.

Hash-based. Accumulators built based on Merkle tree [5, 34] or Bloom-filters [56] do not support batching, aggregation, and ZK proof of batch (non-)membership without general purpose tools such as SNARKs, unlike the techniques proposed in this work.

Hidden-order groups. In the single-proof setting, Camenisch and Lysyanskaya [16] proposed the first *dynamic* accumulator (supports both set difference and set union without requiring to fully recompute the accumulator from scratch) secure under *strong RSA* assumption based on prior accumulator constructions [6, 8]. However, their construction is not in the trapdoorless setting as the trapdoor is used to delete elements from the accumulators. Li et al. [32] proposed the first *universal* accumulator (supports both membership and non-membership proofs) by generalizing the Camenisch and Lysyanskaya accumulators [16] to support non-membership proofs under the strong RSA assumption. They also provided efficient algorithms to update non-membership proofs on changes to the accumulated set.

Boneh et al. [10] support batching and aggregation of membership and non-membership proofs in the distributed and trapdoorless setting. They also leverage their contributions to realize a stateless blockchain [10, 21] in the UTXO setting. However, RSA based constructions have larger proof size and verification cost when compared to the BP-based constructions.

Known-order groups. In the single-proof setting, Nguyen proposed the first accumulator [38] using bilinear-maps and based on the t-strong bilinear Diffie-Hellman assumption. In a later work, Damgård et al. [24] and Au et al. [4] extended the accumulator construction by Nguyen to support constant-sized non-membership proof for a *single* element. This non-membership proof scheme relies on Polynomial Remainder Theorem (PRT) to prove non-membership [55]. However, this construction does not extend to a constant-sized batch non-membership proofs or consider efficient aggregation of non-membership proofs. In our work, we study constant-sized batch proofs and present aggregation techniques for the greatest common divisor (GCD) based non-membership construction, rather than the PRT-based.

Thakur [44] propose batching techniques for BP accumulators (for weak soundness and without rigorous analysis). In their latest version (Sept, 2021), they also propose two PoE protocols. However, one of the PoE approaches assumes that it is possible to exponentiate an arbitrary base to the trapdoor (which is only feasible in the trapdoor-based setting). Connolly et al. [23] also propose a PoE protocol based on [44], under a different assumption (q-co-GSDH). Compared to [44] and [23], our PoE is concretely one exponentiation fewer under the adaptive variant of q-SDH. Finally, neither [44] nor [23] consider aggregation or ZK batch proofs.

Prior works [19, 23, 26, 41] define a batch proof for multiple elements. However, these works either: (1) assume in their soundness definition that the accumulator is well-formed and honestly computed or (2) does not consider aggregation [26] or (3) rely on an accumulator manager [49]. On the other hand, our batch proofs are sound even for an adversarially chosen accumulator value and our techniques support aggregation in the trapdoorless setting.

Camenisch et al. [15] and Zhang et al. [57] proposed BP accumulators that are algebraically quite different from [38]. These schemes have parameters that are equal to the size of the accumulator domain or more.

Vector Commitment (VC). A VC is a primitive closely related to accumulators, that provides a succinct commitment and positional binding to an *ordered* set of values [20]. Catalano and Fiore proposed a technique to transform a VC into an accumulator [20]. However, this approach results in an accumulator scheme with public parameters that are equal to the size of the accumulator domain [20, 21, 27, 31, 47].

Tomescu et al. proposed aggregatable sub-vector commitments in the bilinear-map setting based on Lagrange polynomials and KZG commitments [29, 47]. Their work uses the partial fraction decomposition technique [53] to aggregate VC proofs. We adopt techniques to aggregate proofs in BP accumulators. Campanelli et al. [17] defined incremental aggregation, i.e., aggregation of aggregated proofs for a RSA-based VC. Although inspired by [10], their work cannot be efficiently directly applied to accumulators.

Zero-knowledge (ZK). There are known techniques to efficiently achieve ZK for a single element in the BP [4] and the RSA [9, 16] setting. Naively, a batch ZK RSA proof corresponds to proving that the exponent is the product of multiple distinct elements, which results in a linear size proof. Recent work by Campanelli et al. [18] constructs ZK proofs of batch membership for RSA using SNARKs and get a constant size proof. Their construction can be transformed to prove non-membership (it corresponds to membership of intervals) but it cannot support both membership and non-membership at the same time without having an accumulator manager holding the trapdoor.

3 PRELIMINARIES

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ denote groups of prime order p and let g_1, g_2 be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Also, let $\mathbb{Z}_p$ be a field of prime order and $\mathbb{Z}_p[x]$ be a polynomial ring. We denote the degree of polynomial $I(x) \in \mathbb{Z}_p[x]$ as $\deg(I)$.

Bilinear pairing. A bilinear pairing is an *efficiently* computable map, $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, satisfying the following properties:

- **bilinearity:** $\forall (P, Q, a, b) \in (\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{Z}_p \times \mathbb{Z}_p)$: $e(P^a, Q^b) = e(P^a, Q)^b = e(P, Q^b)^a = e(P, Q)^{ab}$
- **non-degeneracy:** $e(g_1, g_2) \neq 1$

We denote a pairing instance $\mathsf{bp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{BilGen}(1^\lambda)$. When $\mathbb{G}_1 = \mathbb{G}_2 = \langle g \rangle$, the pairing is called *symmetric* and is denoted as $(p, \mathbb{G}, \mathbb{G}_T, e, g)$.

Partial fraction decomposition (PFD). A rational polynomial can be decomposed into simpler fractions [53]. Concretely, let $A(x) = \prod_{i \in I}(x + a_i)$ be a polynomial and let $A'(x)$ be the first derivative of $A(x)$ with respect to x. Then,

$$\frac{1}{A(x)} = \sum_{i \in I} \frac{1}{A'(-a_i)(x + a_i)}.$$

Polynomial remainder theorem (PRT). When a polynomial $A(x)$ is divided by $(x + r)$, the remainder is the evaluation of $A(x)$ at $-r$. Let $q(x)$ denote the quotient polynomial [55]. Then,

$$A(x) = q(x)(x + r) + A(-r).$$

Bézout's theorem (for polynomials). Given $f(x), g(x) \in \mathbb{F}[x]$, there exists $p(x), q(x), h(x) \in \mathbb{F}[x]$ such that [54]:

$$p(x)f(x) + q(x)g(x) = \gcd(f(x), g(x)) = h(x)$$

Moreover, $\deg(p) < \deg(g) - \deg(h)$ and $\deg(q) < \deg(f) - \deg(h)$.

Pedersen vector commitment (PVC). Given a group $\mathbb{G}_1$ or $\mathbb{G}_2$ of prime order p and a vector $\vec{c} = (c_0, \ldots, c_t) \in \mathbb{Z}_p^t$. Let $(g_1, g_2, \ldots, g_t, h) \in \mathbb{G}_1^{t+1}$ or $\mathbb{G}_2^{t+1}$ generators where $\log_{g_i} g_j, i \neq j$ relationship is unknown. In order to commit to the vector $\vec{c}$, one has to pick $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $\mathsf{PVC}(\vec{c}, r) = h^r g_0^{c_0} g_1^{c_1} \ldots g_t^{c_t}$.

The Pedersen commitment scheme [42] (and its vector generalization) is homomorphic, perfectly hiding and computationally binding under the discrete logarithm assumption.

Zero-knowledge proofs. A ZK proof for a relation $\mathcal{R}(x; w)$, where x is the public statement and w is the witness, is a set of algorithms (Setup, Prove, Verify) with the following syntax:

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \rightarrow \mathsf{pp}$: given the security parameter λ and the relation, outputs parameters pp.
- $\mathsf{Prove}(\mathsf{pp}, x, w) \rightarrow \pi$: given the parameters pp, a statement x and a witness w, it returns a proof π for $\mathcal{R}(x; w)$.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{x}, \pi) \rightarrow b \in \{0, 1\}$: given the parameters pp, the statement x and the proof π, it accepts or rejects the proof.

Properties. A ZK proof has to be *correct, sound and zero-knowledge*. Correctness means that if $(x, w) \in \mathcal{R}$, then $\mathsf{Verify}(\mathsf{pp}, \mathsf{x}, \pi) = 1$ with overwhelming probability. We support *knowledge soundness* - if a proof passes verification, then there exists a polynomial time algorithm (the extractor) which by interacting with the prover can extract the witness. Finally, zero-knowledge implies that the proof leaks nothing about the witness, i.e., there exists a simulator with access only to the public statement which can output a valid proof.

3.1 Cryptographic accumulators

An accumulator is a cryptographic primitive that supports a succinct binding commitment to an arbitrary set of values. In this work, we consider trapdoorless, dynamic, and universal accumulators. Following the definitions and notation from [5] and [10], our definitions refer to the *batch* setting, where I is a set of elements. The traditional accumulator algorithms can be derived for $I = \{x\}$.

Definition 3.1 (Trapdoorless Accumulator). Let $\mathcal{D}$ be the domain of the accumulated elements, and consider set $X \in \mathcal{D}^{|X|}$ and set $I \in \mathcal{D}^{|I|}$ (where each element from sets X and I is in $\mathcal{D}$). An accumulator consists of the following PPT algorithms:

(1) $(\mathsf{pp}, \mathcal{D}) \leftarrow \mathsf{Acc.Setup}(1^\lambda)$: Takes security parameter λ, returns the public parameters pp and $\mathcal{D}$ of the accumulated set. A *trusted* entity may use a secret trapdoor to generate pp. The trapdoor

is then destroyed by the setup. In the rest of the document, we assume that $\mathsf{pp} = (\mathsf{pp}, \mathcal{D})$.

(2) $A_X \leftarrow \mathsf{Acc.Commit}_{\mathsf{pp}}(X = \{x_1, \ldots, x_{|X|}\})$: Takes set $X \in \mathcal{D}^{|X|}$, outputs the accumulator digest A_X.

(3) $A'_X \leftarrow \mathsf{Acc.Add}_{\mathsf{pp}}(A_X, X, I)$: Adds set $I \in \mathcal{D}^{|I|}$, $I \cap X = \emptyset$, to the accumulator and returns the new digest A'_X.

(4) $A'_X \leftarrow \mathsf{Acc.Del}_{\mathsf{pp}}(A_X, X, I)$: Removes set I, $I \subseteq X$, from the accumulator and returns the new accumulator value, A'_X.

(5) $\pi_I \leftarrow \mathsf{Acc.MemProve}_{\mathsf{pp}}(X, I)$: Generates a membership proof for set I, $I \subseteq X$.

(6) $\{0,1\} \leftarrow \mathsf{Acc.MemVerify}_{\mathsf{pp}}(A_X, I, \pi_I)$: Returns 1 if the membership proof π_I, for the set I, $I \subseteq X$, is valid against the accumulator digest, A_X.

(7) $\overline{\pi}_I \leftarrow \mathsf{Acc.NonMemProve}_{\mathsf{pp}}(X, I)$: Generates non-membership proof for the set I, $I \cap X = \emptyset$, disjoint from the accumulated set.

(8) $\{0,1\} \leftarrow \mathsf{Acc.NonMemVerify}_{\mathsf{pp}}(A_X, I, \overline{\pi}_I)$: Returns 1 if the non-membership proof $\overline{\pi}_I$, of the set I, $I \cap X = \emptyset$, is valid against the accumulator digest, A_X.

(9) $\pi_I \leftarrow \mathsf{Acc.AggMem}_{\mathsf{pp}}(A_X, I, \{\pi_1, \cdots, \pi_{|I|}\})$: Combines individual membership proofs $\{\pi_1, \cdots, \pi_{|I|}\}$ into a single aggregated proof.

(10) $\overline{\pi}_I \leftarrow \mathsf{Acc.AggNonMem}_{\mathsf{pp}}(A_X, I, \{\overline{\pi}_1, \cdots, \overline{\pi}_{|I|}\})$: Combines individual non-membership proofs $\{\overline{\pi}_1, \cdots, \overline{\pi}_{|I|}\}$ into a single aggregated proof.

The following algorithms update a (non-)membership proof of a single element after changes to the accumulated set.

(11) $\pi'_y \leftarrow \mathsf{Acc.MemProofUpdOnAdd}_{\mathsf{pp}}(A_X, X, y, \pi_y, I)$: Updates the membership proof π_y of element y on addition of set I ($y \notin I, X = X \cup I$) to the accumulator.

(12) $\pi'_y \leftarrow \mathsf{Acc.MemProofUpdOnDel}_{\mathsf{pp}}(A_X, A'_X, X, I, y, \pi_y)$: Updates the membership proof π_y of element y on deletion of set I, $X = X \setminus I$, from the accumulator.

(13) $\overline{\pi}'_y \leftarrow \mathsf{Acc.NonMemProofUpdOnAdd}_{\mathsf{pp}}(A_X, X, y, \overline{\pi}_y, I)$: Updates the non-membership proof $\overline{\pi}_y$ of element y disjoint from set X on addition of set I ($y \notin I, X = X \cup I$) to the accumulator.

(14) $\overline{\pi}'_y \leftarrow \mathsf{Acc.NonMemProofUpdOnDel}_{\mathsf{pp}}(A_X, A'_X, X, I, y, \overline{\pi}_y)$: Updates the non-membership proof $\overline{\pi}_y$ of an element y disjoint from set X on deletion of set I ($X = X \setminus I$) from the accumulator.

3.2 Correctness and soundness

The basic security property for accumulators is *soundness* (sometimes called *undeniability* [33]) which states that an adversary *cannot* construct an accumulator A and a set I for which both π_I and $\overline{\pi}_I$ are simultaneously valid. Below we state *strong soundness* (also found in [10]), which allows the adversary to create the accumulator *without* revealing the accumulated set X to the challenger.

Definition 3.2 (Soundness). For any PPT adversary $\mathcal{A}$, it holds:

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Acc.Setup}(1^\lambda) \\ (A, I, \pi_I, \overline{\pi}_I) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{Acc.MemVerify}_{\mathsf{pp}}(A, I, \pi_I) = 1 \wedge \\ \mathsf{Acc.NonMemVerify}_{\mathsf{pp}}(A, I, \overline{\pi}_I) = 1 \end{array}\right] = \mathsf{negl}(\lambda)$$

Since no trapdoor is needed for algorithms $\mathsf{Acc.Add}_{\mathsf{pp}}$, $\mathsf{Acc.Del}_{\mathsf{pp}}$, $\mathsf{Acc.MemProve}_{\mathsf{pp}}$, $\mathsf{Acc.NonMemProve}_{\mathsf{pp}}$, $\mathcal{A}$ can adaptively update the accumulator and construct honest proofs during the game before coming up with the accumulator value A and proofs $\pi_I, \overline{\pi}_I$.

We formally define correctness of (non-)membership proofs, the correctness of aggregation, and present the accumulator soundness proof under a weak definition in the extended version of our paper. This weak soundness definition assumes that the accumulator value is honestly generated for the set X. Our construction is secure under this weak definition of soundness using the t-SBDH assumption. The weak soundness definition is useful if one wants to avoid the Uber assumption [7] (under which our protocols satisfy strong soundness).

3.3 Accumulator based on bilinear-maps

In this subsection, we review the standard accumulator based on bilinear-pairing (BP) introduced by Nguyen [38] and improved by [24, 40]. Here, we present the classic BP construction that operates on *individual* proofs. In §4 and §5 we show how to *batch* and *aggregate* proofs in the BP setting, and in §6 we present a technique to efficiently update non-membership proofs.

Let X be the accumulator set and $X(s) = \prod_{x \in X}(s + x)$ be the accumulator polynomial.

$\mathsf{pp} \leftarrow \mathsf{Acc.Setup}(1^\lambda)$: Let $\mathsf{bp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ be the output of $\mathsf{BilGen}(1^\lambda)$. Assume that s is a trapdoor randomly sampled from $\mathbb{Z}^*_p$. The public parameters are $\mathsf{pp} = (\mathsf{bp}, \{g_1^{s^i}, g_2^{s^i} \mid 0 \le i \le t\}, \mathcal{D}' = \mathbb{Z}_p)$ where t is the maximum capacity of the accumulator.

$A_X \leftarrow \mathsf{Acc.Commit}_{\mathsf{pp}}(X = \{x_1, \ldots, x_{|X|}\})$: The accumulator digest of the set $X = \{x_1, \cdots, x_n\}$, is $A_X = g_1^{X(s)}$, where the polynomial $X(s) = \prod_{x \in X}(s + x)$.

$A'_X \leftarrow \mathsf{Acc.Add}_{\mathsf{pp}}(A_X, X, I)$: A set of elements I, $I \cap X = \emptyset$, can be added to the accumulator and the digest can be updated as: $A'_X = g_1^{\prod_{x \in X \cup I}(s+x)} = A_X^{\prod_{x_i \in I}(s+x_i)}$.

$A'_X \leftarrow \mathsf{Acc.Del}_{\mathsf{pp}}(A_X, X, I)$: A set of elements $I \subseteq X$ can be deleted from the accumulator and the digest can be updated as: $A'_X = g_1^{\prod_{x \in X \setminus I}(s+x)} = A_X^{1/\prod_{x_i \in I}(s+x_i)}$.

$\pi_y \leftarrow \mathsf{Acc.MemProve}_{\mathsf{pp}}(X, y)$: The proof of membership of an element y can be computed as $\pi_y = g_1^{\prod_{x \in X \setminus \{y\}}(s+x)}$, where X is the accumulated set.

$\{0,1\} \leftarrow \mathsf{Acc.MemVerify}_{\mathsf{pp}}(A_X, y, \pi_y)$: The membership proof of an element $y \in X$ in the accumulator can be verified by performing the following pairing check: $e(\pi_y, g_2^y g_2^s) = e(A_X, g_2)$.

$\overline{\pi}_y = (\alpha, g_1^{\beta(s)}) \leftarrow \mathsf{Acc.NonMemProve}_{\mathsf{pp}}(X, y)$: The non-membership proof of $\{y\} \cap X = \emptyset$ involves computing the Bézout coefficients $\alpha(s)$ and $\beta(s)$ s.t. $\alpha(s) \cdot X(s) + \beta(s) \cdot (s + y) = 1$. As the monomial $(s + y)$ is of degree one, the polynomial $\alpha(s)$ is in fact a *constant*! Thus, the $\overline{\pi}_y = (\alpha, g_1^{\beta(s)}) \in (\mathbb{Z}_p, \mathbb{G}_1)$ is the non-membership proof of element y.

$\{0,1\} \leftarrow \mathsf{Acc.NonMemVerify}_{\mathsf{pp}}(A_X, y, \overline{\pi}_y)$: The non-membership proof can be verified by checking $e(A_X, g_2^\alpha) \cdot e(g_1^{\beta(s)}, g_2^s g_2^y) = e(g_1, g_2)$, where $\overline{\pi}_y = (\alpha, g_2^{\beta(s)})$.

$\pi'_y \leftarrow \mathsf{Acc.MemProofUpdOnAdd}_{\mathsf{pp}}(A_X, X, y, \pi_y, I)$: The membership proof of an element π_y, can be updated on addition of set I, to the accumulated set, $X = X \cup I$, by computing, $\pi'_y = \pi_y^{\prod_{z \in I}(s+z)}$. When $I = \{z\}$, $\pi'_y = A_X \cdot \pi_y^{z-y}$ is the updated proof.

$\pi'_y \leftarrow \mathsf{Acc.MemProofUpdOnDel}_{\mathsf{pp}}(A_X, A'_X, X, I, y, \pi_y)$: The membership proof π_y can be updated on deletion of set I from the accumulated set, $X = X \setminus I$, by computing, $\pi'_y = \pi_y^{\frac{1}{\prod_{z\in I}(s+z)}}$. When $I = \{z\}$, $\pi'_y = (\frac{\pi_y}{\pi_z})^{\frac{1}{z-y}}$ is the updated proof.

Non-membership in BP accumulator. There are two ways to prove non-membership in the BP acccumulator: (1) based on the PRT [4, 24] and (2) based on GCD [40]. The two methods are equivalent, in that they both require $O(|X|)$ field operations. In this work, we focus on the GCD-based method because it allows for efficient batching and aggregation as we show in §4.2 and §5.2. Additionally, we present a new algorithm to update individual non-membership proofs §6.

BP accumulator soundness. The soundness of the BP construction relies on the t-sBDH assumption.

4 BATCHING BP ACCUMULATOR PROOFS

4.1 Membership

Recall that the membership proof of a single element $y \in X$ is defined as $\pi_y = g_1^{\prod_{x\in X\setminus y}(s+x)}$ and the membership proof of y can be verified by checking $e(\pi_y, g_2^{(s+y)}) = e(A_X, g_2)$, where A_X is the digest of the accumulator.

Batch proof. A batch membership proof is a single succinct proof for a set of elements in the accumulator. A batch proof contains a polynomial in the exponent that includes every element in the accumulator except the monomials terms that correspond to values in the set I.

$\pi_I \leftarrow \mathsf{Acc.MemProve}_{\mathsf{pp}}(X, I)$: A batch membership proof for set $I \subseteq X$ is computed as follows: $\pi_I = g_1^{\prod_{x_i\in X\setminus I}(s+x_i)}$

$\{0,1\} \leftarrow \mathsf{Acc.MemVerify}_{\mathsf{pp}}(A_X, I, \pi_I)$: The batch proof can be verified by checking $e(\pi_I, g_2^{\prod_{x_i\in I}(s+x_i)}) = e(A_X, g_2)$.

Asymptotics. The batch membership proof for I consists only of one element in $\mathbb{G}_1$ (as opposed to $|I|\cdot\mathbb{G}_1$ elements, if proved individually). Verifying a batch proof takes $O(|I|\log^2|I|)$ field operations to compute the coefficients of polynomial $I(s) = \prod_{x_i\in I}(s+x_i)$ using FFT, one $|I|$-sized multi-exponentiation in $\mathbb{G}_2$ to compute $g_2^{I(s)}$ and 2 pairings. However, verifying $|I|$ individual proofs requires $|I|$ individual exponentiations (cannot use fast multi-exponentiations), and $2|I|$ pairings.

4.2 Non-membership

A non-membership proof of an element $y \notin X$ leverages the fact that $\gcd(X(s), (s+y)) = 1$. Thus the Bézout coefficients α and $\beta(s)$ satisfy: $\alpha X(s) + \beta(s)(s+y) = 1$. We generalize this observation to prove the non-membership of set $I \cap X = \emptyset$.

Batch proof. We define a batch non-membership proof as a single succinct proof for a set of elements disjoint from the accumulator. Observe that there are no common roots between polynomials $X(s)$ and $I(s) = \prod_{y_i\in I}(s+y_i)$, therefore $\gcd(X(s), I(s)) = 1$. Moreover, the Bézout coefficients $\alpha(s)$ and $\beta(s)$ of $X(s), I(s)$ are non-constant polynomials.

$\overline{\pi}_I \leftarrow \mathsf{Acc.NonMemProve}_{\mathsf{pp}}(X, I)$: A batch non-membership proof for a set I, where $I \cap X = \emptyset$, is $\overline{\pi}_I = (g_2^{\alpha(s)}, g_1^{\beta(s)}) \in (\mathbb{G}_2, \mathbb{G}_1)$ such that $\alpha(s)\cdot X(s) + \beta(s)\cdot\prod_{y_i\in I}(s+y_i) = 1$.

$\{0,1\} \leftarrow \mathsf{Acc.NonMemVerify}_{\mathsf{pp}}(A_X, I, \overline{\pi}_I)$: The batch proof can be verified by checking $e(A_X, g_2^{\alpha(s)}) \cdot e(g_1^{\beta(s)}, g_2^{\prod_{y_i\in I}(s+y_i)}) = e(g_1, g_2)$, where $\overline{\pi}_I = ({g_2}^{\alpha(s)}, {g_1}^{\beta(s)})$.

Asymptotics. The batch non-membership proof consists of only one element from $\mathbb{G}_1$ and one element from $\mathbb{G}_2$. Verifying a batch proof takes $O(|I|\log^2|I|)$ field operations to compute the coefficients of polynomial $I(s)$ using FFT, one $|I|$-sized multi-exponentiation in $\mathbb{G}_2$ to compute $g_2^{I(s)}$ and 2 pairings. However, verifying $|I|$ individual proofs requires $|I|$ individual exponentiations (cannot use fast multi-exponentiations) and $2|I|$ pairings. If $e(g_1, g_2)$ is not precomputed, then batch verification and individual verification requires 3 pairings and $2|I|+1$ pairings, respectively.

4.3 Soundness of batching

Theorem 4.1. *Batch membership and non-membership proofs $\pi_I, \overline{\pi}_I$, are sound, by Definition 3.2, under the adaptive Uber assumption in the AGM.*

Proof. Deferred to the extended version of our paper. □

5 AGGREGATION

5.1 Membership

The BP accumulator based on Nguyen et al. resembles the KZG polynomial commitments [29, 38]. As demonstrated in prior works [14, 25, 47], KZG polynomial commitments can be aggregated using the PFD [53]. We use these techniques to aggregate proofs in the BP accumulator.

$\pi_I \leftarrow \mathsf{Acc.AggMem}_{\mathsf{pp}}(A_X, I, \{\pi_1, \cdots, \pi_{|I|}\})$: Let $I \subseteq X$ be the set of elements to be aggregated, $X(s) = \prod_{x\in X}(s+x)$ be the accumulator polynomial of X, $I(s) = \prod_{x_i\in I}(s+x_i)$ be the accumulator polynomial of I, and $X_i(s) = \prod_{x\in X\setminus\{x_i\}}(s+x)$. The goal is to compute the polynomial $Y(s) = \prod_{x_i\in X\setminus I}(s+x_i)$, which excludes all the monomials that correspond to values in the set I. Using PFD [53]:

$$Y(s) = X(s)\cdot\frac{1}{I(s)} = X(s)\sum_{x_i\in I}\frac{1}{I'(-x_i)(s+x_i)} = \sum_{x_i\in I}\frac{1}{I'(-x_i)}\cdot X_i(s)$$

Thus, the aggregated proof, $\pi_I = {g_1}^{Y(s)} = \prod_{x_i\in I}\pi_i^{c_i}$, where $c_i = \frac{1}{I'(-x_i)}$, I' is the first derivative of I w.r.t s, and π_i is the individual membership proof of element i.

Correctness and soundness. We can see that as long as each individual π_i's are correct, π_I satisfies *aggregation correctness*. Note that Thm. 4.1, is sufficient to argue the soundness of aggregation (Definition 3.2) since the resulting proof after aggregation is a batch proof and thus can be verified by running MemVerify.

Asymptotics. In a field with sufficiently many roots of unity, FFT based polynomial interpolation and multi-point evaluation techniques can be used to compute the polynomial $I(s)$ and evaluate $I'(s)$ at all x_i's in $O(|I|\log^2|I|)$ field operations. Asymptotically, it takes $O(|I|\log^2|I|)$ operations in $\mathbb{Z}_p$ (to compute all the c_i's) and $|I|$ sized multi-exponentiation in $\mathbb{G}_1$ to compute the aggregated proof.

The complexity of verifying an aggregated proof is the same as verifying a batch proof. Note that verification requires a single multi-exponentiation of size $|I|$ in $\mathbb{G}_2$ to compute ${g_2}^{I(s)}$. However, using the PoE protocol, we can outsource the exponentiation cost to an untrusted prover. We discuss this optimization in §8.

5.2 Non-membership

In this section, we give an algorithm to combine multiple individual non-membership proofs into a single non-membership proof.

$\overline{\pi}_I \leftarrow \mathsf{Acc.AggNonMem}_{\mathsf{pp}}(A_X, I, \{\overline{\pi}_1, \cdots, \overline{\pi}_{|I|}\})$: Let I be a set of elements disjoint from the accumulated set X, $\overline{\pi}_i = (\alpha_i, g_1^{\beta_i(s)}) \in \mathbb{Z}_p \times \mathbb{G}_1$ be the non-membership proof of the element $y_i \in I$, $X(s) = \prod_{x \in X}(s+x)$ be the accumulator polynomial, $I(s) = \prod_{y_i \in I}(s+y_i)$ be the accumulator polynomial of I, and $Y_i(s) = \frac{I(s)}{(s+y_i)}$.

Observe that $\gcd(Y_1(s), Y_2(s), \ldots, Y_{|I|}(s)) = 1$. By generalization of Bézout's identity for polynomials, $\exists$ polynomials $c_i(s)$, s.t.:

$$\sum_{i=1}^{|I|} c_i(s) \cdot Y_i(s) = \gcd(Y_1(s), Y_2(s), \ldots, Y_{|I|}(s)) = 1 \tag{1}$$

Lemma 5.1. *The Bézout coefficient polynomials, $c_i(s)$, of Eq. 1 are non-zero constants.*

Proof. Deferred to the extended version of our paper. □

To compute the non-membership proof of I, we need to compute polynomials $\alpha(s)$ and $\beta(s)$ such that:

$$\alpha(s) \cdot X(s) + \beta(s) \cdot \prod_{y_i \in I}(s+y_i) = \gcd(X(s), \prod_{y_i \in I}(s+y_i)) = 1$$

Recall that each individual non-membership proof satisfies:

$$\alpha_i \cdot X(s) + \beta_i(s) \cdot (s+y_i) = 1, \text{where } \deg(\alpha_i) = 0$$

Multiplying by $c_i \cdot Y_i(s)$: $\alpha_i c_i Y_i(s) X(s) + c_i \beta_i(s)(s+y_i) Y_i(s) = c_i Y_i(s)$
$\Rightarrow \alpha_i c_i Y_i(s) X(s) + c_i \beta_i(s) I(s) = c_i Y_i(s)$

Summing over $i = 1$ to $|I|$: $\sum_{i=1}^{|I|} \alpha_i c_i Y_i(s) X(s) + \sum_{i=1}^{|I|} c_i \beta_i(s) I(s) = \sum_{i=1}^{|I|} c_i \cdot Y_i(s)$

Using Eq. 1: $\left(\sum_{i=1}^{|I|} \alpha_i c_i Y_i(s)\right) \cdot X(s) + \left(\sum_{i=1}^{|I|} c_i \beta_i(s)\right) \cdot I(s) = 1$

Therefore,

$$\alpha(s) = \sum_{i=1}^{|I|} \alpha_i c_i Y_i(s) \qquad \beta(s) = \sum_{i=1}^{|I|} c_i \beta_i(s) \tag{2}$$

Thus the non-membership of set I is $\overline{\pi}_I = ({g_2}^{\alpha(s)}, {g_1}^{\beta(s)})$

Correctness and soundness. We can see that as long as each individual $\overline{\pi}_i$'s are correct, $\overline{\pi}_I$ satisfies *aggregation correctness*. The algorithm AggNonMem always outputs $\overline{\pi}_I$ because terms c_i do not depend on trapdoor s (Lemma 5.1), therefore exponentiation by c_i is feasible. We remark that Thm. 4.1, is sufficient to argue the soundness of aggregation (Definition 3.2) since the resulting proof after aggregation is a batch proof and thus can be verified by running NonMemVerify.

Asymptotics. The task of computing the aggregated proof can be divided into: (1) task of computing the Bézout coefficients c_i's, (2) task of computing the coefficients of each $Y_i(s)$, and (3) task of computing $({g_2}^{\alpha(s)}, {g_1}^{\beta(s)})$.

Scheme	RSA	BP
Domain	Prime	$\mathbb{Z}_p$
$\mathsf{Setup}(1^\lambda)$	O(1)	$O(t) \cdot (\mathbb{G}_1 + \mathbb{G}_2)$
Commit	$O(\lvert X\rvert) \cdot \mathbb{G}$	$O(\lvert X\rvert) \cdot \mathbb{G}_1 +$ $O(\lvert X\rvert \log^2 \lvert X\rvert) \cdot \mathbb{Z}_p$
AggMem	$O(\lvert I\rvert \log \lvert I\rvert) \cdot \mathbb{G} +$ $O(\lvert I\rvert \log \lvert I\rvert) \cdot \mathbb{F}$	$O(\lvert I\rvert) \cdot \mathbb{G}_1 +$ $O(\lvert I\rvert \log^2 \lvert I\rvert) \cdot \mathbb{Z}_p$
MemVerify	$O(\lvert I\rvert) \cdot \mathbb{G}$	$2\mathbb{P} + O(\lvert I\rvert) \cdot \mathbb{G}_2 +$ $O(\lvert I\rvert \log^2 \lvert I\rvert) \cdot \mathbb{Z}_p$
MemVerifyPoE	$O(1) \cdot \mathbb{G} +$ $O(\lvert I\rvert) \cdot \mathbb{F}$	$3\mathbb{P} + 1\mathbb{G}_1 + 1\mathbb{G}_2 +$ $O(\lvert I\rvert \log^2 \lvert I\rvert) \cdot \mathbb{Z}_p$
AggNonMem	$O(\lvert I\rvert \log \lvert I\rvert) \cdot \mathbb{G} +$ $O(\lvert I\rvert \log \lvert I\rvert) \cdot \mathbb{F}$	$O(\lvert I\rvert) \cdot (\mathbb{G}_1 + \mathbb{G}_2) +$ $O(\lvert I\rvert^2 \log \lvert I\rvert) \cdot \mathbb{Z}_p$
NonMemVerify	$O(\lvert I\rvert) \cdot \mathbb{G}$	$2\mathbb{P} + O(\lvert I\rvert) \cdot \mathbb{G}_2 +$ $O(\lvert I\rvert \log^2 \lvert I\rvert) \cdot \mathbb{Z}_p$
NonMemVerifyPoE	$O(1) \cdot \mathbb{G} +$ $O(\lvert I\rvert) \cdot \mathbb{F}$	$5\mathbb{P} + 1\mathbb{G}_1 + 1\mathbb{G}_2 +$ $O(\lvert I\rvert \log^2 \lvert I\rvert) \cdot \mathbb{Z}_p$

Table 1: Set X denotes the entire accumulated set and $I \subseteq X$. Let $O(Y) \cdot \mathbb{G}$ denotes one large exponentiation to the product of Y elements or Y exponentiations. All exponentiations in BP can be sped up by a logarithmic factor using multi-exponentiations.

First, the task of computing the Bézout coefficients in Eq. 1 takes $O(|I| \log^2 |I|)$ field operations. Second, the task of computing all Y_i's take $O(|I|^2 \log |I|)$ field operations. This is because computing each $Y_i(s)$ costs $O(|I| \log |I|)$ field operations. Third, the task of computing ${g_2}^{\sum_{i=1}^{|I|} \alpha_i c_i Y_i(s)}$ and ${g_1}^{\sum_{i=1}^{|I|} c_i \beta_i(s)}$ requires a single multi-exponentiation of size $|I|$ in $\mathbb{G}_2$ and $\mathbb{G}_1$, respectively. Thus, in total, it takes $O(|I|^2 \log |I|)$ field operations and multi-exponentiations of size $O(|I|)$ in $\mathbb{G}_1, \mathbb{G}_2$ to compute the aggregated non-membership proof from individual proofs.

The complexity of verifying an aggregated proof is the same as verifying a batch proof. Note that verification requires a single multi-exponentiation of size $|I|$ in $\mathbb{G}_2$ to compute ${g_2}^{I(s)}$. However, using the PoE protocol, we can outsource the exponentiation cost to an untrusted prover. We discuss this optimization in §8.

6 NON-MEMBERSHIP PROOF UPDATES

In this section, we describe a new protocol that allows to efficiently update a GCD-based non-membership proof for the BP accumulator after changes (additions/deletions) to the accumulated set. Let $y \notin X$ and $\overline{\pi}_y = (\alpha, {g_1}^{\beta(s)}) \in \mathbb{Z}_p \times \mathbb{G}_1$ s.t. $\alpha X(s) + \beta(s)(s+y) = 1$. $\overline{\pi}'_y \leftarrow \mathsf{Acc.NonMemProofUpdOnAdd}_{\mathsf{pp}}(A_X, X, y, \overline{\pi}_y, I = \{z\})$: Recall that,

$$\alpha X(s) + \beta(s)(s+y) = 1 \tag{3}$$

Since $y \neq z$, by Bézout's Identity,

$$u(s+z) + v(s+y) = 1, \text{ where } u, v \in \mathbb{Z}_p \tag{4}$$

Goal is to find α' and $\beta'(s)$ s.t.,

$$\alpha' X'(s) + \beta'(s)(s+y) = 1, \text{ where } X'(s) = X(s)(s+z) \tag{5}$$

Multiplying Eq. 3 by $u(s+z)$: $u\alpha X(s)(s+z) + u\beta(s)(s+z)(s+y) = u(s+z)$.
Using Eq. 4:

$$u\alpha X(s)(s+z) + u\beta(s)(s+z)(s+y) = 1 - v(s+y)$$
$$(u\alpha)X'(s) + (v + u\beta(s)(s+z))(s+y) = 1$$

From Eq. 5 we have: $\alpha' = u\alpha$ and $\beta'(s) = v + u\beta(s)(s+z)$

However, it is not possible to compute ${g_1}^{\beta'(s)}$ without the coefficients of $\beta(s)$ because individual proofs only contain $g^{\beta(s)}$. Thus, we simplify $\beta'(s)' = v + u\beta(s)(s+z)$, replace $u(s+z)$ as $1 - v(s+y)$ from Eq. 4: $\beta'(s) = v + \beta(s)(1 - v(s+y)) = v + \beta(s) - v\beta(s)(s+y) = v(1 - \beta(s)(s+y)) + \beta(s)$. Replace $1 - \beta(s)(s+y)$ as $\alpha X(s)$ from Eq. 3: $\beta'(s) = v\alpha X(s) + \beta(s)$.

Thus, $\alpha' = u\alpha$, $\beta'(s) = v\alpha X(s) + \beta(s)$ and $\overline{\pi}'_y = (\alpha', {g_1}^{\beta'(s)}) = (u\alpha, {A_X}^{v\alpha} \cdot {g_1}^{\beta(s)})$.

To compute the updated non-membership proof, $\overline{\pi}'_y = (\alpha', {g_1}^{\beta'(s)})$, we calculate constants u, v in Eq. 4 by running the extended Euclidean algorithm for degree one polynomials, which takes only $O(1)$ operations. We can compute the constant α' with one multiplication operation in $\mathbb{Z}_p$. Similarly, we can compute ${g_1}^{\beta'(s)} = ((A_X)^{\alpha})^{v} \cdot {g_1}^{\beta(s)}$ with one multiplication operation in $\mathbb{Z}_p$, one exponentiation operation in $\mathbb{G}_1$ by $\mathbb{Z}_p$, and one addition in $\mathbb{G}_1$.

$\overline{\pi}'_y \leftarrow \mathsf{Acc.NonMemProofUpdOnDel}_{\mathsf{pp}}(A_X, A'_X, X, I = \{z\}, y, \overline{\pi}_y)$:

Recall that, $\alpha X(s) + \beta(s)(s+y) = 1$. Since $y \neq z$, by Bézout's Identity,

$$u(s+z) + v(s+y) = 1, \text{ where } u, v \in \mathbb{Z}_p \tag{6}$$

Goal is to find α' and $\beta'(s)$ s.t.,

$$\alpha' X'(s) + \beta'(s)(s+y) = 1, \text{ where } X'(s) = \frac{X(s)}{(s+z)} \tag{7}$$

$$\alpha X(s) + \beta(s)(s+y) = 1 \Rightarrow \alpha X'(s)(s+z) + \beta(s)(s+y) = 1$$

Replace $(s+z)$ as $\frac{1-v(s+y)}{u}$ from Eq. 6: $\alpha X'(s)\left(\frac{1-v(s+y)}{u}\right) + \beta(s)(s+y) = 1 \Rightarrow \frac{\alpha}{u}X'(s) + \left(\beta(s) - \frac{v\alpha}{u}X'(s)\right)(s+y) = 1$

Thus, from Eq. 7, $\alpha' = \frac{\alpha}{u}$, $\beta'(s) = \beta(s) - v\alpha' X'(s)$ and $\overline{\pi}'_y = (\alpha', {g_1}^{\beta'(s)}) = \left(\frac{\alpha}{u}, \frac{{g_1}^{\beta(s)}}{A_{X'}^{v\alpha'}}\right)$.

To compute the updated non-membership proof, $\overline{\pi}'_y = (\alpha', {g_1}^{\beta'(s)})$, we calculate constants u, v in Eq. 6 by running the extended Euclidean algorithm for degree one polynomials, which takes only $O(1)$ operations. We can compute the constant α' with one inversion operation and one multiplication operation in $\mathbb{Z}_p$. Similarly, we can compute ${g_1}^{\beta'(s)} = \frac{{g_1}^{\beta(s)}}{A_{X'}^{v\alpha'}}$ with one multiplication operation in $\mathbb{Z}_p$, one inversion operation, one exponentiation operation in $\mathbb{G}_1$ by $\mathbb{Z}_p$, and one addition operation in $\mathbb{G}_1$.

7 ZK BATCH PROOFS

We now show how our batch (non-)membership proofs in BP accumulators can be made zero-knowledge. We consider the following setting: a set $X = \{x_1, \ldots, x_n\}$ of elements is accumulated, and a prover holds witnesses for a subset $I \subseteq X$, where $|I| = d$. We want to prove that $I \subseteq X$ (or $I \cap X = \emptyset$) while *hiding the set* I itself. Additionally, we want to be able to reveal just the size of I or in other words that it includes "at least d elements". This is important since typically a batch proof does not hide the number of batched elements and this is useful in identity systems, sanctions/embargoed lists, e-cash, etc. At the same time, we want to maintain the benefits of batch proofs: (1) the verifier cost should be constant for both membership and non-membership proofs, and (2) the size of the ZK batch proof should remain sublinear.

In the ZK setting, the verifier does not hold the set I or the proof π_I (or $\overline{\pi}_I$) in order to run the verification algorithm. Instead, the prover has to prove in ZK that the pairing equations in the verification algorithm hold. The prover's witness is the set I, the randomness r used in the commitment to I, a proof of membership π_I or a proof of non-membership $\overline{\pi}_I$. The inputs known to the verifier are the public parameters pp, a commitment to the subset C_I, and the accumulator value A_X.

More specifically, the prover (that knows the polynomial $I(x)$ and the commitment randomness r) has to compute ZK proofs for the following relations:

- $\mathcal{R}_{\mathsf{mem}}(\mathsf{pp}, C_I, A_X; \pi_I, r)$ knowledge of π_I such that membership verification holds and knowledge of randomness r
- $\mathcal{R}_{\mathsf{nonmem}}(\mathsf{pp}, C_I, A_X; \overline{\pi}_I, r)$ knowledge of $\overline{\pi}_I$ such that non-membership verification holds and knowledge of randomness r
- If revealing the size, $\mathcal{R}_{\mathsf{degcheck}}(\mathsf{pp}, C_I, d; I, r)$ proves that the set size $|I| = d$ and the set corresponds to C_I. A proof for this relation is analyzed in the following tasks:
  - Proving that $I(x) = \prod_{i=1}^{d}(x + x_i) = x^d + f(x)$, where $\deg(f) \leq d - 1$ using a hiding commitment C_f,
  - Proving well-formedness of C_f in relation to C_I.

The commitment C_I allows the prover to later open all elements in I (if needed) by revealing r. In the following paragraphs, we explain in details how these proofs are constructed.

Notation. Let pp be the following public parameters:

- $g_1, h_1, \mathfrak{g}, \mathfrak{g}_1, \mathfrak{g}_2 \in \mathbb{G}_1$ and $g_2, h_2, \mathfrak{h}, \mathfrak{h}_1, \mathfrak{h}_2 \in \mathbb{G}_2$
- $\mathbf{g_1^s} = [g_1, g_1^s, \ldots, g_1^{s^t}] \in \mathbb{G}_1^{t+1}$, $\mathbf{g_2^s} = [g_2, g_2^s, \ldots, g_2^{s^t}] \in \mathbb{G}_2^{t+1}$
- $\mathbf{a} = [g_2^a, g_2^{as}, \ldots, g_2^{as^t}] \in \mathbb{G}_2^{t+1}$

We view a polynomial $I(x)$ as equivalent to its coefficients that form a vector, therefore for the rest of the paper, we overload the Pedersen VC input with both vectors and polynomials.

7.1 Proving membership ($\mathcal{R}_{\mathsf{mem}}$)

The high level idea of the initial proof is the following: $\mathcal{R}_{\mathsf{mem.init}}$ essentially proves that the verification equation holds for $C_I h_2^{-r}$ and proves knowledge of r.

$$\mathcal{R}_{\mathsf{mem.init}} = \left\{ \begin{array}{c} (\mathsf{pp}, C_I \in \mathbb{G}_2, A_X \in \mathbb{G}_1; \pi_I \in \mathbb{G}_1, r \in \mathbb{Z}_p) : \\ C_I = h_2^r g_2^{I(s)} \wedge e(\pi_I, C_I) \cdot e(\pi_I, h_2)^{-r} = e(A_X, g_2) \end{array} \right\}$$

However, it still does not hide the witness π_I. Using techniques from RingCT [43], $\mathcal{R}_{\mathsf{mem.init}}$ can be transformed into $\mathcal{R}_{\mathsf{mem}}$, which proves the above statement in ZK. Specifically, it adds a proof that $\mathcal{R}_{\mathsf{mem.init}}$ verifies for $\pi_{I,2} g^{-\tau_1}$ and proves knowledge of τ_1, where $\pi_{1,2}$ is the blinded version of the batch proof π_I.

$$\mathcal{R}_{\mathsf{mem}} = \left\{ \begin{array}{c} (\mathsf{pp}, C_I \in \mathbb{G}_2, A_X \in \mathbb{G}_1; \pi_I \in \mathbb{G}_1, r, \tau_1, \tau_2 \in \mathbb{Z}_p) : \\ C_I = h_2^r g_2^{I(s)} \wedge \delta_1 = \tau_1 r \wedge \delta_2 = \tau_2 r \wedge \\ \pi_{I,1} = g_1^{\tau_1} \mathfrak{g}^{\tau_2} \wedge \pi_{I,2} = \pi_I \mathfrak{g}^{\tau_1} \wedge \\ \frac{e(\pi_{I,2}, C_I)}{e(A_X, g_2)} = e(\mathfrak{g}, C_I)^{\tau_1} \cdot e(\mathfrak{g}, h_2)^{-\delta_1} \cdot e(\pi_{I,2}, h_2)^r \end{array} \right\}$$

$\mathcal{R}_{\mathsf{mem}}$ is instantiated with a generalized version of Schnorr's protocol for knowledge of DL as r, τ_1 are scalars.

Protocol for relation $\mathcal{R}_{\mathsf{mem}}$. The interactive version of the protocol is as follows:

- Prover
  - Picks $\tau_1, \tau_2 \leftarrow \mathbb{Z}_p$
  - Computes $\pi_{I,1} = g_1^{\tau_1} \mathfrak{g}^{\tau_2}$ and $\pi_{I,2} = \pi_I \mathfrak{g}^{\tau_1}$
  - Picks $r_r, r_{\tau_1}, r_{\tau_2}, r_{\delta_1}, r_{\delta_2} \in \mathbb{Z}_p$
  - Sends:
    - $\pi_{I,1}, \pi_{I,2}$
    - $R_1 = g_1^{r_{\tau_1}} \mathfrak{g}^{r_{\tau_2}}, R_2 = \pi_{I,1}{}^{r_r} g_1^{-r_{\delta_1}} \mathfrak{g}^{-r_{\delta_2}}$
    - $R_3 = e(\mathfrak{g}, C_I)^{r_{\tau_1}} e(\mathfrak{g}, h_2)^{-r_{\delta_1}} e(\pi_{I,2}, h_2)^{r_r}$
- Verifier sends $c \leftarrow \mathbb{Z}_p$
- Prover sends:
  - $s_r = r_r + cr$
  - $s_{\tau_1} = r_{\tau_1} + c\tau_1, s_{\tau_2} = r_{\tau_2} + c\tau_2$
  - $s_{\delta_1} = r_{\delta_1} + c\delta_1, s_{\delta_2} = r_{\delta_2} + c\delta_2$
- Verifier checks:
  - $R_1 = \pi_{I,1}^{-c} g_1^{s_{\tau_1}} \mathfrak{g}^{s_{\tau_2}}$
  - $R_2 = \pi_{I,1}^{s_r} g_1^{-s_{\delta_1}} \mathfrak{g}^{-s_{\delta_2}}$
  - $R_3 \cdot (\frac{e(\pi_{I,2}, C_I)}{e(A_X, g_2)})^c = e(\mathfrak{g}, C_I)^{s_{\tau_1}} e(\mathfrak{g}, h_2)^{-s_{\delta_1}} e(\pi_{I,2}, h_2)^{s_r}$

Correctness, soundness, and ZK. We defer the security arguments to the extended version of our paper.

Asymptotics. The instantiation of $\mathcal{R}_{\mathsf{mem}}$ consists of 5 group elements and 5 field elements, and has constant prover and verifier.

7.2 Proving non-membership ($\mathcal{R}_{\mathsf{nonmem}}$)

For non-membership, $\mathcal{R}_{\mathsf{mem.init}}$ is replaced by

$$\mathcal{R}_{\mathsf{nonmem.init}} = \left\{ \begin{array}{c} (\mathsf{pp}, C_I \in \mathbb{G}_2, A_X \in \mathbb{G}_1; \\ (\overline{A}, \overline{B}) \in \mathbb{G}_2 \times \mathbb{G}_1, r \in \mathbb{Z}_p) : \\ C_I = h_2^r g_2^{I(s)} \wedge \\ e(A_X, \overline{A}) \cdot e(\overline{B}, C_I) \cdot e(\overline{B}, h)^{-r} = e(g_1, g_2) \end{array} \right\}$$

Similarly, $\mathcal{R}_{\mathsf{nonmem.init}}$ still does not hide the witness π_I. Thus, using techniques from RingCT [43], $\mathcal{R}_{\mathsf{nonmem.init}}$ can be transformed into $\mathcal{R}_{\mathsf{nonmem}}$, which proves the above statement in ZK.

$$\mathcal{R}_{\mathsf{nonmem}} = \left\{ \begin{array}{c} (\mathsf{pp}, C_I \in \mathbb{G}_2, A_X \in \mathbb{G}_1; (\overline{A}, \overline{B}) \in \mathbb{G}_2 \times \mathbb{G}_1, \\ r, \tau_1, \tau_3, \tau_4 \in \mathbb{Z}_p) : \\ C_I = h_2^r g_2^{I(s)} \wedge \delta_3 = \tau_3 r \wedge \delta_4 = \tau_4 r \wedge \\ \overline{A}_2 = \overline{A} \mathfrak{h}^{\tau_1} \wedge \\ \overline{B}_1 = g_1^{\tau_3} \mathfrak{g}^{\tau_4} \wedge \overline{B}_2 = \overline{B} \mathfrak{g}^{\tau_3} \wedge \\ \frac{e(A_X, \overline{A}_2) \cdot e(\overline{B}_2, C_I)}{e(g_1, g_2)} = \begin{array}{c} e(A_X, \mathfrak{h})^{\tau_1} \cdot e(\mathfrak{g}, C_I)^{\tau_3} \cdot \\ e(\mathfrak{g}, h_2)^{-\delta_3} \cdot e(\overline{B}_2, h_2)^r \end{array} \end{array} \right\}$$

The rest of the protocol remains the same. We defer the security arguments to the extended version of our paper.

Protocol for relation $\mathcal{R}_{\mathsf{nonmem}}$. The interactive version of the protocol is as follows:

- Prover
  - Picks $\tau_1, \tau_3, \tau_4 \leftarrow\!\!\$\ \mathbb{Z}_p$
  - Computes
    - $\overline{A}_2 = \overline{A} \mathfrak{h}^{\tau_1}$
    - $\overline{B}_1 = g_1^{\tau_3} \mathfrak{g}^{\tau_4}, \overline{B}_2 = \overline{B} \mathfrak{g}^{\tau_3}$
  - Picks $r_r, r_{\tau_1}, r_{\tau_3}, r_{\tau_4}, r_{\delta_3}, r_{\delta_4} \leftarrow\!\!\$\ \mathbb{Z}_p$
  - Sends:
    - $\overline{A}_2, \overline{B}_1, \overline{B}_2$
    - $R_{2,1} = g_1^{r_{\tau_3}} \mathfrak{g}^{r_{\tau_4}}, R_{2,2} = (\overline{B}_1)^{r_r} g_1{}^{-r_{\delta_3}} \mathfrak{g}^{-r_{\delta_4}}$
    - $R_3 = e(A_X, \mathfrak{h})^{r_{\tau_1}} \cdot e(\mathfrak{g}, C_I)^{r_{\tau_3}} \cdot e(\mathfrak{g}, h_2)^{-r_{\delta_3}} \cdot e(\overline{B}_2, h_2)^{r_r}$
- Verifier sends $c \leftarrow \mathbb{Z}_p$
- Prover sends:
  - $s_r = r_r + cr, s_{\tau_1} = r_{\tau_1} + c\tau_1$
  - $s_{\tau_3} = r_{\tau_3} + c\tau_3, s_{\tau_4} = r_{\tau_4} + c\tau_4$
  - $s_{\delta_3} = r_{\delta_3} + c\delta_3, s_{\delta_4} = r_{\delta_4} + c\delta_4$
- Verifier checks:
  - $R_{2,1} = (\overline{B}_1)^{-c} g_1{}^{s_{\tau_3}} \mathfrak{g}^{s_{\tau_4}}$
  - $R_{2,2} = (\overline{B}_1)^{s_r} g_1{}^{-s_{\delta_3}} \mathfrak{g}^{-s_{\delta_4}}$
  - $R_3 \cdot (\frac{e(A_X, \overline{A}_2) \cdot e(\overline{B}_2, C_I)}{e(g_1, g_2)})^c = e(A_X, \mathfrak{h})^{s_{\tau_1}} \cdot e(\mathfrak{g}, C_I)^{s_{\tau_3}} \cdot e(\mathfrak{g}, h_2)^{-s_{\delta_3}} \cdot e(\overline{B}_2, h_2)^{s_r}$

7.3 Proving degree bound ($\mathcal{R}_{\mathsf{degcheck}}$)

We now want to prove in ZK the following statement: *The prover knows at least d elements.* So far, the verifier only knows that the verification algorithm holds for some commitment C_I. In order to be convinced that C_I is a commitment to a set $I = \{x_1, \ldots, x_d\}$, the verifier has to prove knowledge of a polynomial $I(x)$ of degree d. It is implied that as long as $I(x)$ is a polynomial, it is well-formed as a product of d monomials $(x + x_i)$ ($I(x)$ is part of the accumulator exponent (membership) or coprime to the accumulator exponent(non-membership)). We define the following relation:

$$\mathcal{R}_{\mathsf{degcheck}} = \left\{ \begin{array}{c} (\mathsf{pp}, C_I \in \mathbb{G}_2, d \in \mathbb{Z}_p; r \in \mathbb{Z}_p, I \subset \mathbb{Z}_p) : \\ C_I = h_2^r g_2^{I(s)} \wedge |I| = d \end{array} \right\}$$

For proving the above statement with existing protocols (that prove maximum instead of minimum polynomial degree), we use the following idea: a correct computation of the polynomial $I(x)$ that corresponds to elements in the set I with $|I| = d$, results to the following polynomial:

$$I(x) = \prod_{i=1}^{d} (x + x_i) = x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0$$

Let $f(x) = a_{d-1} x^{d-1} + \cdots + a_1 x + a_0$, Thus, $I(x) = x^d + f(x)$.

In order to prove that the degree of $I(x)$ is at least d it suffices to show that $\deg(f(x)) \leq d - 1$. This implies that there is no term of f that eliminates x^d. Therefore, the degree of polynomial I is at least d.

Proving that $\deg(f) \leq d-1$. In order to prove a polynomial's maximum degree, we follow the technique used in Marlin [22]. Instead of publishing parameters of degree-specific size, we transform the proof to use the accumulator's parameters of size t and we shift the polynomial such that it has degree t instead of d.

Informally: polynomial $f(x)$ gets multiplied with a random polynomial of degree $t - (d - 1)$ (the sparse polynomial $c \cdot x^{t-(d-1)}$ where c is a random scalar would suffice). This is what we call a shift and can be proven to be computed correctly with the use of pairings.

Using knowledge assumptions [28] (and commitment to the same polynomial multiplied with a) the prover shows knowledge of f. In other words, the prover was able to construct the result polynomial using public parameters (that consist of generators raised to powers of s up to t), therefore the degree of f does not exceed $d-1$.

Protocol for $\mathcal{R}_{\text{degcheck}}$. The interactive version of the protocol is as follows:

- Prover
  - Computes: $f(x)$ as $I(x) = \prod_{i=1}^{d}(x+x_i) = x^d + f(x)$, where $\deg(f) \leq d-1$
  - Sends $C_f = \frac{C_I}{g_2^{s^d}}$
- Verifier
  - Sends: $c \in \mathbb{Z}_p$
- Prover
  - Computes: $f(x) \cdot cx^{t-d+1}$ and $r \cdot cx^{t-d+1}$
  - Sends: $C = g_2^{f(s)cs^{t-d+1}} h_2^{r \cdot cs^{t-d+1}} \in \mathbb{G}_2, C^a \in \mathbb{G}_2$
- Verifier checks:
  - $e(g_1, C_I) = e(g_1, C_f) \cdot e(g_1, h^{s^d})$
  - $e(g_1, C) = e(g_1^{cs^{t-d+1}}, C_f), e(g_1, C^a) = e(g_1^a, C)$

Correctness, soundness, and ZK. We defer the security arguments to the extended version of our paper.

Asymptotics. The protocol has $O(d)$ prover cost that comes from multiplying $f(x)$ with the polynomial of degree $t-(d-1)$ and computing its commitment C. It can be made non-interactive using the Fiat-Shamir transformation. The proof size is constant (the prover has to send over constant sized commitments C_I, C_f, C, C^a: a commitment to polynomial $I(x)$, to polynomial $f(x)$, to polynomial $f(x)\cdot c\cdot x^{t-(d-1)}$ and to polynomial $a\cdot f(x)\cdot c\cdot x^{t-(d-1)}$ respectively) and verification cost is also constant (7 pairings).

8 PROOF OF EXPONENTIATION

We present the Proof-of-Exponentiation (PoE) protocol in the known-order group setting. Informally, the prover can convince the verifier that the exponentiation of a group element by the evaluation of a known polynomial at a specific point is correct. That is, given a tuple, $(A_U, A_W, V(x)) \in (\mathbb{G}, \mathbb{G}, \mathbb{Z}_p[x])$, the prover can convince the verifier that $A_W = A_U^{V(s)}$, with a constant sized proof requiring constant number of pairing checks. Given the coefficients of polynomial $V(s)$, naively, computing $g^{V(s)}$ would require as many exponentiations as $\deg(V(s))$. However, with PoE, the verifier has to just perform cheaper polynomial division and *constant* pairing computation instead of performing linear number of group exponentiations.

Since naively verifying batch (non-)membership proofs require a multi-exponentiation of size $|I|$, we can delegate the expensive exponentiations to the prover using the PoE protocol. Note that the PoE protocol is of independent interest and can be used as a building block in other constructions.

For ease of exposition, we present the protocol (Fig. 1) using symmetric pairings. However, the protocol can be instantiated using asymmetric pairings.

$$\mathcal{R}_{\text{PoE}} = \left\{ \begin{array}{c} ((A_U, A_W \in \mathbb{G}, V(x) \in \mathbb{Z}_p[x]); \perp) : \\ A_W = A_U^{V(s)} \in \mathbb{G} \end{array} \right\}$$

$\underline{\text{pp} \leftarrow \text{PoE.Setup}(1^\lambda)}$:

1. $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \text{BilGen}(1^\lambda)$
2. $s \leftarrow\!\$\ \mathbb{Z}_p^*$
3. $\text{pp} := ((p, g, \mathbb{G}, \mathbb{G}_T, e), \{g^{s^i} \mid 0 \leq i \leq t\})$

$\underline{\text{Protocol PoE for } \mathcal{R}_{\text{PoE}}}$:
Params: $\text{pp} \leftarrow \text{PoE.Setup}(1^\lambda)$
Inputs: $(A_U, A_W \in \mathbb{G}, V(x) \in \mathbb{Z}_p[x])$
Claim: $A_W = A_U^{V(s)} \in \mathbb{G}$

1. Verifier sends $\ell \leftarrow\!\$\ \mathbb{Z}_p$
2. Prover computes:
   $q(x), r$ s.t. $V(x) = q(x) \cdot (x+\ell) + r$
   $Q_1 = g^{q(s)}$ using pp
   $Q_2 = g^{q(s)\cdot(\ell+s)}$ using pp
3. Prover sends Q_1, Q_2 to Verifier.
4. Verifier computes: r s.t. $r \equiv V(x) \mod (x+\ell)$
   Accepts if: $e(Q_1, g^{(s+\ell)}) \stackrel{?}{=} e(Q_2, g) \wedge$
   $e(A_U, Q_2) \cdot e(A_U, g^r) \stackrel{?}{=} e(A_W, g)$

Figure 1: PoE protocol. We use Fiat-Shamir transformation to make this protocol into non-interactive. For the ease of exposition we present the construction in the symmetric pairing setting. However, we remark that our implementation uses asymmetric paring.

We present the interactive version of the protocol for the symmetric pairing in Fig. 1, which can be made non-interactive using Fiat-Shamir transformation. The soundness of $\mathcal{R}_{\text{PoE}}$ is defined similar to its RSA counterparts as defined in [51] and [10]. We defer the proof of soundness of our PoE protocol to the extended version of our paper.

9 EVALUATION

In this section, we experimentally compare the aggregation operations in the RSA [10] and BP setting. We implement RSA accumulator using C++17, GNU Multiple Precision Arithmetic library 6.2.1 [52], and OpenSSL 3.0.2 [45]. We choose two 1024-bits prime numbers at random and compute the product to obtain a 2048 RSA modulus (using OpenSSL [45]). We implement [2] the BP accumulator using Golang bindings of the `mcl` library [36, 46]. Specifically, we use BLS12-381, a pairing-friendly elliptic curve. A single group element $\mathbb{G}$ and a field element $\mathbb{F}$ in the RSA setting, by the virtue of the choice of parameters, are 256 and 32 Bytes, respectively. In the elliptic curve group, a single compressed $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ group element requires 48, 96, 576 Bytes, respectively. Moreover, an element in $\mathbb{Z}_p$ requires 32 Bytes. A single exponentiation in the RSA group $\mathbb{G}$ by an exponent at most 256-bits takes 449 μs on an average. However, a single exponentiation in the elliptic curve source groups take $\mathbb{G}_1$ and $\mathbb{G}_2$ takes 106 μs and 250 μs, respectively. As BLS12-381 curve contains numerous roots of unity, we implement FFT based polynomial algorithms to support fast polynomial operations in `go-mcl` [46].

Our implementation is single threaded and all our experiments where performed on an Intel Core i7-4770 CPU @ 3.40GHz with 8 cores and 32 GiB of RAM. Unless stated otherwise, we perform 3 runs of each experiment and report the average.

[2] Our code is available at: https://github.com/accumulators-agg/accumulators

Operation	Sch.	Batch size 2^9	2^{11}	2^{13}	2^{15}	2^{17}
Domain mapping (s)	RSA	0.33	1.31	5.25	20.99	83.95
	BP	0	0	0	0	0
Commit (s)	RSA	0.52	2.09	8.38	33.55	134.37
	BP	0.05	0.24	1.12	5.17	24.28
AggMem (min)	RSA	0.04	0.17	0.8	3.66	16.49
	BP	0	0.01	0.18	2.51	38.35
MemVerify (s)	RSA	0.52	2.1	8.38	33.54	134.37
	BP	0.11	0.46	2.0	8.65	38.14
AggMemPoE (min)	RSA	0.04	0.19	0.86	3.87	17.33
	BP	0	0.02	0.2	2.58	39.12
MemVerifyPoE (s)	RSA	0.33	1.32	5.32	21.4	86.16
	BP	0.03	0.16	0.77	3.8	18.62
AggNonMem (min)	RSA	0.05	0.25	1.16	5.3	23.9
	BP	0.1	1.54	24.54	N/A	N/A
NonMemVerify (s)	RSA	0.72	2.87	11.49	45.98	184.12
	BP	0.11	0.46	2.0	8.65	38.14
AggNonMemPoE (min)	RSA	0.07	0.3	1.37	6.14	27.24
	BP	0.1	1.55	24.58	N/A	N/A
NonMemVerifyPoE (s)	RSA	0.34	1.34	5.33	21.41	86.17
	BP	0.03	0.16	0.77	3.8	18.62

Table 2: Accumulator batching operation costs for different batch sizes. In the first column, (s) denotes seconds and (min) minutes. The costs for RSA operations include the computations required to map to the prime domain. N/A stands for very large costs which are not interesting to compute.

9.1 Aggregation

In Table 1, we present the asymptotic costs for various operations in the BP setting, and in Table 2 we present our corresponding evaluation results.

Public parameters. Both the RSA and the BP accumulators require a trusted setup phase to generate the public parameters. Classgroups based accumulator constructions [13] do not require trusted setup in the unknown-order group setting, but they are too slow in practice. The public parameters in the RSA setting is just the RSA modulus and the group generator. However, in the BP setting, the public parameter consists of $n \cdot \mathbb{G}_1 + n \cdot \mathbb{G}_2$ elements, where n is the maximum size of the accumulated set[3]. For $n = 2^{17}$, the public parameters occupies 18 MiB. With PoE, it is sufficient to store *only* constant number of values (g_1, g_1^s, g_2, g_2^s) from the public parameter by the verifier. Recall that g_1, g_2 are the group generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

Domain mapping. To add values from an arbitrary domain $\mathcal{D}$ to the accumulator set, each element has to be mapped to a value in the specific accumulator domain $\mathcal{D}'$. Since the RSA accumulator requires a prime domain, we implement the standard "hash to prime" algorithm [9, 10], where the hash operation is successively applied on the input until the hash function returns a prime value. We use Blake2s hash implementation in OpenSSL [45] to convert a value from arbitrary domain to Prime domain and GMP's Miller-Rabin's primality testing (15 rounds). For the BP setting, as discussed in §3.3, the specific domain is $\mathbb{Z}_p$. In our implementation we use the BLS12-381 curve for which the group order is around 256-bits [11]. Thus, we can accumulate any arbitrary string of size up to 256-bits without the need of any mapping.

In our experiments, we consider the accumulation of arbitrary 256-bit strings. In Table 2, we report the cost mapping these arbitrary strings to the accumulator domain. Mapping an arbitrary string to the prime domain takes 640 to 894 μs for 15 to 50 rounds of Miller-Rabin primality testing. Before performing an accumulator operation, all elements have to be converted to the accumulator domain. Thus, we include the cost of mapping to the accumulator domain for all operations in Table 1.

Discussion. We now briefly discuss what could go wrong in a universal accumulator if we don't include the mapping process during verification.

Consider the two most popular $\mathsf{H}_{\mathsf{prime}}$ approaches [9, 10][4] for mapping a non-prime element x to a prime y: (1) perform repetitive hashing $H(H(\ldots H(x))) = y$ until a primality test indicates that the output is a prime, then outputs x and r where r is the number of hashing rounds required, (2) perform repetitive hashing of the value $(x||r)$, where r is a random nonce, until $H(x||r) = y$ is a prime. In both cases, when a prover wishes to add element x to the RSA accumulator, it first calculates the mapping y and stores r. The element y is accumulated. When they wish to prove (non-)membership, they provide r as a witness along with the proof to decrease verification costs. However, if a malicious prover can find two numbers of hashing rounds r_1, r_2 for the same element x, that correspond to two elements y_1, y_2 in $\mathcal{D}'$, then if say y_1 was the accumulated value (for x), the malicious prover could use y_2 to argue non-membership for x.

In order to avoid such attacks in universal accumulators[5], before verifying any (non-)membership proof, *all* verifiers need to check that the given prime mapping y is the first one that corresponds to the arbitrary element x and thus need to run all the repetitive steps the prover does. If the proof is a batch proof for $|I|$ elements, the verifiers need to repeat the process $|I|$ times individually for each element. The same holds for updates (addition or deletions) to ensure primality since they might be initiated by an untrusted entity.

Commit. To commit to a set I in the RSA setting, we first compute the product of the elements in the set. Then, we perform modular exponentiation of this large product of size $O(\lambda \cdot |I|)$-bits. However, for BP accumulators, we first compute the coefficient of the accumulator polynomial using the subproduct algorithm and fast polynomial multiplication. Then, we perform a single multi-exponentiation of size $|I|$. Observe that from Table 2, even after subtracting the domain mapping costs from the Commit, BP accumulators are faster! For a set size of 2^{15}, it takes around 12.56 seconds to perform Commit, in the RSA setting, whereas it takes just 5.17 seconds in the BP setting.

[3] Rather than relying on a trusted entity, it is possible to use an MPC based setup ceremony to generate public parameters. We can adopt approaches from real-world MPC ceremonies of Zcash and AZTEC protocol which have successfully generated SNARK parameters for circuit sizes 2^{21} and 2^{27}, respectively [12][50].

[4] A recent work [39], attempts to optimize the "hash to prime" approach, by using Pocklington primality certificates in order to reduce the cost of primality testing on the side of the verifier. However, it still does not guarantee a deterministic mapping.

[5] If the accumulator does not support non-membership, then this attack does not apply.

Operation	RSA (Bytes)	BP (Bytes)
Digest	256	48
Mem. proof	256	48
Non-mem. proof	288	80
Agg. mem. proof (Naive)	256	48
Agg. non-mem. proof (Naive)	3.85* MiB	144* Bytes
Agg. mem. proof (PoE)	512	192
Agg. non-mem. proof (PoE)	1312	336
PoE	256	144 or 96
PoKE	544	×

Table 3: Sizes of accumulator digest and proofs in bytes. Asterisk(*) denotes a batch size of 2^{17}.

Operation	Batch size 2^9	2^{11}	2^{13}	2^{15}
Pedersen Commitment (s)	0.08	0.31	1.24	4.98
Prover $\mathcal{R}_{\mathsf{mem}}$	2.97 ms			
Verifier $\mathcal{R}_{\mathsf{mem}}$	3.66 ms			
Proof size $\mathcal{R}_{\mathsf{mem}}$	0.91 KiB			
Prover $\mathcal{R}_{\mathsf{nonmem}}$	4.65 ms			
Verifier $\mathcal{R}_{\mathsf{nonmem}}$	5.18 ms			
Proof size $\mathcal{R}_{\mathsf{nonmem}}$	1.03 KiB			
Prover $\mathcal{R}_{\mathsf{degcheck}}$ (s)	0.17	0.64	2.57	11.29
Verifier $\mathcal{R}_{\mathsf{degcheck}}$	4.49 ms			
Proof size $\mathcal{R}_{\mathsf{degcheck}}$	0.29 KiB			

Table 4: Single-threaded microbenchmarks for our ZK constructions.

Membership aggregation. We implement the membership proof aggregation algorithm from Boneh et al. to aggregate a set of membership proofs in RSA accumulators [10]. Aggregating a pair of membership proofs involves computing Shamir's trick, which requires computing the Bézout coefficients and performing two exponentiations. However, aggregating membership proofs in BP accumulators, involves $O(|I| \log^2 |I|)$ field operations and one $|I|$-sized multi-exponentiation (Table 1). Thus, we observe that the prover's cost to aggregate is lower for the BP accumulator for set sizes up to $|I| = 2^{15}$. Beyond these set sizes, the field operations in BP dominates aggregation cost. It is not very common for a prover to hold (or wish to aggregate) more than 2^{15} proofs. Thus, for most applications BP should be preferable.

We also implement PoE from Boneh et al. [10] and from §8 to optimize the verification of the aggregated proof in RSA and BP accumulator, respectively. Since the prover overhead in computing the aggregated proof is dominated by field operations in both RSA and BP accumulators, the additional exponentiations overhead incurred by a PoE enabled prover is limited. We observe this in our experiments as the prover engaging PoE additional incurs only around 12.7 and 4.2 seconds for RSA and BP accumulators, respectively, for 2^{15} values (Table 2).

We observe that verifying batch proofs (without PoE) in BP accumulator is 3.5× to 4.7× faster than RSA accumulators. This is because the multi-exponentiations in elliptic curve group $\mathbb{G}_1$ is faster than a single large exponentiation in the RSA group $\mathbb{G}$. In the PoE enabled setting, we observe that BP verification is 4.6× to 11× faster than RSA. In addition to the domain mapping costs, we also include the overhead to compute the Fiat-Shamir coins in our experiments.

Non-membership aggregation. The prover's cost to aggregate non-membership proofs is better for RSA regardless of the use of PoE. This is due to BP's $O(|I|^2 \log |I|)$ field operations that comes from constructing the $Y_i(s)$ terms (§5.2).

In comparison with RSA, verifying batch non-membership proofs in BP is at least 4.6× faster in any case. Observe that verifying a batch non-membership is computationally similar to verifying a batch membership proof in the BP setting, regardless of the usage of PoE (Table 1). Thus, we observe similar performance numbers for NonMemVerify and NonMemVerifyPoE when compared to MemVerify and MemVerifyPoE, respectively, in the BP setting.

Storage and proof sizes. In Table 3, we present the storage overhead of proofs in both RSA and BP setting. For a similar level of security, an RSA group element $\mathbb{G}$ is of size 256 bytes as opposed to an elliptic curve element that is of size 48 Bytes for $\mathbb{G}_1$ and 96 Bytes for $\mathbb{G}_2$. The accumulator value and the batch membership proof consist of one group element in both constructions. Non-membership consists of one group element and one integer for RSA and two group elements in the BP setting. The integer in RSA batch non-membership proof grows linear in the batch size. The PoE proof adds to the proof size one group element in the RSA. However, in the BP setting PoE adds an overhead of either $(\mathbb{G}_1, \mathbb{G}_1)$ or $(\mathbb{G}_1, \mathbb{G}_2)$ depending on whether prover computes a proof for $g_2^{I(s)}$ or $g_1^{I(s)}$, respectively. The RSA non-membership proof can be made succinct using PoE and PoKE [10]. We note that in BP, non-membership proofs do not need PoKE as the proofs are already constant sized. Thus, we observe that membership and non-membership proofs in BP accumulators are 2.5× to 5× smaller and 3.5× smaller than the RSA accumulators, respectively.

9.2 Zero-knowledge batch proofs

We microbenchmark our proposed ZK batch proofs in Table 4.

Public parameters. The constructions for $\mathcal{R}_{\mathsf{mem}}$ and $\mathcal{R}_{\mathsf{nonmem}}$, require the prover and the verifier to store additional generators for Pedersen commitment. However for $\mathcal{R}_{\mathsf{degcheck}}$, since we rely on t-PKE assumption, the prover needs to additionally store $3n \cdot \mathbb{G}_2$ elements (36 MiB). Whereas, the verifier needs to store only $2n \cdot \mathbb{G}$ elements (24 MiB). When $n = 2^{17}$, generating additional $3n \cdot \mathbb{G}_2$ parameter takes around 98 seconds using a single thread.

Prover overhead. To commit to coefficients of a polynomial using Pedersen commitment, we use the values $g_2^{s^i}$ and an independent group generator. Using multi-exponentiation, it takes 4.98 seconds to commit to a batch of size 2^{15}. Recall that, given a commitment to the subset I, prover incurs constant overhead to generate a ZK proof for $\mathcal{R}_{\mathsf{mem}}$ and $\mathcal{R}_{\mathsf{nonmem}}$ regardless of the batch size. Thus,

to prove $\mathcal{R}_{\text{mem}}$ and $\mathcal{R}_{\text{nonmem}}$, it takes 2.97 and 4.65 milliseconds, respectively. To prove a lower bound on the degree of $I(s)$, the prover needs to compute Pedersen commitments on $I(s)$ with and without t-PKE.

Verification time and proof size. The proof of $\mathcal{R}_{\text{mem}}$ is $(4 \cdot \mathbb{G}_1 + \mathbb{G}_T + 5 \cdot \mathbb{Z}_p)$ and the proof of $\mathcal{R}_{\text{nonmem}}$ is $(3 \cdot \mathbb{G}_1 + \mathbb{G}_2 + \mathbb{G}_T + 6 \cdot \mathbb{Z}_p)$. With a 64-bit integer to denote the degree and three elements in $\mathbb{G}_2$, a prover can prove a lower bound on the degree of a polynomial. All the proofs in our scheme can be verified with a constant number of exponentiations and pairing operations.

9.3 Comparison with HARiSA [18]

In this subsection, we argue that our approach to ZK batch proofs of membership can be at least 16× faster than the current state-of-the-art approaches to ZK batch membership proofs in the RSA setting for a reasonable choice of batch size. We also report the performance of our ZK batch proof of non-membership in Fig. 2. HARiSA does not support non-membership.

Experimental setup. We fix the maximum size of the set to 2^{17} elements and measure the performance of computing the zero-knowledge proof of batch (non-)membership while revealing the size of the batched subset. Moreover, we consider an experimental setup where the prover must do maximal work. That is, we assume that the prover: (1) has access only to the individual (non-)membership witness but not the batch membership witness, (2) does not have access to the commitment to the batched subset, and (3) has access to the accumulator digest and public parameters. Thus, the prover incurs the cost of: (1) computing the commitment to set I, (2) aggregating the individual (non-)membership witnesses to obtain batch witness, (3) proving the relation $\mathcal{R}_{\text{mem}}$ or $\mathcal{R}_{\text{nonmem}}$, and (4) proving the relation $\mathcal{R}_{\text{degcheck}}$ for $d = |I|$.

Baseline measure. We compare the performance of our scheme against results of HARiSA [18] by Campanelli et al., which builds a succinct batch proof of membership while preserving the privacy of the batched elements. Their work combines proof of knowledge of exponent (PoKE) along with CP-SNARK for integer arithmetic relations and bound checks to prove batch membership. They implement their construction using LegoGroth16 in C++. Similar to our experiments, they require the prover to compute the batch witness using individual witnesses and their experiments are single

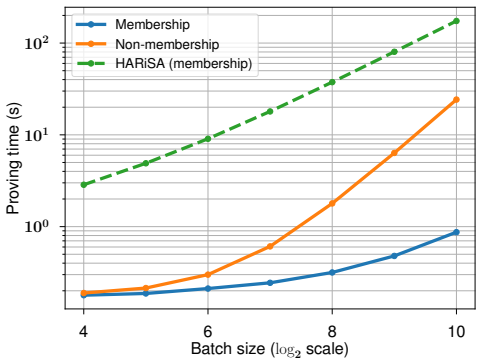

Figure 2: We extrapolate the proving costs using the numbers reported in HARiSA [18]. Note that the results in the RSA setting does include the Hash-to-prime costs.

Scheme		Setup	Verifier	Proof size
HARiSA [18]	Mem.	Trusted	63 ms	1.14 KiB
This work	Mem.	Trusted	7.94 ms	1.2 KiB
	Non-Mem.		9.41 ms	1.32 KiB

Table 5: Verification overhead and proof size.

threaded. Also, recall that in our experiments we reveal the size of the subset I, which is currently not implemented in HARiSA [18].

Proving time. HARiSA reports a prover time of 2.86 and 9.02 seconds for a batch of size 16 and 64, respectively. Recall that their implementation uses LegoGroth16 proof system, thus the prover time is dominated by large FFTs and exponentiations. The performance numbers reported by HARiSA uses Amazon EC2 r5.8xlarge [18, Figure 4], which we extrapolate for various batch sizes in Fig. 2. However, in Fig. 2, the performance of our scheme is measured on an Intel Core i7-4770 CPU@ 3.40GHz. We observe that for a batch size of 16, our approach takes merely 0.18 seconds, whereas HARiSA takes 2.86 seconds, thus resulting in 16× speed up. Clearly, our performance is still an order of magnitude faster even when benchmarked on a much slower machine. Thus, we argue that our order of magnitude performance gain will carry over even when we benchmark our scheme and HARiSA on the same machine.

To the best of our knowledge, there are no known sub-linear proof sized privacy preserving batch non-membership proofs in the RSA setting without generic arguments such as SNARKs. However, SNARKs based approaches are too slow in practice. Thus, we report the performance of our non-membership scheme in the context of membership scheme due to lack of appropriate baseline.

Verification time and proof size. Our approach has a comparable proof size and superior verification speed. We report this in Table 5.

ACKNOWLEDGMENTS

We thank Yupeng Zhang for pointing us to prior work on proving the degree of a polynomial in ZK and Alin Tomescu for productive discussions. Foteini Baldimtsi and Ioanna Karantaidou were partially supported by NSF #171767, NSF #2143287 and awards from Facebook and Google Research. Charalampos Papamanthou is supported in part by awards from Protocol Labs, VMware, JPMorgan, the ACE-PAVE grant from the Algorand Foundation as well as NSF awards CNS-1652259 and CNS-2154732.

REFERENCES

[1] 2015. Hyperledger Indy website. https://www.hyperledger.org/ [Online; accessed 1-May-2022].
[2] Tolga Acar and Lan Nguyen. 2011. Revocation for delegatable anonymous credentials. In *International Workshop on Public Key Cryptography*. Springer, 423–440.
[3] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. 2018. Non-Interactive Zero-Knowledge Proofs for Composite Statements. In *Advances in Cryptology - CRYPTO (Lecture Notes in Computer Science, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, 643–673.
[4] Man Ho Au, Patrick P Tsang, Willy Susilo, and Yi Mu. 2009. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *Topics in Cryptology – CT-RSA 2009*, Marc Fischlin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 295–308.
[5] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov. 2017. Accumulators with Applications to Anonymity-Preserving Revocation. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*. 301–315. https://doi.org/10.1109/EuroSP.2017.13 https://eprint.iacr.org/2017/043.

[6] Niko Barić and Birgit Pfitzmann. 1997. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In *Advances in Cryptology — EUROCRYPT '97*, Walter Fumy (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 480–494.

[7] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. 2020. A classification of computational assumptions in the algebraic group model. In *Annual International Cryptology Conference*. Springer, 121–151.

[8] Josh Benaloh and Michael de Mare. 1993. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In *Advances in Cryptology — EUROCRYPT '93*, Tor Helleseth (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 274–285. https://www.microsoft.com/en-us/research/publication/one-way-accumulators-a-decentralized-alternative-to-digital-signatures/

[9] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. 2021. Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 393–414. https://doi.org/10.1007/978-3-662-64322-8_19

[10] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2019. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In *Advances in Cryptology – CRYPTO 2019*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Springer International Publishing, Cham, 561–586. https://eprint.iacr.org/2018/1188.

[11] Sean Bowe. 2017. Fast amortized Kate proofs. https://electriccoin.co/blog/new-snark-curve/.

[12] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. Cryptology ePrint Archive, Report 2017/1050. https://ia.cr/2017/1050.

[13] Johannes Buchmann and Safuat Hamdy. 2011. A survey on IQ cryptography. In *Public-Key Cryptography and Computational Number Theory*. 1–15.

[14] Vitalik Buterin. 2020. Using polynomial commitments to replace state roots. https://ethresear.ch/t/using-polynomial-commitments-to-replace-state-roots/7095.

[15] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. 2009. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In *Public Key Cryptography – PKC 2009*, Stanisław Jarecki and Gene Tsudik (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 481–500. https://eprint.iacr.org/2008/539.

[16] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology — CRYPTO 2002*, Moti Yung (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 61–76.

[17] Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. 2020. Incrementally Aggregatable Vector Commitments and Applications to Verifiable Decentralized Storage. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 3–35.

[18] Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. 2021. Succinct Zero-Knowledge Batch Proofs for Set Accumulators. Cryptology ePrint Archive, Paper 2021/1672. https://eprint.iacr.org/2021/1672 https://eprint.iacr.org/archive/2021/1672/20220503:104027.

[19] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. 2014. Verifiable Set Operations over Outsourced Databases. In *Public-Key Cryptography – PKC 2014*, Hugo Krawczyk (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 113–130.

[20] Dario Catalano and Dario Fiore. 2013. Vector Commitments and Their Applications. In *International Workshop on Public Key Cryptography*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 55–72. https://eprint.iacr.org/2011/495.

[21] Alexander Chepurnoy, Charalampos Papamanthou, Shravan Srinivasan, and Yupeng Zhang. 2018. Edrax: A Cryptocurrency with Stateless Transaction Validation. Cryptology ePrint Archive, Report 2018/968. https://ia.cr/2018/968.

[22] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 738–768.

[23] Aisling Connolly, Pascal Lafourcade, and Octavio Perez Kempner. 2022. Improved Constructions of Anonymous Credentials from Structure-Preserving Signatures on Equivalence Classes. In *Public-Key Cryptography – PKC 2022*, Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe (Eds.). Springer International Publishing, Cham, 409–438.

[24] Ivan Damgård and Nikos Triandopoulos. 2008. Supporting Non-membership Proofs with Bilinear-map Accumulators. Cryptology ePrint Archive, Report 2008/538. https://eprint.iacr.org/2008/538.

[25] Dankrad Feist and Dmitry Khovratovich. 2020. Fast amortized Kate proofs. https://github.com/khovratovich/Kate.

[26] Esha Ghosh, Olga Ohrimenko, Dimitrios Papadopoulos, Roberto Tamassia, and Nikos Triandopoulos. 2016. Zero-Knowledge Accumulators and Set Algebra. In *Advances in Cryptology – ASIACRYPT 2016*. Springer Berlin Heidelberg, 67–100. https://doi.org/10.1007/978-3-662-53890-6_3

[27] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. 2020. Pointproofs: Aggregating Proofs for Multiple Vector Commitments. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 2007–2023. https://doi.org/10.1145/3372297.3417244 https://eprint.iacr.org/2020/419.

[28] Jens Groth. 2010. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 321–340.

[29] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–194. https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf.

[30] Dmitry Khovratovich and Jason Law. 2017. Sovrin: digital identities in the blockchain era. *Github Commit by jasonalaw October* 17 (2017), 38–99.

[31] Russell W.F. Lai and Giulio Malavolta. 2019. Subvector Commitments with Application to Succinct Arguments. In *Advances in Cryptology – CRYPTO 2019*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Springer International Publishing, Cham, 530–560. https://eprint.iacr.org/2018/705.

[32] Jiangtao Li, Ninghui Li, and Rui Xue. 2007. Universal Accumulators with Efficient Nonmembership Proofs. In *Applied Cryptography and Network Security*, Jonathan Katz and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 253–269. https://www.cs.purdue.edu/homes/ninghui/papers/accumulator_acns07.pdf.

[33] Helger Lipmaa. 2012. Secure Accumulators from Euclidean Rings without Trusted Setup. In *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*, Feng Bao, Pierangela Samarati, and Jianying Zhou (Eds.), Vol. 7341. Springer, 224–240.

[34] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, Carl Pomerance (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.

[35] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*. 397–411. https://doi.org/10.1109/SP.2013.34

[36] Mitsunari Shigeo. 2020. mcl: a portable and fast pairing-based cryptography library. https://github.com/herumi/mcl/ Accessed: 2020-10-14.

[37] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *34th International Symposium on Distributed Computing (DISC 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 179)*, Hagit Attiya (Ed.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 28:1–28:17.

[38] Lan Nguyen. 2005. Accumulators from Bilinear Pairings and Applications. In *Topics in Cryptology – CT-RSA 2005*, Alfred Menezes (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 275–292. https://eprint.iacr.org/2005/123.

[39] Alex Ozdemir, Riad Wahby, Barry Whitehat, and Dan Boneh. 2020. Scaling Verifiable Computation Using Efficient Set Accumulators. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2075–2092. https://www.usenix.org/conference/usenixsecurity20/presentation/ozdemir

[40] Charalampos Papamanthou. 2011. *Cryptography for Efficiency: New Directions in Authenticated Data Structures*. Ph. D. Dissertation. Brown University, Providence, Rhode Island. https://doi.org/10.7301/Z0Z60M99.

[41] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. 2011. Optimal Verification of Operations on Dynamic Sets. In *Advances in Cryptology – CRYPTO 2011*, Phillip Rogaway (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 91–110.

[42] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.

[43] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. 2017. Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In *European Symposium on Research in Computer Security*. Springer, 456–474.

[44] Steve Thakur. 2019. Batching non-membership proofs with bilinear accumulators. Cryptology ePrint Archive, Report 2019/1147. https://eprint.iacr.org/archive/2019/1147/20210929:175523.

[45] The OpenSSL Project. 2003. OpenSSL: The Open Source toolkit for SSL/TLS. (April 2003). www.openssl.org.

[46] Alin Tomescu. 2022. go-mcl. https://github.com/alinush/go-mcl. https://github.com/alinush/go-mcl [Online; accessed 1-May-2022].

[47] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. 2020. Aggregatable Subvector Commitments for Stateless Cryptocurrencies. In *Security and Cryptography for Networks*. Springer International Publishing, Cham, 45–64. https://eprint.iacr.org/2020/527.

[48] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. 2019. Transparency logs via append-only authenticated dictionaries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1299–1316.

[49] Giuseppe Vitto and Alex Biryukov. 2020. Dynamic Universal Accumulator with Batch Update over Bilinear Groups. Cryptology ePrint Archive, Report 2020/777. https://eprint.iacr.org/2020/777.

[50] Thomas Walton-Pocock. 2020. AZTEC CRS: The Biggest MPC Setup in History has Successfully Finished. https://medium.com/aztec-protocol/aztec-crs-the-biggest-mpc-setup-in-history-has-successfully-finished-74c6909cd0c4.
[51] Benjamin Wesolowski. 2019. Efficient Verifiable Delay Functions. In *Advances in Cryptology – EUROCRYPT 2019*, Yuval Ishai and Vincent Rijmen (Eds.). Springer International Publishing, Cham, 379–407. https://eprint.iacr.org/2018/623.
[52] Wikipedia contributors. 2020. The GNU Multiple Precision Arithmetic Library — Wikipedia, The Free Encyclopedia. https://gmplib.org/manual/Extended-GCD
[53] Wikipedia contributors. 2020. Partial fraction decomposition — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Partial_fraction_decomposition&oldid=931350868 [Online; accessed 11-April-2020].
[54] Wikipedia contributors. 2020. Polynomial greatest common divisor — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Polynomial_greatest_common_divisor&oldid=993818674#B%C3%A9zout's_identity_and_extended_GCD_algorithm [Online; accessed 12-December-2020].
[55] Wikipedia contributors. 2020. Polynomial remainder theorem — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Product_rule&oldid=992085655 [Online; accessed 03-December-2020].
[56] Dae Hyun Yum, Jae Woo Seo, and Pil Joong Lee. 2008. Generalized Combinatoric Accumulator. *IEICE - Trans. Inf. Syst.* E91-D, 5 (May 2008), 1489–1491. https://doi.org/10.1093/ietisy/e91-d.5.1489
[57] Y. Zhang, J. Katz, and C. Papamanthou. 2017. An Expressive (Zero-Knowledge) Set Accumulator. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*. 158–173. https://doi.org/10.1109/EuroSP.2017.35