ZAC: Efficient Zero-Knowledge Dynamic Universal Accumulator and Application to Zero-Knowledge Elementary Database

Dang, Hai Van; Phuong, Tran Viet Xuan; Nguyen, Thuc D.; Hoang, Thang

Published in:
2022 IEEE 4th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)

DOI:
10.1109/tps-isa56441.2022.00038

Publication date:
2022

ZAC: Efficient Zero-Knowledge Dynamic Universal Accumulator and Application to Zero-Knowledge Elementary Database

Hai-Van Dang
University of Plymouth
hai-van.dang@plymouth.ac.uk

Tran Viet Xuan Phuong
Old Dominion University
tphuong@odu.edu

Thuc D. Nguyen
University of Science, VNU-HCM
ndthuc@fit.hcmus.edu.vn

Thang Hoang
Virginia Tech
thanghoang@vt.edu

***Abstract*—Zero-knowledge universal accumulator generates the succinct commitment to a set and produces the short (non) membership proof (*universal*) without leaking information about the set (*zero-knowledge*). In order to further support a generic set and zero-knowledge, existing techniques generally combine the zero-knowledge universal accumulator with other protocols, such as digital signatures and hashes to primes, which incur high overhead and may not be suitable for real-world use. It is desirable to commit a set of membership concealing the information with the optimal complexity. We devise ZAC, a new zero-knowledge Dynamic Universal Accumulator by taking the existing cryptographic primitives into account to produce a new efficient accumulator. Our underlying building blocks are Bloom Filter and vector commitment scheme in [19], utilizing the binary expression and aggregation to achieve efficiency, generic set support, zero-knowledge and universal properties. As a result, our scheme is improved in terms of proof size and proof time, also comparable to the RSA-based set accumulator in [8] in the verifying complexity. With 128 bit security, our proof size is 48 bytes while theirs is 1310 bytes and the running time of elliptic curve-based methods is faster than RSA-based counterpart. ZAC is proved to be complete, ϵ-sound and zero-knowledge. Extensively, based on ZAC as building block, we construct a new Zero-Knowledge Elementary Database (ZKEDB), which consumes 5 times less storage space, $\mathcal{O}(\log N)$ less bandwidth, and $\mathcal{O}(\log N)$ more efficient in proving and verification than the state-of-art work in [13] (where N is the domain space size). ZKEDB is proved to be complete, ϵ-sound and zero-knowledge. ZKEDB supports a new type of *select top* ℓ query, and can be extended to non-elementary databases.**

***Index Terms*—zero-knowledge universal dynamic accumulator, zero-knowledge set (database), Bloom filter**

I. Introduction

Cryptographic accumulator primitives introduced by Benaloh et al. [6] plays a crucial part to adapt into the wide applications such as identity escrow and anonymous credentials [11], which enable the efficient membership revocation in the anonymous setting. Technically, a cryptographic accumulator creates a compact *commitment* (i.e. accumulator) C to a set $\mathcal{S}$ with membership *proofs* of its elements. The proofs can be publicly verified against the commitment to prove an element to have been accumulated in the commitment value (i.e. prove x belong to S). By a constraint, the accumulators must satisfy the *set binding* property [12] that an element cannot be proved to belong to the set while it is not. Therefore, the verification of an element x belonging to the accumulated set $\mathcal{S}$ is the underlying mechanism to enable the membership/non-membership proofs. Other desired property, an accumulator is *dynamic* if the commitment and proofs can be updated with constant cost when adding or removing an item from the set. A dynamic accumulator is said to be *universal* [24] if it can support both membership and non-membership proofs. In a non-membership proof, one can prove that an element is not accumulated in the commitment value.

In order to construct a universal dynamic accumulator, Catalano et al. [12] presented a solution by using *Vector Commitment* (VC), a very closely related primitive. VC is a protocol of two stages, i.e. commit-then-prove, with involvement of two main actors: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$. In the committing stage, $\mathcal{P}$ commits to vector $\mathbf{m} = (m_1, \ldots, m_q)$ in such a way that $\mathcal{V}$ cannot learn about $\mathbf{m}$ (*hiding* property). In the proving state, $\mathcal{P}$ can generate a proof to prove that an element is an item of the vector at a specific position (i.e. prove x be value of m_i). VCs must also satisfy the *position binding* property, meaning that an element cannot be proved to be an item at a specific position of the vector while it is not. Later on, in [23], Lai et al. presented the notion of *Subvector Commitments* (SVC). SVCs allow to commit to a vector $\mathbf{m}$, and then prove values of multiple items of the vector. Recently, in [19], Gorbunov et al. proposed a new vector commitment scheme with aggregation called Pointproofs. The scheme allows to aggregate multiple proofs with respect to a same commitment or different commitments into one proof.

Cryptograhic accumulators and vector commitments have been incorporated widely. Among their various applications, zero-knowledge set, first introduced in [28], has a paramount of importance. Zero-knowledge sets (ZKS) allow to (i) commit to a set S (ii) prove if $x \in S$ or $x \notin S$ without revealing anything about S and its cardinality $|S|$. ZKS can be considered as a special case of zero-knowledge elementary database ZKEDB (elementary database D is a set of tuples (x, v_x) where x is the key and its corresponding value is $D(x) = v_x$). ZKEDB allows to (i) commit the database D (ii) then prove that the query result for a key x is $D(x)$ or $\perp$. A ZKS S is an instance of ZKEDB D where the set of keys is S and the corresponding values are all 1. When successfully proving

TABLE I: Comparison of ZAC with prior works.

Scheme	Properties				Performance		
	Dyn.	Gen.	Univ.	zk	Proving time	Verification time	$\|\pi\|$ (bytes)
[6]		✓			$\mathcal{O}(Nm)$ H	$\mathcal{O}(m)$ H	34
[12]	✓		✓		$\mathcal{O}(N \log N \cdot \lambda)$ mult	$\mathcal{O}(N)$ mult	256
[8] (w/o hashing to prime)	✓		✓		$\mathcal{O}(N \log N)$ mult	$\mathcal{O}(\lambda)$ mult + $\mathcal{O}(N \log N)$ $\mathbb{F}$	1310
[8] (w/ hashing to prime)	✓	✓	✓		$\mathcal{O}(N \log N)$ mult + $\mathcal{O}(\lambda)$ primality checks	$\mathcal{O}(\lambda)$ mult + $\mathcal{O}(N \log N)$ $\mathbb{F}$	1310
[32]	✓		✓	✓	$\mathcal{O}(m^2)$ $\mathbb{F}$ + $\mathcal{O}(1)$ mult	$\mathcal{O}(1)$ mult + 2e; $\mathcal{O}(p)$ mult + 3e*	56
Our ZAC scheme	✓	✓	✓	✓	$\mathcal{O}(m)$ H + $\mathcal{O}(q)$ mult	$\mathcal{O}(m)$ H + $\mathcal{O}(q)$ mult + 2e	48

Dyn. means dynamic, **Gen.** means generic set, **Univ.** means universal, **zk** means zero-knowledge, $|\pi|$: proof size. λ : security parameter, N : size of the committed set; $m < N$: size of the proved subset; $q < N$: length of Bloom Filter representation of the committed set; e denotes pairing cost. H, mult and $\mathbb{F}$ denote the hash function, group multiplication operation and multiplication in field of size approximately 2^λ, respectively; *: the cost is $\mathcal{O}(1)$ mult + 2e in case of membership verification and $\mathcal{O}(p)$ mult + 3e in case of non-membership where p is prime number and group size. Proof size $|\pi|$ (in bytes) at 128-bit security for $N = 1000$ and 256-bit hash ([19], Section 5.3)

a query result for a key x to be equal to 1 ($D(x) = 1$), it means $x \in S$. Otherwise, when proving the query result to $\perp$, it means $x \notin S$. ZKEDB can be constructed using vector commitments and Merkle tree as in [15], [28]. However, most of the constructions are expensive in terms of performance overhead.

Motivation. We observe that the construction of Universal Dynamic Accumulators from vector commitments in [12] does not support a generic set and zero-knowledge property without requiring the accompanying with the other protocols like secure digital signature and standard commitment schemes. Consequently, the extra schemes would be costly overhead. Moreover, considering a set of N records of m bits, [12] requires to calculate commitment of 2^m values; therefore, we propose a new model Universal Dynamic Accumulators from vector commitments which reduces the expensive commitment values as N which $N \ll 2^m$; .

Contributions. We present our contributions as two-folds. Firstly, we propose a new efficient zero-knowledge Dynamic Universal Accumulator. Secondly, we design a new efficient Zero-Knowledge Elementary Database. Both protocols are proved to be complete, ϵ-sound and zero-knowledge.

New Efficient Zero-Knowledge Dynamic Universal Accumulator. Table I summarizes the properties and efficiency of our proposed ZAC scheme, compared with the state-of-the-art works. We design a new zero-knowledge accumulator by mainly exploiting the Bloom Filter [17] and the aggregatable vector commitment of [19]. As a result, our proposed scheme, **Z**ero-Knowledge Dynamic Universal **Ac**cumulator (ZAC), generates a short commitment to a generic set and short membership/ non-membership proofs for any subset of a set without leaking the remained elements, the set and subset size (zero-knowledge). For the comparison, we selected the work [6] because it is the state-of-art paper about the accumulator, [12] having the same approach of using vector commitment to construct the set accumulator, [8] using RSA-based approach instead of bilinear map approach and supporting batching, updates as well, [32] being a recent work.

ZAC achieves **succinctness** property in the following metrics:

- The (non)membership proving time is $\mathcal{O}(m)$ H+$\mathcal{O}(q)$ mult.
- The verifying time is $\mathcal{O}(m)$ H + $\mathcal{O}(q)$ mult + 2 e.

The commitment size and the proof size equal to the size of a group element given H, mult, e, group multiplication operation, and pairing functions. The m, q are the size of the proved subset $\hat{S}$ and the parameter of Bloom filter, respectively. Moreover, from a practical point of view, our scheme operates on a smaller group (*e.g.*, ECC) compared with [8] (which is based on RSA group), and thus the arithmetic operations in our scheme is also faster [31].

Comparably, the commitment and membership proof size in our scheme (48 bytes) is 27 times less than the work by Boneh et al. [8] (1310 bytes) with respect to 128-bit security. Our membership proof creating time is less while membership proof verifying time is comparable with [8]. Our scheme is **universal** in the sense that it supports both membership and non-membership proof. In comparison with the membership proof, the overhead of generating and verifying non-membership proof in ours is negligible (bitwise equality and AND operations) while it is approximately doubled in [8]. Additionally, our commitment and proof update cost are more efficient than [8] ($\mathcal{O}(N')$ H + $\mathcal{O}(q)$ mult < $\mathcal{O}(N' \log(N'))$ mult given N' as the number of update items). ZAC supports a generic set (for e.g. a set of European Union countries) without overhead complexity while the other works would consume overhead complexity (i.e. hash-to-prime mapping like in [8]) or redundancy (like in [12]). Finally, ZAC does not leak any extra information about the committed and the proved set, even their size (zero-knowledge).

Regarding the zero-knowledge set (elementary database), the approach of using Merkle tree data structure has proof size and proof computation cost as $\mathcal{O}(\log(\lambda))$ [28], [15], $\mathcal{O}(\lambda \cdot q/\log(q))$ [13], $\mathcal{O}(\lambda/\log(q))$ [26] where λ is the security parameter in such a way that the database size upper bound is 2^λ and q is the branching factor of the tree [26]. The mentioned works currently cannot be extended to support non-elementary database efficiently. Hence, we also aim at designing a zero-knowledge elementary database with proof size and computation cost as $\mathcal{O}(1)$, and easily extending it into a zero-knowledge non-elementary database.

New zero-knowledge (non)elementary database with short proof and new type of query. We design a new zero-knowledge elementary database using ZAC as a building block. Our construction, Zero-Knowledge Elementary Database or ZKEDB, incurs less proving and verifying time and has shorter proof size ($\mathcal{O}(1)$) than the schemes based on the Merkle tree ($\mathcal{O}(\lambda)$ [15], $\mathcal{O}(\lambda \cdot q/\log(q))$ [13], $\mathcal{O}(\lambda/\log(q))$ [26] where λ is the security parameter in such a way that the database size upper bound is 2^λ, q is the

branching factor of the Merkle tree). In addition, our design supports a new type of query, *select top* ℓ *where* $y = a$, with only one proof and verification compared to ℓ proofs and verifications of the Merkle tree based schemes. Ours can be extended to zero-knowledge non-elementary database with short proof while the others cannot.

II. Related Work

Although commitment and vector/ subvector commitment are very closely related, the paper mainly focuses on set accumulator schemes and its application to zero-knowledge elementary database (ZKEDB). Accordingly, the related works discuss about accumulators and ZKEDB instead of commitment schemes.

***Accumulators**:* The primitive *accumulator* was first introduced in [6], eliminating the need for a trusted central authority to accumulate values and being suitable to construct space-efficient protocols for time stamping and membership testing. The participants only need to store the accumulator value (i.e. commitment) instead of the whole list of individual values. Furthermore, the commitment size is independent from the number of the accumulated values. However, the scheme supports subsets of $[1, n-1]$ instead of an arbitrary set, and does not support either non-membership proofs or updates. In [11], Camenisch et al. extended the notion to *dynamic accumulator* which allowed to dynamically add or delete an item into/from the commitment with cost independent of the set size, and update the proofs accordingly at unit cost without any required trapdoor. Their proposal is dynamic, which makes it attractive for the applications with granting/ revoking functionalities like identity escrow or group signature. Besides that, the scheme is provable with respect to the Pedersen commitment scheme (i.e. it can prove a value in a Pedersen commitment is contained in an accumulator), providing *zero-knowledge* proof of membership with overhead computation for the commitment scheme. However, it supports subsets of $[1, n-1]$ instead of an arbitrary set, and does not support non-membership proofs. In [12], Catalano et al. presented a universal accumulator constructed based on a generic vector commitment. The scheme does not support batching updates, and only supports a subset of $[n]$ straightforward. In order to support a generic set, the authors suggested a combination with a digital signature scheme, which incurred an additional overhead complexity. If combined with a commitment scheme, the proposal can conceal actual value of the accumulated elements (*zero-knowledge*) at the cost of computation overhead. In [8], Boneh et al. presented a set of batching and aggregation techniques for accumulators, eliminating the role of a trusted accumulator manager, and supporting non-membership proofs. However, the accumulated values are odd primes. A mapping (i.e. hashing to primes) with additional cost would be necessary if they want to support accumulating to a generic set. In [22], Karantaidou et al. proposed a zero-knowledge universal accumulator using bilinear pairing setting. In order to enable non-membership proving, they convert non-membership proof into membership-proof of the complement set of the initial set. It is not efficient practically for a large domain because it requires accumulating every item of the domain. Recently, in [32], Vitto and Biryukov proposed a new dynamic universal accumulator with batch update over bilinear groups with the involvement of an accumulator manager. Their scheme is efficient but only support accumulating for elements of $(\mathbb{Z}/p\mathbb{Z})^*$.

***Zero-knowledge set (database)**:* In 2003, Micali et al.[28] introduced a first non-interactive zero-knowledge set (ZKS) and immediately yielded zero-knowledge elementary database (ZKEDB). Their construction relies on the usage of Pedersen commitment [30] and Merkle tree. In 2005, Chase et al.[14], [15] constructed zero-knowledge set (database) using mercurial commitments and Merkle tree. They showed that their construction was a generalization of the particular instantiation in [28]. In short, mercurial commitments extends the conventional commitments with four algorithms: hard-commit (i.e. traditional commit) and prove, soft-commit and tease. A soft commitment can never be proved; instead, it can be teased to any value at Prover $\mathcal{P}$'s choice. Meanwhile, a hard commitment to a value x can be proved to x and teased to x. A tease of a commitment to x ensures Verifier $\mathcal{V}$ that if the commitment can be proved then it will be proved to the same value. $\mathcal{P}$ cannot create a commitment that can be teased to x and proved to $x' \neq x$. To construct a ZKEDB $D = (\mathcal{K}, \mathcal{Y})$, Chase et al.[14], [15] built a Merkle tree with nodes as hard or soft commitments. The leaf nodes are numbered by keys ($\{x \in \mathcal{K}\}$), and contain commitments to their values ($\{D(x) \in \mathcal{Y}\}$). The query verification communication and computation cost are $\mathcal{O}(\log(\lambda))$ where λ is the security parameter in such a way that the database size upper bound is 2^λ. By increasing the branching factor of the Merkle tree from 2 to q and using a primitive called trapdoor q-mercurial commitment (qTMC) (which allows to commit to a vector of q messages at once, and later open at specific vector positions without disclosing messages at the other positions), Catalano et al. [13] improved the proof size and computation cost to $\mathcal{O}(\lambda \cdot q/\log(q))$. In [26], Libert et al. introduced trapdoor $q-$mercurial commitments with constant size proofs, and used it to construct a zero-knowledge set (database) with shorter proof and more efficient computation cost, $\mathcal{O}(\lambda/\log(q))$. While there has been improvement in proof size and computation cost in [28], [14], [15], [13], [26], they only support verifying simple statements such as "x does not belong in D" or "x is in D with value $y = D(x)$". In [25], Libert et al. extended verification for range queries over keys and values, which can be exploited to enable further complex queries such as k-nearest neighbors and k smallest or largest values within a given range. In other to support queries over keys and values, the scheme in [25] uses two Merkle trees to the same database, and adds checks to ensure the consistency of the two Merkle trees. In other words, it incurs storage and computation overhead to maintain and compute over the two Merkle trees. Liskov [27] defined the updatable zero-knowledge (elementary) database notion and proposed a construction based on the scheme of Chase et

al. [14], [15]. Camenisch et al. [10] constructed an updatable zero-knowledge (elementary) database scheme based on vector commitments, which achieve less proving/ verifying cost than Merkle tree based schemes. The construction in [10] considers keys of the database as positions, and values of the database as values of a committed vector. Therefore, it only supports keys as numbers in $[N]$ where N is the database size. Despite many constructions for ZKEDB, in our knowledge, there is no construction for zero-knowledge non-elementary database so far.

III. Definitions

Notation. We denote the set $\{1, \ldots, n\}$ as $[n]$, zero vector of size n as 0^n. We denote m_i (or $\mathbf{m}[i]$) as the i-th element in vector $\mathbf{m}$, $\mathbf{m}[\mathcal{S}] := \{m_j : j \in \mathcal{S}\}$ and $\mathbf{m}[-i] := \{m_j\}_{j \neq i}$. $x \xleftarrow{\$} \mathcal{S}$ means x is uniformly chosen at random from a set $\mathcal{S}$, $|\mathcal{S}|$ as the cardinality of set $\mathcal{S}$. Let $\mathsf{negl}()$ be the negligible function.

Zero-knowledge Dynamic Universal Accumulator. We define zero-knowledge Dynamic Universal Accumulator (zkDUA), which permits a prover $\mathcal{P}$ to commit to an arbitrary set $\mathcal{S}$, and then later proves that another set $\hat{\mathcal{S}}$ is the subset of $\mathcal{S}$ (*i.e.* $\hat{\mathcal{S}} \subseteq \mathcal{S}$) or $\hat{\mathcal{S}}$ is not a subset of $\mathcal{S}$ (*i.e.* $\hat{\mathcal{S}} \not\subseteq \mathcal{S}$), without revealing the items as well as the size of $\mathcal{S}$. A zkDUA scheme is a tuple of 9 algorithms $\mathsf{zkDUA} = (\mathsf{Init}, \mathsf{Com}, \mathsf{UpdCom}, \mathsf{ProveM}, \mathsf{ProveN}, \mathsf{UpdPrM}, \mathsf{UpdPrN}, \mathsf{VerifyM}, \mathsf{VerifyN})$ defined as follows.

- $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, N)$: Given the security parameter λ and the bound of set size N as input, it outputs public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{Com}(\mathcal{S}, r, \mathsf{pp})$: Given a set $\mathcal{S}$ and randomness r, it outputs a commitment cm.
- $\mathsf{cm}' \leftarrow \mathsf{UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$: Given two sets $\mathcal{S}, \mathcal{S}'$ and cm as the commitment of $\mathcal{S}$, it outputs a commitment cm' to $\mathcal{S}'$.
- $\hat{\pi} \leftarrow \mathsf{ProveM}(\mathsf{cm}, \mathcal{S}, \hat{\mathcal{S}}, r, \mathsf{pp})$: Given two sets $\mathcal{S}, \hat{\mathcal{S}}$, a commitment cm to $\mathcal{S}$ and randomness r, it outputs a proof $\hat{\pi}$ indicating that $\hat{\mathcal{S}} \subseteq \mathcal{S}$.
- $\bar{\pi} \leftarrow \mathsf{ProveN}(\mathsf{cm}, \mathcal{S}, \bar{\mathcal{S}}, r, \mathsf{pp})$: Given two sets $\mathcal{S}, \bar{\mathcal{S}}$, a commitment cm to $\mathcal{S}$ and randomness r, it outputs a proof $\bar{\pi}$ indicating that $\bar{\mathcal{S}} \not\subseteq \mathcal{S}$.
- $\hat{\pi}' \leftarrow \mathsf{UpdPrM}(\hat{\pi}, \hat{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$: Given three sets $\mathcal{S}, \hat{\mathcal{S}}, \mathcal{S}'$ and a proof $\hat{\pi}$ of $\hat{\mathcal{S}}$ over $\mathcal{S}$, it outputs a new proof $\hat{\pi}'$ to show that $\hat{\mathcal{S}} \subseteq \mathcal{S}'$.
- $\bar{\pi}' \leftarrow \mathsf{UpdPrN}(\bar{\pi}, \bar{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$: Given three sets $\mathcal{S}, \bar{\mathcal{S}}, \mathcal{S}'$ and a proof $\bar{\pi}$ of $\bar{\mathcal{S}}$ over $\mathcal{S}$, it outputs a new proof $\bar{\pi}'$ to show that $\bar{\mathcal{S}} \not\subseteq \mathcal{S}'$.
- $\{0,1\} \leftarrow \mathsf{VerifyM}(\mathsf{cm}, \hat{\mathcal{S}}, \hat{\pi}, \mathsf{pp})$: Given the commitment cm, a set $\hat{\mathcal{S}}$ and a proof $\hat{\pi}$, it outputs 1 if $\hat{\pi}$ is a valid proof for which $\hat{\mathcal{S}} \subseteq \mathcal{S}$, and 0 otherwise.
- $\{0,1\} \leftarrow \mathsf{VerifyN}(\mathsf{cm}, \bar{\mathcal{S}}, \bar{\pi}, \mathsf{pp})$: Given the commitment cm, a set $\bar{\mathcal{S}}$ and a proof $\bar{\pi}$, it outputs 1 if $\bar{\pi}$ is a valid proof for which $\bar{\mathcal{S}} \not\subseteq \mathcal{S}$, and 0 otherwise.

A zkDUA scheme satisfies the following properties.

- **Completeness.** For any $\lambda, q, r, \mathcal{S}, \mathcal{S}', \hat{\mathcal{S}}, \bar{\mathcal{S}}$ such that $\hat{\mathcal{S}} \subseteq \mathcal{S}', \bar{\mathcal{S}} \not\subseteq \mathcal{S}'$, $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, N)$, $\mathsf{cm} \leftarrow \mathsf{Com}(\mathcal{S}, r, \mathsf{pp})$, $\mathsf{cm}' \leftarrow \mathsf{UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$, $\hat{\pi} \leftarrow \mathsf{ProveM}(\mathsf{cm}, \mathcal{S}, \hat{\mathcal{S}}, r, \mathsf{pp})$, $\bar{\pi} \leftarrow \mathsf{ProveN}(\mathsf{cm}, \mathcal{S}, \bar{\mathcal{S}}, r, \mathsf{pp})$, $\hat{\pi}' \leftarrow \mathsf{UpdPrM}(\hat{\pi}, \hat{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$, $\bar{\pi}' \leftarrow \mathsf{UpdPrN}(\bar{\pi}, \bar{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$, it holds that
$$\begin{aligned} &\Pr\left[\mathsf{VerifyM}(\mathsf{cm}', \hat{\mathcal{S}}, \hat{\pi}', \mathsf{pp}) = 1\right] = 1 \text{ and} \\ &\Pr\left[\mathsf{VerifyN}(\mathsf{cm}', \bar{\mathcal{S}}, \bar{\pi}', \mathsf{pp}) = 1\right] = 1 \end{aligned} \tag{1}$$
- **ϵ-Soundness.** For any PPT adversary $\mathcal{A}$, $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, N)$, $(\mathsf{cm}, \mathsf{cm}', \mathcal{S}, \mathcal{S}', \hat{\mathcal{S}}, \hat{\pi}, r) \leftarrow \mathcal{A}(\mathsf{pp})$, $(\mathsf{cm}, \mathsf{cm}', \mathcal{S}, \mathcal{S}', \bar{\mathcal{S}}, \bar{\pi}, r) \leftarrow \mathcal{A}(\mathsf{pp})$, it holds that
$$\Pr\left[\begin{array}{l} \mathsf{cm} = \mathsf{Com}(\mathcal{S}, r, \mathsf{pp}) \\ \mathsf{cm}' = \mathsf{UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}', \mathsf{pp}) \\ \hat{\pi}' = \mathsf{UpdPrM}(\hat{\pi}, \hat{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp}) \\ \mathsf{VerifyM}(\mathsf{cm}', \hat{\mathcal{S}}, \hat{\pi}', \mathsf{pp}) = 1 \\ \hat{\mathcal{S}} \not\subseteq \mathcal{S}' \end{array}\right] \leq \epsilon \text{ and}$$
$$\Pr\left[\begin{array}{l} \mathsf{cm} = \mathsf{Com}(\mathcal{S}, r, \mathsf{pp}) \\ \mathsf{cm}' = \mathsf{UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}', \mathsf{pp}) \\ \bar{\pi}' = \mathsf{UpdPrN}(\bar{\pi}, \bar{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp}) \\ \mathsf{VerifyN}(\mathsf{cm}', \bar{\mathcal{S}}, \bar{\pi}', \mathsf{pp}) = 1 \\ \bar{\mathcal{S}} \subseteq \mathcal{S}' \end{array}\right] \leq \epsilon$$
where ϵ is called as soundness error.
- **Zero-knowledge.** Let D be a binary function defined as: $D(query, S, S_i) = 1$ iff $S_i \subseteq S, D(update, S, S_i) = 1$ iff update successfully. For set $\mathcal{S}$, security parameter λ, bound of set size N, a randomness r, PPT adversary $\mathcal{A}$, and simulator $Sim = (Sim_1, Sim_2)$, consider the following two experiments:

$\mathsf{Real}_{\mathcal{A}}(1^\lambda)$:
1: $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, N)$
2: $\mathcal{A}$ chooses a set $\mathcal{S}_0, |\mathcal{S}_0| \leq N$
3: $\mathsf{cm}_0 \leftarrow \mathsf{Com}(\mathcal{S}_0, r, \mathsf{pp})$
4: Send cm_0 to $\mathcal{A}$
5: $\mathsf{cm} \leftarrow \mathsf{cm}_0, \mathcal{S} \leftarrow \mathcal{S}_0$
Query:
6: **for** each $i \in \{1, \ldots, \ell\}$ **do**
7: $\mathcal{A}$ outputs $(\mathsf{op}, \mathcal{S}_i)$ where $\mathsf{op} \in$ {query,update}
8: **if** $\mathsf{op} =$ query **then**
9: **if** $\mathcal{S}_i \subseteq \mathcal{S}$ **then**
10: $\pi_i \leftarrow \mathsf{ProveM}(\mathsf{cm}, \mathcal{S}, \mathcal{S}_i, r, \mathsf{pp})$
11: **else**
12: $\pi_i \leftarrow \mathsf{ProveN}(\mathsf{cm}, \mathcal{S}, \mathcal{S}_i, r, \mathsf{pp})$
13: Return π_i to $\mathcal{A}$
14: **else**
15: $\mathsf{cm}_i \leftarrow \mathsf{UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}_i, \mathsf{pp})$
16: Return cm_i to $\mathcal{A}$
17: $\mathsf{cm} \leftarrow \mathsf{cm}_i, \mathcal{S} \leftarrow \mathcal{S}_i$
Response:
18: $\mathcal{A}$ outputs a bit b

$\mathsf{Ideal}_{\mathcal{A}}(1^\lambda)$:
1: $\mathsf{pp} \leftarrow Sim_1(1^\lambda, N)$
2: $\mathcal{A}$ chooses a set $\mathcal{S}_0, |\mathcal{S}_0| \leq N$
3: $\mathsf{cm}_0 \leftarrow Sim_1(\mathsf{pp})$
4: Sends cm_0 to $\mathcal{A}$
5: $\mathsf{cm} \leftarrow \mathsf{cm}_0$
Query:
6: **for** each $i \in \{1, \ldots, \ell\}$ **do**
7: $\mathcal{A}$ outputs $(\mathsf{op}, \mathcal{S}_i)$ where $\mathsf{op} \in$ {query,update}
8: **if** $\mathsf{op} =$ query **then**
9: $\pi_i \leftarrow Sim_2(\mathsf{cm}, \mathcal{S}_i, D(op, \mathcal{S}, \mathcal{S}_i))$
10: Return π_i to $\mathcal{A}$
11: **else**
12: $\mathsf{cm}_i \leftarrow Sim_2(\mathsf{cm}, \mathcal{S}_i, D(update, \mathcal{S}, \mathcal{S}_i))$
13: Return cm_i to $\mathcal{A}$
14: $\mathsf{cm} \leftarrow \mathsf{cm}_i, \mathcal{S} \leftarrow \mathcal{S}_i$
Response:
15: $\mathcal{A}$ outputs a bit b

For any PPT adversary $\mathcal{A}$, there exists a simulator Sim such that for any set $\mathcal{S}$

$$|\Pr[\mathsf{Real}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

Note that our completeness and zero-knowledge definitions are slightly different from the notion in [18] in order to capture both membership and non-membership proof scenarios. Similarly, our soundness definition is different from the notion in [21] to cover both membership/non-membership proof and the effect of update function. The inclusion of the function UpdCom covers the scenario where there is no update. Indeed, when $\mathcal{S}' = \mathcal{S}$, UpdCom does nothing.

Zero-knowledge Elementary Database. An elementary database D is defined as a partial function which maps a set of keys into values. The database only supports querying with a key x, which returns its associated value $D(x)$ or $\perp$ if $x \notin D$ [14]. A zero-knowledge elementary database ZKEDB [28], [25] is a tuple of four algorithms $\mathsf{ZKEDB} = (\mathsf{Init}, \mathsf{CommitDB}, \mathsf{ProveQ}, \mathsf{VerifyQ})$ as follows.

- $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, N)$: Given a security parameter λ and the bound of database size N, it outputs public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{CommitDB}(D, r, \mathsf{pp})$: Given a database D and a randomness r, it outputs a commitment cm to D.
- $\pi \leftarrow \mathsf{ProveQ}(\mathsf{cm}, D, x, v, r, \mathsf{pp})$: Given a key x, a value v and the randomness r, it creates a proof to show that $v = D(x)$.
- $\{0,1\} \leftarrow \mathsf{VerifyQ}(\mathsf{cm}, x, v, \pi, \mathsf{pp})$: Given a database commitment cm of D, a key x, the query result v, and the proof π. It outputs 1 if π is the valid proof for $v = D(x)$; otherwise it outputs 0.

A ZKEDB scheme satisfies the completeness, soundness and zero-knowledge properties which are defined in [28], [13].

IV. Our Proposed Method

In this section, we construct our proposed ZAC, an efficient zero-knowledge dynamic accumulator scheme and then show how to enable zero-knowledge elementary database from ZAC afterward. We first present the building blocks that are required to construct our schemes.

A. Building Blocks

Bloom Filter. Bloom Filter is a data structure that generates a compressed presentation of a set and answers the membership queries given the compressed presentation [17], [7], [29]. A Bloom Filter BF for a set $\mathcal{S}$ is a tuple of four algorithms $\mathsf{BF} = (\mathsf{Init}, \mathsf{Gen}, \mathsf{Update}, \mathsf{Check})$ as follows.

- $H \leftarrow \mathsf{Init}(q, k)$: Given two integers $q, k \in \mathbb{N}$ where $k < q$, it outputs k hash functions $H := (h_1, \ldots, h_k)$ where $h_i : \mathcal{S} \rightarrow [q]$.
- $\mathbf{m} \leftarrow \mathsf{Gen}(\mathcal{S})$: Given a set $\mathcal{S}$, it initializes a vector $\mathbf{m} \leftarrow 0^q$. It outputs $\mathbf{m}$ where $\forall s \in \mathcal{S}, i \in [k] : \mathbf{m}[h_i(s)] := 1$ as the compressed presentation of $\mathcal{S}$.
- $\mathbf{m}' \leftarrow \mathsf{Update}(\mathbf{m}, \mathcal{S}, \mathcal{S}')$: Given a compressed presentation $\mathbf{m}$ of a set $\mathcal{S}$ and an updated set $\mathcal{S}'$, it identifies the difference between $\mathcal{S}$ and $\mathcal{S}'$ as $\mathcal{D}$ then outputs $\mathbf{m}'$ where $\forall s \in \mathcal{D}, i \in [k] : \mathbf{m}'[h_i(s)] := 1 - \mathbf{m}[h_i(s)]$ as the presentation of $\mathcal{S}'$.
- $\{0,1\} \leftarrow \mathsf{Check}(\mathbf{m}, s)$: Given a compressed presentation $\mathbf{m}$ and an element s, it outputs a bit $b \leftarrow \wedge_{i \in [k]} \mathbf{m}[h_i(s)]$, in which $b = 0$ if $s \notin \mathcal{S}$, otherwise $b = 1$.

BF is used to approximately test the set membership. In exchange for its efficiency, it may give false positive rate [9], [16], [20] that can be determined by

$$\Pr = \left(1 - \left(1 - \frac{1}{q}\right)^{kN}\right)^k, \tag{2}$$

where N is size of the input set. The false positive rate is bounded and it decreases when increasing q or k.

Vector Commitment with Aggregation. Vector commitment allows the prover to commit to a vector $\mathbf{m}$ and later proves that a specific position of $\mathbf{m}$ is equal to a particular value [19]. A zero-knowledge vector commitment with aggregation zkAVC allows to aggregate multiple proofs of the same commitment into a single proof for succinctness, which is a tuple of 6 PPT algorithms as:

- $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, q)$: Given a security parameter λ and the bound q on the vector size, it outputs public parameters pp.
- $\mathsf{cm} \leftarrow \mathsf{Commit}(\mathbf{m}, r, \mathsf{pp})$: Given a vector $\mathbf{m} \in \mathcal{M}^{q-1}$, it computes the commitment cm of $\mathbf{m}$ under randomness r
- $\mathsf{cm}' \leftarrow \mathsf{UpdateCommit}(\mathsf{cm}, \mathcal{S}, \mathbf{m}[\mathcal{S}], \mathbf{m}'[\mathcal{S}], \mathsf{pp})$: Given a commitment cm and a list of positions $\mathcal{S}$ to update $\mathbf{m}$ to $\mathbf{m}'$, it outputs a commitment cm′ for $\mathbf{m}'$.
- $(\pi, y) \leftarrow \mathsf{Prove}(i, \mathbf{m}, r, \mathsf{pp})$: Given an index $i \in [q-1]$ and a vector $\mathbf{m}$, it outputs $y = \mathbf{m}[i]$ and a proof π.
- $\hat{\pi} \leftarrow \mathsf{Aggregate}(\mathsf{cm}, \mathcal{S}, \mathbf{m}[\mathcal{S}], \{\pi_i\}_{i \in \mathcal{S}}, \mathsf{pp})$: Given a commitment cm, a set of indices $\mathcal{S} \subseteq [q-1]$ with the corresponding values $\mathbf{m}[\mathcal{S}] = \{m_j : j \in \mathcal{S}\}$ and proofs $\{\pi_i : i \in \mathcal{S}\}$, it outputs an aggregated proof $\hat{\pi}$.
- $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{cm}, \mathcal{S}, \mathbf{m}[\mathcal{S}], \hat{\pi}, \mathsf{pp})$: Given a commitment cm, a set of indices $\mathcal{S} \subseteq [q-1]$ with the corresponding values $\mathbf{m}[\mathcal{S}] = \{m_j : j \in \mathcal{S}\}$, and an aggregated proof $\hat{\pi}$, it outputs 1 if the values of vector in the commitment cm at positions identified by $\mathcal{S}$ match $\mathbf{m}[\mathcal{S}]$, and 0 otherwise.

An zkAVC scheme satisfies the standard properties of vector commitment including correctness, binding, hiding and zero-knowledge properties [19]. In our paper, we harness Pointproofs [19] (denoted as PR) as an efficient zkAVC instantiation. For curious readers, we present its detailed construction in Appendix.

B. Our zero-knowledge Dynamic Universal Accumulator

Main idea. Our observation is that BF permits checking set membership of an arbitrary set by representing the set with the fixed length vector. ZAC requires checking whether a set is a subset of another set and it can be considered as checking multiple set membership. On the other hand, zkAVC, particularly PR [4], permits to check multiple elements in a vector via a single proof without leaking other elements. At the high level idea, to enable ZAC, we integrate BF with PR, in which an arbitrary set $\mathcal{S}$ is represented as a fixed-length vector $\mathbf{v}$, and checking a subset $\hat{\mathcal{S}}$ can be done in a succinct manner by proving the opening of some elements

in $\mathbf{v}$ and aggregating all the proofs together. The important properties of BF are one way due to hash functions, hiding the actual size of the set, applicability to any set and supporting membership/non-membership checking, which permits to achieve zero-knowledge, generic set and universal.

It is worth noting that although our construction relies on the protocol PR [4], there is a core difference between our contribution and theirs. The protocol PR is a vector commitment while ours is a set accumulator. The novelty of our work lies in the integration of Bloom Filter and PR, the extension to support non-membership proof and the application to construct zero-knowledge (non)elementary database.

Let $\mathcal{S}$ be the committed set and $\mathbf{v}$ be the BF vector representation of $\mathcal{S}$. To prove $\mathcal{S}' \subseteq \mathcal{S}$, the idea is to show that *all* the elements in $\mathbf{v}$ at positions determined by the BF hashes on all elements in $\mathcal{S}'$ are equal to 1, i.e., $\mathbf{v}[x_{i,j}] = 1$ where $x_{i,j} = h_i(s_j)$ for all $s_j \in \mathcal{S}'$ and $i \in [1,k]$. To prove $\mathcal{S}' \not\subseteq \mathcal{S}$, the idea is to show there exists at least *an* element in $\mathbf{v}$ in one of the positions determined by the BF hashes on all elements in $\mathcal{S}'$ is equal to 0, i.e., $\exists(i, s_j)$ such that $i \in [1,k]$, $s_j \in \mathcal{S}'$ and $\mathbf{v}[x_{i,j}] = 0$ where $x_{i,j} = h_i(s_j)$.

Detailed Algorithm. We present the algorithm of our ZAC scheme in details as below. Note that in our construction, N is the bound of the set size, k, q, ϵ are chosen parameters and the bound of false positive rate for Bloom Filter, h' is a hash function. The way to generate the parameters in ZAC.Init in a trustworthy manner depends on the application.

$\mathsf{pp} \leftarrow \mathsf{ZAC.Init}(1^\lambda, N)$:
1: $q, k \leftarrow$ Compute from Equation 2 given N, ϵ
2: $\mathsf{pp}' \leftarrow \mathsf{PR.Init}(1^\lambda, q)$
3: $H \leftarrow \mathsf{BF.Init}(q-1, k)$
4: $\mathsf{pp} \leftarrow (\mathsf{pp}', H, q, k)$ =0

$\mathsf{cm} \leftarrow \mathsf{ZAC.Com}(\mathcal{S}, r, \mathsf{pp})$:
1: $\mathbf{v} := (v_1, \ldots, v_{q-1}) \leftarrow \mathsf{BF.Gen}(\mathcal{S})$
2: $\mathsf{cm} \leftarrow \mathsf{PR.Commit}(\mathbf{v}, r)$ ▷ $\mathsf{cm} = g_1^{\sum_{i\in[q-1]} v_i \alpha^i + r\alpha^q}$

$\mathsf{cm}' \leftarrow \mathsf{ZAC.UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$:
1: $\mathbf{v} \leftarrow \mathsf{BF.Gen}(\mathcal{S})$, $\mathbf{u} \leftarrow \mathsf{BF.Gen}(\mathcal{S}')$
2: $\mathcal{I} \leftarrow \{i : i \in [q-1], v_i \neq u_i\}$
3: $\mathsf{cm}' \leftarrow \mathsf{PR.UpdateCommit}(\mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}], \mathbf{u}[\mathcal{I}])$ ▷ $\mathsf{cm}' = \mathsf{cm} \cdot g_1^{\sum_{i\in\mathcal{I}}(u_i - v_i)\alpha^i}$

$\hat{\pi} \leftarrow \mathsf{ZAC.ProveM}(\mathsf{cm}, \mathcal{S}, \hat{\mathcal{S}}, r, \mathsf{pp})$:
1: $\mathcal{I} \leftarrow \cup_{\hat{s}\in\hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$
2: $\mathbf{v} := (v_1, \ldots, v_{q-1}) \leftarrow \mathsf{BF.Gen}(\mathcal{S})$
3: **for** each $i \in \mathcal{I}$ **do**
4: $\pi_i \leftarrow \mathsf{PR.Prove}(i, \mathbf{v}, r)$ ▷ $\pi_i = g_1^{\sum_{\ell\in[q-1]-\{i\}} v_\ell \alpha^{q+1-i+\ell} + r\alpha^{q+1-i+q}}$
5: $\pi \leftarrow \mathsf{PR.Aggregate}(\mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}], \{\pi_i : i \in \mathcal{I}\})$ ▷ $\mathsf{PR.Aggregate}(\cdot) = \Pi_{i\in\mathcal{I}}\pi_i^{t_i}$ where $t_i = h'(i, \mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}])$
6: $\hat{\pi} \leftarrow (\pi, 0)$

$\hat{\pi}' \leftarrow \mathsf{ZAC.UpdPrM}(\hat{\pi}, \hat{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$: ▷ $\hat{S} \subseteq \mathcal{S}'$
1: $\mathcal{I} \leftarrow \cup_{\hat{s}\in\hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$
2: $\mathbf{v} \leftarrow \mathsf{BF.Gen}(\mathcal{S})$, $\mathbf{u} \leftarrow \mathsf{BF.Gen}(\mathcal{S}')$
3: $\mathcal{J} \leftarrow \{j : j \in [q-1], v_j \neq u_j\}$
4: $\Delta \leftarrow g_1^{\sum_{i\in\mathcal{I}}[(\sum_{j\in\mathcal{J}\setminus\mathcal{I}}(u_j - v_j)\alpha^{q+1-i+j})t_i]}$ where $t_i = h'(i, \mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{S}])$
5: $\pi \leftarrow$ Extract from $\hat{\pi}$
6: $\hat{\pi}' \leftarrow (\pi \cdot \Delta, 0)$

$b \leftarrow \mathsf{ZAC.VerifyM}(\mathsf{cm}, \hat{\mathcal{S}}, \hat{\pi}, \mathsf{pp})$:
1: $\mathcal{I} \leftarrow \cup_{\hat{s}\in\hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$
2: $\mathbf{v} := (v_1, \ldots, v_q) \leftarrow 0^q$. and set $v_i \leftarrow 1$, $\forall i \in \mathcal{I}$
3: $\pi \leftarrow$ Extract from $\hat{\pi}$
4: $b \leftarrow \mathsf{PR.Verify}(\mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}], \pi)$ ▷ $e\left(\mathsf{cm}, g_2^{\sum_{i\in\mathcal{I}}(\alpha^{q+1-i}t_i)}\right) \stackrel{?}{=} e(\hat{\pi}, g_2) \cdot g_T^{\alpha^{q+1}\sum_{i\in\mathcal{I}}\mathbf{v}_i t_i}$

$\hat{\pi} \leftarrow \mathsf{ZAC.ProveN}(\mathsf{cm}, \mathcal{S}, \hat{\mathcal{S}}, r, \mathsf{pp})$:
1: $\pi \leftarrow$ Extract from $\mathsf{ZAC.ProveM}(\mathsf{cm}, \mathcal{S}, \hat{\mathcal{S}}, r)$
2: $\{x\} \leftarrow \mathcal{I}$ where $\mathbf{v}[x] = 0$ ▷ $\mathcal{I}, \mathbf{v} := \mathsf{BF.Gen}(\mathcal{S})$ calculated in ZAC.ProveM
3: $\hat{\pi} \leftarrow (\pi, \{x\})$

$\hat{\pi}' \leftarrow \mathsf{ZAC.UpdPrN}(\hat{\pi}, \hat{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$: ▷ $\hat{S} \not\subseteq \mathcal{S}$,
1: $\mathcal{I} \leftarrow \cup_{\hat{s}\in\hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$
2: $\mathbf{v} \leftarrow \mathsf{BF.Gen}(\mathcal{S})$, $\mathbf{u} \leftarrow \mathsf{BF.Gen}(\mathcal{S}')$
3: $\mathcal{J} \leftarrow \{j : j \in [q-1], v_j \neq u_j\}$
4: $\Delta \leftarrow g_1^{\sum_{i\in\mathcal{I}}[(\sum_{j\in\mathcal{J}\setminus\mathcal{I}}(u_j - v_j)\alpha^{q+1-i+j})t_i]}$ where $t_i = h'(i, \mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{S}])$
5: $(\pi, \{x\}) \leftarrow \hat{\pi}$
6: $\hat{\pi}' \leftarrow (\pi \cdot \Delta, \{x\})$

$b \leftarrow \mathsf{ZAC.VerifyN}(\mathsf{cm}, \hat{\mathcal{S}}, \hat{\pi}, \mathsf{pp})$:
1: $(\pi, \{x\}) \leftarrow \hat{\pi}$
2: $\mathcal{I} \leftarrow \cup_{\hat{s}\in\hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$
3: $\mathbf{v} := (v_1, \ldots, v_q) \leftarrow 0^q$. and set $v_i \leftarrow 1$, $\forall i \in \mathcal{I} \setminus \{x\}$
4: $b \leftarrow \{x\} \neq \emptyset \wedge \mathsf{PR.Verify}(\mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}], \pi)$

Theorem 1. *ZAC constitutes a zero-knowledge dynamic universal accumulator with completeness, soundness for membership and ϵ-soundness for non-membership proof under the correctness of Bloom Filter and vector commitment in [19].*

Proof. (Proof of completeness) Assuming that there exist $\lambda, q, r, \mathcal{S}, \mathcal{S}', \hat{\mathcal{S}}, \bar{\mathcal{S}}$ such that $\hat{\mathcal{S}} \subseteq \mathcal{S}', \bar{\mathcal{S}} \not\subseteq \mathcal{S}'$, $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda, N)$, $\mathsf{cm} \leftarrow \mathsf{Com}(\mathcal{S}, r, \mathsf{pp})$, $\hat{\pi} \leftarrow \mathsf{ProveM}(\mathsf{cm}, \mathcal{S}, \hat{\mathcal{S}}, r, \mathsf{pp})$, $\bar{\pi} \leftarrow \mathsf{ProveN}(\mathsf{cm}, \mathcal{S}, \bar{\mathcal{S}}, r, \mathsf{pp})$, $\mathsf{cm}' \leftarrow \mathsf{UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$, $\hat{\pi}' \leftarrow \mathsf{UpdPrM}(\hat{\pi}, \hat{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$, $\bar{\pi}' \leftarrow \mathsf{UpdPrN}(\bar{\pi}, \bar{\mathcal{S}}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$. Given $\mathcal{I} \leftarrow \cup_{\hat{s}\in\hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$ (see line 1 in VerifyM), $\mathbf{v}$ as in line 2 in VerifyM, we have $\Pr[\mathsf{PR.Verify}(\mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}], \pi) = 1]$(see line 4 in VerifyM) $= 1$ thanks to the correctness of PR in [19] and BF. Because $\hat{\mathcal{S}} \subseteq \mathcal{S}'$ and $\mathsf{cm}' \leftarrow \mathsf{UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}', \mathsf{pp})$, we deduce $\Pr[\mathsf{VerifyM}(\mathsf{cm}', \hat{\mathcal{S}}, \hat{\pi}', \mathsf{pp}) = 1] = \Pr[\mathsf{PR.Verify}(\mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}], \pi) = 1]$. Hence, $\Pr[\mathsf{VerifyM}(\mathsf{cm}', \hat{\mathcal{S}}, \hat{\pi}', \mathsf{pp}) = 1] = 1$. Similarly $\Pr[\mathsf{VerifyN}(\mathsf{cm}', \bar{\mathcal{S}}, \bar{\pi}', \mathsf{pp}) = 1] = \Pr[\mathsf{PR.Verify}(\mathsf{cm}, \mathcal{I}, \mathbf{v}[\mathcal{I}], \pi) = 1]$(line 4 in VerifyN) $= 1$.

(Proof of soundness) Assuming there exists a PPT adversary $\mathcal{A}(1^\lambda)$ that breaks the soundness of ZAC with non-negligible probability. We construct a PPT adversary $\mathcal{A}'(1^\lambda)$ that emu-

lates the challenger as below.

Setup:
1: Runs $\mathsf{pp} \leftarrow \mathsf{ZAC.Init}(1^\lambda, N)$
2: $\mathcal{A}$ chooses a set $\mathcal{S}_0$, $|\mathcal{S}_0| \leq N$
3: $\mathsf{cm}_0 \leftarrow \mathsf{ZAC.Com}(\mathcal{S}_0, r, \mathsf{pp})$
4: Sends cm_0 to $\mathcal{A}$
Update:
5: Runs $\mathcal{L} \leftarrow \{(\mathsf{cm}_0, \mathcal{S}_0)\}$
6: $(\mathsf{cm}, \mathcal{S}) \leftarrow (\mathsf{cm}_0, \mathcal{S}_0)$
7: **for** each $i \in \{1, \ldots, \ell\}$ **do**
8: $\mathcal{A}$ outputs $\mathcal{S}_i$
9: Runs $\mathsf{cm}_i \leftarrow \mathsf{ZAC.UpdCom}(\mathsf{cm}, \mathcal{S}, \mathcal{S}_i, \mathsf{pp})$
10: Sends cm_i to $\mathcal{A}$
11: $(\mathsf{cm}, \mathcal{S}) \leftarrow (\mathsf{cm}_i, \mathcal{S}_i)$
12: Runs $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{cm}_i, \mathcal{S}_i)\}$
Challenge:
13: $\mathcal{A}$ outputs an index j and a pair $(\hat{S}, \hat{\pi})$
Response:
24: $\mathcal{A}$ outputs a bit b

Assuming $\hat{S} \subseteq \mathcal{S}_j \wedge \mathsf{VerifyN}(\mathsf{cm}_j, \hat{\mathcal{S}}, \hat{\pi}, \mathsf{pp}) = 1$: Given $\mathcal{I} \leftarrow \cup_{\hat{s} \in \hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$ (see line 2 in VerifyN),{x} $\leftarrow$ Extract from $\hat{\pi}$ (see line 1), we have $v_x = 0$ (see line 3). Because $\hat{S} \subseteq \mathcal{S}_j$ and thanks to the correctness of BF, $\{x\} = \emptyset$. According to line 4 in VerifyN, $\mathsf{VerifyN}(\mathsf{cm}_j, \hat{S}, \hat{\pi}, \mathsf{pp})$ returns 0 which contradicts to the assumption. Therefore, ZAC is sound for membership proof.

Assuming $\hat{S} \not\subseteq \mathcal{S}_j \wedge \mathsf{VerifyM}(\mathsf{cm}_j, \hat{\mathcal{S}}, \hat{\pi}, \mathsf{pp}) = 1$: Given $\mathbf{v} \leftarrow \mathsf{BF.Gen}(S_j)$, $\mathcal{I} \leftarrow \cup_{\hat{s} \in \hat{\mathcal{S}}}\{h_1(\hat{s}), \ldots, h_k(\hat{s})\}$ (line 1 in VerifyM) and according to line 2, 4 in VerifyM, PR.Verify($\mathsf{cm}_j, \mathcal{I}, \mathbf{v}[\mathcal{I}], \pi) = 1$ implies $\mathbf{v}[\mathcal{I}]$ are all 1. This event (i.e. $\mathbf{v}[\mathcal{I}]$ are all 1 while $\hat{S} \not\subseteq \mathcal{S}_j$) corresponds to the false positive event of BF which have probability ϵ. Therefore, ZAC is ϵ-sound for non-membership proof.

(Proof of zero-knowledge) We define the simulator $\mathsf{Sim} = (Sim_1, Sim_2)$ as follows.

$\underline{\mathsf{Ideal}_{\mathcal{A}}(1^\lambda)}$
Setup:
1: $Sim_1(1^\lambda, N)$ runs ZAC.Init to generate public parameters pp
2: $\mathcal{A}$ chooses a set $\mathcal{S}_0, |\mathcal{S}_0| \leq N$
3: Sim_1 runs $r_0 \xleftarrow{\$} \mathbb{Z}_p, \mathsf{cm}_0 \leftarrow g_1^{r_0}$
4: Sends cm_0 to $\mathcal{A}$
5: $r \leftarrow r_0, \mathsf{cm} \leftarrow \mathsf{cm}_0, \mathcal{S} \leftarrow \mathcal{S}_0$
6: Stores r_0 and initiates $\mathcal{C} \leftarrow \emptyset$
Query:
7: **for** each $j \in \{1, \ldots, \ell\}$ **do**
8: $\mathcal{A}$ outputs $(\mathrm{op}, \mathcal{S}_j)$ ▷ op ∈ {query,update}
9: **if** op = query **then**
10: **if** $\mathcal{S}_j \notin \mathcal{C}$ **then**
11: Sim_2 runs $\pi_j \leftarrow \Pi_{i \in \mathcal{I}} g_1^{(r\alpha^{q+1-i} - v_i\alpha^{q+1})t_i}$
12: **if** $D(query, \mathcal{S}, \mathcal{S}_j) = 1$ **then**
13: Returns $(\pi_j, 0)$ to $\mathcal{A}$
14: Appends $(\pi_j, 0)$ to $\mathcal{C}$
15: **else**
16: Returns $(\pi_j, \{x\})$ to $\mathcal{A}$ ▷ $x \xleftarrow{\$} \mathbb{Z}_q$
17: Appends $(\pi_j, \{x\})$ to $\mathcal{C}$
18: **else**
19: Responds with the corresponding entries in $\mathcal{C}$
20: **else**
21: Sim_2 runs $r_j \xleftarrow{\$} \mathbb{Z}_p, \mathsf{cm}_j \leftarrow g_1^{r_j}$
22: Return cm_j to $\mathcal{A}$
23: $r \leftarrow r_j, \mathsf{cm} \leftarrow \mathsf{cm}_j, \mathcal{S} \leftarrow \mathcal{S}_j$

Randomness is included in each case of initial set up (line 3), query (line 11) and update (line 21) in $\mathsf{Ideal}_{\mathcal{A}}$ above. In $\mathsf{Real}_{\mathcal{A}}$ (Definition III), randomness is used for commitment (line 2), proofs (line 10,12) and implied for update. It follows that the commitment values and proofs generated in $\mathsf{Real}_{\mathcal{A}}$ are indistinguishable from the randomized values in $\mathsf{Ideal}_{\mathcal{A}}$. Hence, ZAC is zero-knowledge.

Complexity. Let N be the carnality of the input set $\mathcal{S}$ and λ be the security parameter. Given k be the number of used hash functions in Bloom Filter, which is a small constant, and $q < N$ be the length of the compressed string of the set created by Bloom Filter. Let $\mathsf{H}, \mathsf{mult}, \mathbb{F}$ and e be the hash function, group multiplication, field multiplication and pairing operations, respectively.

In ZAC, the proof size and the commitment size are $\mathcal{O}(1)$. Specifically, their cost is a group element regardless of the size of the set. Since BF vector is a binary vector, the commit phase only incurs $\mathcal{O}(Nk)$ H + $\mathcal{O}(q)$ mult invocations. Our scheme uses public parameters of size $\mathcal{O}(q)$. To prove a relationship of a set $\hat{\mathcal{S}}$ over $\mathcal{S}$, the (non)membership proof incurs $\mathcal{O}(|\hat{\mathcal{S}}|)$ H + $\mathcal{O}(q)$ mult invocations. The (non)membership verification incurs $\mathcal{O}(|\hat{\mathcal{S}}|)$ H + $\mathcal{O}(q)$ mult + 2e. Let $\mathcal{S}'$ be the updated set, which differs from $\mathcal{S}$ at n' elements. The cost to update the commitment is $\mathcal{O}(n')$ H + $\mathcal{O}(q)$ mult. The cost to update the proof is $\mathcal{O}(q)$ mult.

C. Our Proposed Zero-Knowledge (Non)Elementary Database

Suppose that an elementary database D includes the keys $\mathcal{K}$ and values $\mathcal{V}$, where $\mathcal{K} \subseteq \{0,1\}^*$, and domain of $\mathcal{V}$ is $[N]$. We then present our proposed ZKEDB construction based on ZAC as below. The idea is to group the records by values to form different sets. For each set, ZAC.Com is called to create its commitment, which is also created for the set of keys. For values with non-existing records, commitments are randomized. Upon a returned record, a membership proof is created according to a commitment identified its value. If returning $\perp$, a non-membership proof is created according to the commitment of the set of keys.

$\underline{\mathsf{pp} \leftarrow \mathsf{ZKEDB.Init}(1^\lambda, N)}$:
1: $\mathsf{pp} = (\mathsf{pp}', H, q, k) \leftarrow \mathsf{ZAC.Init}(1^\lambda, N)$

$\underline{\mathsf{cm} \leftarrow \mathsf{ZKEDB.CommitDB}(D, r, \mathsf{pp})}$
1: **for** $i \leftarrow 1$ to N **do**
2: $\mathcal{S}_i \leftarrow \{x \in \mathcal{K} : D(x) = i\}$
3: **if** $\mathcal{S}_i \neq \emptyset$ **then**
4: $C_i \leftarrow \mathsf{ZAC.Com}(\mathcal{S}_i, r, \mathsf{pp})$
5: **else**
6: $C_i \xleftarrow{\$} \mathbb{G}_1$ ▷ $\mathbb{G}_1$ is multiplicative group (see Appendix VI)
7: $C_{N+1} \leftarrow \mathsf{ZAC.Com}(\mathcal{K}, r, \mathsf{pp})$
8: $\mathsf{cm} \leftarrow \{C_1, \ldots, C_N, C_{N+1}\}$

$\hat{\pi} \leftarrow \mathsf{ZKEDB.ProveQ}(\mathsf{cm}, D, x, v, r, \mathsf{pp})$

1: $(C_v, C_{N+1}) \leftarrow$ Extract from cm
2: **if** $v = \perp$ **then**
3: $\quad \hat{\pi} \leftarrow \mathsf{ZAC.ProveN}(C_{N+1}, \mathcal{K}, \{x\}, r, \mathsf{pp})$
4: **else**
5: $\quad \mathcal{S}_v \leftarrow \{x' \in \mathcal{K} : D(x') = v\}$
6: $\quad \hat{\pi} \leftarrow \mathsf{ZAC.ProveM}(C_v, \mathcal{S}_v, \{x\}, r, \mathsf{pp})$

$b \leftarrow \mathsf{ZKEDB.VerifyQ}(\mathsf{cm}, x, v, \hat{\pi}, \mathsf{pp})$

1: $(C_v, C_{N+1}) \leftarrow$ Extract from cm
2: **if** $v = \perp$ **then**
3: $\quad b \leftarrow \mathsf{ZAC.VerifyN}(C_{N+1}, \{x\}, \hat{\pi}, \mathsf{pp})$
4: **else**
5: $\quad b \leftarrow \mathsf{ZAC.VerifyM}(C_v, \{x\}, \hat{\pi}, \mathsf{pp})$

Types of query support. Although ZKEDB.ProveQ and ZKEDB.VerifyQ currently prove and verify a single item, they can be extended to prove and verify a set of items easily. Thanks to that, ZDB can also support the query *select top* ℓ *where* $y = v$. Given $v = D(x_j), \forall j \in [\ell]$ and randomness r:

$\hat{\pi} \leftarrow \mathsf{ZKEDB.ProveQM}(\mathsf{cm}, D, \{x_j\}_{j\in[\ell]}, v, r, \mathsf{pp})$:

1: $C_v \leftarrow$ Extract from cm
2: $\mathcal{S}_v \leftarrow \{x' \in \mathcal{K} : D(x') = v\}$
3: $\hat{\pi} \leftarrow \mathsf{ZAC.ProveM}(C_v, \mathcal{S}_v, \{x_j\}_{j\in[\ell]}, r, \mathsf{pp})$

$b \leftarrow \mathsf{ZKEDB.VerifyQM}(\mathsf{cm}, \{x_j\}_{j\in[\ell]}, v, \hat{\pi}, \mathsf{pp})$:

1: $b \leftarrow \mathsf{ZAC.VerifyM}(C_v, \{x_j\}_{j\in[\ell]}, \hat{\pi}, \mathsf{pp})$

Theorem 2. ZKEDB *constitutes a complete,* ϵ*-sound and zero-knowledge elementary database.*

ZKEDB can be extended to be zero-knowledge non-elementary database. Due to the limited number of pages, the security proof and extension are not included.

V. Experimental Evaluation

A. Implementation and Configurations

Implementation. We implemented the main functions of ZAC in Rust using the open-source of Pointproofs [4] and standard Bloom Filter [2], [3]: Com to compute a commitment, ProveM to generate a proof of membership, VerifyM to verify a proof of membership. The complexity of the proof of non-membership ProveN and its verification VerifyN are equal to ProveM and VerifyM respectively. The worst case complexity of UpdCom, UpdPrM and UpdPrN are equal to the complexity of Com, ProveM and ProveN respectively. Therefore, we only need to run the experiments for the mentioned three main functions. The implementation is mainly to provide a proof of concept without considering about the programming techniques and optimisation. Therefore, we do not compare our experimented running time with the related works.

Hardware. We benchmarked the performance of our scheme on a virtual machine with 2 AMD EPYC 7452 32-Core CPUs at 2345.593 MHz, 8GB RAM, 128GB hard disk with SATA AHCI controller, Ubuntu 20.04.1 LTS.

Parameter choices. We selected the parameter for Bloom Filter with the expected false positive rate as $\epsilon = 0.01$, the bounded set size is $N = 200$. The optimal size of BF q and number of hash function k are calculated by [3] based on ϵ, N. We ran each experiment is run 30 times and reported the average results.

Dataset and Measurement. We run the program and record the elapsed time of Com, ProveM, VerifyM with the synthetic data provided by Harvard Pilgrim Health Care [5] for the experiments. More specifically, we use the DRUG_CLASS table in the *Sentinel Summary Tables* [1][5] and generate unique patient IDs for all the records. After that, we filter the 200 records with *Analgesic Narcotic Agonists and Combinations* drug class and run the experiments using these records. More detailed, we select randomly $25, 50, 75, 100, 125, 150, 175, 200$ unique patient IDs of the records to form the set $\mathcal{S}$ and generate the commitment. In each case, we choose randomly $1, 5, 10, 15$ unique patient IDs from the commit set $\mathcal{S}$ to form the proving set $\hat{\mathcal{S}}$, and run the program to generate a proof and verify the proof. The running time is measured in milliseconds. We also measure the storage for public commitments, and overhead communication for the witnesses.

B. Results

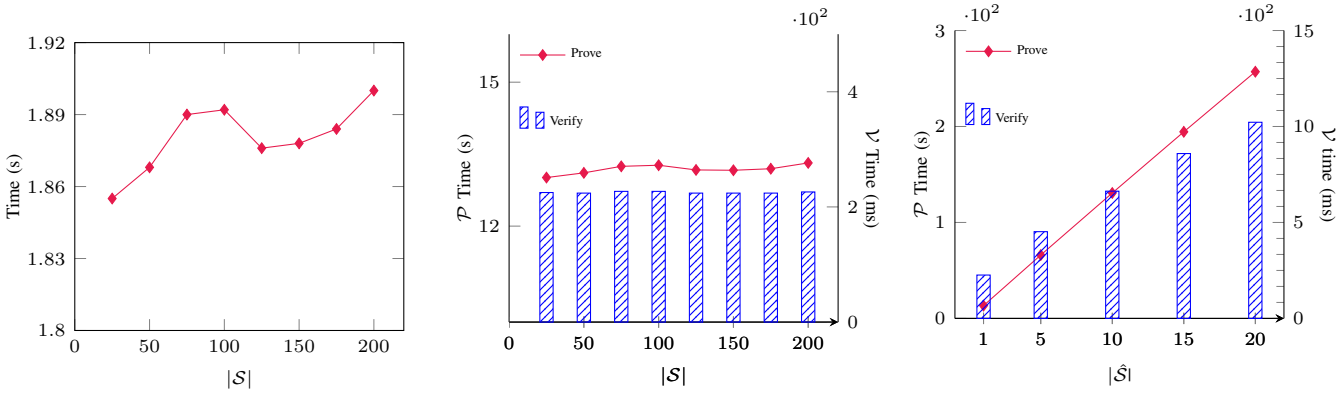
Computation. Figure 1a presents the commit time of our scheme with different committed set sizes. The Com time is composed of BF.Gen($\mathcal{S}$) and PR.Commit time where BF.Gen is much less than PR.Commit. PR.Commit cost depends on the number of non-zero values of BF (see ZAC.Com, line 2). Therefore, the running time are nearly equal among the various size of the committed sets, with very small difference (less than 50 ms).

The ProveM and VerifyM cost are mainly the cost of PR. Prove and PR.Verify respectively which are linear to the size of the proving membership sets (see line 1,4 in ZAC.ProveM, line 1,4 in ZAC.VerifyM). Therefore, the running time is linear to the number of proved items as shown in Figure1b, 1c.

Storage and Communication. In our scheme, the commitment size and the proof size are constant and independent to the sizes of the committed set and the proving set. Concretely, our scheme incurs 49 bytes for the commitment size and proof size.

VI. Conclusion

In this paper, we present an efficient zero-knowledge Dynamic Universal Accumulator scheme (ZAC) enabling to prove subset membership and subset non-membership in zero-knowledge manner. Our results show that the communication complexity is comparable, which is 27 times less than [8]. One of the advantage of ZAC is the proof of non-membership ProveN complexity does not incur any considerable complexity in compared with the proof of membership ProveM. The ZAC can aggregate cross-commitment witnesses into one witness efficiently, unlike /cite[boneh2019batching]. Afterward, we deploy ZAC in the real setting as the scenario of zero-knowledge elementary database, which achieve the better communication and computation cost (i.e. ProveQ, VerifyQ) than the Merkle tree-based approach like [13]. Finally, the proposed



(a) ZAC commit phase

(b) $|\hat{\mathcal{S}}| = 1$

(c) Varied $|\hat{\mathcal{S}}|$

Fig. 1: Performance of our ZAC scheme.

protocol ZAC and ZKEDB are proved to be complete, ϵ-sound and zero-knowledge.

Acknowledgment

Hai-Van Dang is funded by School of Engineering, Computing and Mathematics, University of Plymouth. Thuc D. Nguyen is funded by University of Science, VNU-HCM under Grant No. CNTT2021-17. Thang Hoang is supported by an unrestricted gift from Robert Bosch, and the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

References

[1] Data description. https://www.popmednet.org/resources-and-support/sample-database (last accessed: Jan 2022)
[2] Plum bloom filter. https://docs.rs/plum/0.1.4/plum/struct.StandardBloomFilter.html (last accessed: Jan 2022)
[3] Plum bloom filter code. https://github.com/distrentic/plum.git (last accessed: Jan 2022)
[4] Pointproofs code. https://github.com/algorand/pointproofs (last accessed: Jan 2022)
[5] Synthetic medical data. https://popmednet.atlassian.net/wiki/spaces/DOC/pages/99876870/Test+Databases?preview=%2F99876870%2F99876869%2FTest+Summary+Table+Database.zip (last accessed: Jan 2022)
[6] Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 274–285. Springer (1993)
[7] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)
[8] Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. In: Annual International Cryptology Conference. pp. 561–586. Springer (2019)
[9] Bose, P., Guo, H., Kranakis, E., Maheshwari, A., Morin, P., Morrison, J., Smid, M., Tang, Y.: On the false-positive rate of bloom filters. Information Processing Letters **108**(4), 210–213 (2008)
[10] Camenisch, J., Dubovitskaya, M., Rial, A.: Concise uc zero-knowledge proofs for oblivious updatable databases. In: 2021 34th IEEE Computer Security Foundations Symposium (2021)
[11] Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Annual International Cryptology Conference. pp. 61–76. Springer (2002)
[12] Catalano, D., Fiore, D.: Vector commitments and their applications. In: Int. Workshop on Public Key Cryptography. pp. 55–72. Springer (2013)
[13] Catalano, D., Fiore, D., Messina, M.: Zero-knowledge sets with short proofs. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 433–450. Springer (2008)
[14] Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial commitments with applications to zero-knowledge sets. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 422–439. Springer (2005)
[15] Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial commitments with applications to zero-knowledge sets. Journal of cryptology **26**(2), 251–279 (2013)
[16] Christensen, K., Roginsky, A., Jimeno, M.: A new analysis of the false positive rate of a bloom filter. Information Processing Letters **110**(21), 944–949 (2010)
[17] Derler, D., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-rtt key exchange. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 425–455. Springer (2018)
[18] Ghosh, E., Ohrimenko, O., Papadopoulos, D., Tamassia, R., Triandopoulos, N.: Zero-knowledge accumulators and set algebra. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 67–100. Springer (2016)
[19] Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 2007–2023 (2020)
[20] Grandi, F.: On the analysis of bloom filters. Information Processing Letters **129**, 35–39 (2018)
[21] Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. Journal of the ACM (JACM) **59**(3), 1–35 (2012)
[22] Karantaidou, I., Baldimtsi, F.: Efficient constructions of pairing based accumulators. In: 2021 IEEE 34th Computer Security Foundations Symposium (CSF). pp. 1–16. IEEE (2021)
[23] Lai, R.W., Malavolta, G.: Subvector commitments with application to succinct arguments. In: Annual International Cryptology Conference. pp. 530–560. Springer (2019)
[24] Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: International Conference on Applied Cryptography and Network Security. pp. 253–269. Springer (2007)
[25] Libert, B., Nguyen, K., Tan, B.H.M., Wang, H.: Zero-knowledge elementary databases with more expressive queries. In: IACR International Workshop on Public Key Cryptography. pp. 255–285. Springer (2019)
[26] Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Theory of Cryptography Conference. pp. 499–517. Springer (2010)
[27] Liskov, M.: Updatable zero-knowledge databases. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 174–198. Springer (2005)
[28] Micali, S., Rabin, M., Kilian, J.: Zero-knowledge sets. In: 44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings. pp. 80–91. IEEE (2003)
[29] Naor, M., Yogev, E.: Bloom filters in adversarial environments. In: Annual Cryptology Conference. pp. 565–584. Springer (2015)
[30] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)

[31] Tremel, E.: Real-world performance of cryptographic accumulators. Undergraduate Honors Thesis, Brown University (2013)
[32] Vitto, G., Biryukov, A.: Dynamic universal accumulator with batch update over bilinear groups. In: Cryptographers' Track at the RSA Conference. pp. 395–426. Springer (2022)

APPENDIX

Pointproofs Protocol

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be multiplative groups of prime order p with a non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively and the message space is $\mathcal{M} = \mathbb{Z}_p$. The Pointproofs protocol [19] is presented in details as below.

- $\mathsf{pp} \leftarrow \mathsf{PR.Init}(1^\lambda, q)$:

1: $\alpha \xleftarrow{\$} \mathbb{Z}_p$
2: $g_1^{\mathbf{a}} := \left(g_1^{\alpha}, \ldots, g_1^{\alpha^q}\right)$, $g_1^{\alpha^q \mathbf{a}[-1]} := \left(g_1^{\alpha^{q+2}}, \ldots, g_1^{\alpha^{2q}}\right)$, $g_2^{\mathbf{a}} := \left(g_2^{\alpha}, \ldots, g_2^{\alpha^q}\right)$ where $\mathbf{a} = (\alpha, \alpha^2, \ldots, \alpha^q)$
3: $\mathsf{pp} \leftarrow (g_1^{\mathbf{a}}, g_1^{\alpha^q \mathbf{a}[-1]}, g_2^{\mathbf{a}})$ ▷ $\mathcal{P} \leftarrow g_1^{\mathbf{a}}, g_1^{\alpha^q \mathbf{a}[-1]}$ and $\mathcal{V} \leftarrow g_2^{\mathbf{a}}, g_T^{\alpha^{q+1}} := e(g_1^{\alpha}, g_2^{\alpha^q})$

- $C \leftarrow \mathsf{PR.Commit}(\mathbf{m}, r)$:

1: $C \leftarrow g_1^{\mathbf{a}\cdot\mathbf{m}} g_1^{r\alpha^q}$ ▷ $C = g_1^{r\alpha^q + \Sigma_{i\in[q-1]} m_i \alpha^i}$

- $C' \leftarrow \mathsf{PR.UpdateCommit}(C, \mathcal{S}, \mathbf{m}[\mathcal{S}], \mathbf{m}'[\mathcal{S}])$:

1: $C' \leftarrow C \cdot g_1^{(\mathbf{m}'[\mathcal{S}] - \mathbf{m}[\mathcal{S}])\cdot \mathbf{a}[\mathcal{S}]}$ ▷ $C' = C \cdot g_1^{\Sigma_{i\in\mathcal{S}}(m'_i - m_i)\alpha^i}$

- $\pi_i \leftarrow \mathsf{PR.Prove}(i, \mathbf{m}, r)$:

1: $\mathbf{m}' \leftarrow \mathbf{m}||r$
2: $\pi_i \leftarrow g_1^{\alpha^{q+1-i}\mathbf{m}'[-i]\cdot\mathbf{a}[-i]}$ ▷ $\pi_i = g_1^{\Sigma_{j\in[q]-\{i\}} m'_j \alpha^{q+1-i+j}}$

- $\hat{\pi} \leftarrow \mathsf{PR.Aggregate}(C, \mathcal{S}, \mathbf{m}[\mathcal{S}], \{\pi_i : i \in \mathcal{S}\})$:

1: **for** each $i \in \mathcal{S}$ **do**
2: $t_i \leftarrow h'(i, C, \mathcal{S}, \mathbf{m}[\mathcal{S}])$
3: $\hat{\pi} \leftarrow \Pi_{i\in\mathcal{S}} \pi_i^{t_i}$

- $b \leftarrow \mathsf{PR.Verify}(C, \mathcal{S}, \mathbf{m}[\mathcal{S}], \hat{\pi})$:

1: **for** $i \in \mathcal{S}$ **do**
2: $t_i \leftarrow h'(i, C, \mathcal{S}, \mathbf{m}[\mathcal{S}])$
3: $b \leftarrow e\left(C, g_2^{\Sigma_{i\in\mathcal{S}}(\alpha^{q+1-i} t_i)}\right) \stackrel{?}{=} e(\hat{\pi}, g_2) \cdot g_T^{\alpha^{q+1}\Sigma_{i\in\mathcal{S}} m_i t_i}$

- $\pi \leftarrow \mathsf{PR.AggregateAcross}\left(\{C_j, \mathcal{S}_j, \mathbf{m}_j[\mathcal{S}_j], \hat{\pi}_j\}_{j\in[\ell]}\right)$:

1: **for** $j \in [\ell]$ **do**
2: $t'_j \leftarrow h''(i, \{C_j, \mathcal{S}_j, \mathbf{m_j}[\mathcal{S}_j]\}_{j\in[\ell]})$
3: $\pi \leftarrow \Pi_{j=1}^{\ell} \hat{\pi}_j^{t'_j}$

- $b \leftarrow \mathsf{PR.VerifyAcross}\left(\{C_j, \mathcal{S}_j, \mathbf{m}_j[\mathcal{S}_j]\}_{j\in[\ell]}, \pi\right)$:

1: **for** $j \in [\ell]$ **do**
2: $t'_j \leftarrow h''(i, \{C_j, \mathcal{S}_j, \mathbf{m_j}[\mathcal{S}_j]\}_{j\in[\ell]})$
3: **for** $i \in \mathcal{S}_j$ **do**
4: $t_{j,i} \leftarrow h'(i, C_j, \mathcal{S}_j, \mathbf{m_j}[\mathcal{S}_j])$
5: $b \leftarrow \Pi_{j=1}^{\ell} e\left(C, g_2^{\Sigma_{i\in\mathcal{S}_j}(\alpha^{q+1-i} t_{j,i})}\right)^{t'_j} \stackrel{?}{=} e(\pi, g_2) \cdot g_T^{\alpha^{q+1}\Sigma_{j\in[\ell], i\in\mathcal{S}_j} m_{j,i} t_{j,i} t'_j}$