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 verification 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, $O(\log N)$ less bandwidth, and $O(\log N)$ 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 $\ell$ 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 $S$ 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 $S$ 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 $P$ and Verifier $V$. In the committing stage, $P$ commits to vector $m = (m_1, \ldots, m_q)$ in such a way that $V$ cannot learn about $m$ (hiding property). In the proving state, $P$ 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 $m_k$). 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 $m$, 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.

Cryptographic 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 $\bot$. 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          | Proving time       | Verification time | $|\pi|$ (bytes) |
|----------------|---------------------|--------------------|-------------------|--------------|
|                | Dyn. | Gen. | Univ. | zk | $O(\lambda(N|\pi|)H)$ | $O(\lambda)|\pi|H$ | $O(\lambda)|\pi|$ | $O(|\pi|$ (bytes) |
| [6]            | ✓    | ✓    | ✓    | ✓  | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        |
| [12]           | ✓    | ✓    | ✓    | ✓  | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        |
| [8] (w/o hashing to prime) | ✓    | ✓    | ✓    | ✓  | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        |
| [8] (w/ hashing to prime) | ✓    | ✓    | ✓    | ✓  | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        |
| [32]           | ✓    | ✓    | ✓    | ✓  | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        |
| Our ZAC scheme | ✓    | ✓    | ✓    | ✓  | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        | ✓                        |

Dyn. means dynamic, Gen. means generic set, Univ. means universal, zk means zero-knowledge, $|\pi|$: proof size, $\lambda$: 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 $F$ denote the hash function, group multiplication operation and multiplication in field of size approximately $2^i$, respectively; $+\cdot$: the cost is $O(1)$ mult $+ 2e$ in case of membership verification and $O(p)$ 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 ⊥, 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 $N < 2^m$.

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, $\epsilon$-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, Zero-Knowledge Dynamic Universal Accumulator (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 $O(m) H + O(q)$ mult.
- The verifying time is $O(m) H + O(q)$ mult $+ 2e$.

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 $S$ 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] ($O(N') H + O(q)$ mult $< O(N' \log(N'))$ 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 $O(\log(\lambda))$ [28], [15], $O(\lambda \cdot q/ \log(q))$ [13], $O(\lambda/ \log(q))$ [26] where $\lambda$ is the security parameter in such a way that the database size upper bound is $2^\lambda$ 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 $O(1)$, 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 ($O(1)$) than the schemes based on the Merkle tree ($O(\lambda)$ [15], $O(\lambda \cdot q/\log(q))$ [13], $O(\lambda/ \log(q))$ [26] where $\lambda$ is the security parameter in such a way that the database size upper bound is $2^\lambda$, $q$ is the
branching factor of the Merkle tree). In addition, our design supports a new type of query, select top $\ell$ where $y = \alpha$, with only one proof and verification compared to $\ell$ 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 Birukov 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 recalled to any value at Prover $P$’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 $V$ that if the commitment can be proved then it will be proved to the same value. $P$ cannot create a commitment that can be teased to $x$ and proved to $x' \neq x$. To construct a ZKEDB $D = (K, Y)$, Chase et al.[14], [15] built a Merkle tree with nodes as hard or soft commitments. The leaf nodes are numbered by keys ($\{x \in K\}$), and contain commitments to their values ($\{D(x) \in Y\}$). The query verification communication and computation cost are $O(\log(\lambda))$ where $\lambda$ is the security parameter in such a way that the database size upper bound is $2^\lambda$. 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 $O(\lambda \cdot q/\log(q))$. 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, $O(\lambda/\log(q))$. 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 m[i]) as the i-th element in vector m. m[S] := \{m_j : j \in S\} and m[-i] := \{m_j : j \neq i\}. x \$ S means x is uniformly chosen random from a set S, |S| as the cardinality of set S. Let 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 S, and then later proves that another set \hat{S} is the subset of S (i.e. S \subseteq \hat{S}) or \hat{S} is not a subset of S (i.e. S \nsubseteq \hat{S}), without revealing the items as well as the size of S. A zkDUA scheme is a tuple of 9 algorithms zkDUA = \langle Init, UpdCom, ProveM, ProveN, UpdPrM, UpdPrN, VerifyM, VerifyN \rangle defined as follows.

- \text{pp} \leftarrow \text{Init}(1^{\lambda}, N): Given the security parameter \lambda and the bound of set size N as input, it outputs public parameters pp.
- cm \leftarrow \text{Com}(S, r, pp): Given a set S and randomness r, it outputs a commitment cm.
- cm' \leftarrow \text{UpdCom}(cm, S, S', pp): Given two sets S, S' and cm as the commitment of S, it outputs a commitment cm' to S'.
- \hat{\pi} \leftarrow \text{ProveM}(cm, S, \hat{S}, \hat{r}, pp): Given two sets S, \hat{S}, a commitment cm to S and randomness r, it outputs a proof \hat{\pi} indicating that \hat{S} \subseteq S.
- \hat{\pi} \leftarrow \text{ProveN}(cm, S, \hat{S}, \hat{r}, pp): Given two sets S, \hat{S}, a commitment cm to S and randomness r, it outputs a proof \hat{\pi} indicating that \hat{S} \nsubseteq S.
- \hat{\pi}' \leftarrow \text{UpdPrM}(\hat{\pi}, S, S', pp): Given three sets S, \hat{S}, S' and a proof \hat{\pi} of S over S, it outputs a new proof \hat{\pi}' to show that \hat{S} \subseteq S'.
- \hat{\pi}' \leftarrow \text{UpdPrN}(\hat{\pi}, S, S', pp): Given three sets S, \hat{S}, S' and a proof \hat{\pi} of \hat{S} over S, it outputs a new proof \hat{\pi}' to show that \hat{S} \nsubseteq S'.
- \{0, 1\} \leftarrow \text{VerifyM}(cm, \hat{S}, \hat{\pi}, pp): Given the commitment cm, a set \hat{S} and a proof \hat{\pi}, it outputs 1 if \hat{\pi} is a valid proof for which \hat{S} \subseteq S, and 0 otherwise.
- \{0, 1\} \leftarrow \text{VerifyN}(cm, \hat{S}, \hat{\pi}, pp): Given the commitment cm, a set \hat{S} and a proof \hat{\pi}, it outputs 1 if \hat{\pi} is a valid proof for which \hat{S} \nsubseteq S, and 0 otherwise.

A zkDUA scheme satisfies the following properties.

- **Completeness.** For any \lambda, q, r, S, S', \hat{S}, \hat{S} such that \hat{S} \subseteq S', \hat{S} \nsubseteq S', pp \leftarrow \text{Init}(1^{\lambda}, N), cm \leftarrow \text{Com}(S, r, pp), cm' \leftarrow \text{UpdCom}(cm, S, S', pp), \hat{\pi} \leftarrow \text{ProveM}(cm, S, \hat{S}, \hat{r}, pp), \hat{\pi'} \leftarrow \text{ProveN}(cm, S, \hat{S}, r, pp), \hat{\pi}' \leftarrow \text{UpdPrM}(\hat{\pi}, S, \hat{S}, S', pp), \hat{\pi}' \leftarrow \text{UpdPrN}(\hat{\pi}, \hat{S}, S', pp), \text{it holds that}

\[
\Pr[\text{VerifyM}^{cm}(\hat{\pi}', \hat{\pi}, pp) = 1] = 1 \quad \text{and} \quad \Pr[\text{VerifyN}^{cm}(\hat{\pi}', \hat{\pi}, pp) = 1] = 1
\]

(1)

- **\epsilon-Soundness.** For any PPT adversary \mathcal{A}, pp \leftarrow \text{Init}(1^{\lambda}, N), (cm, cm', S, S', \hat{S}, \hat{\pi}, r) \leftarrow \mathcal{A}(pp), (cm, cm, S, S', \hat{S}, \hat{\pi}, r) \leftarrow \mathcal{A}(pp), it holds that

\[
\Pr[\text{cm} = \text{Com}(S, r, pp)] \leq \epsilon
\]

\[
\Pr[\text{cm'} = \text{UpdCom}(cm, S, S', pp)] \leq \epsilon
\]

\[
\Pr[\text{\hat{\pi}} = \text{ProveM}(\hat{\pi}, S, \hat{\pi}, S', pp)] = 1
\]

\[
\Pr[\text{\hat{\pi}'} = \text{UpdPrM}(\hat{\pi}, S, \hat{\pi}, S', pp)] = 1
\]

\[
\Pr[\text{VerifyM}^{cm'}(\hat{\pi}', \hat{\pi}, pp) = 1] \leq \epsilon
\]

\[
\Pr[\text{VerifyN}^{cm'}(\hat{\pi}', \hat{\pi}, pp) = 1] \leq \epsilon
\]

where \epsilon is called as soundness error.

- **Zero-knowledge.** Let D be a binary function defined as: D(query, S, S_i) = 1 if S_i \subseteq S, D(update, S, S_i) = 1 if update successfully. For set S, security parameter \lambda, bound of set size N, a randomness r, PPT adversary \mathcal{A}, and simulator Sim = (Sim_1, Sim_2), consider the following two experiments:

\[
\text{Real}_A^{(1^{\lambda})}:
\]

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pp \leftarrow \text{Init}(1^{\lambda}, N)</td>
</tr>
<tr>
<td>2</td>
<td>\mathcal{A} chooses a set S_0,</td>
</tr>
<tr>
<td>3</td>
<td>cm_0 \leftarrow \text{Com}(S_0, r, pp)</td>
</tr>
<tr>
<td>4</td>
<td>Send cm_0 to \mathcal{A}</td>
</tr>
<tr>
<td>5</td>
<td>cm \leftarrow cm_0, S_0 \leftarrow S_0</td>
</tr>
<tr>
<td>6</td>
<td>for each i \in {1, \ldots, \ell} do</td>
</tr>
<tr>
<td>7</td>
<td>\mathcal{A} outputs (op, S_i) where op \in {query, update}</td>
</tr>
<tr>
<td>8</td>
<td>if op = query then</td>
</tr>
<tr>
<td>9</td>
<td>if S_i \subseteq S then</td>
</tr>
<tr>
<td>10</td>
<td>\text{ProveM}(cm, S_i, S_i, pp)</td>
</tr>
<tr>
<td>11</td>
<td>\text{else}</td>
</tr>
<tr>
<td>12</td>
<td>\text{ProveN}(cm, S_i, S_i, pp)</td>
</tr>
<tr>
<td>13</td>
<td>Return S_i to \mathcal{A}</td>
</tr>
<tr>
<td>14</td>
<td>\text{else}</td>
</tr>
<tr>
<td>15</td>
<td>cm_i \leftarrow \text{UpdCom}(cm, S_i, pp)</td>
</tr>
<tr>
<td>16</td>
<td>Return cm_i to \mathcal{A}</td>
</tr>
<tr>
<td>17</td>
<td>cm \leftarrow cm_i, S_i \leftarrow S_i</td>
</tr>
<tr>
<td>18</td>
<td>\mathcal{A} outputs a bit b</td>
</tr>
</tbody>
</table>

\[
\text{Ideal}_A^{(1^{\lambda})}:
\]

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pp \leftarrow \text{Sim}_1(1^{\lambda}, N)</td>
</tr>
<tr>
<td>2</td>
<td>\mathcal{A} chooses a set S_0,</td>
</tr>
<tr>
<td>3</td>
<td>cm_0 \leftarrow \text{Sim}_1(1^{\lambda}, pp)</td>
</tr>
<tr>
<td>4</td>
<td>Send cm_0 to \mathcal{A}</td>
</tr>
<tr>
<td>5</td>
<td>cm \leftarrow cm_0</td>
</tr>
<tr>
<td>6</td>
<td>for each i \in {1, \ldots, \ell} do</td>
</tr>
<tr>
<td>7</td>
<td>\mathcal{A} outputs (op, S_i) where op \in {query, update}</td>
</tr>
<tr>
<td>8</td>
<td>if op = query then</td>
</tr>
<tr>
<td>9</td>
<td>\text{Sim}_2(cm, S_i, D(op, S_i))</td>
</tr>
<tr>
<td>10</td>
<td>Return S_i to \mathcal{A}</td>
</tr>
<tr>
<td>11</td>
<td>\text{else}</td>
</tr>
<tr>
<td>12</td>
<td>cm_i \leftarrow \text{Sim}_2(cm, S_i, D(update, S_i))</td>
</tr>
<tr>
<td>13</td>
<td>Return cm_i to \mathcal{A}</td>
</tr>
<tr>
<td>14</td>
<td>cm \leftarrow cm_i, S_i \leftarrow S_i</td>
</tr>
<tr>
<td>15</td>
<td>\mathcal{A} outputs a bit b</td>
</tr>
</tbody>
</table>

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

\[
|\Pr[\text{Real}_A^{(1^{\lambda})}] - \Pr[\text{Ideal}_A^{(1^{\lambda})}]| \leq \text{negl}(\lambda)
\]
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 $S' = S$, 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 $\bot$ if $x \notin D$ [14]. A zero-knowledge elementary database ZKEDB [28], [25] is a tuple of four algorithms $ZKEDB = (\text{Init}, \text{CommitDB}, \text{ProveQ}, \text{VerifyQ})$ as follows.

- $pp \leftarrow \text{Init}(1^\lambda, N)$: Given a security parameter $\lambda$ and the bound of database size $N$, it outputs public parameters $pp$.
- $cm \leftarrow \text{CommitDB}(D, r, pp)$: Given a database $D$ and a randomness $r$, it outputs a commitment $cm$ to $D$.
- $\pi \leftarrow \text{ProveQ}(cm, D, x, v, r, pp)$: Given a key $x$, a value $v$ and the randomness $r$, it creates a proof to show that $v = D(x)$.
- $\{0, 1\} \leftarrow \text{VerifyQ}(cm, x, v, \pi, pp)$: Given a database commitment $cm$ of $D$, a key $x$, the query result $v$, and the proof $\pi$. It outputs 1 if $\pi$ 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 $S$ is a tuple of four algorithms $BF = (\text{Init}, \text{Gen}, \text{Update}, \text{Check})$ as follows.

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

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

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 $m$ and later proves that a specific position of $m$ 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:

- $pp \leftarrow \text{Init}(1^\lambda, q)$: Given a security parameter $\lambda$ and the bound $q$ on the vector size, it outputs public parameters $pp$.
- $cm \leftarrow \text{Commit}(m, r, pp)$: Given a vector $m \in \mathbb{M}^{q-1}$, it computes the commitment $cm$ of $m$ under randomness $r$.
- $cm' \leftarrow \text{UpdateCommit}(cm, S, m[S], m'[S], pp)$: Given a commitment $cm$ and a list of positions $S$ to update $m$ to $m'$, it outputs a commitment $cm'$ for $m'$.
- $(\pi, y) \leftarrow \text{Prove}(i, m, r, pp)$: Given an index $i \in [q-1]$ and a vector $m$, it outputs $y = m[i]$ and a proof $\pi$.
- $\hat{\pi} \leftarrow \text{Aggregate}(cm, S, m[S], \{\pi_i\}_{i \in S}, pp)$: Given a commitment $cm$, a set of indices $S \subseteq [q-1]$ with the corresponding values $m[S] = \{m_j : j \in S\}$ and proofs $\{\pi_i : i \in S\}$, it outputs an aggregated proof $\hat{\pi}$.
- $\{0, 1\} \leftarrow \text{Verify}(cm, S, m[S], \hat{\pi}, pp)$: Given a commitment $cm$, a set of indices $S \subseteq [q-1]$ with the corresponding values $m[S] = \{m_j : j \in S\}$, and an aggregated proof $\hat{\pi}$, it outputs 1 if the values of vector in the commitment $cm$ at positions identified by $S$ match $m[S]$, 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 $S$ is represented as a fixed-length vector $v$, and checking a subset $\hat{S}$ can be done in a succinct manner by proving the opening of some elements
function. The way to generate the parameters in the bound of false positive rate for Bloom Filter, the idea is to show there exists at least \( N \) scheme in details as below. Note that in our construction, \( ZAC \) is a trustworthy manner depends on the application.

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

\[ \begin{align*}
\text{pp} &\leftarrow \text{ZAC.Init}(1^\lambda, N) \\
&1. q, k \leftarrow \text{Compute from Equation 2 given } N, \epsilon \\
&2. pp' \leftarrow \text{PR.Init}(1^\lambda, q) \\
&3. H \leftarrow \text{BF.Init}(q - 1, k) \\
&4. pp \leftarrow (pp', H, q, k) = 0 \\
\end{align*} \]

\[ \begin{align*}
\text{cm} &\leftarrow \text{ZAC.Com}(S, r, pp) \\
&1. v \leftarrow \text{BF.Gen}(S) \\
&2. \text{cm} \leftarrow \text{PR.Commit}(v, r) \\
&3. cm' \leftarrow \text{PR.UpdateCommit}(cm, I, v[I], u[I]) \\
&\text{cm}' = cm \cdot g_1^{\sum_{i \in I}(v[I] - 1)q + \epsilon} \\
\end{align*} \]

\[ \begin{align*}
\hat{\pi} &\leftarrow \text{ZAC.Prove}(cm, S, \hat{S}, r, pp) \\
&1. I \leftarrow \cup_{s \in S}(h_1(s), \ldots, h_k(s)) \\
&2. v \leftarrow (v_1, \ldots, v_\delta) \leftarrow 0^\delta, \text{ and set } v_i \leftarrow 1, \forall i \in I \\
&3. \hat{\pi} \leftarrow \text{Extract from } \hat{\pi} \\
&4. \hat{\pi}' \leftarrow \text{UpdPrM}(\hat{\pi}, \hat{S}, S', \hat{S}, pp) \\
&\text{cm} \leftarrow \text{ZAC.Com}(cm, S, \hat{S}, \hat{S}', pp) \\
\end{align*} \]

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

**Proof.** (Proof of completeness) Assuming that there exist \( \lambda, q, r, S, S', \hat{S}, \hat{S} \) such that \( S \subseteq S', S \not\subseteq S' \), \( \text{pp} \leftarrow \text{Init}(1^\lambda, N) \), \( \text{cm} \leftarrow \text{Com}(S, r, pp) \), \( \hat{\pi} \leftarrow \text{Prove}(cm, S, \hat{S}, r, pp) \), \( \hat{\pi}' \leftarrow \text{UpdPrM}(\hat{\pi}, \hat{S}, S', \hat{S}, pp) \), \( \hat{\pi}' \leftarrow \text{UpdPrM}(\hat{\pi}, \hat{S}, S', \hat{S}, pp) \).

Given \( I \leftarrow \cup_{s \in S}(h_1(s), \ldots, h_k(s)) \) (see line 1 in VerifyM), \( v \) as in line 2 in VerifyM, we have \( Pr[\text{PR.Verify}(cm, I, v[I], \pi) = 1] \) (see line 4 in VerifyM) \( = 1 \) thanks to the correctness of PR in [19] and BF. Because \( S \subseteq S' \) and \( cm' \leftarrow \text{UpdCom}(cm, S, S', pp) \), we deduce \( Pr[\text{VerifyM}(cm', \hat{\pi}, \hat{\pi}', pp) = 1] = Pr[\text{PR.Verify}(cm, I, v[I], \pi) = 1] \). Hence, \( Pr[\text{VerifyM}(cm', \hat{\pi}, \hat{\pi}', pp) = 1] = 1 \). Similarly \( Pr[\text{VerifyN}(cm', \hat{\pi}, \hat{\pi}', pp) = 1] = Pr[\text{PR.Verify}(cm, I, v[I], \pi) = 1] \) (line 4 in VerifyN) \( = 1 \).

(Proof of soundness) Assuming there exists a PPT adversary \( A'(1^\lambda) \) that breaks the soundness of ZAC with non-negligible probability. We construct a PPT adversary \( A'(1^\lambda) \) that emu-
lates the challenger as below.

\[\text{Setup:} \]
1. Runs \( pp \leftarrow \text{ZAC.Init}(1^\lambda, N) \)
2. \( A \) chooses a set \( S_0 \), \(|S_0| \leq N \)
3. \( cm_0 \leftarrow \text{ZAC.Com}(S_0, r, pp) \)
4. Sends \( cm_0 \) to \( A \)
5. Runs \( L \leftarrow \{\text{ZAC.} \}
6. \( cm(S) \leftarrow \text{cm}(S_0) \)
7. for each \( i \in \{1, \ldots, \ell\} \)

Assuming \( Z \subseteq S_j \wedge \text{VerifyN(cm}_j, S, \pi, pp) = 1 \): Given \( I \leftarrow \bigcup_{i \in S_j} \{h_1(\delta), \ldots, h_k(\delta)\} \) (see line 2 in VerifyN), \( \pi \leftarrow \text{Extract from } \pi \) (see line 1), we have \( v_x = 0 \) (see line 3). Because \( S \subseteq S_j \) and thanks to the correctness of BF, \( \{x\} = \emptyset \). According to line 4 in VerifyN, VerifyN(cm_j, S, \pi, pp) returns \( 0 \) which contradicts to the assumption. Therefore, ZAC is zero-knowledge.

(Proof of zero-knowledge) We define the simulator \( \text{Sim} = (\text{Sim}_1, \text{Sim}_2) \) as follows.

\[\text{Ideal}_A(1^\lambda)\]
1. \( \text{Sim}_1(1^\lambda, N) \) runs ZAC.Init to generate public parameters \( pp \)
2. \( A \) chooses a set \( S_0, |S_0| \leq N \)
3. \( \text{Sim}_1 \) runs \( r_0 \sim Z_p, cm_0 \leftarrow g_1^{r_0} \)
4. Sends \( cm_0 \) to \( A \)
5. \( r \leftarrow r_0, cm \leftarrow cm_0, S \leftarrow S_0 \)
6. Stores \( r_0 \) and initiates \( C \leftarrow \emptyset \)

Query:
1. for each \( j \in \{1, \ldots, \ell\} \) do
2. \( A \) outputs \( \langle \text{op}, S_j \rangle \) \( \text{or } \text{op} \in \{\text{query, update}\} \)
3. if \( \text{op} = \text{query} \) then
4. if \( S_j \notin C \) then
5. \( \text{Sim}_2 \) runs \( \pi_j \leftarrow \Pi_{i < i} (g_1^{r_0 + 1 - i - i_0 \alpha + 1}) \)
6. if \( D(\text{query}, S, S_j) = 1 \) then
7. Returns \( \langle \pi_j, 0 \rangle \) to \( A \)
8. Appends \( (\pi_j, 0) \) to \( C \)
9. else
10. Returns \( \langle \pi_j, \{x\} \rangle \) to \( A \)
11. \( x \leftarrow Z_q \)
12. Appends \( (\pi_j, \{x\}) \) to \( C \)
13. end
14. else
15. Responses with the corresponding entries in \( C \)
16. end
17. Sim_2 runs \( r_j \sim Z_p, cm_j \leftarrow g_1^{r_j} \)
18. Return \( cm_j \) to \( A \)
19. \( r \leftarrow r_j, cm \leftarrow cm_j, S \leftarrow S_j \)
20. end

Response:
21. \( A \) outputs a bit \( b \)

Randomness is included in each case of initial set up (line 3), query (line 11) and update (line 21) in Ideal_A above. In Real_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 Real_A are indistinguishable from the randomized values in Ideal_A. Hence, ZAC is zero-knowledge.

Complexity. Let \( N \) be the carnality of the input set \( S \) and \( \lambda \) 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 \( H, \text{mult}, F \) and \( e \) be the hash function, group multiplication, field multiplication and pairing operations, respectively.

In ZAC, the proof size and the commitment size are \( O(1) \). 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 \( O(Nk) \) \( H + O(q) \) mult invocations. Our scheme uses public parameters of size \( O(q) \). To prove a relation of a set \( S \) over \( S \), the (non)membership proof incurs \( O(|S|) \) \( H + O(q) \) mult invocations. The (non)membership verification incurs \( O(|S|) \) \( H + O(q) \) mult + 2e. Let \( S' \) be the updated set, which differs from \( S \) at \( n' \) elements. The cost to update the commitment is \( O(n') \) \( H + O(q) \) mult. The cost to update the proof is \( O(q) \) mult.

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

Suppose that an elementary database \( D \) includes the keys \( K \) and values \( V \), where \( K \subseteq \{0, 1\}^* \), and domain of \( V \) is \( \mathbb{N} \). We then present our proposed ZKEDB construction based on ZAC 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.

\[pp \leftarrow \text{ZKEDB.Init}(1^\lambda, N):\]
1. \( pp = (pp', H, q, k) \leftarrow \text{ZAC.Init}(1^\lambda, N)\)

\[cm \leftarrow \text{ZKEDB.CommitDB}(D, r, pp)\]
1. for \( i \leftarrow 1 \) to \( N \) do
2. \( S_i \leftarrow \{x \in K: D(x) = i\} \)
3. if \( S_i \neq \emptyset \) then
4. \( C_i \leftarrow \text{ZAC.Com}(S_i, r, pp)\)
5. else
6. \( C_i \leftarrow G_1 \) \( \leftarrow G_3 \) is multiplicative group (see Appendix VI)
7. \( C_{N+1} \leftarrow \text{ZAC.Com}(K, r, pp)\)
8. \( cm \leftarrow \{C_1, \ldots, C_N, C_{N+1}\}\)
A. Implementation and Configurations

Implementation. We implemented the main functions of ZAG 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 $\epsilon, N$. 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 Analogic 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 $S$ and generate the commitment. In each case, we choose randomly 1, 5, 10, 15 unique patient IDs from the commit set $S$ to form the proving set $\hat{S}$, 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($S$) 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 is 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 Figure 1b, 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[bonche2019batching]. 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
protocol ZAC and ZKEDB are proved to be complete, $\epsilon$-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](http://www.cyberinitiative.org).

**REFERENCES**

APPENDIX

Pointproofs Protocol

Let $G_1, G_2, G_T$ be multiplicative groups of prime order $p$ with a non-degenerate bilinear pairing $e : G_1 \times G_2 \to G_T$. $g_1, g_2$ are generators of $G_1, G_2$ respectively and the message space is $\mathcal{M} = Z_p$. The Pointproofs protocol [19] is presented in details as below.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$\mathbf{p} \leftarrow \text{PR.Init}(1^\lambda, q)$:</td>
</tr>
<tr>
<td>1.1</td>
<td>$\alpha \leftarrow Z_p$</td>
</tr>
<tr>
<td>1.2</td>
<td>$g_1^a := (g_1^{a^2}, \ldots, g_1^a), \quad g_1^{a^2} := (g_1^{a^2}, \ldots, g_1^{a^2})$</td>
</tr>
<tr>
<td>1.3</td>
<td>$g_2^a := (g_2^{a^2}, \ldots, g_2^a)$ where $a = (\alpha, \alpha^2, \ldots, \alpha^q)$</td>
</tr>
<tr>
<td>1.4</td>
<td>$\mathbf{p} \leftarrow (\hat{g}_1, \hat{g}_2) \quad \triangleright \mathcal{P} \leftarrow (g_1^{a^2}, g_1^{a^2})$ and $\mathcal{V}$</td>
</tr>
<tr>
<td>1.5</td>
<td>$\hat{\mathcal{V}}$ $\leftarrow e(g_1^{a^2}, g_2^{a^2})$</td>
</tr>
<tr>
<td>2.</td>
<td>$\mathbf{C} \leftarrow \text{PR.Commit}(m, r)$:</td>
</tr>
<tr>
<td>2.1</td>
<td>$\mathbf{C} \leftarrow g_1^{a</td>
</tr>
<tr>
<td>2.2</td>
<td>$\mathcal{C} \leftarrow g_1^{r\alpha^2 + \Sigma_{i \in [q-1]} m_{i} a^i}$</td>
</tr>
<tr>
<td>3.</td>
<td>$\mathbf{C}' \leftarrow \text{PR.UpgradeCommit}(C, S, m[S], m'[S])$:</td>
</tr>
<tr>
<td>3.1</td>
<td>$\mathcal{C}' \leftarrow C \cdot (m'[(S)-m[S])] \cdot a[S]$</td>
</tr>
<tr>
<td>3.2</td>
<td>$\mathbf{C}' \leftarrow C \cdot g_1^{\Sigma_{i \in S} (m'[S]-m[S]) a^i}$</td>
</tr>
<tr>
<td>4.</td>
<td>$\pi_1 \leftarrow \text{PR.Prove}(i, m, r)$:</td>
</tr>
<tr>
<td>4.1</td>
<td>$\mathbf{m}' \leftarrow m[m] \cdot r$</td>
</tr>
<tr>
<td>4.2</td>
<td>$\pi_{1'} \leftarrow g_1^{\alpha^{q+1} - m'[S] a[S]}$</td>
</tr>
<tr>
<td>5.</td>
<td>$\hat{\pi} \leftarrow \text{PR.Aggregate}(C, S, m[S], {\pi_i : i \in S})$:</td>
</tr>
<tr>
<td>5.1</td>
<td>$\hat{\pi} \leftarrow \Pi_{i \in S} \pi_{1'}^{i}$</td>
</tr>
<tr>
<td>6.</td>
<td>$b \leftarrow \text{PR.Verify}(C, S, m[S], \hat{\pi})$:</td>
</tr>
<tr>
<td>6.1</td>
<td>$t_i \leftarrow h^i(i, C, S, m[S])$</td>
</tr>
<tr>
<td>6.2</td>
<td>$b \leftarrow e \left( C, g_2^{\Sigma_{i \in S} (a^{q+1} - t_i)} \right) \leftarrow e(\hat{\pi}, g_2) \cdot g_T^{</td>
</tr>
<tr>
<td>7.</td>
<td>$\pi \leftarrow \text{PR.AggregateAcross}({C_j, S_j, m_j[S_j], \hat{\pi}<em>j}</em>{j \in [t]}$:</td>
</tr>
<tr>
<td>7.1</td>
<td>$\pi \leftarrow \Pi_{j=1}^t \pi_{j'}^{i}$</td>
</tr>
<tr>
<td>8.</td>
<td>$b \leftarrow \text{PR.VerifyAcross}({C_j, S_j, m_j[S_j]}_{j \in [t]}, \pi)$:</td>
</tr>
<tr>
<td>8.1</td>
<td>$b \leftarrow e \left( C, g_2^{\Sigma_{i \in S_j} (a^{q+1} - t_{j,i})} \right) \leftarrow e(\pi, g_2) \cdot g_T^{\alpha^{q+1} \Sigma_{j \in S_j} m_j, t_{j,i}, t'_{j,i}}$</td>
</tr>
</tbody>
</table>