An Expressive (Zero-Knowledge) Set Accumulator

Yupeng Zhang*, Jonathan Katz†, and Charalampos Papamanthou*
* *Dept. of Electrical and Computer Engineering, University of Maryland*
†*Dept. of Computer Science, University of Maryland*
Email: *{*zhangyp, cpap*}*@umd.edu*, †*jkatz@cs.umd.edu*

***Abstract*—We present a new construction of an expressive *set accumulator*. Unlike existing cryptographic accumulators, ours provides succinct proofs for a large collection of operations over accumulated sets, including intersection, union, set difference, `SUM`, `COUNT`, `MIN`, `MAX`, and `RANGE`, as well as arbitrary nestings of the above. We also show how to extend our accumulator to be *zero-knowledge*. The security of our accumulator is based on extractability assumptions and other assumptions that hold in the generic group model.**

Our construction has asymptotically optimal verification complexity and proof size, constant update complexity, and public verifiability/updatability—namely, any client who knows the public key and the last accumulator value can verify the supported operations and update the accumulator. The expressiveness of our accumulator comes at the cost of quadratic prover time. However, we show that the cryptographic operations involved are cheap compared to those incurred by generic approaches (e.g., SNARKs) that are equally expressive: our prover runs faster for sets of up to 5 million items.

Our accumulator serves as a powerful cryptographic tool with many applications. For example, it can be applied to efficiently support verification of a rich collection of SQL queries when used as a drop-in replacement in existing verifiable database systems (e.g., IntegriDB, CCS 2015).

1. Introduction

With the development of cloud computing, it has become common to delegate storage of data to external parties. Ensuring the correctness of operations (queries, computation, ...) performed on such data is of significant interest.

More specifically, in the setting we consider here a *data owner* outsources storage of data to a *server*. Multiple *clients* can then issue queries on the data, but want to verify the correctness of the answers provided by the server. The data owner and the clients can also potentially issue incremental updates to the data over time, and this should be cheaper than uploading everything again from scratch. We study a central problem in this setting, namely that of *verifiable set operations*. That is, we consider the data to be a collection of m sets $S_1, S_2, \ldots, S_m$ over some universe. Clients will store some constant-size cryptographic representations of each of the sets (i.e., they will store m digests $d_1, d_2, \ldots, d_m$) which they receive from the data owner. Our scheme enables the clients to verifiably learn the results of various operations performed on these sets, e.g., to learn the set $S_1 \cap S_2$, or the minimum element of $S_1 \cup S_2$. Such operations are motivated by their connections to several real-world applications including keyword search and SQL database queries.

We provide verifiability guarantees for the above setting by introducing a new *cryptographic accumulator*. Originally introduced by Benaloh and de Mare [7], and subsequently refined by Baric and Pfitzmann [2] and by Camenisch and Lysyanskaya [18], a *cryptographic accumulator* is a primitive that produces a succinct representation d_S of a set of elements S such that anyone having access to d_S can verify various operations on S by checking constant-size proofs. While previous constructions focused on queries such as set membership [7], [2], [18], [34], [17], non-membership [30], [21], and basic set operations [38], [19], our new construction combines, for the first time, the following features:

1) *Expressiveness*. Our new accumulator supports membership, non-membership, intersection, union, set difference, `SUM`, `COUNT`, `MIN`, `MAX`, and `RANGE`, as well as *arbitrary nestings* of these operations (e.g., $\texttt{COUNT}((\mathcal{A} \cup \mathcal{B}) \cap (\mathcal{C} \setminus \mathcal{D}))$).
2) *Efficiency*. Proof size and verification complexity only depend on the query size $|q|$ and the size of the result $|R|$, but not on the sizes of the original sets or intermediate results (in nested queries). In particular, proof size is $O(|q|)$ and verification complexity is $O(|q| + |R|)$.
3) *Public verifiability and updatability*. Any client with the public key and the accumulation values can verify correctness of computations, and clients given permission of the data owner can also update the sets and their respective digests. Our construction achieves $O(1)$ complexity for both insertion and deletion. This is a substantial improvement over previous work [7], [34], where updates can take quasilinear time without the trapdoor (known only to the data owner).
4) *Zero-knowledge*. We extend the basic construction to support zero-knowledge set operations, functions, nested queries, and updates.

The enhanced expressiveness of our approach comes at the cost of quadratic prover time. However, the cryptographic operations involved are cheap compared to equally expressive, quasilinear-time, generic constructions such as SNARKs [16], [4], [6], [5]. As we show in Section 6, our prover is indeed faster than prior work for sets containing fewer than 5 million items.

1.1. Technical Overview

We describe the intuition behind our scheme in this section. We first present a construction to support the most fundamental set operation: intersection. For sets $\mathcal{A}, \mathcal{B} \subset \mathbb{F}_q^*$, we set the accumulator values to be $\mathcal{A}_s = g^{\sum_{i \in \mathcal{A}} s^i} = g^{s+s^2+s^3}$ and $\mathcal{B}_{r,s} = g^{\sum_{i \in \mathcal{B}} r^i s^{q-i}} = g^{rs^{q-1}+r^3s^{q-3}+r^5s^{q-5}}$, where s, r are uniform (secret) points in $\mathbb{F}_q^*$. Note that the product of the two exponents gives

$$\begin{aligned}
&(s+s^2+s^3)\cdot(rs^{q-1}+r^3s^{q-3}+r^5s^{q-5}) \\
&\quad = \ (rs^q + r^3s^{q-2} + r^5s^{q-4}) + (rs^{q+1} + r^3s^{q-1} \\
&\qquad\quad + r^5s^{q-3}) + (rs^{q+2} + r^3s^q + r^5s^{q-2}) \\
&\quad = \ (r+r^3)s^q + Q(s,r),
\end{aligned}$$

for some bivariate polynomial Q. Only when some element i is in the intersection $\mathcal{I} = \mathcal{A} \cap \mathcal{B} = \{1, 3\}$ of both sets is the term $r^i s^q$ in the result, and thus the coefficient of s^q is $\sum_{i \in \mathcal{I}} r^i$. To compute the intersection, the server sends the result $\mathcal{I}$ along with the proof $g^{Q(s,r)}$, and by accessing the two accumulator values, the client can check the above relation in the exponent using bilinear pairings. Membership, union, and set difference can be reduced to intersection—see Section 3.3.

For other functions, if we let $\mathcal{A}(s) = s + s^2 + s^3$ be the polynomial in the exponent of $\mathcal{A}_s$, then the size of $\mathcal{A}$ is simply $\mathcal{A}(1)$, and the sum of the elements of $\mathcal{A}$ is $\mathcal{A}'(1)$ (where $\mathcal{A}'$ denotes the formal derivative of the polynomial $\mathcal{A}$). Also, the minimum element in the set $\mathcal{A}$ is given by the lowest-degree term in the polynomial $\mathcal{A}(s)$. We utilize these ideas to support verification of `COUNT`, `SUM`, `MAX`, and `MIN` queries. `RANGE` queries can be reduced to `MAX`, `MIN`, and other set-operation queries.

Supporting nested queries. For nested queries, one challenge is to ensure that the proof size is proportional to the size of the result and the size of the query, and not to the sizes of intermediate results or the original sets. The techniques above cannot give efficient proofs for nested queries (e.g., $(\mathcal{A} \cap \mathcal{B}) \cup \mathcal{C}$), because they would require returning the intermediate result $\mathcal{I} = \mathcal{A} \cap \mathcal{B}$ which will (in general) be too large.

In our approach the server returns only an *accumulator* of $\mathcal{I}$ and the client verifies its correctness. The client then uses this accumulator to verify $\mathcal{I} \cup \mathcal{C}$ using the scheme above. In Section 3.8 we show how to realize this by introducing additional accumulator values for each set.

1.2. Related Work

A detailed comparison with related work can be found in Table 1.

Accumulators and authenticated data structures. There are authenticated data structures [41], [32] that support (non-)membership and range queries, but with logarithmic proofs and update time. They do not support efficient proofs for more complicated queries, such as set intersection.

The *RSA accumulator* was proposed by Benaloh and de Mare [7], and supports membership queries with constant-size proofs. Camenisch and Lysyanskaya [18] extend it to the dynamic setting and augment it with zero-knowledge proofs. Efficient dynamic RSA accumulators for non-membership queries were given by Li et al. [30]. RSA accumulators do not support general set operations, and cannot handle efficient deletion without a trapdoor.

A *bilinear accumulator* was proposed by Nguyen [34]. It supports set-membership queries. Damgard et al. [21] extend it to support non-membership proofs. Papamanthou et al. [38] propose a scheme based on bilinear accumulators that supports set operations including intersection, union, and set difference. This was extended by Canetti et al. [19] to support nesting of the above three operations, based on an extractability assumption [8]. Zhang et al. [45] further extend the scheme to support `SUM` queries. However, this accumulator cannot support functions including `COUNT`, `MIN`, `MAX`, and `RANGE`.[1] Moreover, updates require quasilinear time without the trapdoor. We refer to [22] for a more complete list of prior work.

Ghosh et al. [25] propose zero-knowledge set operations based on the bilinear accumulator. Their scheme is not as expressive as ours, and in particular they do not consider zero knowledge for nested operations. On the other hand, their prover is more efficient than ours, and runs in linear (as opposed to quadratic) time.

Camenisch et al. [17] propose an accumulator based on bilinear groups that has some similarities to our scheme. However, they only support membership queries and cannot support nested operations.

NIZK argument for vector product. A non-interactive zero-knowledge (NIZK) argument for vector product was proposed by Groth [27]. That construction is similar to our scheme and can be used for set operations (e.g., by viewing the intersection of two sets as the inner product of their indicator vectors) but not nested queries. Lipmaa [31] improves the complexity of that scheme to quasilinear. Interestingly, Lipmaa's improved construction can support nested queries but does not support functions such as `SUM`, `COUNT`, `MAX`, `MIN`, and `RANGE`.

Generic approaches. General-purpose verifiable computation [23], [9], [10], [24], [11] and implementations thereof [40], [39], [4], [16], [42], [6], [5], [20] can provide expressive zero-knowledge set accumulators. To apply these approaches, the sets and supported operations would need to be compiled into a circuit or a RAM program. Although it is not hard to compile each set operation individually, it is somewhat complicated and expensive to generate circuits/RAM programs that can handle all possible nestings

1. Note that supporting such functions is only interesting when nested queries are considered, as otherwise the client can simply precompute and sign the results of such functions for each outsourced set. As such, in Table 1 we refer to the ability to support such functions in nested queries .

reference	set ops	functions	nested queries	setup time	prover time	proof size	verification time	update time
RSA [7], [2], [18], [26]	✗	✗	✗	$O(n)$	$O(n)$	$O(1)$	$O(\|R\|)$	$O(n)$
bilinear [34], [38], [19], [45], [25]	✓	only SUM [45]	✗	$\tilde{O}(n)$	$\tilde{O}(n)$	$O(d)$	$O(d+\|R\|)$	$\tilde{O}(n)$
NIZK product argument [27], [31]	✓	✗	✓	$O(n)$	$\tilde{O}(n)$	$O(d)$	$O(d+\|R\|)$	$O(1)$
circuit-based [39],[6,libsnark],[20]	✓	✓	✗	$O(n)$	$\tilde{O}(n)$	$O(1)$	$O(d+\|R\|)$	✗
RAM-based [16], [4], [6], [5]	✓	✓	✓	$O(n)$	$\tilde{O}(n)$	$O(1)$	$O(d+\|R\|)$	$\Omega(\log n)$
our work	✓	✓	✓	$O(n)$	$O(n^2)$	$O(d)$	$O(d+\|R\|)$	$O(1)$

TABLE 1. COMPARISON WITH PRIOR WORK. THE TOTAL NUMBER OF ELEMENTS IN ALL SETS INVOLVED IS n. EACH NESTED QUERY CAN BE EXPRESSED AS A QUERY TREE, AND WE LET d BE THE TOTAL NUMBER OF NODES IN THE TREE. $|R|$ IS THE SIZE OF THE FINAL RESULT. FUNCTIONS REFERS TO THE SET OF FUNCTIONS {SUM, COUNT, MIN, MAX, RANGE}. SETUP/UPDATE TIME IS WITHOUT USE OF ANY TRAPDOOR. WE USE $\tilde{O}(n)$ FOR $O(n^{1+o(1)})$. NOTE THAT ALTHOUGH RAM-BASED SCHEMES ARE AS EXPRESSIVE AS OUR WORK AND SOMETIMES HAVE ASYMPTOTICALLY BETTER PERFORMANCE, THEY ARE LESS EFFICIENT IN PRACTICE (SEE SECTION 6).

of those operations. Moreover, updates are not supported efficiently by circuit-based approaches, and introduce high overhead for RAM-based approaches. Detailed performance comparisons can be found in Section 6.

2. Preliminaries

Define $[z] = \{1, 2, \dots, z\}$. We let k denote the security parameter and let PPT stand for "probabilistic polynomial time." We use $\mathcal{A}$ for a set, and $\mathcal{A}(x)$, $\mathcal{A}(x, y)$ for characteristic polynomials of a set $\mathcal{A}$; these will be defined in Section 3.1. We use $(\mathbf{a}; \mathbf{b}) \leftarrow (\mathsf{A}||\mathsf{B})$ to denote running algorithms A and B together, where A outputs $\mathbf{a}$ and B outputs $\mathbf{b}$.

2.1. Bilinear Pairings

Let $\mathbb{G}$, $\mathbb{G}_T$ be two cyclic groups of order p with $g \in \mathbb{G}$ a generator. An efficiently computable bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ satisfies $e(P^a, Q^b) = e(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$. We denote by $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^k)$ running a PPT algorithm BilGen on input 1^k to generate parameters for a bilinear pairing. For simplicity we assume symmetric pairings in this paper, but our scheme can be adapted to use asymmetric pairings as well,

We rely on the following cryptographic assumptions.

Assumption 1 (*q*-SBDH [13]) *For all polynomials q and all* PPT *algorithms* A,

$$\Pr\left[pub \leftarrow \mathsf{BilGen}(1^k); s \leftarrow \mathbb{Z}_p; \sigma = (pub, g^s, \cdots, g^{s^q}); (c, h) \leftarrow \mathsf{A}(\sigma) :\ h = e(g, g)^{1/(c+s)}\right] \approx 0.$$

Assumption 2 *For any $\mathcal{W} \subset [q] \times [q]$ and every PPT adversary* Adv *there is a PPT extractor ε such that*

$$\Pr\Big[pub \leftarrow \mathsf{BilGen}(1^k); \alpha, s, r \leftarrow \mathbb{Z}_p^*; \sigma = (pub, \{g^{s^i r^j}\}_{(i,j)\in\mathcal{W}}, \{g^{\alpha s^i r^j}\}_{(i,j)\in\mathcal{W}}); (c, \hat{c}; \{a_{ij}\}_{(i,j)\in\mathcal{W}}) \leftarrow (\mathsf{Adv}||\varepsilon)(\sigma, z) :\ \hat{c} = c^\alpha \wedge c \neq \prod_{(i,j)\in\mathcal{W}} g^{a_{ij} s^i r^j}\Big] \approx 0$$

for any auxiliary information that is generated independently of α.[2]

Above, $(y; y') \leftarrow (\mathsf{Adv}||\varepsilon)(x)$ denotes Adv, given input x and a uniform random tape, outputs y, and ε, given the same random tape, outputs y'.

Assumption 2 is a generalization of the q-PKE assumption [8] to bivariate polynomials. We show in Appendix A that it can be reduced to a variant of the q-PKE assumption with univariate polynomials. The standard q-PKE assumption is a special case of Assumption 2 obtained by setting $r = 1$ and $\mathcal{W} = \{1\} \times [q]$.

Assumption 3 *For every PPT time adversary $\mathcal{A}$*

$$\Pr\left[pub \leftarrow \mathsf{BilGen}(1^\lambda); s, r, \alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p; \sigma = pk; (G(\cdot), h) \leftarrow \mathcal{A}(\sigma) :\ h = g^{G(s)r^q}\right] \approx 0,$$

where pk is defined in Section 3.1, $G(\cdot)$ is not a constant polynomial, and the degree of $G(\cdot)$ is less than $2q$.

Assumption 3 is a variant of the BDHE assumption [14]. We reduce this assumption to an assumption that holds in the generic group model in Section B in the Appendix. Later in our proofs, we sometimes use lemmas derived from these assumptions by changing the public parameters. The lemmas can be found in Section C of the Appendix and the proofs are trivial and omitted.

2.2. Expressive Set Accumulators

Our expressive set accumulator (**ESA**) scheme is parameterized by a set of supported queries Q and a set of updates UPD. For example, for our construction, Q includes (1) intersection $\cap$, union $\cup$, difference $\setminus$, symmetric difference $\triangle$. These functions take two sets as input and output a set. (2) COMPLEMENT, RANGE(a, b). These functions take one set as input and output a set. (3) MIN, MAX, SUM, COUNT, $\in_x$, $\notin_x$. These functions take one set as input and output a value with type integer or boolean. (The output can also be viewed as a set with one element.) (4) Nested queries consist of functions above. In a nested query, the input of

2. See [12], [15], [20] for background on this requirement.

an function can either be existing sets in the set collections or an intermediate set generated by prior functions in the nested query. The updates considered in our construction are insertion and deletion: $\mathsf{UPD} = \{(\texttt{ADD}, x), (\texttt{REMOVE}, x)\}$

An **ESA** scheme consists of the following PPT algorithms:

1) $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$: The algorithm is run by the owner. On input the security parameter k and a universe $\mathcal{U}$, it outputs a secret key sk and a public key pk;
2) $d_{\mathcal{A}} \leftarrow \mathsf{setup}(\mathcal{A}, \mathsf{sk}, \mathsf{pk})$: The algorithm is run by the owner. On input a set $\mathcal{A} \subseteq \mathcal{U}$ and the secret key sk, it computes the digest $d_{\mathcal{A}}$ of $\mathcal{A}$. In our construction, the digest can also be computed using pk only;
3) $d_{\mathcal{A}'} \leftarrow \mathsf{update}(d_{\mathcal{A}}, \mathsf{upd}, \mathsf{pk})$: The algorithm is run by the client. On input a digest $d_{\mathcal{A}}$, an update $\mathsf{upd} \in \mathsf{UPD}$ and the public key pk, it outputs the new digest $d_{\mathcal{A}'}$ of the updated set $\mathcal{A}'$.
4) $\{\pi_Q, \mathcal{R}\} \leftarrow \mathsf{Query}(\mathcal{A}_1, \ldots, \mathcal{A}_l, Q, \mathsf{pk})$: The algorithm is run by the server. On input a query $Q \in \mathsf{Q}$, sets $\mathcal{A}_1, \ldots, \mathcal{A}_l$ and the public key pk, it returns the result $\mathcal{R} = Q(\mathcal{A}_1, \ldots, \mathcal{A}_l)$ along with a proof π_Q;
5) $\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}_1}, \ldots, d_{\mathcal{A}_l}, Q, \pi_Q, \mathcal{R}, \mathsf{pk})$: The algorithm is run by the client. On input digests $d_{\mathcal{A}_1}, \ldots, d_{\mathcal{A}_l}$ of sets $\mathcal{A}_1, \ldots, \mathcal{A}_l$, a proof π_Q for a query Q, an answer $\mathcal{R}$ and public key pk, it outputs either `accept` or `reject`.

Correctness and soundness. We consider only correctness and soundness here; zero knowledge is defined in Section 4. Correctness of an **ESA** scheme is defined in the natural way and is omitted. For the soundness, consider the following experiment based on an **ESA** scheme specified by the algorithms above and an attacker Adv, and parameterized by security parameter k:

- **Step 1:** For universe $\mathcal{U}$ picked by Adv, experiment runs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$ and sends pk to Adv. Adv outputs l sets $\mathcal{A}_1, \ldots, \mathcal{A}_l$. The experiment stores sets $\mathcal{A}_1, \ldots, \mathcal{A}_l$ and computes $d_{\mathcal{A}_1} \leftarrow \mathsf{setup}(\mathcal{A}_1, \mathsf{sk}), \ldots, d_{\mathcal{A}_l} \leftarrow \mathsf{setup}(\mathcal{A}_l, \mathsf{sk})$.
- **Step 2:** Adv can run either of the following two procedures polynomially many times:
  - Update:
    - Adv outputs $u_i \in \mathsf{UPD}$, an update for set $\mathcal{A}_i$, for $i \in \{1, \ldots, l\}$. The experiments updates sets $\mathcal{A}_i$ and the experiment sets $d_{\mathcal{A}'_i} \leftarrow \mathsf{update}(d_{\mathcal{A}_i}, u_i, \mathsf{pk})$.
  - Query:
    - Adv outputs $(i_1, \ldots, i_{l'}, Q, \mathcal{R}, \pi)$ where $Q \in \mathsf{Q}$ is a query and $i_1, \ldots, i_{l'} \in \{1, \ldots, l\}$. Let $b \leftarrow \mathsf{Verify}(d_{\mathcal{A}_{i_1}}, \ldots, d_{\mathcal{A}_{i_{l'}}}, Q, \pi, \mathcal{R}, \mathsf{pk})$.
    - Event attack occurs if $b = \texttt{accept}$ but $\mathcal{R} \neq Q(\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_{l'}})$.

Definition 1 *An* **ESA** *scheme is* **sound** *if for all* PPT *adversaries* Adv*, the probability that event* attack *occurs in the above experiment is negligible.*

3. Construction

We describe an **ESA** scheme that supports queries Q and updates UPD, as described in the previous section. We only consider correctness and soundness in this section, and will extend the construction to zero-knowledge in the next section.

Theorem 1 *The* **ESA** *scheme below is correct and sound under Assumption 1, 2 and 3.*

3.1. Setup

Given a set $\mathcal{A}$, we define the polynomial $\mathcal{A}(x) = \sum_{i \in \mathcal{A}} x^i$. The accumulator of $\mathcal{A}$ is

$$g^{\mathcal{A}(s)} = g^{\sum_{i \in \mathcal{A}} s^i},$$

where s is randomly chosen in $\mathbb{Z}_p$. With this accumulator, note that the total number of elements in $\mathcal{A}$ is $\mathcal{A}(1)$ (i.e. polynomial $\mathcal{A}(x)$ evaluated at 1); the sum of all elements is $\mathcal{A}'(1)$ (i.e. the derivative of polynomial $\mathcal{A}(x)$ evaluated at 1); the minimum is the lowest degree of $\mathcal{A}(x)$. Techniques to evaluate a polynomial on the exponent and its derivative will be presented in Section 3.5, and we propose a scheme to retrieve the minimum degree of $\mathcal{A}(x)$ in Section 3.6. `MAX` can be handled similarly to `MIN`. To support set intersection, define the polynomial $\mathcal{A}(x, y) = \sum_{i \in \mathcal{A}} x^i y^{q-i}$, and introduce another accumulator value

$$g^{\mathcal{A}(r,s)} = g^{\sum_{i \in \mathcal{A}} r^i s^{q-i}},$$

where r is also randomly chosen from the field. We present the algorithms to support intersection in Section 3.3 and show how intersection implies all other set operations. In Section 3.7, we show that `RANGE` can be reduced to intersection, `MIN`, and `MAX`. We now present the algorithms of the scheme in detail.

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$:. Let $\mathcal{U} = [q]$, where $q = poly(k)$. Choose $\mathsf{sk} = (s, r, \alpha, \beta, \gamma, \delta)$ at random from $\mathbb{Z}_p^*$. Run $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^k)$, and set pk as:

- pub.
- g^α, g^β, g^γ, g^δ.
- g^{s^q}, g^{r^q}.
- g^{s^i}, $g^{\alpha s^i}$, g^{r^i}, $g^{\beta r^i}$, $g^{\gamma r^i s^{q-i}}$ for $i \in [q-1]$.
- $g^{r^i s^j}$, $g^{\delta r^i s^j}$ for $(i, j) \in ([2q-1] \setminus \{q\}) \times ([2q-1] \setminus \{q\})$.

Elements with $\alpha, \beta, \gamma, \delta$ in the public key will be used in the construction for an extractability assumption [8]. Note that while the public key of our construction above is different than the public key of Assumptions 1, 2 and 3, we show in Section C in the Appendix that the extra information in the public key does not make the adversary's task any easier.

$d_{\mathcal{A}} \leftarrow \mathsf{setup}(\mathcal{A}, \mathsf{pk})$: It sets the digest $d_{\mathcal{A}}$ as a vector of four group elements $[\mathcal{A}_s \;\; \mathcal{A}_{s,r} \;\; \mathcal{A}_r \;\; \mathcal{A}_{r,s}]$ where

$$\mathcal{A}_s = g^{\mathcal{A}(s)}, \mathcal{A}_{s,r} = g^{\mathcal{A}(s,r)}, \mathcal{A}_r = g^{\mathcal{A}(r)}, \mathcal{A}_{r,s} = g^{\mathcal{A}(r,s)}.$$

As explained in the overview above, $\mathcal{A}_s$ and $\mathcal{A}_{r,s}$ are used for set operations and functions. We also include $\mathcal{A}_r$

and $\mathcal{A}_{s,r}$ to support nested queries, which will be discussed in Section 3.8. Under this setting, s and r are symmetric, e.g, $\mathcal{A}_r$ and $\mathcal{A}_{s,r}$ can also be used for intersection.

Complexity. The size of the public key is $O(q^2)$, where q is the size of the universe. The setup complexity is $O(n)$ for a set with n elements.

3.2. Updates

Updates are very simple. To insert/delete an element, one simply multiplies/divides each accumulator value by the corresponding value in pk for the new element. The algorithm for update is presented below.

$\underline{d_{\mathcal{A}'} \leftarrow \mathsf{update}(d_{\mathcal{A}}, \mathsf{upd}, \mathsf{pk})}$: If upd is an insertion of element $w \in \mathcal{U}$ into set $\mathcal{A}$, update the group elements contained in $d_{\mathcal{A}}$ as $\mathcal{A}'_s = \mathcal{A}_s \times g^{s^w}, \mathcal{A}'_{s,r} = \mathcal{A}_{s,r} \times g^{s^w r^{q-w}}, \mathcal{A}'_r = \mathcal{A}_r \times g^{r^w}$ and $\mathcal{A}'_{r,s} = \mathcal{A}_{r,s} \times g^{r^w s^{q-w}}$. Deletions are performed similarly by dividing.

Complexity. The complexity for both insertion and deletion is $O(1)$. In contrast, RSA accumulators [7] and bilinear accumulators [34] only support efficient private update by the data owner with the trapdoor. The public update complexity is linear for RSA accumulators (for deletion) and quasilinear for bilinear accumulators.

3.3. Set Intersection

As mentioned in Section 1.1, to compute the intersection $\mathcal{I} = \mathcal{A} \cap \mathcal{B}$, we rely on the following relationship: $\mathcal{A}(s) \times \mathcal{B}(r,s) = \mathcal{I}(r) \times s^q + q(s,r)$ where $q(s,r)$ is a polynomial of s, r without the s^q term. The server sends the answer $\mathcal{I}$ and the proof $\mathcal{I}_r = g^{\mathcal{I}(r)}, \mathcal{I}_{r,\beta} = g^{\beta\mathcal{I}(r)}, \mathsf{Q}_{s,r} = g^{q(s,r)}, \mathsf{Q}_{s,r,\delta} = g^{\delta q(s,r)}, \mathsf{L}_r = g^{\mathcal{I}(r)/r}$. The client checks the relationship using bilinear pairings. The formal algorithms are as follows:

$\underline{\{\pi, \mathcal{R}\} \leftarrow \mathsf{Query}(\mathcal{A}, \mathcal{B}, \cap, \mathsf{pk})}$: Let $\mathcal{I} = \mathcal{A} \cap \mathcal{B}$ be the intersection. Compute polynomials $q(x,y)$ such that $\mathcal{A}(y) \times \mathcal{B}(x,y) = \left(\sum_{i \in \mathcal{I}} x^i\right) \times y^q + q(x,y) = \mathcal{I}(x) \times y^q + q(x,y)$ and $l(y)$ such that $\mathcal{I}(y) = l(y) \times y$. Compute $\mathcal{I}_r = g^{\mathcal{I}(r)}, \mathcal{I}_{r,\beta} = g^{\beta\mathcal{I}(r)}, \mathsf{Q}_{s,r} = g^{q(s,r)}, \mathsf{Q}_{s,r,\delta} = g^{\delta q(s,r)}, \mathsf{L}_r = g^{l(r)}$ and Output $\mathcal{R} = I$ and $\pi = \mathcal{I}_r, \mathcal{I}_{r,\beta}, \mathsf{Q}_{s,r}, \mathsf{Q}_{s,r,\delta}, \mathsf{L}_r$.

$\underline{\{\mathtt{accept}, \mathtt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}}, d_{\mathcal{B}}, \cap, \pi, \mathcal{I}, \mathsf{pk})}$: Output $\mathtt{accept}$ if and only if $e(\mathcal{A}_s, \mathcal{B}_{r,s}) = e(\mathcal{I}_r, g^{s^q}) \times e(\mathsf{Q}_{s,r}, g)$, $e(\mathcal{I}_r, g^{\beta}) = e(\mathcal{I}_{r,\beta}, g)$, $e(\mathsf{Q}_{s,r}, g^{\delta}) = e(\mathsf{Q}_{s,r,\delta}, g)$, $e(\mathcal{I}_r, g) = e(\mathsf{L}_r, g^r)$ and $g^{\sum_{i \in \mathcal{I}} r^i} = \mathcal{I}_r$.

Proof (Soundness proof for intersection). Suppose an adversary Adv returns $\mathcal{I}^*$ and $\mathcal{I}^*_r \neq g^{\mathcal{I}(r)}$ and passes the verification, then by Assumption 2, Adv with all but negligible probability, can use extractor ε_1 and ε_2 to derive a_i for $i = 0, \ldots, q-1$ and b_{ij} for $i, j = 0, \ldots, q-1, q+1, \ldots, 2q-1$ such that $\mathcal{I}^*_r = g^{\sum a_i r^i}$ and $\mathsf{Q}_{s,r} = g^{\sum b_{ij} s^i r^j}$. By the check $e(\mathcal{I}_r, g) = e(\mathsf{L}_r, g^r)$, if $a_0 \neq 0$, then $e(g,g)^{1/r} = (e(\mathsf{L}_r, g)/e(g^{-\sum_{i=1}^{q-1} a_i r^i}, g))^{a_0^{-1}}$, which breaks Lemma 1. Therefore, $a_0 = 0$ with all but negligible probability.

By the first check,

$$
\begin{aligned}
& e(g^{\mathcal{A}_s}, g^{\mathcal{B}_{r,s}}) = e(g^{\sum a_i r^i}, g^{s^q}) \times e(g^{\sum b_{ij} s^i r^j}, g) \\
\Leftrightarrow \quad & g^{\mathcal{I}(r)s^q + q(s,r)} = g^{(\sum a_i r^i)s^q + \sum b_{ij} s^i r^j} \\
\Leftrightarrow \quad & g^{(\mathcal{I}(r) - \sum a_i r^i)s^q} = g^{\sum b_{ij} s^i r^j - q(s,r)}.
\end{aligned}
$$

As $a_0 = 0$, $\mathcal{I}(r) - \sum a_i r^i$ is a non-constant polynomial of r and the right side can be evaluated by Adv, breaking Assumption 3. Therefore, $\mathcal{I}^*_r = g^{\mathcal{I}(r)}$ with all but negligible probability. Finally, if $\mathcal{I}^*_r = g^{\mathcal{I}(r)}$ but $\mathcal{I}^* \neq \mathcal{I}$, let min be the smallest element in $\mathcal{I} \cup \mathcal{I}^*$, then $g^{\mathcal{I}(r)} = g^{\mathcal{I}^*(r)} \Leftrightarrow g^{1/r} = g^{(\sum_{i \in \mathcal{I} \cup \mathcal{I}^* \setminus \{\mathsf{min}\}} r^i)/r^{\mathsf{min}+1}}$, breaking Lemma 1.

Complexity. Proof size is $O(1)$ and verification complexity is $O(|\mathcal{I}|)$, both optimal. Note that the verifier does not need all of the public key. Prover complexity is $O(|\mathcal{A}||\mathcal{B}|)$, dominated by the polynomial multiplication, as the two polynomials are sparse. Reducing prover complexity remains an open problem.

3.4. Other Set Operations

Once one verifies one of the accumulator values of the intersection $\mathcal{I}_r$, it is trivial to compute the accumulator value of the result of all other set operations.

1) Union $\mathcal{F} = \mathcal{A} \cup \mathcal{B}$: It is $\mathcal{F}_r = \mathcal{A}_r \mathcal{B}_r / \mathcal{I}_r$.
2) Difference $\mathcal{D} = \mathcal{A} \setminus \mathcal{B}$: It is $\mathcal{D}_r = \mathcal{A}_r / \mathcal{I}_r$ (or $\mathcal{D}_r = \mathcal{B}_r / \mathcal{I}_r$ for $B \setminus A$).
3) Symmetric difference $\mathcal{S} = \mathcal{A} \triangle \mathcal{B}$: It is $\mathcal{S}_r = \mathcal{A}_r \mathcal{B}_r / \mathcal{I}_r^2$.
4) $\bar{\mathcal{A}} = \mathtt{COMPLEMENT}(\mathcal{A})$: It is $\bar{\mathcal{A}}_r = \mathcal{U}_r / \mathcal{A}_r$, where $\mathcal{U}_r = g^{\sum_{i=1}^{q-1} r^i}$.
5) Subset $\mathcal{A} \subseteq \mathcal{B}$: It is $\mathcal{A} = \mathcal{A} \cap \mathcal{B}$ (i.e. $\mathcal{A}_r = \mathcal{I}_r$).
6) $x \in \mathcal{A}$: It is $\{x\} = \{x\} \cap \mathcal{A}$ (i.e. $\mathcal{I}_r = g^{r^x}$).
7) $x \notin \mathcal{A}$: It is $\emptyset = \{x\} \cap \mathcal{A}$ (i.e. $\mathcal{I}_r = 1$).

With the accumulator of the result of the above operations, the result itself can be verified by computing the accumulator value from scratch and checking the equality.

3.5. Sum and Count

With the accumulator value $\mathcal{A}_s = g^{\mathcal{A}(s)}$, the total number of elements in $\mathcal{A}$ is $\mathsf{count} = \mathcal{A}(1)$ and the summation of all elements in $\mathcal{A}$ is $\mathsf{sum} = \mathcal{A}'(1)$. We use a similar scheme to evaluate the polynomial in the exponent and its derivative to that in [37]. Note that $\mathcal{A}(x) - \mathcal{A}(1) = (x-1)a(x)$ and $\mathcal{A}(x) - \mathcal{A}(1) - \mathcal{A}'(1)(x-1) = (x-1)^2 b(x)$, where $a(x)$ and $b(x)$ can be computed in linear time[3]. Therefore, the server sends $g^{a(s)}$ as the proof for $\mathtt{COUNT}$ and $g^{b(s)}, \mathcal{A}(1)$ as the proof for $\mathtt{SUM}$. The client checks the relationships above through bilinear pairing. Detailed algorithms are below.

$\underline{\{\pi, v\} \leftarrow \mathsf{Query}(\mathcal{A}, \mathit{COUNT}, \mathsf{pk})}$: Compute $a(x) = (\mathcal{A}(x) - \mathcal{A}(1))/(x-1)$ and output $\pi = g^{a(s)}$ and $v = \mathcal{A}(1)$.

3. In general, polynomial division takes quasi-linear time. However, the special form of $\mathcal{A}(x)$ allows computing $a(x)$ and $b(x)$ in linear time. E.g., $a(x) = (\mathcal{A}(x) - \mathcal{A}(1))/(x-1) = \sum_{i \in \mathcal{A}} (x^i - 1)/(x-1) = \sum_{i \in \mathcal{A}} \sum_{j=1}^{i} x^j$.

$\underline{\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}}, \mathit{COUNT}, \pi, v, \mathsf{pk})}$: Output $\texttt{accept}$ if and only if $e(\mathcal{A}_s/g^v, g) = e(g^{s-1}, \pi)$.

Proof (Soundness proof for `COUNT`). Suppose an adversary Adv returns $v \neq \mathcal{A}(1)$ and passes the verification. Then

$$\begin{aligned} e(\mathcal{A}_s/g^v, g) &= e(g^{s-1}, \pi) \\ \Leftrightarrow e(g^{\mathcal{A}(s)-v}, g) &= e(g^{s-1}, \pi) \\ \Leftrightarrow e(g,g)^{a(s)(s-1)+\mathcal{A}(1)-v} &= e(g,\pi)^{s-1} \\ \Leftrightarrow e(g,g)^{1/(s-1)} &= (e(g,\pi)/e(g^{a(s)}, g))^{(\mathcal{A}(1)-v)^{-1}}. \end{aligned}$$

Adv can evaluate the right side, which breaks Lemma 1.

$\underline{\{\pi, v\} \leftarrow \mathsf{Query}(\mathcal{A}, \mathit{SUM}, \mathsf{pk})}$: Compute $b(x) = (\mathcal{A}(x) - \mathcal{A}(1) - \mathcal{A}'(1)(x-1))/(x-1)^2$, set $\pi_1 = g^{b(s)}$, $\pi_2 = \mathcal{A}(1)$ and output $\pi = \pi_1, \pi_2$ and $v = \mathcal{A}'(1)$.

$\underline{\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}}, \mathit{SUM}, \pi, v, \mathsf{pk})}$: Output $\texttt{accept}$ if and only if $e(\mathcal{A}_s, g) = e(g^{(s-1)^2}, \pi_1) \times e(g^{v(s-1)+\pi_2}, g)$.

Proof (Soundness proof for `SUM`). Suppose an adversary Adv returns $\pi_2 \neq \mathcal{A}(1)$ and passes the verification. Then

$$\begin{aligned} e(\mathcal{A}_s, g) &= e(g^{(s-1)^2}, \pi_1) \times e(g^{v(s-1)+\pi_2}, g) \\ \Leftrightarrow e(g^{\mathcal{A}(s)}, g) &= e(g^{(s-1)^2}, \pi_1) \times e(g^{v(s-1)+\pi_2}, g) \\ \Leftrightarrow e(g^{b(s)(s-1)^2+(\mathcal{A}'(1)-v)(s-1)+\mathcal{A}(1)-\pi_2}, g) &= e(g^{(s-1)^2}, \pi_1) \\ \Leftrightarrow e(g,g)^{1/(s-1)} &= \left(\frac{e(g^{s-1}, \pi_1)}{e(g^{b(s)(s-1)+(\mathcal{A}'(1)-v)}, g)}\right)^{(\mathcal{A}(1)-\pi_2)^{-1}} \end{aligned}$$

Adv can evaluate the right side, which breaks Lemma 1; if $\pi_2 = \mathcal{A}(1)$ and $v \neq \mathcal{A}'(1)$, continuing the above formula,

$$e(g,g)^{1/(s-1)} = (e(g,\pi_1)e(g^{b(s)}, g))^{(\mathcal{A}'(1)-v)^{-1}}$$

Again, Adv can evaluate the right side, which breaks Lemma 1;

Complexity. The proof size and the verification time are both $O(1)$, while the prover time is $O(n)$, where n is the size of the set.

3.6. Minimum and Maximum

The minimum value in set $\mathcal{A}$ is simply the lowest degree of $\mathcal{A}(s)$. In this section, we propose a scheme to extract it. Let min be the minimum, then $\mathcal{A}(s) - s^{\mathsf{min}}$ has a factor $s^{\mathsf{min}+1}$. We let the server return $g^{(\mathcal{A}(s)-s^{\mathsf{min}})/s^{\mathsf{min}+1}}$ as the proof, and let the client reconstruct the accumulator $\mathcal{A}_s$. Detailed algorithms are as follows:

$\underline{\{\pi, v\} \leftarrow \mathsf{Query}(\mathcal{A}, \mathit{MIN}, \mathsf{pk})}$: Output $v = \mathsf{min}$ and $\pi = g^{(\mathcal{A}(s)-s^{\mathsf{min}})/s^{\mathsf{min}+1}}$.

$\underline{\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}}, \mathit{MIN}, \pi, v, \mathsf{pk})}$: Output $\texttt{accept}$ if and only if

$$e(\mathcal{A}_s, g) = e(g^{s^v}, g) \times e(\pi, g^{s^{v+1}}).$$

Proof (Soundness proof for `MIN`). Suppose an adversary Adv returns $v < \mathsf{min}$ and passes the verification, then

$$\begin{aligned} e(\mathcal{A}_s, g) &= e(g^{s^v}, g) \times e(\pi, g^{s^{v+1}}) \\ \Leftrightarrow e(g^{s^v}, g) &= e(g^{\mathcal{A}(s)}, g)/e(\pi, g^{s^{v+1}}) \\ \Leftrightarrow e(g,g)^{1/s} &= e(g,g)^{\mathcal{A}(s)/s^{v+1}}/e(\pi, g). \end{aligned}$$

which breaks Lemma 1

Suppose Adv returns $v > \mathsf{min}$. Define the sets $\mathcal{B} = \{i \in \mathcal{A} : i < v\}$ and $\mathcal{C} = \{i \in \mathcal{A} : i \geq v\}$. By the condition in algorithm $\mathsf{Verify}()$, we have

$$\begin{aligned} e(\mathcal{A}_s, g) &= e(g^{s^v}, g) \times e(\pi, g^{s^{v+1}}) \\ \Leftrightarrow e(g,g)^{\sum_{i\in\mathcal{B}} s^i + \sum_{i\in\mathcal{C}} s^i} &= e(g,g)^{s^v} e(\pi, g)^{s^{v+1}} \\ \Leftrightarrow e(g,g)^{\frac{\sum_{i\in\mathcal{B}} s^i}{s^v}} &= e(g,g)e(\pi, g^s)e(g,g)^{-\sum_{i\in\mathcal{C}} s^{i-v}} .(1) \end{aligned}$$

Let now s^{min}, where $\mathsf{min} < v$, be the greatest common divisor of $\sum_{i\in\mathcal{B}} s^i$ and s^v (note that min is the correct minimum). Then there exists polynomials $a(s)$ and $b(s)$ such that

$$a(s)\sum_{i\in\mathcal{B}} s^i + b(s)s^v = s^{\mathsf{min}} \Leftrightarrow a(s)\sum_{i\in\mathcal{B}} s^i = s^{\mathsf{min}} - b(s)s^v .$$

Therefore continuing from Relation 1 we have

$$\begin{aligned} e(g,g)^{\frac{a(s)\sum_{i\in\mathcal{B}} s^i}{s^v}} &= e(g^{a(s)}, g)e(\pi, g^{sa(s)})e(g,g)^{-a(s)\sum_{i\in\mathcal{C}} s^{i-v}} \\ \Leftrightarrow e(g,g)^{1/s^{v-\mathsf{min}}} &= e(g^{a(s)}, g)e(\pi, g^{sa(s)}) \times \\ &\quad e(g,g)^{-a(s)\sum_{i\in\mathcal{C}} s^{i-v}} e(g, g^{b(s)}) \\ \Leftrightarrow e(g,g)^{1/s} &= e(g^{a(s)}, g^{s^{v-\mathsf{min}-1}})e(\pi, g^{s^{v-\mathsf{min}}a(s)}) \times \\ &\quad e(g^{s^{v-\mathsf{min}-1}}, g^{-a(s)\sum_{i\in\mathcal{C}} s^{i-v}})e(g^{s^{v-\mathsf{min}-1}}, g^{b(s)}) . \end{aligned}$$

which breaks Lemma 1.

Note that the maximum corresponds to the lowest degree in terms of s in $\mathcal{A}_{r,s}$. Therefore, a similar scheme can be used to extract the maximum.

$\underline{\{\pi, v\} \leftarrow \mathsf{Query}(\mathcal{A}, \mathit{MAX}, \mathsf{pk})}$: Output $v = \mathsf{max}$ and $\pi = g^{\frac{\mathcal{A}(r,s) - r^{\mathsf{max}} s^{q-\mathsf{max}}}{s^{q-\mathsf{max}+1}}}$.

$\underline{\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}}, \mathit{MAX}, \pi, v, \mathsf{pk})}$: Output $\texttt{accept}$ if and only if $e(\mathcal{A}_{r,s}, g) = e(g^{r^v s^{q-v}}, g) \times e(\pi, g^{s^{q-v+1}})$.
The proof is similar to `MIN` and is omitted.

Complexity. The proof size and the verification time are both $O(1)$, while the prover time is $O(n)$, where n is the size of the set.

3.7. Range Queries

We describe the algorithm for the case of range query in this section. It relies on the algorithms for intersection, minimum and maximum. In general, for range $[l, r]$, the idea is to decompose $\mathcal{A}$ to $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ such that: (1) $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ are pairwise disjoint; (2) $\mathcal{B} \cup \mathcal{C} \cup \mathcal{D} = \mathcal{A}$; (3) $max(\mathcal{B}) < l$, $min(\mathcal{C}) \geq l$, $max(\mathcal{C}) \leq r$ and $min(\mathcal{D}) > r$. $\mathcal{C}$ is the answer to the range query. However, the challenge is that $\mathcal{B}$ and $\mathcal{D}$ should not be returned, otherwise the proof is not efficient. Therefore, we let the server return the digests of $\mathcal{B}$ and $\mathcal{D}$. Note that these two digests are not honestly computed, but we will show that the strategy still works for our construction.

$\{\pi, \mathcal{C}\} \leftarrow \mathsf{Query}(\mathcal{A}, \mathit{RANGE}(l,r), \mathsf{pk})$: $\mathtt{RANGE}(l,r)$ is a range query on $\mathcal{A}$ with range $[l,r]$. Decompose $\mathcal{A}$ to $\mathcal{B} = \{i : i \in \mathcal{A}, i < l\}$, $\mathcal{C} = \{i : i \in \mathcal{A}, l \leq i \leq r\}$ and $\mathcal{D} = \{i : i \in \mathcal{A}, i > r\}$. Compute the following:

1) $\mathcal{B}_s$, $\mathcal{B}_{r,s}$, $\mathcal{D}_s$.
2) $\mathcal{B}(x) - \mathcal{B}(y,x) = Z(x,y) \times (y-1)$.
3) $\{\pi_{\mathcal{BC}}, \emptyset\} \leftarrow \mathsf{Query}(\mathcal{B}, \mathcal{C}, \cap, \mathsf{pk})$.
4) $\{\pi_{\mathcal{BD}}, \emptyset\} \leftarrow \mathsf{Query}(\mathcal{B}, \mathcal{D}, \cap, \mathsf{pk})$.
5) $\{\pi_{\mathcal{CD}}, \emptyset\} \leftarrow \mathsf{Query}(\mathcal{C}, \mathcal{D}, \cap, \mathsf{pk})$.
6) $\{\pi_1, max_{\mathcal{B}}\} \leftarrow \mathsf{Query}(\mathcal{B}, \mathtt{MAX}, \mathsf{pk})$.
7) $\{\pi_2, min_{\mathcal{D}}\} \leftarrow \mathsf{Query}(\mathcal{D}, \mathtt{MIN}, \mathsf{pk})$.

Output $\mathcal{C}$ and $\pi = \mathcal{B}_s$, $\mathcal{B}_{s,\alpha} = g^{\alpha\mathcal{B}(s)}$, $\mathcal{B}_{r,s}$, $\mathcal{B}_{r,s,\gamma} = g^{\gamma\mathcal{B}(r,s)}$, $\mathcal{D}_s$, $\mathcal{D}_{s,\alpha} = g^{\alpha\mathcal{D}(s)}$, $\mathsf{Z}_{s,r} = g^{Z(s,r)}$, $\pi_{\mathcal{BC}}, \pi_{\mathcal{BD}}, \pi_{\mathcal{CD}}, \pi_1, max_{\mathcal{B}}, \pi_2, min_{\mathcal{D}}$.

$\{\mathtt{accept}, \mathtt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}}, \mathit{RANGE}(l,r), \pi, \mathcal{C}, \mathsf{pk})$: Output $\mathtt{accept}$ if and only if

1) $e(\mathcal{B}_s, g^{\alpha}) = e(\mathcal{B}_{s,\alpha}, g)$, $e(\mathcal{B}_{r,s}, g^{\gamma}) = e(\mathcal{B}_{r,s,\gamma}, g)$, $e(\mathcal{D}_s, g^{\alpha}) = e(\mathcal{D}_{s,\alpha}, g)$.
2) $e(\mathcal{B}_s / \mathcal{B}_{r,s}, g) = e(g^{r-1}, \mathsf{Z}_{s,r})$.
3) $\mathtt{accept} \leftarrow \mathsf{Verify}(d_{\mathcal{B}}, d_{\mathcal{C}}, \cap, \pi_{\mathcal{BC}}, \emptyset, \mathsf{pk})$.
4) $\mathtt{accept} \leftarrow \mathsf{Verify}(d_{\mathcal{B}}, d_{\mathcal{D}}, \cap, \pi_{\mathcal{BD}}, \emptyset, \mathsf{pk})$.
5) $\mathtt{accept} \leftarrow \mathsf{Verify}(d_{\mathcal{C}}, d_{\mathcal{D}}, \cap, \pi_{\mathcal{CD}}, \emptyset, \mathsf{pk})$.
6) $\mathcal{B}_s \times g^{\sum_{i \in \mathcal{C}} s^i} \times \mathcal{D}_s = \mathcal{A}_s$.
7) $\mathtt{accept} \leftarrow \mathsf{Verify}(d_{\mathcal{B}}, \mathtt{MAX}, \pi_1, max_{\mathcal{B}}, \mathsf{pk})$.
8) $\mathtt{accept} \leftarrow \mathsf{Verify}(d_{\mathcal{D}}, \mathtt{MIN}, \pi_2, min_{\mathcal{D}}, \mathsf{pk})$.
9) $max_{\mathcal{B}} < l, min_{\mathcal{C}} \geq l, max_{\mathcal{C}} \leq r, min_{\mathcal{D}} > r$, where $min_{\mathcal{C}}$ is the minimum element in $\mathcal{C}$ and $max_{\mathcal{C}}$ is the maximum element in $\mathcal{C}$.

Proof (Soundness proof for RANGE). By check 1 and Assumption 2, the adversary, with all but negligible probability, can use extractors ε_1, ε_2, ε_3 to derive coefficients b_i, b'_i, d_i such that $\mathcal{B}_s = g^{\sum_{i=0}^{q-1} b_i s^i}$, $\mathcal{B}_{r,s} = g^{\sum_{i=0}^{q-1} b'_i r^i s^{q-i}}$ and $\mathcal{D}_s = g^{\sum_{i=0}^{q-1} d_i s^i}$.
Claim 1: $\mathcal{B}_s$ and $\mathcal{B}_{r,s}$ share the same coefficients in the polynomial on the exponent, with all but negligible probability. (i.e. $\forall i : b_i = b'_i$.)
Proof of Claim 1:
By check 2, if $\exists i : b_i \neq b'_i$, then

$$e(\mathcal{B}_s / \mathcal{B}_{r,s}, g) = e(g^{r-1}, \mathsf{Z}_{s,r})$$

$$\Leftrightarrow e(g^{\sum_{i=0}^{q-1}(b_i - b'_i)s^i + (r-1)Z(s,r)}, g) = e(g, \mathsf{Z}_{s,r})^{r-1}$$

$$\Leftrightarrow e(g,g)^{(\sum_{i=0}^{q-1}(b_i - b'_i)s^i)/(r-1)} = e(g, \mathsf{Z}_{s,r}) e(g^{-Z(s,r)}, g).$$

The adversary can evaluate the right side, which breaks Lemma 2. Therefore, $\forall i : b_i = b'_i$ with all but negligible probability.
Claim 2: Let $\mathcal{C}_s = g^{\sum_{i=0}^{q-1} c_i s^i}$ and $\mathcal{D}_s = g^{\sum_{i=0}^{q-1} d_i s^i}$, ($\mathcal{C}_s$ is computed by the client, and shares the same c_i with $\mathcal{C}_{r,s}$.) Then $\forall i \in [q]$, at most one of b_i, c_i, d_i is nonzero.
Proof of Claim 2: The proof is the same as for intersection. For example, if $\exists i \in [q] : b_i \neq 0$ and $c_i \neq 0$, then by the check of $\mathcal{B} \cap \mathcal{C}$, the adversary can evaluate $g^{G(r)s^q}$, which breaks Assumption 3.
Claim 3: $\mathcal{B}_s, \mathcal{B}_{r,s}, \mathcal{D}_s$ are the correct accumulators for sets $\mathcal{B}, \mathcal{D}$ such that $\mathcal{B}, \mathcal{C}, \mathcal{D}$ are pairwise disjoint and $\mathcal{B} \cup \mathcal{C} \cup \mathcal{D} = \mathcal{A}$.
Proof of Claim 3: Let $\mathcal{A}_s = g^{\sum_{i=0}^{q-1} a_i s^i}$, where $a_i \in \{0,1\}$. By check 6, if $\exists i : a_i \neq b_i + c_i + d_i$, then if there is only one index i such that $a_i \neq b_i + c_i + d_i$ (we use k to denote it), then we have

$$\mathcal{A}_s = \mathcal{B}_s \times \mathcal{C}_s \times \mathcal{D}_s \Leftrightarrow g^{\sum_{i=0}^{q-1} a_i s^i} = g^{\sum_{i=0}^{q-1}(b_i + c_i + d_i)s^i}$$

$$\Leftrightarrow g^{(a_k - b_k - c_k - d_k)s^k} = 1.$$

Which is impossible as $s \in \mathbb{Z}_p^*$; if there is more than one index i such that $a_i \neq b_i + c_i + d_i$, we use k to denote the smallest one, then we have

$$\mathcal{A}_s = \mathcal{B}_s \times \mathcal{C}_s \times \mathcal{D}_s$$

$$\Leftrightarrow g^{\sum_{i=0}^{q-1} a_i s^i} = g^{\sum_{i=0}^{q-1}(b_i + c_i + d_i)s^i}$$

$$\Leftrightarrow g^{\sum_{i=k+1}^{q-1}(a_i - b_i - c_i - d_i)s^i + (a_k - b_k - c_k - d_k)s^k} = 1$$

$$\Leftrightarrow g^{1/s} = g^{-\frac{1}{a_k - b_k - c_k - d_k} \sum_{i=k+1}^{q-1}(a_i - b_i - c_i - d_i)s^{i-k-1}}$$

$$\Leftrightarrow e(g,g)^{1/s} = e(g,g)^{-\frac{1}{a_k - b_k - c_k - d_k} \sum_{i=k+1}^{q-1}(a_i - b_i - c_i - d_i)s^{i-k-1}}.$$

Which breaks Lemma 1. Therefore, $\forall i : a_i = b_i + c_i + d_i$. As $a_i \in \{0,1\}$, by Claim 2, if $a_i = 0$, then $b_i = c_i = d_i = 0$; if $a_i = 1$, one of b_i, c_i, d_i equals 1 and the other two are 0s. This is equivalent to Claim 3.

By the security of queries $\mathtt{MAX}$ and $\mathtt{MIN}$, $max_{\mathcal{B}}$ is the maximum of $\mathcal{B}$ and $min_{\mathcal{D}}$ is the minimum of $\mathcal{D}$, with all but negligible probability. As $max_{\mathcal{B}} < l, min_{\mathcal{C}} \geq l, max_{\mathcal{C}} \leq r, min_{\mathcal{D}} > r$, together with Claim 3, $\mathcal{C}$ is the correct answer to the range query.

Complexity. The proof size is $O(1)$ and the verification time is $O(|\mathcal{C}|)$. The prover time is $O(n^2)$, where n is the size of the set, which is dominated by the algorithms for intersection.

3.8. Nested Queries

We utilize a similar idea to [19] to support nested queries. For each (single) function in Q, instead of outputting the result $\mathcal{R}$, we propose an algorithm $\mathsf{QueryNested}()$ that outputs the digest of the result $d_{\mathcal{R}}$ with a proof, instead of the result $\mathcal{R}$ itself. Similarly, we have $\mathsf{VerifyNested}()$ that verifies the correctness of $d_{\mathcal{R}}$ without accessing $\mathcal{R}$. In this way, to support a nested query, we can apply QueryNested() and VerifyNested() recursively for each single function in the nested query. For example, to retrieve the result of $\mathcal{A} \cap \mathcal{B} \cup \mathcal{C}$, the server first sends the digest $d_{\mathcal{I}}$ of the intermediate result $\mathcal{I} = \mathcal{A} \cap \mathcal{B}$ with its proof, and the client verifies it through VerifyNested(). Then with the correct digest of $\mathcal{I}$, the client can verifiably compute $\mathcal{R} = \mathcal{I} \cup \mathcal{C}$. The verification complexity is proportional to the number of single functions in the nested query and the size of the final result, but not to the size of any intermediate result or the original set.

Verifying the digest of an intersection. We first describe such QueryNested() and VerifyNested() algorithms for intersection. Recall in Section 3.3, the client can already check the correctness of the accumulator value $\mathcal{I}_r$ for $\mathcal{I}$, without seeing $\mathcal{I}$. ($\mathcal{I}$ is only used to compute the digest from scratch and check the equality in the last step). We simply extend it to also retrieve the other three values $\mathcal{I}_s$, $\mathcal{I}_{s,r}$ and $\mathcal{I}_{r,s}$. For $\mathcal{I}_{s,r}$, we apply the same construction as the proof 2 and check 2 for `RANGE` query in Section 3.7, i.e. $\mathcal{I}(r) - \mathcal{I}(s,r) = (s-1)Z(s,r)$ if $\mathcal{I}_{s,r}$ is correctly computed. For $\mathcal{I}_s$ and $\mathcal{I}_{r,s}$, as in our construction, s and r are totally symmetric, we rerun the same algorithms for the server and the client with s and r switched. In this way, the digest $d_{\mathcal{I}}$ can be verified without seeing the result $\mathcal{I}$. The algorithm is as follows:

```
{$\pi, d_{\mathcal{R}}$} ← QueryNested($\mathcal{A}, \mathcal{B}, \cap$, pk):
  1) Run the original algorithm Query() as in Section 3.3 without outputting $\mathcal{R}$. Let $\pi_1$ be the proof except the element $\mathcal{I}_r$.
  2) Compute $\mathcal{I}(y) - \mathcal{I}(x,y) = (x-1)Z(x,y)$. Set $\mathsf{Z}_{s,r} = g^{Z(s,r)}$, $\mathcal{I}_{s,r} = g^{\mathcal{I}(s,r)}$ and $\mathcal{I}_{s,r,\gamma} = g^{\gamma \mathcal{I}(s,r)}$.
  3) Repeat 1 and 2 with $s$ and $r$ switched. Let $\pi_2$, $\mathsf{Z}_{r,s}$ and $\mathcal{I}_{r,s,\gamma}$ be the corresponding proofs.
  4) Output $d_{\mathcal{I}} = \mathcal{I}_s, \mathcal{I}_r, \mathcal{I}_{s,r}, \mathcal{I}_{r,s}$ and $\pi = \pi_1, \pi_2, \mathsf{Z}_{s,r}, \mathsf{Z}_{r,s}, \mathcal{I}_{s,r,\gamma}, \mathcal{I}_{r,s,\gamma}$.

{accept, reject} ← VerifyNested($d_{\mathcal{A}}, d_{\mathcal{B}}, \cap, \pi, d_{\mathcal{R}}$, pk):
  1) Run the original algorithm Verify() as in Section 3.3 without the last check.
  2) Check if $e(\mathcal{I}_r/\mathcal{I}_{s,r}, g) = e(g^{s-1}, \mathsf{Z}_{s,r})$ and $e(\mathcal{I}_{s,r}, g^{\gamma}) = e(\mathcal{I}_{s,r,\gamma}, g)$.
  3) Repeat 1 and 2 with $s$ and $r$ switched.
  4) Output accept if and only if all checks above pass.
```

Proof (Soundness proof for intersection (nested)). Let $d^*_{\mathcal{I}} = \mathcal{I}^*_s, \mathcal{I}^*_r, \mathcal{I}^*_{s,r}, \mathcal{I}^*_{r,s}$ be the digest returned by the adversary Adv. By check 1 and the same proof as in Section 3.3, $\mathcal{I}^*_r = \mathcal{I}_r$, otherwise Adv breaks either Lemma 1 or Assumption 3.

By the second part of check 2 and Assumption 2, Adv, with all but negligible probability, can use extractors ε to derive coefficients a'_i such that $\mathcal{I}_{s,r} = g^{\sum_{i=0}^{q-1} a'_i s^i r^{q-i}}$. Meanwhile, we use a_i to denote the coefficient of $\mathcal{I}^*_r = \mathcal{I}_r = g^{\sum_{i=0}^{q-1} a_i r^i}$. Similar to the proof of Claim 1 in Section 3.7, by check 2, if $\exists i : a_i \neq a'_i$, then

$$e(\mathcal{I}_r/\mathcal{I}_{s,r}, g) = e(g^{s-1}, \mathsf{Z}_{s,r})$$

$$\Leftrightarrow e(g^{\sum_{i=0}^{q-1}(a_i - a'_i)r^i + (s-1)Z(s,r)}, g) = e(g, \mathsf{Z}_{s,r})^{s-1}$$

$$\Leftrightarrow e(g,g)^{(\sum_{i=0}^{q-1}(a_i - a'_i)r^i)/(s-1)} = e(g, \mathsf{Z}_{s,r}) e(g^{-Z(s,r)}, g).$$

The adversary can evaluate the right side, which breaks Lemma 2. Therefore, $\forall i : a_i = a'_i$ with all but negligible probability, thus $\mathcal{I}^*_{s,r} = \mathcal{I}_{s,r}$.

As s and r are symmetric, step 3 guarantees that $\mathcal{I}^*_s = \mathcal{I}_s$ and $\mathcal{I}^*_{r,s} = \mathcal{I}_{r,s}$.

The digest of the result for other set operations in a nested query can be computed by the relationship described in Section 3.4.

Verifying the digest of a range query. The algorithms in Section 3.7 still work if we let the server return the digest instead of $\mathcal{C}$, with the consistency check $\mathcal{C}(s) - \mathcal{C}(r,s) = (r-1)Z(s,r)$ for the polynomials on the exponent of $\mathcal{C}_s, \mathcal{C}_{r,s}$. In addition, as $\mathcal{C}$ is not directly returned, we need to rely on `MAX` and `MIN` queries to retrieve $\max_{\mathcal{C}}$ and $\min_{\mathcal{C}}$. As in the intersection, $\mathcal{C}_r$ and $\mathcal{C}_{s,r}$ can be retrieved through the same algorithms with s and r switched. The formal algorithms and the security proof are omitted.

Complexity. The proof size is $O(d)$ and the verification complexity is $O(d + |\mathcal{R}|)$ for a nested query with d single queries and the final result $\mathcal{R}$. (E.g. $d = 2$ in $\mathcal{A} \cap \mathcal{B} \cup \mathcal{C}$.)

Other functions inside nested queries. We support other functions inside nested queries. E.g., `MAX`($\mathcal{A}$)$\cup$`MIN`($\mathcal{B}$). However, as the result of a function is a single number, Query() and Verify() need not to be changed. The client retrieves the result of a function, views it as a set, computes its digest and proceeds to the next query. All these can be done in $O(1)$ time and thus the complexity above remains unchanged.

4. Zero-Knowledge Expressive Accumulators

In this section, we extend **ESA** to zk-**ESA**, providing perfect zero-knowledge operations. Our zk-**ESA** supports zero-knowledge set operations, `SUM`/`COUNT`, `MAX`/`MIN` and nested queries. We recall that zero-knowledge nested queries and functions are not supported in prior work (e.g., [25]).

4.1. Zero-Knowledge Definition

The security property of the accumulator in Definition 1 only guarantees that a malicious server cannot return an incorrect answer to pass the verification. However, the client may learn extra information about the sets through the proof. We introduce a notion of zero-knowledge proofs for our accumulator, which is the same as that for RSA accumulators in [18]. It guarantees that even an unbounded malicious client cannot learn anything about the set beyond what is queried. We formalize this in a similar way to the definition of zero-knowledge sets in [25]. We require that there exists a simulator with no access to the sets such that a malicious client cannot distinguish whether he is interacting with the algorithms of the scheme or with the simulator.

To build a zero-knowledge accumulator, the setup algorithm defined in Section 2.2 needs to additionally output an auxiliary information $\mathsf{aux}_{\mathcal{A}}$ for a set $\mathcal{A}$ to hide the information of the set. This auxiliary information is sent to the server. Algorithm Query takes the auxiliary information associated with its input sets to generate proofs, and update also updates the auxiliary information. We will use the algorithms with the auxiliary information in the zero-knowledge definition below. We first describe two experiments involving a challenger, an adversary Adv and a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2, \mathsf{Sim}_3, \mathsf{Sim}_4)$:

$\mathsf{Real}_{\mathsf{Adv}}(1^k)$:
1. The challenger runs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$ and sends pk to Adv.
2. Adv picks sets $\mathcal{A}_1, \ldots, \mathcal{A}_l$ and sends them to the challenger.
3. The challenger runs $(d_{\mathcal{A}_i}, \mathsf{aux}_{\mathcal{A}_i}) \leftarrow \mathsf{setup}(\mathcal{A}_i, \mathsf{sk})$ for $i = 1, \ldots, l$ and returns $d_{\mathcal{A}_1}, \ldots, d_{\mathcal{A}_l}$.
4. Adv runs the following polynomially many times:
4.1. The challenger runs $(d_{\mathcal{A}'_i}, \mathsf{aux}_{\mathcal{A}'_i}) \leftarrow \mathsf{update}(d_{\mathcal{A}_i}, \mathsf{upd}, \mathsf{pk})$ for $i \in \{1, \ldots, l\}$, forwards $d_{\mathcal{A}'_i}$ to Adv.
4.2. Adv sends a query Q and indices $i_1, \ldots, i_{l'}$ to the challenger. The challenger runs $\{\pi_Q, \mathcal{R}\} \leftarrow \mathsf{Query}(\mathcal{A}_{i_1}, \ldots, \mathcal{A}_{i_l}, Q, \mathsf{pk}, \mathsf{aux}_{\mathcal{A}_{i_1}}, \ldots, \mathsf{aux}_{\mathcal{A}_{i_{l'}}})$ and returns the output.
5. Adv outputs a bit b.

$\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^k)$:
1. The simulator runs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Sim}_1(1^k, \mathcal{U})$ and sends pk to Adv.
2. Adv picks sets $\mathcal{A}_1, \ldots, \mathcal{A}_l$.
3. Without seeing $\mathcal{A}_1, \ldots, \mathcal{A}_l$, the simulator runs $(d_{\mathcal{A}_i}, \mathsf{aux}_{\mathcal{A}_i}) \leftarrow \mathsf{Sim}_2(\mathsf{sk})$ for $i = 1, \ldots, l$ and returns $d_{\mathcal{A}_1}, \ldots, d_{\mathcal{A}_l}$.
4. Adv runs the following polynomially many times:
4.1. The simulator runs $(d_{\mathcal{A}'_i}, \mathsf{aux}_{\mathcal{A}'_i}) \leftarrow \mathsf{Sim}_3(d_{\mathcal{A}_i}, \mathsf{pk})$ for $i \in \{1, \ldots, l\}$, forwards $d_{\mathcal{A}'_i}$ to Adv.
4.2. Adv sends a query Q and indices $i_1, \ldots, i_{l'}$ to the simulator. With oracle access to the answer $\mathcal{R}$, the simulator runs $\{\pi_Q, \mathcal{R}\} \leftarrow \mathsf{Sim}_4(Q, \mathsf{sk}, \mathsf{pk}, \mathsf{aux}_{\mathcal{A}_{i_1}}, \ldots, \mathsf{aux}_{\mathcal{A}_{i_{l'}}})$ and returns the output.
5. Adv outputs a bit b.

Definition 2 *An* **ESA** *scheme is perfect zero-knowledge if for all unbounded adversaries* Adv *and all sets* $\mathcal{A}_1, \ldots, \mathcal{A}_l$*, there exists a* PPT *simulator* Sim *such that*

$$\Pr[\mathsf{Real}_{\mathsf{Adv}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda) = 1] = 0.$$

Note that in the update algorithm and the zero-knowledge definition above, the identity of the updated set is leaked during the update. (i.e. the updated set is the set whose digest is changed.) However, this information can be easily hided by rerandomizing and refreshing the digests of all sets during each update.

At the end of this section, we are able to prove that the following theorem holds for our zk-**ESA** scheme.

Theorem 2 *The zk-***ESA** *scheme proposed in this section is correct, sound and perfect zero-knowledge according to Definition 1 and Definition 2, under Assumption 1, 2 and 3.*

4.2. Setup

Algorithm $\mathsf{genkey}(1^k, \mathcal{U})$ picks $\mathsf{sk} = s, r, t, \alpha, \beta, \gamma, \delta$ at random from $\mathbb{Z}_p^*$. It runs $pub = (p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BilGen}(1^k)$. It sets pk as:

- pk' as defined in Section 3.1.
- g^t.
- $g^{ts^i}, g^{\alpha ts^i}$ for $i \in [q-1]$.
- $g^{tr^i}, g^{\beta tr^i}$ for $i \in [q-1]$.
- $g^{tr^i s^j}, g^{\delta tr^i s^j}$ for $(i, j) \in ([2q-1] \setminus \{q\}) \times ([2q-1] \setminus \{q\})$.
- $g^{tr^i s^q}, g^{\gamma tr^i s^q}$ for $i \in [q-1]$.

Algorithm $\mathsf{setup}(\mathcal{A}, \mathsf{sk})$ adds a randomizer on the exponent of each element in the original **ESA**. It sets the accumulator $d_{\mathcal{A}}$ of a set $\mathcal{A}$ as $\mathcal{A}_s = g^{\mathcal{A}(s) + r_1 ts^{\mathsf{min}+1}}$,$\mathcal{A}_r = g^{\mathcal{A}(r) + r_2 tr^{\mathsf{min}+1}}$, $\mathcal{A}_{s,r} = g^{\mathcal{A}(s,r) + r_3 ts^{\mathsf{max}-1} r^{q-\mathsf{max}+1}}$, $\mathcal{A}_{r,s} = g^{\mathcal{A}(r,s) + r_4 tr^{\mathsf{max}-1} s^{q-\mathsf{max}+1}}$, where min and max are the minimum and maximum elements in $\mathcal{A}$ and r_1, r_2, r_3, r_4 are randomly selected from $\mathbb{Z}_p^*$. It sends $\mathsf{aux}_{\mathcal{A}} = (r_1, r_2, r_3, r_4)$ to the prover.

4.3. Zero-Knowledge Set Operations

We first give a zero knowledge protocol for intersection. For simplicity, we use min, max, r_1, r_2, r_3, r_4 for set $\mathcal{A}$ and $\mathsf{min}', \mathsf{max}', r_5, r_6, r_7, r_8$ for set $\mathcal{B}$ in this section. The server views s, r, t as variables and computes

$$(\sum_{i \in \mathcal{A}} s^i + r_1 ts^{\mathsf{min}+1}) \times (\sum_{i \in \mathcal{B}} r^i s^{q-i} + r_8 tr^{\mathsf{max}'-1} s^{q-\mathsf{max}'+1})$$

$$\begin{aligned} =\ & \sum_{i \in \mathcal{I}} r^i s^q + \sum_{i \in \mathcal{A} \setminus \mathcal{I}, j \in \mathcal{B} \setminus \mathcal{I}} r^j s^{q+i-j} + \\ & r_1 ts^{\mathsf{min}+1} \sum_{i \in \mathcal{B}} r^i s^{q-i} + r_8 tr^{\mathsf{max}'-1} s^{q-\mathsf{max}'+1} \sum_{i \in \mathcal{A}} s^i \\ =\ & \mathcal{I}(r) s^q + q(s, r, t, r_1, r_8). \end{aligned}$$

The server computes the proof as $\mathcal{Q}_1 = g^{q(s,r,t,r_1,r_8)}$, $\mathcal{Q}_2 = g^{\delta q(s,r,t,r_1,r_8)}$ and returns the result $\mathcal{I}$. The client verifies the relationship above through pairings. The formal algorithms are as follows:

$\{\pi, \mathcal{R}\} \leftarrow \mathsf{Query}(\mathcal{A}, \mathcal{B}, \cap, \mathsf{pk}, \mathsf{aux}_{\mathcal{A}}, \mathsf{aux}_{\mathcal{B}})$: Let $\mathcal{I} = \mathcal{A} \cap \mathcal{B}$ be the intersection. Compute polynomial $q(s, r, t, r_1, r_8)$ defined above. Compute $\mathcal{Q}_1 = g^{q(s,r,t,r_1,r_8)}$, $\mathcal{Q}_2 = g^{\delta q(s,r,t,r_1,r_8)}$ and Output $\mathcal{R} = \mathcal{I}$ and $\pi = \mathcal{Q}_1, \mathcal{Q}_2$.

$\{\texttt{accept}, \texttt{reject}\} \leftarrow \mathsf{Verify}(d_{\mathcal{A}}, d_{\mathcal{B}}, \cap, \pi, \mathcal{I}, \mathsf{pk})$: Output `accept` if and only if $e(\mathcal{A}_s, \mathcal{B}_{r,s}) = e(g^{\sum_{i \in \mathcal{I}} r^i}, g^{s^q}) e(\mathcal{Q}_1, g)$ and $e(\mathcal{Q}_1, g^\delta) = e(\mathcal{Q}_2, g)$.

We first show that the soundness still holds, and then prove the scheme is zero-knowledge.

Proof (Soundness proof for zk-intersection). Suppose an adversary Adv returns $\mathcal{I}^* \neq \mathcal{I}$ and passes the verification, then by Assumption 2, Adv with all but negligible probability, can use extractor ε to extract a_{ij}, b_{ij} for

$i, j = 0, \ldots, q-1, q+1, \ldots, 2q-1$ and c_i for $i = 0, \ldots, q-1$ such that $\mathcal{Q}_1 = g^{\sum a_{ij} s^i r^j + \sum b_{ij} t s^i r^j + \sum c_i t r^i s^q}$.

By the first check,

$$e(\mathcal{A}_s, \mathcal{B}_{r,s}) = e(g^{\sum_{i \in \mathcal{I}^*} r^i}, g^{s^q}) e(\mathcal{Q}_1, g)$$

$$g^{\mathcal{I}(r)s^q + q(s,r,t,r_1,r_8)} = g^{\sum_{i \in \mathcal{I}^*} r^i s^q + \sum a_{ij} s^i r^j + \sum b_{ij} t s^i r^j + \sum c_i t r^i s^q}$$

$$g^{(\mathcal{I}(r) - \sum_{i \in \mathcal{I}^*} r^i) s^q} = g^{\sum a_{ij} s^i r^j + \sum b_{ij} t s^i r^j + \sum c_i t r^i s^q - q(s,r,t,r_1,r_8)}$$

As $\mathcal{I}^* \neq \mathcal{I}$, $\mathcal{I}(r) - \sum_{i \in \mathcal{I}^*} r^i$ is a non-constant polynomial of r and the right side can be evaluated by Adv. This breaks a variant of Assumption 3 with pk defined in Section 4.2, which can be easily reduced to Assumption 3.

Proof (Zero-knowledge proof). We present the real protocol first, followed by the interaction with a simulator. We will show that the corresponding steps in the real world and the ideal world are indistinguishable.

$\mathsf{Real}_{\mathsf{Adv}}(1^\lambda)$:
1. The challenger runs $\mathsf{pk} \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$ where pk is defined in Section 4.2, sends pk to Adv.
2. Adv picks sets $\mathcal{A}, \mathcal{B}$ and sends them to the challenger.
3. The challenger sets $d_\mathcal{A}, d_\mathcal{B}$ as defined above and sends $d_\mathcal{A}, d_\mathcal{B}$ to Adv.
4. Adv sends the intersection query.
The challenger returns the answer $\mathcal{I}$ and the proof $\mathcal{Q}_1 = g^{q(s,r,t,r_1,r_8)}$, $\mathcal{Q}_2 = g^{\delta q(s,r,t,r_1,r_8)}$.
5. Adv outputs a bit b.

$\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$:
1. The simulator picks $s, r, t, \alpha, \beta, \gamma, \delta$ and outputs $\mathsf{pk} \leftarrow \mathsf{Sim}_1(1^k, \mathcal{U})$ where pk is defined in Section 4.2. The simulator stores $s, r, t, \alpha, \beta, \gamma, \delta$.
2. Adv picks sets $\mathcal{A}, \mathcal{B}$.
3. Sim sets $d_\mathcal{A} = (g^{r_1}, g^{r_2}, g^{r_3}, g^{r_4})$ and $d_\mathcal{B} = (g^{r_5}, g^{r_6}, g^{r_7}, g^{r_8})$, where $r_1, \ldots, r_8$ are randomly chosen in $\mathbb{Z}_p^*$ and sends $d_\mathcal{A}$, $d_\mathcal{B}$ to Adv.
4. Adv sends the intersection query.
With the oracle access to the answer $\mathcal{I}$, Sim sends the answer $\mathcal{I}$ and the proof $\mathcal{Q}_1 = g^{r_1 r_8 - \mathcal{I}(r)s^q}$, $\mathcal{Q}_2 = g^{\delta(r_1 r_8 - \mathcal{I}(r)s^q)}$.
5. Adv outputs a bit b.

In Step 1 of both worlds, pk is computed as in Section 4.2, thus the two cases are obviously indistinguishable. There is no information received from the challenger/simulator in Step 2. In Step 3 of the ideal world, for each possible value of r_1, there exist a unique r_1' s.t. $\mathcal{A}_s = g^{r_1} = g^{\mathcal{A}(s) + r_1' t s^{\mathsf{min}+1}}$. The same argument holds for $r_2, \ldots, r_8$. In Step 4, $\mathcal{Q}_1 = g^{r_1 r_8 - \mathcal{I}(r)s^q} = g^{q(s,r,r_1',r_8')}$ for the same r_1', r_8' obtained from step 2. As r_1, r_8 are randomly chosen and Adv has no access to them, the values above in the two cases are indistinguishable. $\mathcal{Q}_2$ is uniquely determined by $\mathcal{I}$ and $\mathcal{Q}_1$. Therefore, the two cases are indistinguishable.

Membership can be reduced to $\{x\} \cap \mathcal{A} = \{x\}$ and subset is reduced to $\mathcal{A} \cap \mathcal{B} = \mathcal{A}$. Similar proofs of correctness, soundness and zero-knowledge apply. The protocols for other set operations will be discussed in the next section.

4.4. Zero-Knowledge Nested Queries

It is challenging to extend the zero knowledge protocol in Section 4.3 to support nested queries. The reason is that our solution in Section 3.8 requires the server to return the accumulator of an intersection $d_\mathcal{I}$ instead of $\mathcal{I}$. The client checks the validity of $d_\mathcal{I}$ and uses it to continue performing other operations in the nested query. With zk-**ESA**, the accumulator is randomized, the server can simply return a random number and any random number is a valid accumulator for some set. Therefore, in zk-**ESA**, we require the algorithms to satisfy the following security property: for any adversary Adv, there exist a PPT extractor ε that $\Pr[(\{\pi_Q, d_\mathcal{R}\}; \mathsf{aux}_\mathcal{R}) \leftarrow (\mathsf{Adv}||\varepsilon)(\mathcal{A}, \mathcal{B}, Q, \mathsf{pk}, \mathsf{aux}_\mathcal{A}, \mathsf{aux}_\mathcal{B});\ \mathtt{accept} \leftarrow \mathsf{Verify}(d_\mathcal{A}, d_\mathcal{B}, Q, \pi_Q, d_\mathcal{R}, \mathsf{pk}) : (d_\mathcal{R}, \mathsf{aux}_\mathcal{R}) \neq \mathsf{setup}(\mathcal{R}, sk)] \approx 0$. I.e., if the accumulator returned passes the verification, the server can use the extractor to extract the auxiliary information and express the accumulator in the correct form defined in Section 4.2.

We are going to give the protocol for zero knowledge intersection in a nested query below and sketch the security proof according to this property, followed by the proof of zero knowledge.

$\{\pi, d_\mathcal{I}\} \leftarrow \mathsf{QueryNested}(\mathcal{A}, \mathcal{B}, \cap, \mathsf{pk}, \mathsf{aux}_\mathcal{A}, \mathsf{aux}_\mathcal{B})$:

1) Compute $q(s, r, t, r_1, r_8)$ as defined in Section 4.3, set $\mathcal{I}_r = g^{\mathcal{I}(r) + r_9 t r^{\mathsf{min}''+1}}$, where min'' is the minimum element in $\mathcal{I}$. Output $\pi_1 = g^{\beta(\mathcal{I}(r) + r_9 t r^{\mathsf{min}''+1})}$,$\pi_2 = g^{q(s,r,t,r_1,r_8) - r_9 t r^{\mathsf{min}''+1} s^q}$, $\pi_3 = g^{\delta(q(s,r,t,r_1,r_8) - r_9 t r^{\mathsf{min}''+1} s^q)}$.
2) Compute $\mathcal{I}(r) + r_9 t r^{\mathsf{min}''+1} - \mathcal{I}(s,r) - r_{11} t s^{\mathsf{max}''-1} r^{q-\mathsf{max}''+1} = (s-1)Z(s,r) + t(r_9 r^{\mathsf{min}''+1} - r_{11} s^{\mathsf{max}''-1} r^{q-\mathsf{max}''+1})$. Set $\mathsf{Z} = g^{Z(s,r)}$, $\mathcal{I}_{min} = g^{r_9 r^{\mathsf{min}''+1}}$, $\mathcal{I}_{min,\beta} = g^{\beta r_9 r^{\mathsf{min}''+1}}$,$\mathcal{I}_{max} = g^{r_{11} s^{\mathsf{max}''-1} r^{q-\mathsf{max}''+1}}$, $\mathcal{I}_{max,\gamma} = g^{\gamma r_{11} s^{\mathsf{max}''-1} r^{q-\mathsf{max}''+1}}$, $\mathcal{I}_{s,r} = g^{\mathcal{I}(s,r) + r_{11} t s^{\mathsf{max}''-1} r^{q-\mathsf{max}''+1}}$ and $\mathcal{I}_{s,r,\gamma} = g^{\gamma(\mathcal{I}(s,r) + r_{11} t s^{\mathsf{max}''-1} r^{q-\mathsf{max}''+1})}$.
3) Repeat 1 and 2 with s and r switched.
4) Output $d_\mathcal{I} = \mathcal{I}_s, \mathcal{I}_r, \mathcal{I}_{s,r}, \mathcal{I}_{r,s}$ and π to include all the proofs.

$\{\mathtt{accept}, \mathtt{reject}\} \leftarrow \mathsf{VerifyNested}(d_\mathcal{A}, d_\mathcal{B}, \cap, \pi_o, d_\mathcal{R}, \mathsf{pk})$:

1) Check if $e(\mathcal{A}_s, \mathcal{B}_{r,s}) = e(\mathcal{I}_r, g^{s^q}) e(\pi_2, g))$, $e(\pi_1, g) = e(\mathcal{I}_r, g^\beta)$, $e(\pi_3, g) = e(\pi_2, g^\delta)$.
2) Check if $e(\mathcal{I}_r / \mathcal{I}_{s,r}, g) = e(g^{s-1}, \mathsf{Z}_1) e(g^t, \mathcal{I}_{min} / \mathcal{I}_{max})$, $e(\mathcal{I}_{s,r}, g^\gamma) = e(\mathcal{I}_{s,r,\gamma}, g)$, $e(\mathcal{I}_{min}, g^\beta) = e(\mathcal{I}_{min,\beta}, g)$ and $e(\mathcal{I}_{max}, g^\gamma) = e(\mathcal{I}_{max,\gamma}, g)$.
3) Repeat 1 and 2 with s and r switched.
4) Output $\mathtt{accept}$ if and only if all checks above pass.

Proof (Soundness proof for zk-nested intersection). We consider $\mathcal{I}_r$ first. Suppose an adversary Adv returns $\mathcal{I}_r^*$ that is not an accumulator value of $\mathcal{I}$ and passes the verification, then by Assumption 2, Adv with all but negligible probability, can use an extractor ε_1 to derive b_{ij}, c_{ij} for

$i, j = 0, \dots, q-1, q+1, \dots, 2q-1$ and d_i for $i = 0, \dots, q-1$ such that $\pi_2 = g^{\sum b_{ij} s^i r^j + \sum c_{ij} t s^i r^j + \sum d_i t r^i s^q}$. He can use another extractor ε_2 to derive e_i, f_j for $i = 0, \dots, q-1, j = 0, \dots, 2q-1$ such that $\mathcal{I}_r = g^{\sum e_i r^i + \sum f_j t r^j}$ By the first check,

$$\begin{aligned}
& e(\mathcal{A}_s, \mathcal{B}_{r,s}) = e(\mathcal{I}_r, g^{s^q}) e(\pi_2, g) \\
\Leftrightarrow\quad & g^{\mathcal{I}(r)s^q + q(s,r,t,r_1,r_2)} = \\
& g^{(\sum e_i r^i + \sum f_j t r^j) s^q + \sum b_{ij} s^i r^j + \sum c_{ij} t s^i r^j + \sum d_i t r^i s^q} \\
\Leftrightarrow\quad & g^{(\mathcal{I}(r) - \sum e_i r^i)) s^q} = \\
& g^{\sum f_j t r^j s^q + \sum b_{ij} s^i r^j + \sum c_{ij} t s^i r^j + \sum d_i t r^i s^q - q(s,r,t,r_1,r_2)}
\end{aligned}$$

As $\mathcal{I}^*$ is not an accumulator value of $\mathcal{I}$, $\mathcal{I}(r) - \sum e_i r^i$ is a non-constant polynomial of r and the right side can be evaluated by Adv. This breaks a variant of Assumption 3 with pk defined in Section 4.2.

By Check 2 and Assumption 2, Adv with all but negligible probability, can use an extractor ε_3 to derive e'_i, f'_i for $i = 0, \dots, q-1$ such that $\mathcal{I}_{s,r} = g^{\sum e'_i s^i r^{q-i} + \sum f'_i t s^i r^{q-i}}$. He can use another extractor ε_4 to derive the coefficients for $\mathcal{I}_{min}$ and $\mathcal{I}_{max}$. For simplicity, instead of denoting their coefficients, we denote $\mathcal{I}_{min}/\mathcal{I}_{max} = g^{Z_3(s,r,t)}$, which is sufficient for the proof. Suppose $\exists i : e'_i \neq e_i$, e.g., $\mathcal{I}_{s,r}$ is not the correct value for the accumulator of $\mathcal{I}$, then by Check 2,

$$\begin{aligned}
& e(\mathcal{I}_r/\mathcal{I}_{s,r}, g) = e(g^{s-1}, \mathsf{Z}) e(g^t, \mathcal{I}_{min}/\mathcal{I}_{max}) \\
\Leftrightarrow\quad & e(g^{\sum(e_i - e'_i) r^i + \sum(f_i - f'_i) t r^i + (s-1) Z'(s,r) + t Z_2(s,r)}, g) = \\
& e(g^{s-1}, \mathsf{Z}) e(g^t, g^{Z_3(s,r,t)}) \\
\Leftrightarrow\quad & e(g,g)^{\frac{\sum(e_i - e'_i) r^i + \sum(f_i - f'_i) t r^i + t Z_2(s,r) - t Z_3(s,r,t)}{s-1}} = \\
& e(g, \mathsf{Z}/g^{Z'(s,r)})
\end{aligned}$$

As $\exists i : e'_i \neq e_i$, $\sum(e_i - e'_i) r^i$ is a non-constant polynomial and the right side can be evaluated by Adv, which breaks a variant[4] of Lemma 2.

As s, r are symmetric, by Check 3, the correctness of $\mathcal{I}_s, \mathcal{I}_{r,s}$ can be proved in the same way above with s and r switched.

The intuition is that the verification guarantees that $\mathcal{I}_r$ returned by the server must have $\sum_{i \in \mathcal{I}} r^i$ on the exponent. The server has the freedom to choose the randomizer, and to ensure zero knowledge, he will add $r_9 t r^{\mathsf{min}''+1}$ on the exponent[5].

Proof (Zero-knowledge proof).

4. On the left side, the polynomial has one more variable t than Lemma 2. This can be easily reduced to Lemma 2 by a black-box reduction.

5. The server can set for example $r_9 = 0$ and still passes the verification. However, the server loses the zero knowledge guarantee of his sets, but the soundness of the result is still maintained.

$\mathsf{Real}_{\mathsf{Adv}}(1^\lambda)$:
1. The challenger runs $\mathsf{pk} \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$ where pk is defined in Section 4.2, sends pk to Adv.
2. Adv picks sets $\mathcal{A}, \mathcal{B}$ and sends them to the challenger.
3. The challenger sets $d_\mathcal{A}, d_\mathcal{B}$ as defined above and sends $d_\mathcal{A}, d_\mathcal{B}$ to Adv.
4. Adv sends the intersection query.
The challenger returns the answer $d_\mathcal{I}$ and the proof π defined above.
5. Adv outputs a bit b.

$\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$:
1. The simulator picks $s, r, t, \alpha, \beta, \gamma, \delta$ and outputs $\mathsf{pk} \leftarrow \mathsf{Sim}_1(1^k, \mathcal{U})$ where pk is defined in Section 4.2. The simulator stores $s, r, t, \alpha, \beta, \gamma, \delta$.
2. Adv picks sets $\mathcal{A}, \mathcal{B}$.
3. Sim sets $d_\mathcal{A} = (g^{r_1}, g^{r_2}, g^{r_3}, g^{r_4})$ and $d_\mathcal{B} = (g^{r_5}, g^{r_6}, g^{r_7}, g^{r_8})$, where $r_1, \dots, r_8$ are randomly chosen in $\mathbb{Z}_p^*$ and sends $d_\mathcal{A}$, $d_\mathcal{B}$ to Adv.
4. Adv sends the intersection query.
Sim sends the answer $\mathcal{I}_r = g^{r_9}, \mathcal{I}_{s,r} = g^{r_{11}}$ and the proof $\pi_1 = g^{\beta r_9}, \pi_2 = g^{r_1 r_8 - r_9 s^q}, \mathsf{Z} = g^{\frac{r_9 - r_{11}}{s-1} - \frac{r_{13} - r_{14}}{t}}, \mathcal{I}_{min} = g^{\frac{r_{13}}{t}}, \mathcal{I}_{min,\beta} = g^{\beta \frac{r_{13}}{t}}, \mathcal{I}_{max} = g^{\frac{r_{14}}{t}}, \mathcal{I}_{max,\gamma} = g^{\gamma \frac{r_{14}}{t}}$. Sim repeats the above with s and r switched.
5. Adv outputs a bit b.

In Step 1 of both worlds, pk is computed as in Section 4.2, thus the two cases are obviously indistinguishable. There is no information received from the challenger/simulator in Step 2. In Step 3 of the ideal world, for each possible value of r_1, there exist a unique r'_1 s.t. $\mathcal{A}_s = g^{r_1} = g^{\mathcal{A}(s) + r'_1 t s^{\mathsf{min}+1}}$. The same argument holds for $r_2, \dots, r_8$. In Step 4 of the ideal world, for each possible value of r_9, there exist a unique r'_9 s.t. $\mathcal{I}_r = g^{r_9} = g^{\mathcal{I}(r) + r'_9 t r^{\mathsf{min}''+1}}$. In Step 4, $\pi_1 = g^{\beta r_9} = g^{\beta(\mathcal{I}(r) + r'_9 t r^{\mathsf{min}''+1})}$. Similarly, π_2, Z in the proof of the ideal world is equal to their counter part in the real world masked by a unique and different random number. In addition, $\mathcal{I}_{min}$ and $\mathcal{I}_{max}$ are uniformly distributed in both worlds and $\mathcal{I}_{min,\beta}, \mathcal{I}_{max,\gamma}$ are uniquely determined by them. As $r_1, \dots, r_{14}$ are randomly chosen and Adv has no access to them, the values above in the two cases are indistinguishable.

Other set operations. Other set operations cannot be reduced to intersection trivially in zk-**ESA**. E.g., in section 3.8, to get the digest of $\mathcal{U} = \mathcal{A} \cup \mathcal{B}$, the client can himself compute $d_\mathcal{U} = d_\mathcal{A} \times d_\mathcal{B} / d_\mathcal{I}$. This no longer holds, because there is a randomizer on the exponent of each accumulator value.

Instead, we propose a more complicated protocol as following: the client first computes $d_{\mathcal{U}^*} = d_\mathcal{A} \times d_\mathcal{B} / d_\mathcal{I}$. Note that as mentioned above, this is not the correct accumulator for $\mathcal{U}$, and only the randomizer part is wrong. E.g.,

$$\mathcal{U}^*_s = g^{\mathcal{U}(s) + r_1 t s^{\mathsf{min}+1} + r_2 t s^{\mathsf{min}'+1} - r_3 t s^{\mathsf{min}''+1}},$$

but the correct one should be $g^{\mathcal{U}(s) + r_4 t s^{\mathsf{min}'''+1}}$. Therefore, the client further asks the server to return the correct digest $d_\mathcal{U}$, and runs the zero knowledge intersection protocol with the server to check $\mathcal{U}^* \cap \mathcal{U} = \mathcal{U}^*$ and $\mathcal{U} \cap \mathcal{U}^* = \mathcal{U}$, using the digests $d_{\mathcal{U}}$ and $d_{\mathcal{U}^*}$. In this way, it guarantees that $d_{\mathcal{U}}$ returned by the server is the correct digest for $\mathcal{U}$ and the client can further use it to perform other operations in the nested query.

Other set operations can be handled in similar ways.

4.5. Zero-Knowledge Sum and Count

In this section, we give zero-knowledge protocols for SUM and COUNT. To obtain the result of a count query for set $\mathcal{A}$, the client wants to evaluate the polynomial on the exponent of $\mathcal{A}_s$ at $s = 1, t = 0$. We use $F(x, y)$ to denote the polynomial, then $F(x, y) - F(1, 0) = (x - 1)q_1(x, y) + yq_2(y)$. Notice when $F(x, y) = \mathcal{A}(x) + r_1 y x^{\mathsf{min}+1}$, the degree of y in $q_1(x, y)$ is 1 and $q_2(y) = r_1$. The server then selects randomly $r_5 \in \mathbb{Z}_p^*$ and sets $\pi_1 = g^{q_1(s,t)+r_5 t}$ and $\pi_2 = g^{q_2(t)-r_5(s-1)}$. The server returns the answer $F(1, 0)$ together with the proofs. To verify, the client checks $e(\mathcal{A}_s/g^{F(1,0)}, g) \overset{?}{=} e(g^{s-1}, \pi_1)e(g^t, \pi_2)$.

The correctness and soundness can be proved similarly to that in [37]. We give the proof of zero-knowledge below.

Proof (Zero-knowledge proof for COUNT).

$\mathsf{Real}_{\mathsf{Adv}}(1^\lambda)$:
1. The challenger runs $\mathsf{pk} \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$ where pk is defined in Section 4.2, sends pk to Adv.
2. Adv picks sets $\mathcal{A}$ and sends them to the challenger.
3. The challenger sets $d_{\mathcal{A}}$ as defined in Section 4.2 and sends it to Adv.
4. Adv sends the count query.
The challenger returns the answer count and the proof $pi_1 = g^{q_1(s,t)+r_5 t}$, $\pi_2 = g^{q_2(t)-r_5(s-1)}$.

$\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$:
1. The simulator picks $s, r, t, \alpha, \beta, \gamma, \delta$ and outputs $\mathsf{pk} \leftarrow \mathsf{Sim}_1(1^k, \mathcal{U})$ where pk is defined in Section 4.2. The simulator stores $s, r, t, \alpha, \beta, \gamma, \delta$.
2. Adv picks sets $\mathcal{A}$.
3. Sim sets $d_{\mathcal{A}} = (g^{r_1 t}, g^{r_2 t}, g^{r_3 t}, g^{r_4 t})$, where r_1, r_2, r_3, r_4 are randomly chosen in $\mathbb{Z}_p^*$ and sends $d_{\mathcal{A}}$ to Adv.
4. Adv sends the count query.
With the oracle access to the answer count, Sim sends the answer count and the proof $\pi_1 = g^{\frac{r_5}{s-1}}$, $\pi_2 = g^{\frac{r_1 t - \mathsf{count} - r_5}{t}}$.

In step 1 of both worlds, pk is computed as in Section 4.2, thus the two cases are obviously indistinguishable. There is no information received from the challenger/simulator in step 2. In step 3 of the ideal world, for each possible value of $\mathcal{A}_s = g^{r_1 t}$, there exist a r_1' s.t. $\mathcal{A}_s = g^{r_1 t} = g^{\mathcal{A}(s)+r_1' t s^{\mathsf{min}_{\mathcal{A}}+1}}$. The same argument holds for other accumulator values. Therefore, Adv cannot distinguish the two cases through $\mathcal{A}_s$. In step 4 of both worlds, π_1 is uniformly distributed and there exists a unique π_2 for each π_1. Therefore, the two cases are indistinguishable.

SUM also relies on the polynomial evaluation on the exponent and can be handled similarly.

4.6. Zero-Knowledge Maximum and Minimum

We first describe the zero-knowledge protocol for MIN. To generate the proof, the server sets $\pi = g^{(\mathcal{A}(s)-s^{\mathsf{min}})/s^{\mathsf{min}+1}+r_1 t}$ and returns the result min together with the proof π. The client then checks if $e(\mathcal{A}_s, g) \overset{?}{=} e(g^{s^{\mathsf{min}}}, g)e(\pi, g^{s^{\mathsf{min}+1}})$.

The correctness and soundness follows directly from the original proofs for MIN in Section 3.6. We give the proof of zero-knowledge below.

Proof (Zero-knowledge proof for MIN).

$\mathsf{Real}_{\mathsf{Adv}}(1^\lambda)$:
1. The challenger runs $\mathsf{pk} \leftarrow \mathsf{genkey}(1^k, \mathcal{U})$ where pk is defined in Section 4.2, sends pk to Adv.
2. Adv picks sets $\mathcal{A}$ and sends them to the challenger.
3. The challenger sets $d_{\mathcal{A}}$ as defined in Section 4.2 and sends it to Adv.
4. Adv sends the MIN query.
The challenger returns min and $\pi = g^{(\mathcal{A}(s)-s^{\mathsf{min}})/s^{\mathsf{min}+1}+r_1 t}$.

$\mathsf{Ideal}_{\mathsf{Adv},\mathsf{Sim}}(1^\lambda)$:
1. The simulator picks $s, r, t, \alpha, \beta, \gamma, \delta$ and outputs $\mathsf{pk} \leftarrow \mathsf{Sim}_1(1^k, \mathcal{U})$ where pk is defined in Section 4.2. The simulator stores $s, r, t, \alpha, \beta, \gamma, \delta$.
2. Adv picks sets $\mathcal{A}$.
3. Sim sets $d_{\mathcal{A}} = (g^{r_1}, g^{r_2}, g^{r_3}, g^{r_4})$, where r_1, r_2, r_3, r_4 are randomly chosen in $\mathbb{Z}_p^*$ and sends $d_{\mathcal{A}}$ to Adv.
4. Adv sends the MIN query.
With the oracle access to the answer min, Sim sends min and $\pi = g^{\frac{r_1 - s^{\mathsf{min}}}{s^{\mathsf{min}+1}}}$.

In step 3 of the ideal world, for each possible value of $\mathcal{A}_s = g^{r_1}$, there exist a r_1' s.t. $\mathcal{A}_s = g^{r_1} = g^{\mathcal{A}(s)+r_1' t s^{\mathsf{min}+1}}$. The same argument holds for other accumulator values. Therefore, Adv cannot distinguish the two cases through $\mathcal{A}_s$. In step 4 of the ideal world, $\pi = g^{\frac{r_1 - s^{\mathsf{min}}}{s^{\mathsf{min}+1}}} = g^{(\mathcal{A}(s)-s^{\mathsf{min}})/s^{\mathsf{min}+1}+r_1' t}$ for the same r_1'. Therefore, the two cases are indistinguishable.

To support zero-knowledge MAX, a similar protocol to MIN is applied to the accumulator value $\mathcal{A}_{r,s}$.

4.7. Zero-Knowledge Updates

Zero-knowledge updates are again very simple in our construction. The client first checks the validity of the update using zero-knowledge membership query (i.e. $x \notin \mathcal{A}$ for $(\mathtt{ADD}, x)$ and $x \in \mathcal{A}$ for $(\mathtt{REMOVE}, x)$). If the update is valid, the client updates the digest through multiplication/division. Finally, the client asks the server to rerandomize the digest, and checks the new digest is for the updated set using intersections, as described in Section 4.4.

In addition, to hide which set is updated, the client chooses a random element for each set that is not updated, refreshes all digests and sends all random elements to the server.

5. Applications

5.1. Keyword Search

Our scheme can be applied to verifiable *keyword search* implemented by the *inverted index* data structure [1]. The file IDs of all the documents containing a common term in the dictionary are grouped into a set indexed by that term. A keyword search query for terms w_1 and w_2 returns all documents that contain both terms. This can be naturally modeled as an intersection of the two sets indexed by w_1 and w_2, and thus supported by our **ESA**.

In addition, it is common in practice to have range constrains on the result. E.g., in keyword search in the email inbox, a keyword search query could be "return emails that contain terms w_1 and w_2 and are received between time t_1 and t_2". To support such query, we embed a timestamp in the most significant bits of the file ID. For file i, the new file ID is $\mathsf{ID'}_i = t_i||\mathsf{ID}_i$, where $||$ means concatenation. Note that ID' still uniquely determines a file. Then the inverted index data structure is constructed and the sets indexed by w_1 and w_2 contain groups of ID's. An intersection of the two sets gives all ID's that contain the two terms. After that, a `RANGE` query for $[t1||00\ldots0, t2||11\ldots1]$ on the result of intersection gives those received between t_1 and t_2 (assuming ID is in binary representation). Such a `RANGE` on the result of an intersection can be supported by our **ESA** efficiently. In contrast, in prior work (e.g., [38]), either the verification is not optimal or additional data structures (e.g., *segment tree*) are used. Moreover, we can even support more complicated keyword search queries such as "return the total number of emails that contain terms w_1 and w_2 and are received between time t_1 and t_2, except those contain terms w_3 and w_4 between t_3 and t_4", which can be translated to a nested query containing `COUNT`, `RANGE`, intersection and set difference.

5.2. SQL Database Queries

Our scheme can be used for verifying database SQL queries. Verifiable database has been studied by various prior works [35], [28], [36], [43], [44], [29] and the most expressive system, *IntegriDB*, is proposed by Zhang et. al in [45], supporting update, SQL range queries, `JOIN`, `SUM`, `MAX/MIN`, `COUNT`, `AVG` and limited nesting of such queries. It applies the bilinear accumulator as a building block for set operations and `SUM` queries.

IntegriDB requires the accumulator to support algorithms for authenticated intersection, union, set difference and sum, which are all supported by our **ESA**. Therefore, by replacing the bilinear accumulator with **ESA**, we can support the same set of SQL queries as in [45]. Moreover, We will show that by applying our **ESA**, we can further support public update, join query with duplicates in nested queries, `DISTINCT` query (unique values in a column) and improve the complexity of `MAX/MIN`.

Public updatability. One limitation of IntegriDB is that it only supports efficient update from the data owner with the secret key, which is inherited from the bilinear accumulator. As mentioned in Section 3.2, it takes quasilinear time to insert/delete an element by a client without the secret key in bilinear accumulator (same as computing from scratch). Even for the data owner with the secret key, one key operation used in IntegriDB during the update is combining the accumulators of two disjoint sets (e.g. compute $d_{\mathcal{A}\cup\mathcal{B}}$ given $d_{\mathcal{A}}$ and $d_{\mathcal{B}}$, where $\mathcal{A}\cap\mathcal{B} = \emptyset$). As these sets are not stored by the data owner, IntegriDB relies on extra information stored in the authenticated data structure, which is the encryption of the exponent for each accumulator.

These problems can be solved by applying our **ESA** as the accumulator. First, as mentioned in Section 3.2, it takes $O(1)$ time for both insertion and deletion by a client without the secret key. Second, given the accumulators for two disjoint sets, the accumulator for the union is simply their product. Therefore, we support public update.

Join with duplicates. Another open problem listed in [45] is how to support join query with duplicates in a nested query. A join query in SQL returns the *Cartesian product* of the matching elements in the joined columns (and their corresponding rows). E.g., the result of a join query on $c_1 = [1,1,1,2,2,3]$ and $c_2 = [1,1,2,2,4]$ is $[1,1,1,1,1,1,2,2,2,2]$. In IntegriDB, a Merkle tree [33] is built for each of c_1 and c_2, where the leaves stores all unique values and their cardinalities in the column. Besides, a bilinear accumulator is computed by modeling the columns as multi-sets. To support a join query, the client first retrieves the intersection of the two columns using the accumulator (intersection of multi-sets $\mathcal{I} = \{1,1,2,2\}$), then for each unique value in the intersection, the client issues a search query to the Merkle tree to determine its cardinality in each column and computes the Cartesian product himself. The proof size and the verification complexity is $O(|R|\log n)$, where $|R|$ is the size of the result and n is the maximum size of the joined columns.

This technique cannot be generalized to a join query inside a nested query. E.g., if a SQL sum query is applied on top of the join query, then the client must compute the summation himself after getting the result of join, which is not efficient. How to support join queries with duplicates in nested queries is listed as an open problem in [45].

By using our **ESA**, we can both support join queries with duplicates in nested queries and improve the complexity. We construct the accumulator value for the column c_1 as: $acc(c_1) = g^{\sum n_i s^{x_i}} = g^{3s+2s^2+s^3}$, where n_i is the cardinality of element x_i. Similarly for c_2, we set $acc(c_2) = g^{2rs^{q-1}+2r^2s^{q-2}+r^4s^{q-4}}$. Notice that $acc(c_1) \times acc(c_2) = (6r+4r^2)\times s^q + Q(s,r)$, where the coefficient $6r+4r^2$ before s^q is exactly the exponent of the accumulator for the result of join $[1,1,1,1,1,1,2,2,2,2]$. Therefore, by running the same algorithms for intersection as described in Section 3.3, the joined column in the result can be verified. With this improvement, the proof size is $O(1)$ and the verification complexity is $O(|R|)$, both of which are totally independent of the size of the original database.

Moreover, the solution above can be easily generalized

TABLE 2. COMPARING WITH RAM-BASED GENERIC SCHEME. QUERY IS $\texttt{SUM}((\mathcal{A}\cap\mathcal{B})\cup(\mathcal{C}\cap\mathcal{D}))$. $\mathcal{A},\mathcal{B},\mathcal{C},\mathcal{D}$ HAVE n ELEMENTS EACH.

n		setup time	prover time	verification time	proof size	update time	server storage
1,000	[6]	9,000s	4,300s	19ms	288Bytes	??	23GB
	ESA	0.01s	3.6s	9ms	540Bytes	2.4×10^{-6}s	60MB
10,000	[6]	90,000s	43,000s	19ms	288Bytes	??	230GB
	ESA	0.1s	360s	9ms	540Bytes	2.4×10^{-6}s	6GB
5×10^6	[6]	4.5×10^7s	2.15×10^7s	19ms	288Bytes	??	1.2×10^5GB
	ESA	5s	9×10^7s	9ms	540Bytes	2.4×10^{-6}s	1.5×10^6GB

to nested queries. It is not hard to prove that SUM, COUNT, MAX/MIN, RANGE functions are still working with the modified accumulator. Therefore, a SQL sum query on the result of a join query is supported by applying a SUM on the digest of the result of join. The proof size and verification complexity are both $O(1)$, which is optimal.

Max/Min. IntegriDB relies on the tree structure to support SQL max/min query. E.g., to retrieve the maximum of a column, the server first sends the claimed maximum max, and the client then queries the range $[\mathsf{max}, \infty]$ to an authenticated interval tree [45] and accepts if and only if the result contains a single element max. The proof size and the verification complexity are logarithmic on the size of the column and it requires there is no duplicate in the column.

ESA can answer it directly using the MAX query for set. The proof size and the verification complexity are $O(1)$ and we support duplicates.

Unique values. DISTINCT query is used in SQL to return all unique values in a column. Similar to join query, we view each column as a multi-set, and encode the cardinality n_i of each element i in the accumulator: $acc(c) = g^{\sum n_i s^i}$. **ESA** can then support the distinct query as following: the server returns the value/cardinality pair (i, n_i) of each unique value and the client computes the accumulator from scratch and checks the equality with the one stored. The proof size and the verification complexity are both linear to the number of unique values. The same trick does not work in IntegriDB using bilinear accumulator. The proof could be the same, but the complexity to computes the bilinear accumulator $g^{\Pi(s+i)^{n_i}}$ from scratch is quasilinear in $n = \sum n_i$, which could be much more than the number of the unique values.

6. Comparison with Generic Schemes

Generic systems for verifiable computation (VC) can be classified into two categories: circuit-based and RAM-based. To apply circuit-based VC on expressive verifiable set operations, queries are compiled to a circuit and the VC system is run on the resulting circuit. As indicated in Table 1, circuit-based VC does not support arbitrary nesting of the queries. A bound on the degree of the nested query must be known in advance. Update is also not supported and the sets must be hard coded to the circuit.

RAM-based VC can support all queries and has better prover time and proof size than our **ESA**, as shown in Table 1. However, it is slower in practice for sets of reasonable size. In this section, we compare the performance of **ESA** to an efficient RAM-based generic VC system, SNARKs for C [6]. For SNARKs for C, the sets would be hardcoded in a program that takes a query as input and outputs the result. We estimate the performance by first expressing the queries in TinyRAM and then using this to determine the three parameters that affect performance: the number of instructions (L), the number of cycles (T) and the size of the input and output (N). We then use those parameters along with [6, Fig.9] to estimate the running time and the server storage (evaluation key size). We were unable to estimate the update time for SNARKs for C.

For **ESA**, we use the benchmark from an efficient bilinear group implementation: ate-pairing[6] (the same library is also used in SNARKs for C). In order to compare the performance on the same machine, we first find the number of CPU cycles for operations such as multiplication and pairing, then use the number of CPU cycles to estimate the running time on the same machine used in [6]. A multiplication of two elements in the group takes around 2500 CPU cycles and a pairing takes 1.4 million CPU cycles, as reported in the benchmark. These roughly correspond to 0.6 microsecond and 0.3ms on the computed used in [6], with a 3.40 GHz Intel Core i7-4770 CPU and 32 GB of RAM. Note that these are the only two operations used in our construction. Exponentiation, which is much more expensive than multiplication, is never used in **ESA**, as all the coefficients on the exponent of accumulators are 1s. Therefore, **ESA** is very efficient in practice.

As a representative example, we test the query $\texttt{SUM}((\mathcal{A}\cap\mathcal{B})\cup(\mathcal{C}\cap\mathcal{D}))$, where every set of $\mathcal{A},\mathcal{B},\mathcal{C},\mathcal{D}$ has n elements. We also show the time to delete one element from set $\mathcal{A}$. As shown in Table 2, **ESA** is faster in all aspects even for sets that contain 10000 elements. In particular, the setup time is faster than SNARKs for C by orders of magnitude. Note that the gap remains the same for larger sets, as the setup complexity is $O(n)$ for both schemes. The prover time is faster by $1194\times$ for $n = 1000$ and $120\times$ for $n = 10000$. Only when the size of the sets reaches 5 million, the prover time starts to be slower for **ESA**. The verification time for **ESA** is only 9ms, which is 2 times faster than SNARKs for C. The overhead of the proof size for **ESA** is only $1.9\times$, both are only several hundred of bytes. The server storage is smaller by 97% in **ESA** for a universe of 10,000. Although the prover time, proof size and server storage will

6. https://github.com/herumi/ate-pairing

eventually be larger than for SNARKs for C for larger sets and more complicated nested queries, Table 2 already shows practicality of **ESA** for reasonable sets and queries.

In addition, we also consider a more involved solution using circuit-based VC. The client constructs a circuit-based VC for intersection. The VC takes two digests as input from the client, takes two sets from the server as the external input, verifies the validity of the two sets, computes their intersection, and finally outputs the digest of this intersection. In this way, the size of the clients input and the output are both constant. After executing one operation, the client can get back the digest of the intermediate result and feed it to the next operation in a nested query. In this solution, the digest should be updatable for dynamic sets.

We estimate the performance of this solution. We use the incremental hash in [3] and implement the same query as in Table 2. For n=10,000, it roughly takes 120000s of prover time using this approach with libsnark [4]. The scheme is always slower than RAM-based VC, thus we do not include it in the table.

Acknowledgments

This research was supported in part by an NSF award #1514261 and by a NIST grant.

References

[1] Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press New York (1999)

[2] Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In EUROCRYPT97.

[3] Bellare, M., Micciancio, D.: A new paradigm for collision-free hashing: Incrementality at reduced cost. In Eurocrypt 1997.

[4] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: CRYPTO 2013

[5] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In CRYPTO 2014.

[6] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: USENIX Security (2014)

[7] Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In EUROCRYPT93.

[8] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference.

[9] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS 2012

[10] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: STOC 2013

[11] Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: TCC 2013

[12] Bitansky, N., Canetti, R., Paneth, O., Rosen, A.: On the existence of extractable one-way functions. In: STOC 2014

[13] Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. Journal of Cryptology, 2008.

[14] Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In EUROCRYPT 2005.

[15] Boyle, E., Pass, R.: Limits of extractability assumptions with distributional auxiliary input. In ASIACRYPT 2015.

[16] Braun, B., Feldman, A.J., Ren, Z., Setty, S.T.V., Blumberg, A.J., Walfish, M.: Verifying computations with state. In: SOSP, 2013

[17] Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: PKC 2009.

[18] Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In CRYPTO 2002.

[19] Canetti, R., Paneth, O., Papadopoulos, D., Triandopoulos, N.: Verifiable set operations over outsourced databases. In: PKC 2014.

[20] Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In IEEE Symposium on Security and Privacy, 2015.

[21] Damgård, I., Triandopoulos, N.: Supporting non-membership proofs with bilinear-map accumulators. IACR Cryptology ePrint Archive 2008, 538 (2008)

[22] Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: CT-RSA, 2015

[23] Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Crypto 2010.

[24] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: EUROCRYPT. pp. 626–645 (2013)

[25] Ghosh, E., Ohrimenko, O., Papadopoulos, D., Tamassia, R., Triandopoulos, N.: Zero-knowledge accumulators and set operations. Cryptology ePrint Archive 2015.

[26] Goodrich, M.T., Tamassia, R., Hasić, J.: An efficient dynamic and distributed cryptographic accumulator*. In: Information Security 2002.

[27] Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: ASIACRYPT 2010, pp. 321–340. Springer (2010)

[28] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data. pp. 121–132. ACM (2006)

[29] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Authenticated index structures for aggregation queries. TISSEC 13(4), 32 (2010)

[30] Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: ACNS 2007.

[31] Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Theory of Cryptography, pp. 169–189. Springer (2012)

[32] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., Stubblebine, S.G.: A general model for authenticated data structures. Algorithmica 39(1), 21–41 (2004)

[33] Merkle, R.C.: A certified digital signature. In Crypto ’89.

[34] Nguyen, L.: Accumulators from bilinear pairings and applications. In: Topics in Cryptology–CT-RSA 2005, pp. 275–292. Springer (2005)

[35] Pang, H., Tan, K.L.: Authenticating query results in edge computing. In: International Conference on Data Engineering, 2004.

[36] Papadopoulos, S., Papadias, D., Cheng, W., Tan, K.L.: Separating authentication from query execution in outsourced databases. In ICDE’09.

[37] Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC. pp. 222–242 (2013)

[38] Papamanthou, C., Tamassia, R., Triandopoulos, N.: Optimal verification of operations on dynamic sets. In: CRYPTO 2011.

[39] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE S&P 2013.

[40] Setty, S.T.V., McPherson, R., Blumberg, A.J., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). In: NDSS (2012)

[41] Tamassia, R.: Authenticated data structures. In: Algorithms-ESA 2003.

[42] Vu, V., Setty, S.T.V., Blumberg, A.J., Walfish, M.: A hybrid architecture for interactive verifiable computation. In: IEEE Symposium on Security and Privacy. 2013

[43] Yang, Y., Papadias, D., Papadopoulos, S., Kalnis, P.: Authenticated join processing in outsourced databases. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. pp. 5–18. ACM (2009)

[44] Yang, Y., Papadopoulos, S., Papadias, D., Kollios, G.: Authenticated indexing for outsourced spatial databases. The International Journal on Very Large Data Bases 18(3), 631–648 (2009)

[45] Zhang, Y., Katz, J., Papamanthou, C.: IntegriDB: Verifiable SQL for outsourced databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015.

Appendix A. Justification of Assumption 2

Assumption 4 *Let $\mathcal{W}$ be a set of q elements in $\mathbb{Z}_p^*$. For every PPT adversary* Adv *there exists a PPT extractor ε such that*

$$\Pr\Big[pub \leftarrow \mathsf{BilGen}(1^k); \alpha, s \leftarrow \mathbb{Z}_p^*; \sigma = (pub, \{g^{s^i}\}_{i\in\mathcal{W}}, \{g^{\alpha s^i}\}_{i\in\mathcal{W}}); (c, \hat{c}; \{a_i\}_{i\in\mathcal{W}}) \leftarrow (\mathsf{Adv}||\varepsilon)(\sigma, z) : \hat{c} = c^\alpha \wedge c \neq \prod_{i\in\mathcal{W}} g^{a_i s^i}\Big] \approx 0.$$

for any auxiliary information that is generated independently of α.

Assumption 4 is a variant of the standard q-PKE assumption, in which the powers are consecutive starting from 1 (e.g., $\mathcal{W} = \{1, \ldots, q\}$).

Assumption 2 can be easily reduced to Assumption 4 by setting $r = s^t$ (for an unknown t). In this way, $s^i r^j = s^{i+tj}$ and Assumption 2 is equivalent to Assumption 4 with the cardinality of $\mathcal{W}$ being q^2.

Appendix B. Justification of Assumption 3

We now prove Assumption 3 holds. To do that, we rely on the following assumption which obviously holds in the generic group model [14]—we omit the proof in this paper.

Assumption 5 *For every non-uniform probabilistic polynomial time adversary $\mathcal{A}$*

$$\Pr\Big[pub \leftarrow \mathsf{BilGen}(1^\lambda); r \leftarrow \mathbb{Z}_p; \sigma = (pub, \{g^{r^i}\}) \text{ for } i = [4q^2] \setminus \{3q, 5q, 7q, \ldots, q(4q-1)\}; (a_1, \ldots, a_{2q-1}, h) \leftarrow \mathcal{A}(\sigma) : \ h = g^{\sum_{i=1}^{2q-1} a_i r^{(2i+1)q}}\Big] \approx 0.$$

Informally, the public key includes all powers of g^{r^i} from 1 to $4q^2$, except odd multiples $3q, 5q, \ldots, q(4q-1)$. The assumption says that it is infeasible for an adversary to output an combination of these missing powers.

Proof. Suppose there exist an adversary Adv that breaks this assumption, we construct and adversary Adv' to break Assumption 5 as following: $pub \leftarrow \mathsf{BilGen}(1^\lambda)$; $r \leftarrow \mathbb{Z}_p$; given $\sigma = (pub, \{g^{r^i}\})$ for $i = [4q^2] \setminus \{3q, 5q, 7q, \ldots, q(4q-1)\}$, Adv' sets $s = r^{2q}$ (without knowing r) and randomly chooses $\alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p^*$ and sets $\sigma' = pk$, pk is defined in Section 3.1. Note that Adv' can use σ to compute all terms in pk when $s = r^{2q}$. (E.g., $g^{r^i s^j} = g^{r^{2qj+i}}$ for $(i, j) \in [2q-1] \setminus \{q\} \times [2q-1] \setminus \{q\}$.) Adv' then feed σ' to Adv to get $G(\cdot), h$ such that $h = g^{G(s)r^q}$. Notice that $G(s)r^q$ contains all the missing powers in σ (e.g., $sr^q = r^{3q}, s^2 r^q = r^{5q}$). Adv' sets $a_1, \ldots, a_{2q-1}$ as the coefficients in $G(s)$ and outputs $a_1, \ldots, a_{2q-1}, h$, which breaks Assumption 5.

Appendix C. Lemmas Derived from Assumptions

Lemma 1 *Based on Assumption 1, for every PPT* Adv

$$\Pr\Big[pub \leftarrow \mathsf{BilGen}(1^\lambda); s, r, \alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p; \sigma = pk; (c, h) \leftarrow \mathsf{Adv}(\sigma) : \ h = e(g, g)^{1/(c+s)}\Big] \approx 0.$$

Lemma 2 *Based on Assumption 1, for every PPT* Adv

$$\Pr\Big[pub \leftarrow \mathsf{BilGen}(1^\lambda); s, r, \alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p; \sigma = pk; (w(\cdot), c, h) \leftarrow \mathsf{Adv}(\sigma) : \ h = e(g, g)^{\frac{w(s)}{r+c}}\Big] \approx 0.$$

Lemma 3 *Based on Assumption 1, for every PPT* Adv

$$\Pr\Big[pub \leftarrow \mathsf{BilGen}(1^\lambda); s, r, \alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p; \sigma = pk; (w(\cdot), c, h) \leftarrow \mathsf{Adv}(\sigma) : \ h = e(g, g)^{\frac{w(s)}{r+cs}}\Big] \approx 0.$$

where pk is defined in Section 3.1, and $w(\cdot)$ is a non-zero polynomial.