Authenticating Aggregate Range Queries over Multidimensional Dataset

Jia XU
National University of Singapore
Email: xujia@comp.nus.edu.sg

Ee-Chien CHANG
National University of Singapore
Email: changec@comp.nus.edu.sg

Abstract

We are interested in the integrity of the query results from an outsourced database service provider. Alice passes a set D *of d-dimensional points, together with some authentication tag* T*, to an untrusted service provider Bob. Later, Alice issues some query over* D *to Bob, and Bob should produce a query result and a proof based on* D *and* T*. Alice wants to verify the integrity of the query result with the help of the proof, using only the private key. In this paper, we consider aggregate query conditional on multidimensional range selection. In its basic form, a query asks for the total number of data points within a d-dimensional range. We are concerned about the number of communication bits required and the size of the tag* T*. We give a method that requires $O(d^2)$ communication bits to authenticate an aggregate query conditional on d-dimensional range selection. Besides counting, summing and finding of the minimum can also be supported. Furthermore, our scheme can be extended slightly to authenticate d-dimensional usual (non-aggregate) range selection query with $O(d^2)$ bits communication overhead, improving known results that require $O(\log^{d-1} N)$ communication overhead, where N is the number of data points in the dataset.*

Keywords

Authentication, Aggregate Query, Database Outsourcing

1. Introduction

Alice has a set D of d-dimensional points. She preprocesses the dataset D using her private key to generate some authentication tag T. She sends (outsources) D and T to an untrusted service provider Bob. Then Alice deletes the original copy of dataset D and tag T from her local storage. Later Alice (or Charlie, in the public key setting) may issue a query over D to Bob, for example, an aggregate query conditional on a multidimensional range selection, and Bob should produce the query result and a proof based on D and T. Alice (or Charlie, in the public key setting) wants to authenticate the query result, using only her private key (using Alice's public key, in the public key setting).

We are concerned about the communication cost and the storage overhead on Bob's side. Such requirements exclude the following two straightforward approaches: (1) Bob sends back the whole dataset D with its tag T; (2) During preprocessing, Alice generates and signs answers to all possible queries.

The problem we study in this paper fits in the framework of the outsourced database applications [16], [41], which emerged in early 2000s as an example of "software-as-a-service" (SaaS). By outsourcing database management, backup services and other IT needs to a professional service provider, companies can reduce expensive cost in purchase of equipments and even more expensive cost in hiring or training qualified IT specialists to maintain the IT services [20].

Researches in secure outsourced database focus on two major aspects: privacy [41], [42], [43], [44] (i.e. protect the data confidientiality against both the service provider and any third party), and integrity [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30] (i.e. authenticate the soundness and completeness of query results returned by the service provider). In the latter aspect, a lot of works are done for "identity query" [22], i.e. the query result is a subset of the database. Only a few works [7] are devoted to the authentication of aggregate query.

In this paper, we are interested in the authentication of aggregate range query over static multidimensional dataset with sublinear (w.r.t. the number of data points within the query range) communication bits. A d-dimensional aggregate range query specifies a d-dimensional rectangular range (which can be represented by its two "end points"), and its outcome is the aggregated value over all points inside the range. Figure 1 shows a 2D aggregate range query. The aggregate operations we consider in this paper include counting, summing, and finding of the maximum and minimum. Besides aggregate query, we also discuss the usual range selection query as an extension.

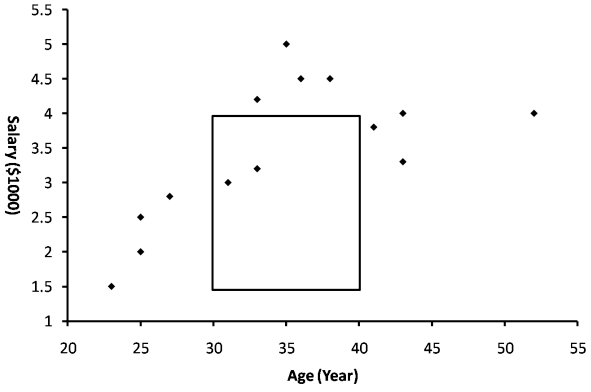
Our results. We propose a scheme, which we call MAIA (Multidimensional Aggregate query Integrity Authentication). For a dataset D with N d-dimensional points, the number of communication bits required is in $O(d^2)$. The storage overhead on Bob's side is $O(dN)$, which is linear in the size of D. Although the data points are d-dimensional, by

Table 1: Performance of different authentication schemes for aggregate range query or range selection query. Note that this table consists of two parts: (1) The first two rows are for aggregate query; (2) The rest four rows are for range selection query, and our scheme MAIA appears twice in this table, since it can authenticate both aggregate query and range selection query (with modifications in Section 6.1.3).

Scheme	Communication overhead (bits)	Key Size	Storage overhead	Computation (Verifier Alice)	Computation (Prover Bob)	Dimension d	Query
PDAS [7]	$O(\|S\| \log N)$	$O(1)$	$O(N)$	$O(\|S\| \log N)$	$O(\|S\| + K^2)$	$d = 1$	SUM,COUNT
MAIA (This paper)	$O(d^2)$	$O(d)$	$O(dN)$	$O(d \log N)$†	$O(dN \log N)$‡	$d \geq 1$	SUM,COUNT,MIN,MAX
Martel *et al.* [17]	$O(\log^{d-1} N + \|S\|)$	-	-	-	-	$d \geq 1$	Range Selection
Atallah *et al.* [26]	$O(1)$	$O(1)$	$O(N)$	$O(\|S\|)$	$O(1)$	$d = 1, 2$	Range Selection
Chen *et al.* [15]	$O(\log^d M)$	-	$O(N \log^d M)$	$O(\log^d M)$	$O(\log^d M)$	$d \geq 1$	Range Selection
MAIA (Section 6.1.3)	$O(d^2)$	$O(d)$	$O(dN)$	$O(d \log N)$†	$O(dN \log N)$‡	$d \geq 1$	Range Selection

N: The number of tuples in the dataset.
K: The number of servers in PDAS [7].
†: $O(d \log N)$ modular multiplications.
S: The set of tuples satisfying the query condition.
M: The domain size of attributes in Chen *et al.* [15].
‡: $O(dN \log N)$ modular exponentiations.

Figure 1: An example of 2D aggregate range query: How many employees with age between 30 and 40 have salary between \$1500 and \$4000? The query range is $[30, 40] \times [1500, 4000]$ and the query result is 2.



ignoring some dimensions, we can make an ℓ-dimensional range selection query, where $1 \leq \ell \leq d$. After preprocessing, our scheme is able to authenticate ℓ-dimensional aggregate range query (for any ℓ dimensions) with $O(\ell^2)$ communication bits.

The main reason that our scheme achieves efficient communication complexity, is that Alice can compress the messages, which could be viewed as a sequence of pseudorandom numbers, before sending them to Bob, and Bob can uncompress them, using the redactable signature scheme proposed by Johnson *et al.* [39].

The main idea of our construction is as follows: Let us consider the aggregate query on counting, and let S be the set of data points within the range. Recall that Bob should send a value N_0 to Alice and prove that $N_0 = |S|$. In the preprocessing, the dataset is normalized (Section 3.1.1), and each data point $\mathbf{x}$ is associated with a tag vector $\mathbf{tag}_k(\mathbf{e}, \boldsymbol{\mu}_\mathbf{x})$, where each component of the vector $\boldsymbol{\mu}_\mathbf{x}$ is some pseudorandom number generated from Alice's private key k, and $\mathbf{e} = (\underbrace{1, 1, 1, \cdots, 1}_{d \text{ 1's}})$. Here the function $\mathbf{tag}$ is homomorphic: $\mathbf{tag}(\mathbf{x} + \mathbf{y}, \mathbf{u} + \mathbf{v}) = \mathbf{tag}(\mathbf{x}, \mathbf{u}) \otimes \mathbf{tag}(\mathbf{y}, \mathbf{v})$, where the binary operator $\otimes$ denotes the direct product, i.e. for any vector $\mathbf{u} = (u_1, u_2, \cdots, u_d)$ and vector $\mathbf{v} = (v_1, v_2, \cdots, v_d)$, $\mathbf{u} \otimes \mathbf{v}$ denotes the vector $(u_1 v_1, u_2 v_2, \cdots, u_d v_d)$. In the first step of a query session, under our scheme, Alice is able to provide sufficient information (refered as *Help-Info* in Section 5) for Bob to generate a vector $\mathbf{tag}(0,\ \boldsymbol{\alpha} \otimes \boldsymbol{\mu}_\mathbf{x})$ for each $\mathbf{x}$ within the query range, but not for data point outside the range, using low communication cost, where $\boldsymbol{\alpha}$ is a vector of random nonces chosen by Alice.

Bob's proof consists of two parts. The first part of proof ensures that Bob indeed computes the aggregate value using points inside the query range only. Using homomorphic property of $\mathbf{tag}$, Bob is able to compute $\Psi = \mathbf{tag}_k(\sum_{\mathbf{x} \in S} \mathbf{e},\ \sum_{\mathbf{x} \in S} \boldsymbol{\mu}_\mathbf{x})$ from the tag values that are obtained from Alice in the setup, and $\Psi_{\boldsymbol{\alpha}} = \mathbf{tag}_k(0,\ \boldsymbol{\alpha} \otimes \sum_{\mathbf{x} \in S} \boldsymbol{\mu}_\mathbf{x})$ from $\mathbf{tag}(0,\ \boldsymbol{\alpha} \otimes \boldsymbol{\mu}_\mathbf{x})$'s that are generated using *Help-Info* provided by Alice in a query session. Bob presents $(\Psi, \Psi_{\boldsymbol{\alpha}})$ as the first part of proof for $N_0 = |S|$, and Alice can verify the consistency between Ψ and $\Psi_{\boldsymbol{\alpha}}$ using her private key k and the random nonce $\boldsymbol{\alpha}$. More precisely, Alice checks whether this equality holds: Let $\Psi = (\psi_1, \psi_2, \cdots, \psi_d)$ and $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_d)$,

$$(\psi_1^{\alpha_1}, \psi_2^{\alpha_2}, \cdots, \psi_d^{\alpha_d}) \quad = \quad \Psi_{\boldsymbol{\alpha}} \ \otimes \ \mathbf{tag}_k(\boldsymbol{\alpha} \otimes \ N_0 \mathbf{e},\ 0),$$

where the count number N_0 and (partial) proof Ψ and $\Psi_{\boldsymbol{\alpha}}$ come from Bob, and Alice keeps the secret nonce $\boldsymbol{\alpha}$ and computes $\mathbf{tag}_k(\boldsymbol{\alpha} \otimes \ N_0 \mathbf{e},\ 0)$. Note that if Ψ is computed honestly, we have $(\psi_1^{\alpha_1}, \psi_2^{\alpha_2}, \cdots, \psi_d^{\alpha_d}) =$

$\mathbf{tag}_k(\boldsymbol{\alpha} \otimes \ N_0\mathbf{e},\ \boldsymbol{\alpha} \otimes \sum_{\mathbf{x}\in S} \boldsymbol{\mu}_{\mathbf{x}})$.

The second part of proof ensures the completeness of the query result. Although the first part of proof can prevent *collusion attack*[1], it cannot prevent a dishonest Bob from excluding some data points within the query range, which we call *undercounting*. Furthermore, it cannot prevent a dishonest Bob from *double-counting*, e.g. claiming that the size of S was $2|S|$ with the partial proof $\Psi \otimes \Psi = \mathbf{tag}(2\sum_{\mathbf{x}\in S}\mathbf{e},\ 2\sum_{\mathbf{x}\in S}\boldsymbol{\mu}_{\mathbf{x}})$ and $\Psi_{\boldsymbol{\alpha}} \otimes \Psi_{\boldsymbol{\alpha}} = \mathbf{tag}(0,\ 2\left(\boldsymbol{\alpha} \otimes \sum_{\mathbf{x}\in S}\boldsymbol{\mu}_{\mathbf{x}}\right))$. With the second part of the proof, Bob convinces Alice that he does not undercount or double count, by dividing the domain into a few sub-regions and showing sums among these sub-regions are consistent (Similar strategy is also used in Chen *et al.* [15]). To achieve efficiency, the sub-regions have to be crafted carefully.

It is worthy to point out that, the above description is only for illustration of our ideas. So we purposely choose different notations $\mathbf{tag}$ for the tag fucntion and $\boldsymbol{\mu}_{\mathbf{x}}$ for the vector of random numbers to avoid confusion with the detailed description in Section 5. The actual construction relies on our specially designed tag function Tag (defined in Section 3), which consists of three components. Each component takes different role: (1) The first component prevents dishonest Bob to bring points outside the query range into the query result. (2) The second component binds different dimensions of the same point together, so dishonest Bob is not able to mix different dimensions of different points together to forge a new data point. (3) The third component is for the second part of proof. It ensures that undercounting or double-counting can be detected.

We design the tag function Tag using three cyclic multiplicative subgroups of $\mathbb{Z}_n^*$, for some proper composite modulus n. It may also be possible to design an alternative tag function using bilinear map, to achieve similar goal (the three roles described above). For example, the MRQED scheme, proposed by Shi *et al.* [1], might be a candidate (after some modifications). However, there are at least two major differences: (1)Like our scheme MAIA, authentication scheme for aggregate query based on MRQED (with modifications), just fits in the private key setting, although MRQED itself is a public key encryption scheme for the problem in paper [1]. MRQED is based on bilinear map, but our Tag function does not utilize bilinear map yet. (2) MRQED based approach may have $O(d^2 \log N)$ communication cost and $O(dN \log N)$ storage overhead on Bob's side, where d is the dimension and N is the number of points in the dataset. In contrast, our scheme MAIA takes $O(d^2)$ communication bits, independent of the dataset size and the query range, and $O(dN)$ storage overhead. For higher dimension d, in particularly, when $d > \log(dN)/\log\log N$, the complexities of MAIA will surpass (asymptotically smaller than) MRQED based scheme in communication cost, storage overhead, key size, and computation cost. In the other direction, it may be possible to apply some techniques in this paper to improve or make a tradeoff among complexities of MRQED.

The main contributions in this paper can be summarized as below:

1) We propose MAIA (Section 5), to authenticate multidimensional aggregate range queries, based on a specially designed homomorphic MAC function Tag (Section 4).
2) MAIA is efficient (See Table 1) and takes only $O(\ell^2)$ communication bits for an ℓ-dimensional aggregate range query, for each $1 \le \ell \le d$, independent of the query range size and dataset size.
3) We prove that MAIA is secure (Theorem 4, Corollary 5) under reasonable assumptions (Assumption 1, Assumption 2, etc.).
4) We extend MAIA to authenticate multidimensional range selection query (Section 6.1.3). The performance is showed in Table 1.

We also show a way to support multiple queriers using private key version of MAIA in Section 6.2.2.

2. Related work

Cryptography. Privacy-preserving computation and integrity verification are two major aspects of the security of outsourced computing. Many works in cryptography can be casted as privacy-preserving computation over ciphertexts, including homomorphic encryption [8], [9], Attribute Based Encryption [2], [3], Predicate Encryption [4], [1], [5], [6], and Order Preserving Encryption [10].

Shi *et al.* [1] proposed MRQED (Multi-Dimensional Range Query over Encrypted Data), a public key encryption scheme supporting multidimensional range queries over ciphertexts. Both MRQED [1] and MAIA deal with multidimensional range selection and have to prevent collusion attack cross different queries. But they are essentially different in at least these aspects: (1) MRQED dealt with privacy, and MAIA deals with integrity. (2) In MAIA, there is an aggregate operation after multidimensional range selection, and the verification of aggregated value is an additional requirement and not easy to handle when communication cost is concerned. (3) Besides collusion attack, MAIA also faces other challenges, like undercounting and double-counting attacks, which have no counterparts in researches of privacy-preserving computation, like MRQED.

Several works [11], [12], [13], [14] in verification of integrity of data stored in remote storage server also adopted some homomorphic and/or aggregatable verification tags to achieve efficient communication cost.

1. In this paper, the collusion attack means that, an adversary could utilize the information gathered through interactions with the verifier when authenticating previous queries, to cheat in the authentication of the next new query. It's a collusion across different query sessions.

Secure Outsourced Database. There are roughly four categories of approaches for outsourced database authentication in the literatures [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30]. (1) (Homomorphic, or aggregatable) Cryptographic primitives, like collision-resistant hash, digital signature, commitment [20], [31], [7]; (2) Merkle Hash Tree and variants (has a typical $O(\log |N|)$ complexity for proof size) [28], [24], [23]; (3) Computational geometry approach [17], [23], [26] (4) Inserting and auditing fake tuples [25].

Thompson *et al.* [7] proposed a scheme called PDAS, with superlinear communication cost, more precisely $O(\min\{|S|\log N,\ N\})$(S is the subset of tuples selected by the query condition), and linear storage overhead, to authenticate 1D aggregate (more precisely, SUM,COUNT) queries. PDAS is based on Shamir's threshold secret sharing [32] and Pedersen commitment scheme [33], and protected the privacy of the aggregated attributes from the verifier. The authors briefly mentioned that their scheme could handle aggregate queries conditional on multidimensional range selections over insensitive attributes (stored in plaintext), but no details are provided. It seems that PDAS still requires superlinear communication cost for multidimensional aggregate query. It is worthy to point out that, the techniques in PDAS(like secret sharing, and how to compare two integers privately) may be integrated into MAIA to provide similar privacy protection, without much tradeoff of communication cost (it may increase by the factor of τ, where τ is the bit-length of the maximum attribute value in the dataset).

3. Formulation

In this section, we formulize the problem and security model. We formally describe the dataset and query in Section 3.1, and gives a general security model in Section 3.2 for authentication of outsourced computing. Here in Table 2, we summarize the notations in this paper.

3.1. Problem

3.1.1. Dataset and Normalization. Let $\mathtt{D} \subset \mathbb{Z}^d$ be a set of N d-dimensional points. Let $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_N$ be an (arbitrary but fixed) enumeration of elements in D, where each point $\mathbf{x}_m = (x_{m,1}, x_{m,2}, \cdots, x_{m,d}), x_{m,j} \in \mathbb{Z}, 1 \le m \le N, 1 \le j \le d$.

Let us consider the *normalization* of D. Normalization is a technique commonly deployed in computational geometry [34]. That is, for each dimension $1 \le j \le d$, we sort[2] all points in D along the j-th dimension in increasing order, in order to get the rank, denoted as $i_{m,j}$, of $x_{m,j}, 1 \le m \le N$, among $\{x_{m',j} : 1 \le m' \le N\}$. As a result, each point $\mathbf{x}_m = (x_{m,1}, x_{m,2}, \cdots, x_{m,d})$ has a corresponding d-dimensional rank vector $\mathbf{i}_m = (i_{m,1}, i_{m,2}, \cdots, i_{m,d})$, where $i_{m,j}$ is the rank of $x_{m,j}$ along j-th dimension of D, $1 \le m \le N, 1 \le j \le d$. One can view $\mathbf{x}_m$ and $\mathbf{i}_m$ as two different coordinates of the same point w.r.t. two different coordinate systems. To distinguish these two types of coordinates, we call $\mathbf{x}_m$ as *attribute vector* or simply *attribute*, and $\mathbf{i}_m$ as *index vector* or simply *index*.

We denote with $\mathsf{Idx} : \mathtt{D} \to [N]^d$ the mapping from attribute $\mathbf{x}_m$ to index $\mathbf{i}_m$ w.r.t. dataset D. So $\mathsf{Idx}(\mathbf{x}_m) = \mathbf{i}_m$.

During the setup, Alice will compute the normalized dataset of D. To process a query, Bob can do the conversion between attributes and indices online, and Alice is able to verify the correctness. For convenience of presentation, in this paper we focus on range query over indices. The more realistic case, where range query is over attributes (e.g. the query in Figure 1), is handled in Section 6.1.1.

3.1.2. Query. Firstly, we describe the range selection query. Let $\mathbf{B} = [a_1, b_1] \times [a_2, b_2] \cdots \times [a_\ell, b_\ell] \times [N]^{d-\ell} \subseteq [N]^d$ be an ℓ-dimensional (rectangular) range in $[N]^d$, $1 \le \ell \le d$. The result to the range selection query with range $\mathbf{B}$ is the set of points in D whose index is in $\mathbf{B}$, i.e. the set $S_{\mathbf{B}} = \{\mathbf{x}_m \in \mathtt{D} : \mathsf{Idx}(\mathbf{x}_m) \in \mathbf{B}, 1 \le m \le N\}$.

Next, we look at aggregate range queries. Let $\iota \in [d]$. The sum query $\textsc{Sum}(\mathbf{B}, \iota)$ asks for the the summation of attribute values along the ι-th dimension of all points in $S_{\mathbf{B}}$, i.e. $\sum_{\mathbf{x} \in S_{\mathbf{B}}} \mathbf{x}[\iota]$. Similarly, the min query $\textsc{Min}(\mathbf{B}, \iota)$ asks for the minimum attribute value along the ι-th dimension of all points in $S_{\mathbf{B}}$, i.e. $\min_{\mathbf{x} \in S_{\mathbf{B}}} \mathbf{x}[\iota]$, and the count query $\textsc{Count}(\mathbf{B})$ asks for the number of points in $S_{\mathbf{B}}$, i.e. $|S_{\mathbf{B}}|$.

Remark. Although the above definition of ℓ-dimensional range restricts the very first ℓ dimensions, our scheme (with trivial modifications) can handle the general case: range selection over any ℓ out of d dimensions.

2. Ties can be broken in an arbitrary way. For example, if $x_{m_1,j} = x_{m_2,j}$, then $x_{m_1,j}$ is "smaller" than $x_{m_2,j}$ iff $m_1 < m_2$.

Table 2: Summary of Key Notations

- $\mathbb{Z}$: Set of all non-negative integers.
- $x \xleftarrow{\$} S$: x is uniformly randomly chosen from a finite set S.
- $[a, b]$: The set $\{a, a+1, a+2, \cdots, b-1, b\}$, where $a < b$ are non-negative integers.
- $[b]$: The set $\{1, 2, 3, \cdots, b\}$, where b is a positive integer.
- $\mathbf{i}[j]$: i_j, where $\mathbf{i} = (i_1, i_2, \cdots, i_j, \cdots)$ is a vector of dimension $\ge j$.
- $|S|$: The size of the set (or multiset) S.
- Prime: Set of all odd primes.
- $\mathsf{Tag}(x, i; \xi)$: Randomized function Tag with input (x, i) and random coin ξ (Section 4).
- MAIA: It stands for "Multidimensional Aggregate query Integrity Authentication" and it is our main scheme (Section 5).
- ProVer: It stands for "Prover-Verifier" and it is an interactive algorithm (Section 5).
- CollRes: It stands for "Collusion Resistant" and it is a subroutine called by algorithm ProVer (Section 5).

3.2. Security Model

First, we formulize the authentication problem described in Section 1. Next, we will state the assumptions for the security of our scheme.

Definition 1 ($\mathcal{RC}$)**.** A $\mathcal{RC}$ (*Remote Computing*) protocol for a key-ed function $F : \mathbb{M} \times \mathbb{K}_F \to \{0,1\}^*$, between Alice and Bob, consists of a setup phase and a query phase. The setup phase consists of a key generating algorithm KGen and data encoding algorithm DEnc; the query phase consists of a sequence of query sessions, and each query session consists of a pair of interactive algorithms, namely the evaluator Eval and the extractor Ext. These four algorithms $(\mathsf{KGen}, \mathsf{DEnc}, \langle \mathsf{Eval}, \mathsf{Ext} \rangle)$ run in the following way:

1) Alice generates a key $k : k \leftarrow \mathsf{KGen}(1^\kappa)$.
2) Alice encodes data $x \in \mathbb{M}$: $(p_x, s_x) \leftarrow \mathsf{DEnc}(x, k)$, then sends p_x to Bob and keeps s_x.
3) Alice selects a function key $r \in \mathbb{K}_F$.
4) Algorithm $\mathsf{Eval}(p_x)$ on the Bob's side, interacts with Algorithm $\mathsf{Ext}(s_x, r, k)$ on the Alice's side, to compute $y \leftarrow \langle \mathsf{Eval}(p_x), \mathsf{Ext}(s_x, r, k) \rangle$. If $y = \perp$, then Alice reject. Otherwise, Alice believes that y is equal to $F(x, r)$.

Definition 2 ($\mathcal{PRC}$)**.** A $\mathcal{RC}$ protocol $(\mathsf{KGen}, \mathsf{DEnc}, \langle \mathsf{Eval}, \mathsf{Ext} \rangle)$ w.r.t. function $F : \mathbb{M} \times \mathbb{K}_F \to \{0,1\}^*$, is called $\mathcal{PRC}$ (*Provable Remote Computing*) protocol, if the following two conditions hold

- correctness: for any $x \in \mathbb{M}$, $\mathbb{P}_{\mathsf{Eval}}(x) \succeq 1 - negl(\kappa)$ (asymptotically larger or equal);
- soundness: for any PPT adversary $\mathcal{A}$, for any $x \in \mathbb{M}$, $\mathbb{P}_{\mathcal{A}}(x) \succeq 1 - negl(\kappa)$ (asymptotically larger or equal),

where $\mathbb{P}_{\mathsf{Eval}}$ and $\mathbb{P}_{\mathcal{A}}$ are defined as

$$\mathbb{P}_{\mathsf{Eval}}(x) \triangleq \mathsf{Pr}\left[\begin{array}{l} k \leftarrow \mathsf{KGen}(1^\kappa); \\ (p_x, s_x) \leftarrow \mathsf{DEnc}(x, k); \\ r \xleftarrow{\$} \mathbb{K}_F; \\ \zeta \leftarrow \langle \mathsf{Eval}(p_x), \mathsf{Ext}(s_x, r, k) \rangle : \\ \zeta = F(x, r) \end{array}\right],$$

$$\mathbb{P}_{\mathcal{A}}(x) \triangleq \mathsf{Pr}\left[\begin{array}{l} k \leftarrow \mathsf{KGen}(1^\kappa); \\ (p_x, s_x) \leftarrow \mathsf{DEnc}(x, k); \\ \textit{for } 1 \le i \le poly(\kappa) \\ \quad r_i \xleftarrow{\$} \mathbb{K}_F; \\ \quad \zeta_i \leftarrow \langle \mathcal{A}(p_x, \mathsf{view}_{\mathcal{A}}^{\mathsf{Ext}}), \mathsf{Ext}(s_x, r, k) \rangle; \\ r \xleftarrow{\$} \mathbb{K}_F; \\ \zeta \leftarrow \langle \mathcal{A}(p_x, \mathsf{view}_{\mathcal{A}}^{\mathsf{Ext}}), \mathsf{Ext}(s_x, r, k) \rangle : \\ \zeta = \perp \ \vee \ \zeta = F(x, r) \end{array}\right].$$

The probability is taken over all random coins used by related algorithms, $poly(\cdot)$ is an arbitrary but fixed polynomial function, and $\mathsf{view}_{\mathcal{A}}^{\mathsf{Ext}}$ is a state variable[3] describing all random coins chosen by $\mathcal{A}$ and all messages $\mathcal{A}$ received from Ext during previous interactions.

In this paper, the function $F(\cdot, \cdot)$ that we are dealing with, is the aggregate SUM query over dataset D. More precisely, the first argument of F is D and the second argument is the query $\textsc{Sum}(\mathbf{B}, \iota)$ as defined in Section 3.1, and the output of F is the expected result to $\textsc{Sum}(\mathbf{B}, \iota)$ w.r.t. dataset D.

Let the group generator $\mathcal{G}$ be a randomized algorithm, which takes as input a security parameter 1^κ and outputs a tuple $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$, where $n = (2p+1)(2q+1)(2r+1)$ is a randomly chosen κ bits composite; all of $p, q, r, 2p+1, 2q+1$ and $2r+1$ are distinct primes; p, q and r are of the same bit-length; G_p, G_q and G_r are three cyclic multiplicative subgroups of $\mathbb{Z}_n^*$, of order p, q and r respectively; and g_p, g_q and g_r are randomly chosen generators of G_p, G_q and G_r respectively.

Assumption 1. *Algorithm $\mathcal{G}(1^\kappa)$ is run to obtain $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$. Let group $G = G_p \times G_q \times G_r$. The following two distributions are computationally indistinguishable,*

- $\mathcal{X} \triangleq \{\mathcal{X}_\kappa \xleftarrow{\$} G\}$*, i.e. $\mathcal{X}_\kappa$ is uniformly randomly distributed over G;*
- $\mathcal{Y} \triangleq \{\mathcal{Y}_\kappa \xleftarrow{\$} G_r\}$*, i.e. $\mathcal{Y}_\kappa$ is uniformly randomly distributed over G_r.*

Assumption 2. *Algorithm $\mathcal{G}(1^\kappa)$ is run to obtain $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$. Let group $G = G_p \times G_q \times G_r$. Define function $\mathcal{R} : G \to \mathbb{Z}_r$, such that for any $(x, y, z) \in \mathbb{Z}_p \times \mathbb{Z}_q \times \mathbb{Z}_r$, $\mathcal{R}(g_p^x g_q^y g_r^z) \triangleq z$. Let W be a set $\{w_i \in G : \mathcal{R}(w_i) \xleftarrow{\$} \mathbb{Z}_r\}$. Given only (n, W), it is hard to compute (A, B), such that*

$$A \not\equiv 1 \pmod{n}, \ B = \mathcal{R}(A) \mod r.$$

Assumption 1 can be considered as a variant version of the p-Subgroup Assumption [35], and also a variant version of the Subgroup Membership Problem ($\mathcal{SMP}$) Assumption [36], and Assumption 2 is closely related to the Projection Problem ($\mathcal{PP}$) Assumption [36].

4. Homomorphic Verification MAC

Definition 3. Algorithm $\mathcal{G}(1^\kappa)$ is run to obtain $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$. Randomly choose (s_1, s_2, s_3) from $\mathbb{Z}_p^* \times \mathbb{Z}_q^* \times \{0,1\}^\kappa$. Define (randomized) function $\mathsf{Tag} : \mathbb{Z} \times [N] \to \mathbb{Z}_n^*$ with key $k = (s_1, s_2, s_3, n, g_p, g_q, g_r, p, q, r)$ as follows:

$$\mathsf{Tag}_k(x, i; \xi) \triangleq g_p^{x + \frac{s_1}{h(i)}} \ g_q^{-x + s_2 \xi} \ g_r^{f_{s_3}(i)} \mod n$$

where the random coin $\xi \xleftarrow{\$} \mathbb{Z}_q^*$, $h(\cdot) : \mathbb{Z} \to \mathsf{Prime}$ is a collision-resistant[4] hash function, and $\{f_s\}_{s \in \{0,1\}^\kappa}$ is some

3. The adversary $\mathcal{A}$ may keep updating this state variable.

4. In other words, h should be *division intractable*. See Gennaro *et al.* [37] for the definition of "division intractable".

function (e.g. PRF [38]).

When we use the tag function Tag, we do not keep the values of ξ. For simplicity, we may just write $\mathsf{Tag}(x, i)$. As mentioned in the introduction, the tag function Tag consists of three components $\mathsf{P}, \mathsf{Q}, \mathsf{R}$:

$$\begin{aligned}
\mathsf{P}(x, s_1/h(i)) &= g_p^{x+\frac{s_1}{h(i)}} \mod n, \\
\mathsf{Q}(x, s_2\xi) &= g_q^{-x+s_2\xi} \mod n, \\
\mathsf{R}(f_{s_3}(i)) &= g_r^{f_{s_3}(i)} \mod n, \\
\mathsf{Tag}_k(x, i; \xi) &= \mathsf{P}(x, s_1/h(i))\mathsf{Q}(x, s_2\xi)\mathsf{R}(f_{s_3}(i)) \mod n.
\end{aligned}$$

Note that P, Q and R are homomorphic: for any inputs $y_1, y_2, z_1, z_2 \in \mathbb{Z}$,

$$\begin{aligned}
\mathsf{P}(y_1, z_1)\mathsf{P}(y_2, z_2) &\equiv \mathsf{P}(y_1 + y_2, z_1 + z_2) \pmod n; \\
\mathsf{Q}(y_1, z_1)\mathsf{Q}(y_2, z_2) &\equiv \mathsf{Q}(y_1 + y_2, z_1 + z_2) \pmod n; \\
\mathsf{R}(y_1)\mathsf{R}(y_2) &\equiv \mathsf{R}(y_1 + y_2) \pmod n.
\end{aligned}$$

Furthermore, $\mathsf{P}(y_1, z_1)^p \equiv \mathsf{Q}(y_1, z_1)^q \equiv \mathsf{R}(y_1)^r \equiv 1 \pmod n$. We point out that, with the key k, one can "extract"[5] out P, Q or R component from a tag value generated using Tag; without the key k, it is hard to do so, due to Assumption 2.

Lemma 1. *Given $\{(x_i, i, \mathsf{Tag}_k(x_i, i)) : x_i \in \mathbb{Z}\}_{1 \le i \le N}$, it is computationally hard to forge a tuple $(y, j, \mathsf{Tag}_k(y, j))$, under Assumption 1 and Assumption 2, and assuming that $f_{s_3}(\cdot)$ is PRF.*

The proof is given in Appendix A. Under Tag with key $k = (s_1, s_2, s_3, n, g_p, g_q, g_r, p, q, r)$, z is a valid MAC for (x, i), if

$$z^{qr} \equiv \left(g_p^{qr}\right)^{x+\frac{s_1}{h(i)}} \pmod n; \quad (1)$$

$$z^{pq} \equiv g_r^{pq\ f_{s_3}(i)} \pmod n. \quad (2)$$

Definition 4. Algorithm $\mathcal{G}(1^\kappa)$ is run to obtain $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$. Let $(s_{\ell,1}, s_{\ell,2}, s_{\ell,3}) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^* \times \{0,1\}^\kappa$ for each $1 \le \ell \le d$ and $k' = (s_{1,1}, s_{1,2}, s_{1,3}, s_{2,1}, s_{2,2}, s_{2,3}, \cdots, s_{d,1}, s_{d,2}, s_{d,3})$. Define function $\mathsf{TaG} : \mathbb{Z}^d \times [N]^d \to (\mathbb{Z}_n^*)^d$ with key $k = (k', n, g_p, g_q, g_r, p, q, r)$ as follows: For $\mathbf{x} = (x_1, x_2, \cdots, x_d) \in \mathbb{Z}^d$, $\mathbf{i} = (i_1, i_2, \cdots, i_d) \in [N]^d$,

$$\begin{aligned}
\mathsf{TaG}_k(\mathbf{x}, \mathbf{i}; \xi) \triangleq (&\mathsf{Tag}_{k_1}(x_1, i_1, \xi), \mathsf{Tag}_{k_2}(x_2, i_2, \xi), \\
&\cdots, \mathsf{Tag}_{k_d}(x_d, i_d, \xi))
\end{aligned}$$

where for each $\ell, 1 \le \ell \le d$, $k_\ell = (s_{\ell,1}, s_{\ell,2}, s_{\ell,3}, n, g_p, g_q, g_r, p, q, r)$, and the random coin $\xi \in \mathbb{Z}_q^*$.

Remark.

- For every d-dimensional point in $\mathbb{Z}^d$, the vector-valued function TaG defines a tag value consisting of d elements from $\mathbb{Z}_n^*$.
- The random number ξ binds together all d elements from $\mathbb{Z}_n^*$ generated using Tag.
- In the applications of TaG in this paper, we do not keep the values of ξ's. For simplicity, we may just write $\mathsf{TaG}(\mathbf{x}, \mathbf{i})$.

5. To "extract" the P component from a tag value t generated using Tag, one can just raise t to power qr, where q and r are from key k.

5. MAIA: Multidimensional Aggretate query Ingegrity Authentication

Recall that Section 1 gives an overview of our scheme. For illustration, we will first describe the scheme for $d = 2$ in Section 5.1. Then we will present the scheme for general case $d \ge 1$ in Section 5.2. We analyze the security of the scheme in Section 5.3 and complexities in Section 5.4.

The main reason that our scheme achieves efficient communication complexity, is that Alice can compress the *Help-Info* and Bob can uncompress the compressed data to generate a proof. The Compress and Uncompress algorithms are described as follows:

Compress and Uncompress. Johnson *et al.* [39] (in Section 5 "Set Homomorphic Signatures" of that paper) proposed a redactable signature scheme, based on the signature scheme in Gennaro *et al.* [37]. We wrap this redactable signature scheme as two algorithms Compress and Uncompress: Let (n, g_p) be defined as in Definition 3, $pk \leftarrow n$; $sk \leftarrow (n, g_p, \phi(n))$, for a set $\mathcal{I}$ of integers,

- $\mathsf{Compress}_{sk}(\mathcal{I})$: Output $\sigma \leftarrow g_p^{\prod_{i \in \mathcal{I}} h(i)^{-1}} \mod n$.
- $\mathsf{Uncompress}_{pk}(\mathcal{I}, \sigma, \widetilde{\mathcal{I}})$: If $\widetilde{\mathcal{I}} \not\subseteq \mathcal{I}$, output $\perp$. Otherwise, output $\widetilde{\sigma}$: $\widetilde{\sigma} \leftarrow \sigma^{\prod_{i \in \mathcal{I} \setminus \widetilde{\mathcal{I}}} h(i)} \mod n$.

5.1. Illustration of MAIA over a 2D Dataset

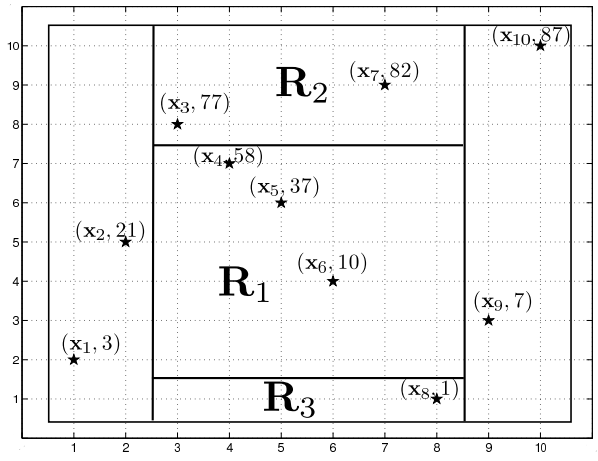
Let us consider the dataset shown in Table 3 and Figure 2. Table 3 and Figure 2 show a normalized (Section 3.1.1) dataset $\mathtt{D}$, consisting of 10 points $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{10}$. Let us consider the query that asks for the sum of the second attribute values of all points within the range $[3, 8] \times [2, 7]$ (i.e. the region $\mathbf{R}_1$ in Figure 2). Recall that this query is denoted with notation $\textsc{Sum}(\mathbf{R}_1, 2)$ (See Section 3.1.2).

5.1.1. Setup Phase. In the setup phase, Alice derives two sub-keys k_1 and k_2 from the private key k. For each data point $\mathbf{x}_m = (x_{m,1}, x_{m,2})$ with index $\mathbf{i}_m = (i_{m,1}, i_{m,2})$, Alice chooses a random number ξ, and generates a vector $(\mathsf{Tag}_{k_1}(x_{m,1}, i_{m,1}; \xi), \mathsf{Tag}_{k_2}(x_{m,2}, i_{m,2}; \xi))$, denoted as $\mathsf{T}_{\mathbf{x}_m}$. Then Alice sends the dataset $\mathtt{D}$ together with all $\mathsf{T}_{\mathbf{x}_m}$'s to Bob. Figure 3 shows the construction of setup for general case.

Table 3: The dataset $\mathtt{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{10}\}$. . For each $1 \leq m \leq 10$, we write $\mathbf{x}_m = (x_{m,1}, x_{m,2})$, and $(i_{m,1}, i_{m,2})$ is the index vector of $\mathbf{x}_m$, i.e. $i_{m,1}$ is the rank of $x_{m,1}$ among $\{x_{m',1} : 1 \leq m' \leq 10\}$. Similar for $i_{m,2}$ and $x_{m,2}$.

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$	$\mathbf{x}_7$	$\mathbf{x}_8$	$\mathbf{x}_9$	$\mathbf{x}_{10}$
$x_{m,1}$	5	7	11	24	31	45	58	61	83	97
$i_{m,1}$	1	2	3	4	5	6	7	8	9	10
$x_{m,2}$	3	21	77	58	37	10	82	1	7	87
$i_{m,2}$	2	5	8	7	6	4	9	1	3	10

Figure 2: The set D of 10 points $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{10}$ are displayed on the $[1, 10] \times [1, 10]$ grid, where $\mathbf{x}_m$'s, $1 \leq m \leq 10$, are as defined in Table 3. Each point $\mathbf{x}_m$ is labeled with $(\mathbf{x}_m, x_{m,2})$, and located at position $(i_{m,1}, i_{m,2})$. The 2D range $[3, 8] \times [2, 7]$ selects the region denoted by "$\mathbf{R}_1$".



Recall that the tag function Tag consists of three components $\mathsf{P}, \mathsf{Q}, \mathsf{R}$ (Section 4): Let dimension $j \in \{1, 2\}$,

$$\begin{aligned} \mathsf{Tag}_{k_j}(x, i; \xi) &= g_p^{x + \frac{s_{j,1}}{h(i)}} \; g_q^{-x + s_{j,2}\xi} \; g_r^{f_{s_{j,3}}(i)} \\ &= \mathsf{P}(x, s_{j,1}/h(i))\mathsf{Q}(x, s_{j,2}\xi)\mathsf{R}(f_{s_{j,3}}(i)). \end{aligned}$$

Here $s_{j,1}, s_{j,2}$ and $s_{j,3}$ are from the sub-key k_j, and P, Q and R are some homomorphic functions (See Section 4 for details). As mentioned in Section 4, with the private key, Alice can extract $\mathsf{P}(x, s_{j,1}/h(i))$, $\mathsf{Q}(x, s_{j,2}\xi)$, and $\mathsf{R}(f_{s_{j,3}}(i))$ from the value of $\mathsf{Tag}_{k_j}(x, i; \xi)$, using some function Υ_p, Υ_q and Υ_r respectively (Notice that in the costruction in Figure 4 and Figure 5, Alice just raises a value t, generated using Tag, up to a power qr to cancel out Q and R components and retain only P component). Without the private key, it is hard to break $\mathsf{Tag}_{k_j}(x, i; \xi)$ into P, Q and R components, by Assumption 2.

5.1.2. Query Phase. In a query session, Alice issues an aggregate range query to Bob, for example, the query $\textsc{Sum}(\mathbf{R}_1, 2)$: what is the sum of the attribute values (precisely $x_{m,2}$) of all points inside range $\mathbf{R}_1$. Bob should compute the query result $58 + 37 + 10 = 105$, and convince Alice that 105 is the correct answer. The two major challenges to Alice are: (1) How to prevent collusion attack, i.e. prevent Bob from utilizing *Help-Info* gathered from Alice in previous query sessions to cheat in the current session; (2) How to ensure completeness, i.e. prevent Bob from undercounting and double-counting (e.g. Bob may "remove" the point $\mathbf{x}_5$ from range $\mathbf{R}_1$, or "forge" a new point $\mathbf{x}'_6$ as a duplicate copy of $\mathbf{x}_6$). As mentioned in Section 1, Alice will use secret random nonces α's in a proper way to prevent collusion attack and check the complementary region to ensure completeness.

Recall $\mathbf{R}_1 = [3, 8] \times [2, 7]$ is a 2D range. Let $\mathbf{C} = [3, 8] \times [1, 10]$ be a 1D range. Suppose Alice somehow "knows" the correct result W to the query $\textsc{Sum}(\mathbf{C}, 2)$ and the valid proof Π for W. The mechanism to prevent collusion attack is captured in an algorithm called CollRes (stands for "Collusion Resistant"). Alice interacts with Bob to simulate CollRes with the query $\textsc{Sum}(\mathbf{R}_1, 2)$ as input, and gets results W_1 and a partial proof π_1. To ensure that 105 is computed from all points inside range $\mathbf{R}_1$, without duplicating or missing, Alice checks the complementary region $\mathbf{C} \setminus \mathbf{R}_1$, which is just the union of 2D ranges $\mathbf{R}_2$ and $\mathbf{R}_3$. Running CollRes on queries $\textsc{Sum}(\mathbf{R}_2, 2)$ and $\textsc{Sum}(\mathbf{R}_3, 2)$ separately, Alice obtains (W_2, π_2) and (W_3, π_3) from Bob respectively, where W_2 and W_3 are query results, and π_2 and π_3 are partial proofs. Alice checks (1) $W \stackrel{?}{=} W_1 + W_2 + W_3$; (2) whether (π_1, π_2, π_3) is consistant with Π. Here π_1 can be viewed as the first part of proof for the query $\textsc{Sum}(\mathbf{R}_1, 2)$, and (Π, π_2, π_3) can be viewed as the second part of proof, as mentioned in Section 1.

The whole interactive process between Alice and Bob is captured in an algorithm called ProVer (stands for "Prover-Verifier"). ProVer will call three instances of CollRes as subroutine on query $\textsc{Sum}(\mathbf{R}_1, 2)$, $\textsc{Sum}(\mathbf{R}_2, 2)$ and $\textsc{Sum}(\mathbf{R}_3, 2)$ respectively. Meanwhile, ProVer will recursively call itself on query $\textsc{Sum}(\mathbf{C}, 2)$.

Part I. We describe the interactive algorithm CollRes on a 2D aggregate range query $\textsc{Sum}(\mathbf{R}_1, 2)$ step by step as follows. The construction of CollRes for general case is given in Figure 4.

A1: Alice issues the query $\textsc{Sum}(\mathbf{R}_1,\ 2)$, and sends δ_1 and δ_2 to Bob, where δ_1 and δ_2 are computed as follows,
  a) Alice selects two independent random nonces α_1 and α_2, and computes the *Help-Info*
  $\varpi_1 = \{\mathsf{P}(0, 1/h(i))^{\alpha_1} : \quad i \in [3, 8]\}$ and
  $\varpi_2 = \{\mathsf{P}(0, 1/h(i))^{\alpha_2} : \quad i \in [2, 7]\}$.
  b) Alice compresses ϖ_1 and ϖ_2 using algorithm Compress (details in Step A1(b) in Figure 4), and randomizes the compressed data to obtain δ_1 and δ_2 (details in Step A1(c) in Figure 4).

B1: Bob generates the query result w_2, and the proof $(w_1, \psi_1, \psi_2, \widetilde{\psi}_1, \widetilde{\psi}_2)$, in the following way, and sends them to Alice.

a) Bob finds the set $S = \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$ of points in the range $\mathbf{R}_1$.

b) Bob uncompresses δ_1 and δ_2 to obtain ϖ_1 and ϖ_2, using algorithm Uncompress, and computes

$$w_j = \sum_{\mathbf{x}\in S} \mathbf{x}[j]; \quad \psi_j = \prod_{x\in S} \mathsf{T}_{\mathbf{x}}[j];$$
$$\widetilde{\psi}_j = \prod_{\mathbf{x}\in S}^{\mathbf{i}=\mathsf{Idx}(\mathbf{x})} \mathsf{P}\big(0, 1/h(\mathbf{i}[j])\big)^{\alpha_j}$$

Note that $\mathsf{T}_{\mathbf{x}}$ is the tag vector of $\mathbf{x}$ generated in the setup phase, $\mathsf{T}_{\mathbf{x}}[j]$ refers to the j-th component of the vector $\mathsf{T}_{\mathbf{x}}$, and $\mathbf{i} = \mathsf{Idx}(\mathbf{x})$ is the index of $\mathbf{x}$, as defined in Section 3.1.

A2: Alice checks the correctness of w_2 in two steps:

a) Alice extracts P components from ψ_j and $\widetilde{\psi}_j$ using private key, and checks the consistency between these two P components using the secret random nonce α_j: For dimension $j \in \{1, 2\}$,

$$\Upsilon_p(\psi_j^{\alpha_j}) \stackrel{?}{\equiv} \mathsf{P}(\alpha_j w_j, 0)\Upsilon_p(\widetilde{\psi}_j^{s_{j,1}}) \pmod n.$$

b) Alice verifies the consistency of the random variables ξ's between ψ_1 and ψ_2:

$$\frac{\Upsilon_q(\psi_1^{s_{2,2}})}{\mathsf{Q}(s_{2,2}w_1, 0)} \stackrel{?}{\equiv} \mathsf{Q}(0, s_{1,2}s_{2,2}\sum_{\mathbf{x}\in S}\xi_{\mathbf{x}}) \stackrel{?}{\equiv} \frac{\Upsilon_q(\psi_2^{s_{1,2}})}{\mathsf{Q}(s_{1,2}w_2, 0)} \pmod n.$$

Remark on Correctness.

1) Let $\Lambda_j = \sum_{\mathbf{x}\in S}^{\mathbf{i}=\mathsf{Idx}(\mathbf{x})} 1/h(\mathbf{i}[j])$ for $j \in \{1, 2\}$. Due to the homomorphism of P, one can derive the following equalities,

$$\begin{aligned}\Upsilon_p(\psi_j) &= \mathsf{P}\big(w_j, s_{j,1}\sum_{\mathbf{x}\in S}^{\mathbf{i}=\mathsf{Idx}(\mathbf{x})} 1/h(\mathbf{i}[j])\big)\\ &= \mathsf{P}(w_j, s_{j,1}\Lambda_j),\\ \Upsilon_p(\widetilde{\psi}_j) &= \mathsf{P}\big(0, \alpha_j\sum_{\mathbf{x}\in S}^{\mathbf{i}=\mathsf{Idx}(\mathbf{x})} 1/h(\mathbf{i}[j])\big)\\ &= \mathsf{P}(0, \alpha_j\Lambda_j).\end{aligned}$$

2) If w_j's, ψ_j's, and $\widetilde{\psi}_j$'s are computed honestly according to the protocol, we have

$$\begin{aligned}\Upsilon_p(\psi_j^{\alpha_j}) &= \mathsf{P}(\alpha_j w_j,\ \alpha_j s_{j,1}\Lambda_j)\\ &= \mathsf{P}(\alpha_j w_j,\ 0)\mathsf{P}(0,\ \alpha_j s_{j,1}\Lambda_j)\\ &= \mathsf{P}(\alpha_j w_j,\ 0)\Upsilon_p(\widetilde{\psi}_j^{s_{j,1}}) \pmod n.\end{aligned}$$

Hence, the equivalence in Step A2 (a) holds.

Remark on Soundness.

1) Bob is not able to forge a valid tag value w.r.t. Tag due to Lemma 1. Bob could not compute $\widetilde{\psi}_j$'s neither, without *Help-Info* from Alice.
2) If Bob intends to include the point of index (i_1, i_2) outside the range $\mathbf{R}_1$ to the answer (W.L.O.G, assume $i_1 \notin [a_1, b_1]$), he cannot forge $\mathsf{P}(0, 1/h(i_1))^{\alpha_1}$ without knowing α_1, so the proof he generated cannot pass the verification in Step A2(a) described above.
3) If Bob intends to combine the first dimenison of the point $\mathbf{x}_7$ and the second dimension of the point $\mathbf{x}_2$ to forge a point inside the range $\mathbf{R}_1$, the proof he generated cannot pass the verification in Step A2(b), because $\mathbf{x}_2$ and $\mathbf{x}_7$ have different secret binding variables ξ's.
4) Bob could not take use of *Help-Info* recevied from Alice in previous sessions, to cheat in the processing of the current query, because α_j's are secret random nonces. Namely, collusion attack can be prevented.

Part II. We describe the interactive algorithm ProVer on a 2D aggregate range query $\textsc{Sum}(\mathbf{R}_1, 2)$ step by step as follows. The construction of ProVer for general case is given in Figure 6.

A1: Alice chooses three ranges $\mathbf{R}_2$, $\mathbf{R}_3$, and $\mathbf{C}$, based on range $\mathbf{R}_1$, where $(\mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3)$ is a partition of 1D range $\mathbf{C}$.

B1: Bob interacts with Alice to simulate CollRes on queries $\textsc{Sum}(\mathbf{R}_1, 2)$, $\textsc{Sum}(\mathbf{R}_2, 2)$ and $\textsc{Sum}(\mathbf{R}_3, 2)$, and simulate ProVer on query $\textsc{Sum}(\mathbf{C}, 2)$.

A2: (1) Alice interacts with Bob to simulate CollRes on queries $\textsc{Sum}(\mathbf{R}_1, 2)$, $\textsc{Sum}(\mathbf{R}_2, 2)$ and $\textsc{Sum}(\mathbf{R}_3, 2)$, and receives their answers $(W_1, \psi_{1,1}, \psi_{1,2})$, $(W_2, \psi_{2,1}, \psi_{2,2})$, $(W_3, \psi_{3,1}, \psi_{3,2})$, respectively. Recursively, Alice interacts with Bob to simulate ProVer on query $\textsc{Sum}(\mathbf{C}, 2)$, and receives answer $(W_4, \psi_{4,1})$.

(2) Assuming the answer $(W_4, \psi_{4,1})$ is correct, Alice verifies the completeness of the answer to $\textsc{Sum}(\mathbf{R}_1, 2)$, by checking the consistency among W's and among pseduorandom $f_{s_{j,3}}(i)$'s:

$$\begin{aligned}W_1 + W_2 + W_3 &= W_4\\ \Upsilon_r(\psi_{1,1})\Upsilon_r(\psi_{2,1})\Upsilon_r(\psi_{3,1}) &\equiv \Upsilon_r(\psi_{4,1}) \pmod n\end{aligned}$$

Note that $\Upsilon_r(\psi_{1,1})$ is an "encrypted" form of the sum of $f_{s_{j,3}}(i)$'s for range $\mathbf{R}_1$.

Remark.

1) The above procedure can prevent undercounting and double-counting attacks based on the assumption that we have a solution to handle 1D case. For 1D aggregate query $\textsc{Sum}(\mathbf{C}, 2)$ (Recall that $\mathbf{C} = [3, 8] \times [1, 10]$), Alice has full information of $\mathcal{I} = \{\mathbf{i}[1] : \mathbf{i} \in \mathbf{C} \wedge$ *there is some data point at location* $\mathbf{i}\} = [3, 8]$,

so she can reconstruct $\{f_{s_{1,3}}(i) : i \in \mathcal{I}\}$, and verifies the answer $(W_4, \psi_{4,1})$ to query $\text{SUM}(\mathbf{C}, 2)$ by checking $f_{s_{1,3}}(i)$ directly:

$$\Upsilon_r(\psi_{4,1}) \equiv g_r^{\sum_{i\in\mathcal{I}} f_{s_{1,3}}(i)} \pmod{n}.$$

2) For 1D aggregate query, the set (e.g. $\mathcal{I}$ as in the above example) of indices of points in the query range, forms a sequence of consecutive integers, so it can be expressed efficiently with two boundary integers (e.g. $[3, 8]$ represents $\mathcal{I}$). This is the reason that our scheme achieves constant communication complexity in 1D case.
3) For higher dimensional aggregate query, the set (e.g. $\mathcal{I} = \{(\mathbf{i}[1], \mathbf{i}[2]) : \mathbf{i} \in \mathbf{R}_1 \wedge$ *there is some data point at location* $\mathbf{i}\}$ for query $\text{SUM}(\mathbf{R}_1, 2)$) of indices of points in the query range, cannot be expressed compactly as in 1D case (Note that, in general, $\mathcal{I} \subsetneq [3, 8] \times [2, 7]$). So Alice switches to verify the P component of Tag to audit $1/h(i)$'s. Alice comuptes $\varpi_j = \{\mathsf{P}(0, 1/h(i))^{\alpha_j} : i \in [a_j, b_j]\}$ embeding a random once α_j, and compress ϖ_j before sending it to Bob. From comprssed ϖ_j, Bob is able to recover ϖ_j, and to construct $\widetilde{\psi_j}$'s as a part of the proof. Due to the compression of ϖ_j's, our scheme can achieve efficient communication complexity (i.e. $O(\ell^2)$ for ℓ-dimensional aggregate query. See Section 5.4).

5.2. The Main Construction: Generalize to high dimension

In this subsection, we formally describe the $\mathcal{RC}$ protocol $\mathsf{MAIA} = (\mathsf{KGen}, \mathsf{DEnc}, \langle \mathsf{Eval}, \mathsf{Ext} \rangle)$ (See Section 3.2 for $\mathcal{RC}$ protocol) for multidimensional aggregate range query. The key generating algorithm KGen and data encoding algorithm DEnc are given in Figure 3. Note that for convenience of presentation, we call $\langle \mathsf{Eval}, \mathsf{Ext} \rangle$ of MAIA as ProVer. In Figure 5, we present the interactive proof, denoted as ProVer^1, for 1D aggregate range query. In Figure 6, we present the interactive proof, denoted as ProVer^ℓ, for ℓ-dimensional ($2 \le \ell \le d$) aggregate range query. ProVer^ℓ will call a subroutine $\mathsf{CollRes}^\ell$ (for ℓ-dimensional query), which is given in Figure 4. In the output $(w_0, \psi_{0,1}, \psi_{0,2}, \cdots, \psi_{0,\ell}, \psi_{0,\iota})$ of ProVer^ℓ, w_0 is the result to the query and the rest is the proof.

5.3. Security Analysis

In previous subsection, we have described our authentication scheme for multidimensional aggreagte query. That is: (1) MAIA^1, the solution to 1D case, consists of two algorithms KGen and DEnc in Figure 3 for setup phase, and an interactive algorithm $\langle \mathsf{Eval}, \mathsf{Ext} \rangle$ (namely ProVer^1) in Figure 5 for query phase; (2) MAIA^ℓ, the solution to ℓ-dimension case, consists of two algorithms KGen and DEnc in Figure 3 for setup phase, and an interactive algorithm $\langle \mathsf{Eval}, \mathsf{Ext} \rangle$ (namely ProVer^ℓ) in Figure 6 for query phase. Recall that the SUM query is defined in Section 3.1.

Figure 3: Setup phase of MAIA for dataset $\mathtt{D}$.

(Alice) $\mathsf{KGen}(1^\kappa)$. Randomly choose key k as in Definition 4. Output k.

(Alice) $\mathsf{DEnc}(\mathtt{D}; k)$.

1) Alice normalizes the dataset $\mathtt{D}$ to obtain an index $\mathsf{Idx}(\mathbf{x}) \in [N]^d$ for each point $\mathbf{x} \in \mathtt{D}$ (See Section 3.1.1 for "normalize" and Idx), where N is the number of points in $\mathtt{D}$.
2) For each $\mathbf{x} \in \mathtt{D}$, Alice randomly chooses ξ from $\mathbb{Z}_q^*$, and create a tag value: $\mathsf{T}_\mathbf{x} \leftarrow \mathsf{TaG}_k(\mathbf{x}, \mathsf{Idx}(\mathbf{x}); \xi)$.
3) Alice sends $\mathtt{D}$ and $\{\mathsf{T}_\mathbf{x} : \mathbf{x} \in \mathtt{D}\}$ to Bob, and removes their local copy.

Theorem 2. MAIA^1 *is a $\mathcal{PRC}$ protocol (as defined in Definition 2) w.r.t. 1D* SUM *query (as defined in Section 3.1), under Assumption 1 and Assumption 2, and assuming that $f_s(\cdot)$ is PRF.*

Lemma 3. *Under Assumption 1 and Assumption 2, and assuming that $f_s(\cdot)$ is a PRF,* MAIA^ℓ *is a $\mathcal{PRC}$ protocol w.r.t. ℓ-dimensional aggregate* SUM *query , if* $\mathsf{MAIA}^{\ell-1}$ *is a $\mathcal{PRC}$ protocol w.r.t. $(\ell-1)$-dimensional aggregate* SUM *query, where $2 \le \ell \le d$.*

The proof of Theorem 2 is given in Appendix B, and a sketch of proof of Lemma 3 is given in Appendix C. From Theorem 2 and Lemma 3, we conclude the following Theorem 4 as our main theorem.

Theorem 4. *For any $\ell, 1 \le \ell \le d$,* MAIA^ℓ *is a $\mathcal{PRC}$ protocl, w.r.t. ℓ-dimensional aggregate* SUM *query, under Assumption 1 and Assumption 2, and assuming that $f_s(\cdot)$ is a PRF.*

5.4. Complexity Analysis

The communication complexities of $\mathsf{CollRes}^\ell$ and ProVer^ℓ are only dependent on ℓ, regardless of the query range or the size of dataset $\mathtt{D}$. The communication cost of $\mathsf{CollRes}^\ell$ is $O(\ell)$. Denote the communication cost of ProVer^ℓ as $C(\ell)$. ProVer is a recursive algorithm. We have recurrence: $C(1) = O(1)$; $\quad C(\ell) = C(\ell-1) + 3O(\ell)$. Hence $C(\ell) = O(\ell^2)$. More precisely, $C(\ell)$ is about $2.5\ell^2\kappa$ bits.

It is straightforward to see that the round complexity of ProVer is $O(\ell)$. We can reduce it to $O(1)$, by running

Figure 4: $\mathsf{CollRes}^{\ell}$: A subroutine of ProVer. Processing query $\textsc{Sum}(\mathbf{B}, \iota)$, where $\iota \in [d]$, $1 \le \ell \le d$, and $\mathbf{B} = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_\ell, b_\ell] \times [N]^{d-\ell}$.

Alice has key k; Bob has dataset D and tags $\{\mathsf{T}_{\mathbf{x}} : \mathbf{x} \in \mathtt{D}\}$.

A1: Alice computes the *Help-Info* ϖ_j's and randomizes them to obtain δ_j's, in the following way. Then Alice sends $(a_1, a_2, \cdots, a_\ell; b_1, b_2, \cdots, b_\ell; \iota)$ and $(\delta_1, \delta_2, \cdots, \delta_\ell, \delta_\iota)$ to Bob.

a) If $\iota \notin [\ell]$, let $a_\iota = 1$ and $b_\iota = N$.

b) Let $sk = (n, g_p, \phi(n))$. For each $j \in [\ell] \cup \{\iota\}$, $\varpi_j \leftarrow \mathsf{Compress}_{sk}([a_j, b_j])$.

c) For each $j \in [\ell] \cup \{\iota\}$, $\alpha_j \xleftarrow{\$} \mathbb{Z}_p^*$, $\xi \xleftarrow{\$} \mathbb{Z}_q^*$, $\lambda \xleftarrow{\$} \mathbb{Z}_r^*$, $\delta_j \leftarrow \varpi_j^{\alpha_j} g_q^{\xi} g_r^{\lambda} \mod n$.

B1: Bob generates the query result w_ι and the proof $(w_1, w_2, \cdots, w_\ell; \psi_1, \psi_2, \cdots, \psi_\ell, \psi_\iota; \widetilde{\psi}_1, \cdots, \widetilde{\psi}_\ell, \widetilde{\psi}_\iota)$ in the following way, and sends them to Alice.

a) Let the set $S = \{\mathbf{x} \in \mathtt{D} : \mathsf{Idx}(\mathbf{x}) \in \mathbf{B}\}$ and $S_I = \{\mathsf{Idx}(\mathbf{x}) : \mathbf{x} \in S\}$.

b) For each $\mathbf{i} \in S_I$, $j \in [\ell] \cup \{\iota\}$, $u_{\mathbf{i},j} \leftarrow \mathsf{Uncompress}([a_j, b_j], \delta_j, \mathbf{i}[j])$.

c) For $j \in [\ell] \cup \{\iota\}$,
$w_j \leftarrow \sum_{\mathbf{x} \in S} \mathbf{x}[j]$;
$\psi_j \leftarrow \prod_{\mathbf{x} \in S} \mathsf{T}_{\mathbf{x}}[j] \mod n$;
$\widetilde{\psi}_j \leftarrow \prod_{\mathbf{i} \in S_I} u_{\mathbf{i},j} \mod n$.

Note that $S_I \subseteq \mathbf{B}$. Recall that $\mathsf{T}_{\mathbf{x}}$ denotes the tag vector of $\mathbf{x}$ generated in Figure 3, and $\mathsf{T}_{\mathbf{x}}[1]$ denotes the 1st dimension of the vector $\mathsf{T}_{\mathbf{x}}$.

A2: Alice accepts Bob's reply and outputs $(w_\iota, \psi_1, \psi_2, \cdots, \psi_\ell, \psi_\iota)$, if the following $(2\ell + 1)$ equations hold:

$$\forall j \in [\ell] \cup \{\iota\},$$
$$\left(\frac{\psi_j}{g_p^{w_j}}\right)^{qr\alpha_j} \equiv \widetilde{\psi}_j^{qrs_{j,1}} \pmod{n}; \quad (3)$$
$$\forall j \in [\ell],$$
$$\left(\psi_j g_q^{w_j}\right)^{prs_{\iota,2}} \equiv \left(\psi_\iota g_q^{w_\iota}\right)^{prs_{j,2}} \pmod{n}. \quad (4)$$

Otherwize reject it and output $\perp$.

all recursive calls to $\mathsf{ProVer}^{\ell-1}, \mathsf{ProVer}^{\ell-2}, \cdots, \mathsf{ProVer}^{1}$, simultaneously and in parallel.

The storage overhead on Bob's side, i.e. the total size of all tags, is $O(dN)$: dN elements from $\mathbb{Z}_n^*$. The storage overhead on Alice's side, i.e. the key size, is $O(d)$: $(3d+7)$ elements from $\{0,1\}^{\le \kappa}$. The computation complexity per query on Bob's side is $O(\ell N^2)$ (modular exponentiations) for ℓ-dimensional aggregate range query. The dominant computation step is Uncompress in Step B1(b) in Figure 4. The computation complexity per query on Alice's side is $O(\ell N)$ (modular multiplications). The dominant computation step is Compress in Step A1(b) in Figure 4. We can reduce the computation cost on Alice's side to $O(\ell \log N)$ (modular multiplications) at the cost of additional $O(N)$ storage on Bob's side, and reduce the computation cost on Bob's side to $O(\ell N \log N)$ (modular exponentiations; the constant factor behind big-O notation is about 1 and the base of log is 2), with $O(N)$ temporary storage. We save the details. The summary of complexities of MAIA compared with related schemes is given in Figure 1 in Section 1.

Figure 5: ProVer^{1}: The query phase of MAIA^{1}. Processing query $\textsc{Sum}(\mathbf{B}, \iota)$, where $\iota \in [d]$ and $\mathbf{B} = [a_1, b_1] \times [N]^{d-1}$.

Alice has key k and runs algorithm Ext; Bob has dataset D and tags $\{\mathsf{T}_{\mathbf{x}} : \mathbf{x} \in \mathtt{D}\}$ and runs algorithm Eval.

B1: Bob interacts with Alice to simulate $\mathsf{CollRes}^{1}$ on query $\textsc{Sum}(\mathbf{B}, \iota)$.

A1: a) Alice interacts with Bob to simulate $\mathsf{CollRes}^{1}$ on query $\textsc{Sum}(\mathbf{B}, \iota)$. If rejected, then Alice rejects the current interaction and output $\perp$. Otherwise, let $(w_1, w_\iota, \psi_1, \psi_\iota, \widetilde{\psi}_1, \widetilde{\psi}_\iota)$ be the message that Alice received from Bob in Step A2 of $\mathsf{CollRes}^{1}$ in Figure 4.

b) Alice accepts Bob's reply and output $(w_\iota, \psi_1, \psi_\iota)$, if all of the following equations hold. Otherwise, rejcet it and output $\perp$. Let $\mathcal{I} = [a_1, b_1]$; all equations are modulo n;

$$\psi_1^{pq} \equiv g_r^{pq\left(\sum_{i \in \mathcal{I}} f_{s_{1,3}}(i) \mod r\right)}; \quad (5)$$
$$\psi_1^{qr} \equiv \left(g_p^{qr}\right)^{w_1 + \sum_{i \in \mathcal{I}} \frac{s_{1,1}}{h(i)} \mod p}. \quad (6)$$

Note that the computation time for decryption in MRQED [1] is $O(\log^d N)$ (bilinear map group operations). When $d > \log(dN)/\log\log N$, we have $dN \log N \le \log^d N$, not to mention that MRQED uses bilinear map operations and MAIA uses only modular exponentiations and multiplications.

6. Extension and Performance of MAIA

In this section, we extend MAIA to support: (1) range query over attributes; (2) COUNT, MIN and MAX queries; (3) range selection query. After that, we discuss some speedup methods, and show a way to support multiple queriers with private key version of MAIA.

6.1. Extension of MAIA

Figure 6: ProVer^{ℓ}: The query phase of MAIA^{ℓ}. Processing query $\textsc{Sum}(\mathbf{B}, \iota)$, where $\iota \in [d]$, $2 \le \ell \le d$, and $\mathbf{B} = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_\ell, b_\ell] \times [N]^{d-\ell}$.

Alice has key k and runs algorithm Ext; Bob has dataset $\mathtt{D}$ and tags $\{\mathsf{T}_{\mathbf{x}} : \mathbf{x} \in \mathtt{D}\}$ and runs algorithm Eval.

A1: Alice chooses three ranges $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$, where
$\mathbf{B}_1 = [a_1, b_1] \times \cdots \times [a_{\ell-1}, b_{\ell-1}] \times [1, a_\ell - 1] \times [N]^{d-\ell}$,
$\mathbf{B}_2 = [a_1, b_1] \times \cdots \times [a_{\ell-1}, b_{\ell-1}] \times [b_\ell + 1, N] \times [N]^{d-\ell}$,
$\mathbf{B}_3 = [a_1, b_1] \times \cdots \times [a_{\ell-1}, b_{\ell-1}] \times [N]^{d-\ell+1}$.
Note that $(\mathbf{B}, \mathbf{B}_1, \mathbf{B}_2)$ is a partition of $\mathbf{B}_3$.

B1: Bob
- a) Interacts with Alice to simulate $\mathsf{CollRes}^{\ell}$ on query $\textsc{Sum}(\mathbf{B}, \iota)$, $\textsc{Sum}(\mathbf{B}_1, \iota)$, and $\textsc{Sum}(\mathbf{B}_2, \iota)$, sequentially.
- b) Interacts with Alice to simulate $\mathsf{ProVer}^{\ell-1}$ on query $\textsc{Sum}(\mathbf{B}_3, \iota)$.

A2: Alice

a) Interacts with Bob to simulate $\mathsf{CollRes}^{\ell}$ on query $\textsc{Sum}(\mathbf{B}, \iota)$, $\textsc{Sum}(\mathbf{B}_1, \iota)$, and $\textsc{Sum}(\mathbf{B}_2, \iota)$, sequentially. If any of the above three queries is rejected, then reject the current query and output $\perp$. Otherwise denote the three outputs as $(W_0, \psi_{0,1}, \psi_{0,2}, \cdots, \psi_{0,\ell}, \psi_{0,\iota})$, $(W_1, \psi_{1,1}, \psi_{1,2}, \cdots, \psi_{1,\ell}, \psi_{1,\iota})$, and $(W_2, \psi_{2,1}, \psi_{2,2}, \cdots, \psi_{2,\ell}, \psi_{2,\iota})$.

b) Interacts with Bob to simulate $\mathsf{ProVer}^{\ell-1}$ on query $\textsc{Sum}(\mathbf{B}_3, \iota)$. If it is rejected, then reject the current query and output $\perp$. Otherwise denote the output as $(W_3, \psi_{3,1}, \psi_{3,2}, \cdots, \psi_{3,\ell-1}, \psi_{3,\iota})$.

c) Accepts Bob's reply and output $(W_0, \psi_{0,1}, \psi_{0,2}, \cdots, \psi_{0,\ell}, \psi_{0,\iota})$, if the following equations hold,

$$W_0 + W_1 + W_2 = W_3 \quad (7)$$
$$\forall \vartheta \in [\ell - 1] \cup \{\iota\},$$
$$(\psi_{0,\vartheta}\psi_{1,\vartheta}\psi_{2,\vartheta})^{pq} \equiv (\psi_{3,\vartheta})^{pq} \pmod{n}. \quad (8)$$

Otherwise, reject and output $\perp$.

6.1.1. Conversion from Attribute to Index. As we have mensioned previously, our scheme can handle the more realistic case: aggregate over attributes conditional on range selection over attributes, e.g. the query in Figure 1. Our strategy is that, Alice translates attribute values to index values online with the help of Bob, and Alice is able to verify the correctness of the translation.

Suppose Alice's query is $\textsc{Sum}(\widehat{\mathbf{B}}, \iota)$, i.e. what is the sum of attribute values along the ι-th dimension of all points $\mathbf{x} \in \widehat{\mathbf{B}}$, where $\widehat{\mathbf{B}} = [\hat{y}_1, \hat{z}_1] \times [\hat{y}_2, \hat{z}_2] \times \cdots \times [\hat{y}_\ell, \hat{z}_\ell] \times \mathbb{Z}^{d-\ell}$, $\ell \ge 2$, and for each $1 \le j \le \ell, \hat{y}_j, \hat{z}_j \in \mathbb{Z}$. Alice will have one additional round of communication (Step A0 and B0) with Bob just before Alice's first step A1 and Bob's first step B1 in Figure 4. In Step A0, Alice sends $\{(\hat{y}_j, \hat{z}_j) : 1 \le j \le \ell\}$ to Bob. Next, in Step B0, Bob sends back the translated index and its proof $\{(a_j, y_j, t_{a,j}) : 1 \le j \le \ell\}$ and $\{(b_j, z_j, t_{b,j}) : 1 \le j \le \ell\}$. Alice accepts the translation, if for each $j, 1 \le j \le \ell$, (1) $y_j \ge \hat{y}_j$ and $z_j \le \hat{z}_j$; (2) $t_{a,j}$ and $t_{b,j}$ are valid MACs of (y_j, a_j) and (z_j, b_j), respectively, under Tag w.r.t. key k_j (See Eq 1 and Eq 2). As a result, the range $\widehat{\mathbf{B}}$ over attributes is translated into range $\mathbf{B} = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_\ell, b_\ell] \times [N]^{d-\ell}$ over indices.

Particularly in case $\ell = 1$, Bob has to send back $\Pi = \{(a_l, y_l, t_{a,l}), (a_r, y_r, t_{a,r}), (b_l, z_l, t_{b,l}), (b_r, z_r, t_{b,r})\}$ for $(\hat{y}, \hat{z})$. Alice accepts the translation, if[6] (1) $y_l \le \hat{y} \le y_r$ and $z_l \le \hat{z} \le z_r$; (2) every 3-tuple in Π is a valid data-MAC pair under Tag w.r.t. key k_1. In this way, range $[\hat{y}, \hat{z}]$ over attribute values is translated into range $[a_r, b_l]$ over indices.

Note that in 1D case, Alice has to check the two side boundaries, while in higher dimensional case, Alice just checks one side boundary.

6.1.2. Extend to COUNT, MIN Query. COUNT is just a special case of SUM: summing a constant attribute. In the setup, Alice attaches a new constant attribute value 1 to each point $\mathbf{x} \in \mathtt{D}$. As a result, the dimension of each point $\mathbf{x}$ will increase by 1. Then Alice performs the SUM query for this constant attribute using MAIA over the resulting dataset.

Furthermore, MIN and MAX query can be converted to COUNT query. The conversion is based on this property: For any set S of numbers, $c = \min S \Leftrightarrow c \in S \wedge |S| = |\{x : x \in S \wedge x \ge c\}|$. Suppose Alice asks Bob for the minimum attribute value along the ι-th dimension of points within range $\widehat{\mathbf{B}}$ (as given in Section 6.1.1) . Bob returns some value c, with the proof of $c \in S$, where S is the set of attribute values along the ι-th dimension of all points within range $\widehat{\mathbf{B}}$, i.e. $S = \{\mathbf{x}[\iota] : \mathbf{x} \in \widehat{\mathbf{B}} \cap \mathtt{D}\}$. Then Alice issues two COUNT queries to Bob: (1) $\textsc{Count}(\widehat{\mathbf{B}})$, i.e. the size of set S; (2) $\textsc{Count}\left(\widehat{\mathbf{B}} \bigcap \left(\mathbb{Z}^{\iota-1} \times [c, +\infty) \times \mathbb{Z}^{d-\iota}\right)\right)$, i.e. the size of set $\{x : x \in S \wedge x \ge c\}$. Bob is expected to return the two count numbers with proofs following the (extended) MAIA protocol. Alice believes c is the minimum value if all proofs are valid and the two count nubmers are equal.

Similarly, MAX query can be authenticated. So we conclude that: MAIA can handle all of SUM, COUNT, MIN, MAX aggregate query.

Corollary 5. *For any $\ell, 1 \le \ell \le d$, MAIA^{ℓ} (with slight modifications) is a $\mathcal{PRC}$ (Provable Remote Computing) protocol*

6. We omit the two extreme cases: (1) $\hat{y}$ is too small, e.g. $\hat{y} < 0$; (2) $\hat{z}$ is too large, e.g. $\hat{z} = +\infty$.

(as defined in Definition 2) w.r.t. ℓ-dimensional aggregate SUM, COUNT, MIN, *or* MAX *Query, under Assumption 1 and Assumption 2, and assuming that $f_s(\cdot)$ is a PRF.*

6.1.3. Multidimensional Range Selection Query. Besides aggregate range query, our scheme (with slight modifications) can also authenticate multidimensional range selection query, by authenticating a weighted sum of selected attribute values.

Suppose Alice wants to select the ι-th dimension of all points within range $\widehat{\mathbf{B}}$. She does it in three steps: (1) Issue the range selection query to Bob, and Bob returns a set $\{\mathbf{x}[\iota] : \mathbf{x} \in S\}$ of attribute values without any proof, where S denotes the set of points within range $\widehat{\mathbf{B}}$; (2) Issue query $\textsc{Count}(\widehat{\mathbf{B}})$ to Bob using the extended MAIA; (3) Simulate (modified) CollRes with Bob to authenticate a weighted sum query with range $\widehat{\mathbf{B}}$. More precisely, Alice asks for the weighted sum $w_j \leftarrow \sum_{\mathbf{x}\in S} \mathbf{x}[j]\ \mathcal{H}(\mathbf{x}[\iota])$ with hash of attribute as weights, where $\mathcal{H}$ is some collision-resistant hash function. So in Bob's first step B1 in Figure 4, the sum $w_j \leftarrow \sum_{\mathbf{x}\in S} \mathbf{x}[j]$ is replaced by $w_j \leftarrow \sum_{\mathbf{x}\in S} \mathbf{x}[j]\ \mathcal{H}(\mathbf{x}[\iota])$, and the computations of ψ_j's and $\widetilde{\psi}_j$'s are modified correspondingly.

Assume all (legitimate) attribute values are distinct. From (3), Alice can verify the correctness of the result set returned in (1). With the help of the count number from (2), Alice can ensure the completeness of the range selection query.

6.2. Performance of MAIA

6.2.1. Tightest Bounding Box. The tightest ℓ-dimensional bounding box for a range $\mathbf{B} = [a_1, b_1] \times \cdots \times [a_\ell, b_\ell] \times [N]^{d-\ell}$, is the minimum ℓ-dimensional rectangular range $\bar{\mathbf{B}}$ that satisfies the following conditions: (1) $\bar{\mathbf{B}} \subseteq \mathbf{B}$; (2) $\textsc{Count}(\mathbf{B}) = \textsc{Count}(\bar{\mathbf{B}})$. For example, in Figure 2, the tightest 2D bounding box for range $\mathbf{R}_1$ is $[4, 6] \times [4, 7]$.

We can find the tightest bounding box for range $\mathbf{B}$ in this way: Let $S_I = \{\mathbf{i} \in \mathbf{B} : \exists \mathbf{x} \in \mathtt{D}, \mathsf{Idx}(\mathbf{x}) = \mathbf{i}\}$. For each $1 \le j \le d$, let $\bar{a}_j = \min_{\mathbf{i}\in S_I} \mathbf{i}[j]$ and $\bar{b}_j = \max_{\mathbf{i}\in S_I} \mathbf{i}[j]$. Then the tightest ($\ell$-dimensional) bounding box is $\bar{\mathbf{B}} = [\bar{a}_1, \bar{b}_1] \times [\bar{a}_2, \bar{b}_2] \times \cdots \times [\bar{a}_\ell, \bar{b}_\ell] \times [N]^{d-\ell}$. We can replace a_ι, b_ι in Step A1(a) of CollRes in Figure 4 with $\bar{a}_\iota, \bar{b}_\iota$.

In certain scenarioes, $\bar{\mathbf{B}} \subsetneq \mathbf{B}$. Alice can ask Bob for the tightest bounding box $\bar{\mathbf{B}}$ for a query range $\mathbf{B}$ and just generate *Help-Info* for $\bar{\mathbf{B}}$, instead of $\mathbf{B}$ itself. In the example dataset in Figure 2, the (uncompressed) *Help-Info* for range $\mathbf{R}_1$, i.e. $[3, 8]\times[2, 7]$, consists of $(8-3+1)+(7-2+1) = 12$ elements from $\mathbb{Z}_n^*$. In contrast, the (uncompressed) *Help-Info* for the tightest bounding box $[4, 6] \times [4, 7]$, consists of only $(6 - 4 + 1) + (7 - 4 + 1) = 7$ elements. Note that the running time of Compress (in Step A1(b) in Figure 4) and Uncompress (in Step B1(b) in Figure 4) w.r.t. range $\bar{\mathbf{B}}$ is proportional to the sum of widths of the query interval along each dimension $j \in [\ell] \cup \{\iota\}$, i.e. $\sum_{j\in[\ell]\cup\{\iota\}} (\bar{b}_j - \bar{a}_j + 1)$.

Furthermore, Alice may sort the first ℓ dimensions in decreasing order according to the width of the query interval, and process each dimension in this order. That is, Alice will process dimension j before dimension j', if $(\bar{b}_j - \bar{a}_j + 1) > (\bar{b}_{j'} - \bar{a}_{j'} + 1)$, $j, j' \in [\ell]$.

6.2.2. Mitigation of Computation Cost and Support of Multiple Queriers. In actual application, one may employ our scheme MAIA in this way: The setup is as in Figure 3. In a query session, anyone, say Charlie, issues a query Q to Bob, and Bob should return a query result protected by Bob's signature, i.e. Bob returns (W, σ), where W is the result to query Q, and σ is Bob's signature on (Q, W). After receiving (W, σ), Charlie may decide whether (or when) to ask Alice to verify the correctness of W. If Alice receives a verification request from Charlie, she will run MAIA with query Q to verify the query result W by interacting with Bob. Using MAIA in this way, will not weaken the benefits of outsourcing significantly, because Alice just need run a lightweight verification server with very small storage (for the key and some temporary data, and the size of temporary data is in $O(d^2)$ where d is the dimension).

7. Conclusion

We proposed a scheme to authenticate aggregate range query over static multidimensional dataset, and the communication complexity (in term of bits) is sublinear and independent of the query range size and the dataset size. Aggregate operations we considered in this paper include summing, counting, and finding of the minimum and maximum. Our scheme and techniques can be useful in various other applications. In particular, we showed that our scheme with slight modifications can authenticate multidimensional range selection query, and improved the communication overhead significantly compared with previous works. How to convert our scheme to the public key setting using bilinear map, remains an open problem.

References

[1] Shi, E., Bethencourt, J., Chan, T.H.H., Song, D., Perrig, A.: Multi-dimensional range query over encrypted data. In: SP '07: IEEE Symposium on Security and Privacy. (2007) 350–364

[2] Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: SP '07: IEEE Symposium on Security and Privacy. (2007) 321–334

[3] Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: CCS '06: ACM conference on Computer and communications security. (2006) 89–98

[4] Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: TCC '07: Theory of Cryptography. (2007) 535–554

[5] Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: EUROCRYPT '08: International Conference on Advances in Cryptology. (2008) 146–162

[6] Shi, E., Waters, B.: Delegating capabilities in predicate encryption systems. In: ICALP '08: International colloquium on Automata, Languages and Programming. (2008) 560–578

[7] Brian Thompson, Danfeng Yao, S.H.W.G.H., Sander, T.: Privacy-preserving computation and verification of aggregate queries on outsourced databases. In: PETS '09: Privacy Enhancing Technologies Symposium (PETS). (2009)

[8] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2) (1978) 120–126

[9] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT '99: International Conference on Advances in Cryptology. (1999) 223–238

[10] Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: EUROCRYPT '09: International Conference on Advances in Cryptology. (2009) 224–241

[11] Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: CCS '07: ACM conference on Computer and communications security. (2007) 598–609

[12] Chang, E.C., Xu, J.: Remote integrity check with dishonest storage server. In: ESORICS '08: European Symposium on Research in Computer Security. (2008) 223–237

[13] Shacham, H., Waters, B.: Compact proofs of retrievability. In: ASIACRYPT '08: International Conference on the Theory and Application of Cryptology and Information Security. (2008) 90–107

[14] Bowers, K.D., Juels, A., Oprea, A.: Hail: A high-availability and integrity layer for cloud storage. Cryptology ePrint Archive, Report 2008/489 (2008) http://eprint.iacr.org/.

[15] Chen, H., Ma, X., Hsu, W.W., Li, N., Wang, Q.: Access control friendly query verification for outsourced data publishing. In: ESORICS '08: European Symposium on Research in Computer Security. (2008) 177–191

[16] Devanbu, P.T., Gertz, M., Martel, C.U., Stubblebine, S.G.: Authentic third-party data publication. In: IFIP '00: IFIP TC11/ WG11.3 Working Conference on Database Security. (2000) 101–112

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

[18] Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.G.: Authentic data publication over the internet. J. Comput. Secur. **11**(3) (2003) 291–314

[19] Pang, H., Jain, A., Ramamritham, K., Tan, K.L.: Verifying completeness of relational query results in data publishing. In: SIGMOD '05: ACM SIGMOD international conference on Management of data. (2005) 407–418

[20] Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. Trans. Storage **2**(2) (2006) 107–138

[21] Pang, H., Tan, K.L.: Verifying completeness of relational query answers from online servers. ACM Trans. Inf. Syst. Secur. **11**(2) (2008) 1–50

[22] Sion, R.: Query execution assurance for outsourced databases. In: VLDB '05: International conference on Very large data bases. (2005) 601–612

[23] Cheng, W., Tan, K.L.: Query assurance verification for outsourced multi-dimensional databases. J. Comput. Secur. **17**(1) (2009) 101–126

[24] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: SIGMOD '06: ACM SIGMOD international conference on Management of data. (2006) 121–132

[25] Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: VLDB '07: International conference on Very large data bases. (2007) 782–793

[26] Atallah, M.J., Cho, Y., Kundu, A.: Efficient data authentication in an environment of untrusted third-party distributors. In: ICDE '08: IEEE International Conference on Data Engineering. (2008) 696–704

[27] Yang, Y., Papadias, D., Papadopoulos, S., Kalnis, P.: Authenticated join processing in outsourced databases. In: SIGMOD '09: ACM SIGMOD international conference on Management of data. (2009) 5–18

[28] Mouratidis, K., Sacharidis, D., Pang, H.: Partially materialized digest scheme: an efficient verification method for outsourced databases. The VLDB Journal **18**(1) (2009) 363–381

[29] HweeHwa PANG, Jilian ZHANG, K.M.: Scalable verification for outsourced dynamic databases. In: VLDB '09: International Conference on Very Large Data Bases. (2009)

[30] Goodrich, M.T., Tamassia, R., Triandopoulos, N.: Super-efficient verification of dynamic outsourced databases. In: CT-RSA '08: The Cryptographers' Track at the RSA Conference. (2008) 407–424

[31] Haber, S., Horne, W., Sander, T., Yao, D.: Privacy-preserving verification of aggregate queries on outsourced databases. Technical report, HP Laboratories (2006) HPL-2006-128.

[32] Shamir, A.: How to share a secret. Commun. ACM **22**(11) (1979) 612–613

[33] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO '91: Advances in Cryptology. (1991) 129–140

[34] Preparata, F.P., Shamos, M.I.: Computational geometry: an introduction. (1985)

[35] Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: EUROCRYPT '98: International Conference on Advances in Cryptology. (1998) 308–318

[36] Gonzalez Nieto, J.M., Boyd, C., Dawson, E.: A public key cryptosystem based on a subgroup membership problem. Des. Codes Cryptography **36**(3) (2005) 301–316

[37] Gennaro, R., Halevi, S., Rabin, T.: Secure hash-and-sign signatures without the random oracle. In: EUROCRYPT '99: International Conference on Advances in Cryptology. (1999) 123–139

[38] Goldreich, O.: Foundations of Cryptography: Volume 1. (2006)

[39] Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In: CT-RSA '02: The Cryptographer's Track at the RSA Conference on Topics in Cryptology. (2002) 244–262

[40] Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptol. **17**(4) (2004) 297–319

[41] Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: SIGMOD '02: ACM SIGMOD international conference on Management of data. (2002) 216–227

[42] Hacigümüs, H., Iyer, B.R., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: DASFAA. (2004) 125–136

[43] Mykletun, E., Tsudik, G.: Aggregation queries in the database-as-a-service model. In: IFIP '06: IFIP WG 11.3 Working Conference on Data and Applications Security. (2006) 89–103

[44] Ge, T., Zdonik, S.B.: Answering aggregation queries in a secure system model. In: VLDB '07: International conference on Very large data bases. (2007) 519–530

Appendix A. Proof of Lemma 1

We define $\mathcal{P}$ and $\mathcal{Q}$ similar to $\mathcal{R}$. Define function $\mathcal{P} : G \rightarrow \mathbb{Z}_p$, such that for any $(x, y, z) \in \mathbb{Z}_p \times \mathbb{Z}_q \times \mathbb{Z}_r$, $\mathcal{Q}(g_p^x g_q^y g_r^z) \triangleq x$. Define function $\mathcal{Q} : G \rightarrow \mathbb{Z}_q$, such that for any $(x, y, z) \in \mathbb{Z}_p \times \mathbb{Z}_q \times \mathbb{Z}_r$, $\mathcal{Q}(g_p^x g_q^y g_r^z) \triangleq y$.

LEMMA 1. *Given $\{(x_i, i, \mathsf{Tag}_k(x_i, i)) : x_i \in \mathbb{Z}\}_{1 \leq i \leq N}$, it is computationally hard to forge a tuple $(y, j, \mathsf{Tag}_k(y, j))$, under Assumption 1 and Assumption 2, and assuming that $f_{s_3}(\cdot)$ is PRF.*

Proof: Suppose that some PPT adversary $\mathcal{A}$ takes as input a set $S = \{(x_i, i, t_i = \mathsf{Tag}_k(x_i, i)) : x_i \in \mathbb{Z}\}_{1 \leq i \leq N}$, and outputs $(x', i', t') \notin S$, where (x', i', t') satisfies Eq 1 and Eq 2 with non-negligible probability.

Assumption 2 holds for $\mathcal{R}$, since $f_{s_3}(\cdot)$ is PRF.

Case 1: $i' \in [N]$. Let $(x_{i'}, i', t_{i'}) \in S$ be the tuple with index $i = i'$ in the set S. Since $(x', i', t') \notin S$, we have $(x_{i'}, i', t_{i'}) \neq (x', i', t')$, i.e. $(x_{i'}, t_{i'}) \neq (x', t')$

Since both $(x_{i'}, i', t_{i'})$ and (x', i', t') satisfy Eq 2,

$$\mathcal{R}(t_{i'}) = \mathcal{R}(t')$$

Let $A = t_{i'}\ t'^{-1} \pmod n$. If $t_{i'} \neq t'$, with overwhelming high probability, $A \not\equiv 1 \pmod n$, and $B = \mathcal{R}(A) = 0$. The tupe (A, B) is a contradiction with Assumption 2. Otherwise, $t_{i'} = t'$ and $x_{i'} \neq x'$. Since both $(x_{i'}, i', t_{i'})$ and (x', i', t') satisfy Eq 1, we have

$$\begin{aligned} & \mathcal{P}(t_{i'}) = \mathcal{P}(t') \\ \Rightarrow\ & x_{i'} + \frac{s_1}{h(i')} \equiv x' + \frac{s_1}{h(i')} \pmod p \\ \Rightarrow\ & x_{i'} \equiv x' \pmod p. \end{aligned}$$

Hence, $p|(x_{i'} - x')$. With this information, one can factorize n. Note that Assumption 1 or Assumption 2 implies that factorization of n is hard.

Case 2: $i' \notin [N]$. This will result in a contradiction with the assumption that $f_{s_3}(\cdot)$ is PRF. We save the details. Note that in our use of Tag in this paper, this case will not consider as a valid forgery, since we are only interested in index $i \in [N]$. □

Appendix B. Proof of Theorem 2

THEOREM 2. MAIA^1 *is a $\mathcal{PRC}$ protocol (as defined in Definition 2) w.r.t. 1D* SUM *query (as defined in Section 3.1), under Assumption 1 and Assumption 2, and assuming that $f_s(\cdot)$ is PRF.*

Note: (1) We can apply Assumption 2 on both $\mathcal{Q}$ and $\mathcal{R}$. (2) All messages Alice sends to Bob (which may be recorded in $\mathsf{view}_{\mathcal{A}}^{\mathsf{Ext}}$) is pseudorandom, by Assumption 1. (3) As specified in Definition 3, the hash function $h(\cdot)$ is division-intractable [37]. So Compress and Uncompress work as desired with o.h.p.

Proof: Le $\mathbb{P}_{\mathcal{A}}$ be defined as in Definition 2. The correctness part is straightforward. We focus on the soundness part, i.e. whether $\mathbb{P}_{\mathcal{A}} \succeq 1 - negl$ holds.

Let $(w_1, w_\iota, \psi_1, \psi_\iota, \widetilde{\psi_1}, \widetilde{\psi_\iota})$ denote the message generated by an honest Bob (i.e. Eval), and $(w_1^*, w_\iota^*, \psi_1^*, \psi_\iota^*, \widetilde{\psi_1^*}, \widetilde{\psi_\iota^*})$ denote the 1 message generated by malicious Bob (i.e. adversary $\mathcal{A}$).

1) If ψ_1 satisfies Eq 5, then $\psi_1^* = \psi_1$ with overwhelming high probability, otherwise $\mathcal{R}(\frac{\psi_1}{\psi_1^*}) = 0$ is in contradiction with Assumption 2.
2) Since both (w_1, ψ_1) and (w_1^*, ψ_1^*) satisfy Eq 6, and $\psi_1^* = \psi_1$, we have $p|(w_1 - w_1^*)$. Since Assumption 1

implies that n is hard to factorize, we have $w_1^* = w_1$ with o.h.p.

3) Since $(w_1, \psi_1, w_\iota, \psi_\iota)$ satisfies Eq 4, we have

$$prs_{\iota,2}\left(\mathcal{Q}(\psi_1) + w_1\right) \equiv prs_{1,2}\left(\mathcal{Q}(\psi_\iota) + w_\iota\right) \quad (9)$$

$$\Rightarrow \mathcal{Q}(\psi_\iota) \equiv s_{\iota,2}s_{1,2}^{-1}(\mathcal{Q}(\psi_1) + w_1) - w_\iota \pmod{q} \quad (10)$$

$(w_1^*, \psi_1^*, w_\iota^*, \psi_\iota^*)$ also satisfies Eq 4. So we have

$$\mathcal{Q}(\psi_\iota^*) \equiv s_{\iota,2}s_{1,2}^{-1}(\mathcal{Q}(\psi_1^*) + w_1^*) - w_\iota^* \pmod{q} \quad (11)$$

Note we have $w_1^* = w_1, \psi_1^* = \psi_1$. Combining Eq 10 and Eq 11, we have

$$\mathcal{Q}(\frac{\psi_\iota^*}{\psi_\iota}) \equiv \mathcal{Q}(\psi_\iota^*) - \mathcal{Q}(\psi_\iota) = w_\iota - w_\iota^* \pmod{q}.$$

If $\frac{\psi_\iota^*}{\psi_\iota} \not\equiv 1$, then $(\frac{\psi_\iota^*}{\psi_\iota}, w_\iota - w_\iota^*)$ is in contradiction with Assumption 2. Hence, we have $\psi_\iota^* \equiv \psi_\iota$ and $w_\iota = w_\iota^*$, with o.h.p.

In summary, $(w_1, w_\iota, \psi_1, \psi_\iota) = (w_1^*, w_\iota^*, \psi_1^*, \psi_\iota^*)$ with o.h.p. Hence, $\mathbb{P}_{\mathcal{A}} \succeq 1 - negl$. □

Appendix C. Proof of Lemma 3

LEMMA 3. *Under Assumption 1 and Assumption 2, and assuming that $f_s(\cdot)$ is a PRF,* MAIA^ℓ *is a $\mathcal{PRC}$ protocol w.r.t. ℓ-dimensional aggregate* SUM *query , if* $\mathsf{MAIA}^{\ell-1}$ *is a $\mathcal{PRC}$ protocol w.r.t. $(\ell-1)$-dimensional aggregate* SUM *query.*

Proof: (Sketch) The correctness part is straightforward. We focus on the soundness part, i.e. whether $\mathbb{P}_{\mathcal{A}} \succeq 1 - negl$ holds.

Let $\widetilde{\mathtt{D}} = (\mathtt{D}, \{\mathsf{T}_{\mathbf{x}} : \mathbf{x} \in \mathtt{D}\})$. The function computated by MAIA^ℓ is

$$F(\widetilde{\mathtt{D}}, \textsc{Sum}(\mathbf{B}, \iota)) = (w_0, \psi_{0,1}, \psi_{0,2}, \cdots, \psi_{0,\ell}, \psi_{0,\iota}),$$

where w_0 is the answer to $\textsc{Sum}(\mathbf{B}, \iota)$, and $\psi_{0,j}$'s are some aggregated values over Tag values (Check $\mathsf{CollRes}^\ell$ and MAIA^ℓ in Figure 4 and Figure 6 for details).

Suppose the output of MAIA^ℓ (precisely, output of Alice in ProVer^ℓ) is not $\perp$.

Step 1: Verifying 1D query $a_j \le \mathbf{x}[j] \le b_j$. The first $(\ell+1)$ equations (Eq 3) in Step A2 of $\mathsf{CollRes}^\ell$ in Figure 4, intend to verify 1D query $a_j \le \mathbf{x}[j] \le b_j$ for each dimenion $j \in [\ell] \cup \{\iota\}$ separately, by checking the P componemt of Tag values.

Let $F_1(w_j, \psi_j) = \psi_j g_p^{-w_j}$ and $F_2(\widetilde{\psi}_j) = \widetilde{\psi}_j^{qrs_{j,1}}$. In Figure 4, Eq 3 can be expressed as $F_1(w_j, \psi_j)^{qr\alpha_j} \equiv F_2(\widetilde{\psi}_j)$, where α_j is the secret random nonce chosen by Alice. Note that $F_1(\cdot)$ and $F_2(\cdot)$ are homormorphic functions with secret key $(g_p, q, r, s_{j,1})$ and the adversary cannot compute $F_1(\cdot)$ and $F_2(\cdot)$ by themselves. If $F_1(w_1, \psi_1)^\beta = F_2(\widetilde{\psi}_1)$ and $F_1(w_2, \psi_2)^\beta = F_2(\widetilde{\psi}_2)$, then

$$\left(F_1\left(w_1 + w_2, \psi_1\psi_2\right)\right)^\beta = F_2\left(\widetilde{\psi}_1\widetilde{\psi}_2\right) \quad (12)$$

$$\left(F_1\left(w_1 - w_2, \frac{\psi_1}{\psi_2}\right)\right)^\beta = F_2\left(\frac{\widetilde{\psi}_1}{\widetilde{\psi}_2}\right) \quad (13)$$

By Assumption 1, $F_1(w, \psi)$'s are pseudorandom, and all *Help-Info* sent in previous sessions (recorded in $\mathsf{view}_{\mathcal{A}}^{\mathsf{Ext}}$) are pseudorandom to the adversary $\mathcal{A}$. We claim that, the only way that the adversary cheats and passes the verifications of $F_1(\cdot)^\beta = F_2(\cdot)$ is through exploiting homomorphism of F_1 and F_2. The additive homomorphism (Eq 12) and subtractive homomorphism (Eq 13) just correspond to double-counting (twice or more times) and undercounting, respectively, of some points inside the 1D query range $[N]^{j-1} \times [a_j, b_j] \times [N]^{d-j}$.

This claim can be proved with similar techniques used in proof of BLS signature [40].

Step 2: Verifying Conjunctions of 1D Range Query. The last ℓ equations (Eq 4) in Step A2 of $\mathsf{CollRes}^\ell$ in Figure 4, intend to verify the conjunctions of all ℓ 1D range queries, i.e. $\bigwedge_{1\le j\le \ell} a_j \le \mathbf{x}[j] \le b_j$, by checking the binding variable ξ's in Tag values.

Similar with Step 1, since $s_{j,2}$'s, $j \in [\ell] \cup \{\iota\}$, are secret random numbers, the only way the adversary can cheat and pass this verification is to exploit homomorphism, which is actually double-counting (twice or more times) and miss-counting of some points inside the ℓ-dimensional query range $[a_1, b_1] \times \cdots \times [a_\ell, b_\ell] \times [N]^{d-\ell}$.

Step 3: Verifying Complementary Region. We assume $\mathsf{MAIA}^{\ell-1}$ is a $\mathcal{PRC}$. So the answer $(W_3, \psi_{3,1}, \psi_{3,2}, \cdots, \psi_{3,\ell-1}, \psi_{3,\iota}))$ to $\textsc{Sum}(\mathbf{B}_3, \iota)$ (See Figure 6) is authentic with o.h.p.

Since $\mathcal{R}(\psi_{i,j})$'s $(1 \le i \le 3, j \in [\ell] \cup \{\iota\})$ are pseudorandom and secret, by Assumption 2, PPT adversary cannot find two distinct multisets $K_1 = \{\psi_{i_1}, \psi_{i_2}, \cdots, \psi_{i_m}\}$ and $K_2 = \{\psi_{j_1}, \psi_{j_2}, \cdots, \psi_{j_{m'}}\}$ with non-negligible probability, such that

$$\sum_{1\le v\le m} \mathcal{R}(\psi_{i_v}) \equiv \sum_{1\le v\le m'} \mathcal{R}(\psi_{j_v}) \pmod{r}.$$

So that the adversary cannot pass verifications of ψ's in Eq 8 in Step A2 in Figure 6, by miss-counting elements in K_1 and double-counting elements in K_2.

Summary. For the above reasons, the output of MAIA^ℓ is $F(\widetilde{\mathtt{D}}, \textsc{Sum}(\mathbf{B}, \iota))$ with o.h.p. Hence, $\mathbb{P}_{\mathcal{A}} \ge 1 - negl$.

□