Authenticating Aggregate Range Queries over Multidimensional Dataset

Jia Xu
National University of Singapore
xujia@comp.nus.edu.sg

Ee-Chien Chang
National University of Singapore
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 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, Multidimensional Aggregate Query, Secure Outsourced Database, Provable Remote Computing

1. INTRODUCTION

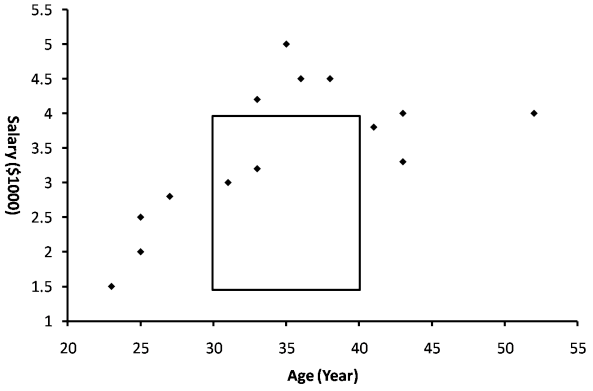
Alice has a set D of d-dimensional points. She preprocesses the dataset D using her private key to generate some authentication tag T. She sends (outsources) D and T to an untrusted service provider Bob. Then Alice deletes the original copy of dataset D and tag T from her local storage. Later Alice may 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 wants to authenticate the query result, using only her private key.

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 [20, 51], 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 [24].

Researches in secure outsourced database focus on two major aspects: privacy [51, 52, 53, 54] (i.e. protect the data confidentiality against both the service provider and any third party), and integrity [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34] (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" [26], i.e. the query result is a subset of the database. Aggregate range query is arguably more challenging and only a few works (e.g. [7]) are devoted to the authentication of aggregate query.

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.



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 $\mathtt{D}$ with N d-dimensional points, the number of communication bits required is in $O(d^2)$ per query. The storage overhead on Bob's side is $O(dN)$, which is linear w.r.t. the size of $\mathtt{D}$.

We now describe our main ideas in three steps: (1) We describe a plausible scheme, which is insecure and requires large amount of communication, to authenticate aggregate range query. (2) We point out problems in this scheme and difficulties to implement it. (3) We summarize the key techniques we used to conquer such problems or difficulties.

Plausible Scheme

For simplicity, let $\mathtt{D}$ be a set of N 1D points. Let T be an additive homomorphic authentication tag function, such that $\mathsf{T}(\boldsymbol{x}+\boldsymbol{y}) = \mathsf{T}(\boldsymbol{x})\mathsf{T}(\boldsymbol{y})$. During setup, Alice preprocesses the dataset $\mathtt{D}$ with a private key, to generate a tag value $\mathsf{T}(\boldsymbol{x})$ for each point $\boldsymbol{x} \in \mathtt{D}$. Next, Alice sends $\mathtt{D}$ and tag values $\{\mathsf{T}(\boldsymbol{x}) : \boldsymbol{x} \in \mathtt{D}\}$ to Bob and deletes everything except the private key from her storage. Let us consider a summing query conditional on range $\mathbf{B}$ which asks for the sum $\sum_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}} \boldsymbol{x}$. Bob is expected to send to Alice a number $\boldsymbol{X}$ as the query result, and a proof to show that indeed $\boldsymbol{X} = \sum_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}} \boldsymbol{x}$.

To process this query, Alice sends an auxiliary message (called as *Help-Info*) $\{(\mathsf{T}(\boldsymbol{x}))^{\boldsymbol{\alpha}} : \boldsymbol{x} \in \mathbf{B}\}$ to Bob, in order to help Bob to generate a proof for the query result. Here the secret random nonce $\boldsymbol{\alpha}$ prevents Bob from abusing this *Help-Info* for other queries. Bob is expected to:

1. sum all $\boldsymbol{x} \in \mathtt{D}\cap\mathbf{B}$ to obtain $\boldsymbol{X}$: $\boldsymbol{X} = \sum_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}} \boldsymbol{x}$;
2. aggregate all $\mathsf{T}(\boldsymbol{x})$'s for $\boldsymbol{x} \in \mathtt{D}\cap\mathbf{B}$ to obtain $\boldsymbol{\Psi}_1$:

$$\boldsymbol{\Psi}_1 = \prod_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}} \mathsf{T}(\boldsymbol{x}) = \mathsf{T}\left(\sum_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}} \boldsymbol{x}\right);$$

3. aggregate all $(\mathsf{T}(\boldsymbol{x}))^{\boldsymbol{\alpha}}$'s for $\boldsymbol{x} \in \mathtt{D}\cap\mathbf{B}$ to obtain $\boldsymbol{\Psi}_2$:

$$\boldsymbol{\Psi}_2 = \prod_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}} (\mathsf{T}(\boldsymbol{x}))^{\boldsymbol{\alpha}} = \left(\mathsf{T}\left(\sum_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}} \boldsymbol{x}\right)\right)^{\boldsymbol{\alpha}}.$$

Bob sends back $\boldsymbol{X}$ as query result and $(\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2)$ as proof. Alice can verify the correctness, by checking the following two equalities:

$$\begin{aligned} \boldsymbol{\Psi}_1 &\stackrel{?}{=} \mathsf{T}(\boldsymbol{X}) \\ \boldsymbol{\Psi}_1^{\boldsymbol{\alpha}} &\stackrel{?}{=} \boldsymbol{\Psi}_2 \end{aligned}$$

For data point $\boldsymbol{x} \in \mathtt{D}\cap\mathbf{B}$, Bob has both $\mathsf{T}(\boldsymbol{x})$ and $(\mathsf{T}(\boldsymbol{x}))^{\boldsymbol{\alpha}}$. For any point $\boldsymbol{x} \notin \mathtt{D}\cap\mathbf{B}$, Bob cannot provide either $\mathsf{T}(\boldsymbol{x})$ or $(\mathsf{T}(\boldsymbol{x}))^{\boldsymbol{\alpha}}$, assuming it is difficult for Bob to forge $\mathsf{T}(\boldsymbol{x})$ or $(\mathsf{T}(\boldsymbol{x}))^{\boldsymbol{\alpha}}$ without the private key and the random nonce $\boldsymbol{\alpha}$. Hence, Bob is unable to include points outside $\mathtt{D}\cap\mathbf{B}$ in the computation of query result $\boldsymbol{X}$ and proof $(\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2)$.

Problems and Difficulties

We identify 3 problems in the above scheme: (1) Bob still can cheat by double counting or undercounting some points in the set $\mathtt{D}\cap\mathbf{B}$. For example, Bob may send $2\boldsymbol{X}$ as query result and $(\boldsymbol{\Psi}_1\times\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2\times\boldsymbol{\Psi}_2)$ as proof. Due to homomorphism of T, Alice will consider this proof valid. (2) Additive homomorphism is unlikely secure [49], and the assumption of unforgeability of $\mathsf{T}(\boldsymbol{x})$ and $(\mathsf{T}(\boldsymbol{x}))^{\boldsymbol{\alpha}}$ may not hold. (3) The size of *Help-Info* is linear w.r.t. the size of query range $\mathbf{B}$, which is huge and possibly comparable to the domain size of a data point. This implies large computation cost (to generate *Help-Info*) and communication cost (to send *Help-Info*).

Our solution

We employ or design several key techniques to solve these problems, including:

1. To prevent double counting or undercounting, Alice will check the complement query with range $\mathbf{B}^{\mathsf{c}}$, which asks for the value of $\sum_{\boldsymbol{x}\in\mathtt{D}\cap\mathbf{B}^{\mathsf{c}}} \boldsymbol{x}$. Nevertheless, a malicious Bob may undercount some points in $\mathtt{D}\cap\mathbf{B}$ and compensate it by double counting some points in $\mathtt{D}\cap\mathbf{B}^{\mathsf{c}}$. To prevent such cheating, we introduce a secret and random attribute to each data point and aggregate it alongside with the requested value. Note such strategy that verifies completeness by checking the complement is also used in Chen *et al.* [19].
2. To decrease the size of query range $\mathbf{B}$, we "normalize" the dataset $\mathtt{D}$, so that the values of normalized data points are in $[1, N]$. Note N is the number of points in $\mathtt{D}$.
3. To further decrease the size of *Help-Info*, we separate each dimension of a query range, and are able to *compress Help-Info* for each dimension. For example, let $\mathbf{B} = [a_1, b_1]\times[a_2, b_2]$ be a 2D range. Alice compresses *Help-Info* for 1D range $[a_1, b_1]$ to obtain ϖ_1, and compresses *Help-Info* for 1D range $[a_2, b_2]$ to obtain ϖ_2. From (ϖ_1, ϖ_2), Bob is able to reconstruct *Help-Info* for the 2D range $\mathbf{B}$. Such separation of dimensions might cause collusion attack [1].
4. We specially design an authentication tag function, which (1) allows Bob to do aggregation to generate a short proof; (2) is unforgeable; (3) allows *Help-Info* to be compressed and yet prevents collusion attack; (4) binds a secret and random attribute to each point and allows Alice to verify completeness by checking the complement.

Our tag function, denoted as Tag, consists of three components DTag, ITag and CTag. Let $\boldsymbol{i}$ be the normalized value of point $\boldsymbol{x} \in \mathtt{D}$.

1. $\mathsf{DTag}(\boldsymbol{x})$ binds different dimensions of a d-dimensional point $\boldsymbol{x}$, so Bob cannot forge a new point by mixing and combining different dimensions of different points (e.g. $(\boldsymbol{x}_2[1], \boldsymbol{x}_1[2], \boldsymbol{x}_1[3], \ldots, \boldsymbol{x}_1[d])$);

Table 1: Performance of different authentication schemes for aggregate range query or range selection query. This table consists of two parts: the first two rows are for aggregate query; the rest four rows are for range selection query. Our scheme MAIA appears twice in this table, since it can authenticate both aggregate query and range selection query (with modifications in Section 6.2.4). Note $dN \log N \leq \log^d N$, if $d > \log(dN)/\log\log N$.

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, MEDIAN, QUANTILE
Atallah *et al.* [30]	$O(1)$	$O(1)$	$O(N)$	$O(\|S\|)$	$O(1)$	$d = 1, 2$	Range Selection
Martel *et al.* [21]	$O(\log^{d-1} N + \|S\|)$	-	-	-	-	$d \geq 1$	Range Selection
Chen *et al.* [19]	$O(\log^d M)$	-	$O(N \log^d M)$	$O(\log^d M)$	$O(\log^d M)$	$d \geq 1$	Range Selection
MAIA (Section 6.2.4)	$O(d^2)$	$O(d)$	$O(dN)$	$O(d \log N)$†	$O(dN \log N)$‡⋆	$d \geq 1$	Range Selection

N: The number of tuples in the dataset.
K: The number of servers in PDAS [7].
†: $O(d \log N)$ modular multiplications.
⋆: If the query range is 1D, the cost is $O(|S|)$.
S: The set of tuples satisfying the query condition.
M: The domain size of attributes in Chen *et al.* [19].
‡: $O(dN \log N)$ modular exponentiations.

2. $\mathsf{ITag}(\boldsymbol{i})$ is an authentication tag of $\boldsymbol{i}$ and allows Alice to compress the *Help-Info* (The compression is based on redactable signature scheme [49]);
3. $\mathsf{CTag}(\boldsymbol{i})$ prevents double counting and undercounting.

With private key, Alice is able to "extract" $\mathsf{DTag}(\boldsymbol{x})$, $\mathsf{ITag}(\boldsymbol{i})$ or $\mathsf{CTag}(\boldsymbol{i})$ from the value of $\mathsf{Tag}(\boldsymbol{x}, \boldsymbol{i})$. Without the private key, Bob cannot decompose $\mathsf{Tag}(\boldsymbol{x}, \boldsymbol{i})$. We construct Tag using three cyclic multiplicative subgroups of $\mathbb{Z}_n^*$, where n is the product of three safe primes [5, 55].

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

1. We propose a scheme called MAIA (Section 5), to authenticate multidimensional aggregate range queries, based on a specially designed tag function Tag (Section 4).
2. MAIA is efficient (See Table 1) and takes only $O(d^2)$ communication bits for a d-dimensional aggregate range query, independent of the query range size and dataset size.
3. We prove that MAIA is secure (Theorem 2) under reasonable assumptions (Assumption 1, Assumption 2, etc.).
4. We extend MAIA to authenticate multidimensional range selection query (Section 6.2.4). The performance is showed in Table 1.

2. RELATED WORK

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 [8,9,11], 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 across 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. It is unlikely to solve our problem using MRQED as a black-box, and we doubt whether it is possible to design an alternative solution based on MRQED without significant modifications. However, MAIA and MRQED indeed can be benefited from each other, especially in how to represent a range compactly. We present an alternative compression method in Section 6.2.1 based on binary interval tree, which is employed by MRQED. On the other hand, it is possible to design an alternative solution for MRQED problem using redactable signature scheme [49], which we employ as our compression method.

Several works [12, 13, 14, 15, 16, 17] 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.

There are roughly four categories of approaches for outsourced database authentication in the literatures [20,21,22, 23,24,25,26,27,28,29,30,31,32,33,34]. (1) (Homomorphic, or aggregatable) Cryptographic primitives, like collision-resistant hash, digital signature, commitment [24, 35, 7]. (2) Merkle Hash Tree and variants [32, 28, 27]. (3) Computational geometry approach [21, 27, 30]. (4) Inserting and auditing fake tuples [29].

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 condi-

tion), and linear storage overhead, to authenticate 1D aggregate (more precisely, SUM,COUNT) queries. PDAS is based on Shamir's threshold secret sharing [36] and Pedersen commitment scheme [37], 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) can 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 introduce some key notations in Section 3.1, formally describe the dataset and query in Section 3.2, give a general security model in Section 3.3 for authentication of outsourced computing, and introduce our security assumptions in Section 3.4.

3.1 Notations

For the convenience of presentation, we introduce several operators $\oplus, \otimes$ and $\circledast$ on vectors as follows: Let $\boldsymbol{u} = (u_1, u_2, \ldots, u_d)$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_d)$ be two d-dimensional vectors, where $u_j, v_j \in \mathbb{N}$ for $1 \le j \le d$.

$$\oplus: \ \boldsymbol{u} \oplus \boldsymbol{v} = (u_1 + v_1, u_2 + v_2, \ldots, u_d + v_d); \quad (1)$$

$$\otimes: \ \boldsymbol{u} \otimes \boldsymbol{v} = (u_1 v_1, u_2 v_2, \ldots, u_d v_d); \quad (2)$$

$$\circledast: \ \boldsymbol{u} \circledast \boldsymbol{v} = (u_1^{v_1}, u_2^{v_2}, \ldots, u_d^{v_d}). \quad (3)$$

We also introduce scalar multiplication $a\boldsymbol{u} = (au_1, au_2, \ldots, au_d)$, where a is a scalar, and modulo operator **mod** on vectors:

$$\boldsymbol{u} \quad \textbf{mod } n = (u_1 \mod n, \ldots, u_d \mod n).$$

Additionally, we have vector $\boldsymbol{e} = (\underbrace{1, 1, \ldots, 1}_{d \text{ 1's}})$.

Here in Table 2, we summarize the key notations in this paper.

3.2 Dataset and Query

We give a simplified (i.e. normalized [38]) data model, and the main construction in Section 5 will be based on this simplified data model. Later, in Section 6.2.2, we extend our scheme to general data model. We emphasize that this simplification does not lose generality, and serves two purposes: (1) reduce computation cost (precisely, reduce the size of *Help-Info*); (2) simplify the presentation.

Our data model consists of a set $\mathtt{D}$ and a vector-valued *attribute* function Att, such that: (1) $\mathtt{D} = \{\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_N\} \subseteq [N]^d$, and for each $j \in [d]$, the set $\{\boldsymbol{i}_1[j], \boldsymbol{i}_2[j], \ldots, \boldsymbol{i}_N[j]\} = [N]$; (2) $\mathsf{Att} : [N]^d \to \mathbb{X}^d$, where $\mathbb{X}^d \subset \mathbb{N}^d$ is the domain of attributes.

Let $\mathbf{R} = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d] \subseteq [N]^d$ be a rectangular range. A range selection query with range $\mathbf{R}$ over the dataset $\mathtt{D}$ asks for the set of points $\boldsymbol{i}$'s in $\mathtt{D} \cap \mathbf{R}$, and an aggregate sum query with range $\mathbf{R}$ over the dataset $\mathtt{D}$, denoted as $\textsc{Sum}(\mathbf{R})$, asks for the sum $\bigoplus_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}} \mathsf{Att}(\boldsymbol{i})$.

Table 2: Summary of Key Notations

- $\mathbb{N}$: 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 $a \le b$ are non-negative integers.
- $[b]$: The set $\{1, 2, 3, \cdots, b\}$, where b is a positive integer.
- $|S|$: The size of the set (or multiset) S.
- $\boldsymbol{i}[j]$: i_j, where $\boldsymbol{i} = (i_1, i_2, \cdots, i_j, \cdots)$ is a vector of dimension greater than or equal to j.
- $\boldsymbol{e}$: The d-dimensional vector with each dimension equal to 1.
- $\oplus, \otimes, \circledast$: Addition, multiplication and exponentiation operators over vectors.
- Prime: Set of all odd primes.
- $\mathsf{Tag}(\boldsymbol{x}, \boldsymbol{i}; \xi)$: Randomized function Tag with input $(\boldsymbol{x}, \boldsymbol{i})$ 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).

3.3 Security Model

We formulize the authentication problem described in Section 1.

Let us view the query on a database as the function $F : \mathbb{D} \times \mathbb{Q} \to \{0,1\}^*$, where $\mathbb{D}$ is the domain of databases, $\mathbb{Q}$ is the domain of queries, and the output of F is a binary string. We define a remote computing protocol as follow:

DEFINITION 1 ($\mathcal{RC}$). *A $\mathcal{RC}$ (*Remote Computing*) protocol for a function $F : \mathbb{D} \times \mathbb{Q} \to \{0,1\}^*$, 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 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. *Given security parameter κ, Alice generates a key k: $k \leftarrow \mathsf{KGen}(1^\kappa)$.*
2. *Alice encodes database $x \in \mathbb{D}$: $(p_x, s_x) \leftarrow \mathsf{DEnc}(x, k)$, then sends p_x to Bob and keeps s_x.*
3. *Alice selects a query $r \in \mathbb{Q}$.*
4. *Algorithm $\mathsf{Eval}(p_x)$ on Bob's side, interacts with algorithm $\mathsf{Ext}(s_x, r, k)$ on Alice's side, to compute $y \leftarrow \langle \mathsf{Eval}(p_x), \mathsf{Ext}(s_x, r, k) \rangle$. If $y = \perp$, then Alice rejects. Otherwise, Alice believes that y is equal to $F(x, r)$.*

In the setup phase, Alice executes Step (1) and (2). The query phase consists of multiple query sessions. In each query session, Alice and Bob execute Step (3) and (4).

We say a $\mathcal{RC}$ protocol is provable, if the following conditions hold: (1) Alice accepts with o.h.p. (overwhelming high probability), when Bob follows the protocol honestly; (2) Alice rejects with o.h.p., when Bob returns a wrong result. Here we consider adversaries, i.e. malicious Bob, who are allowed to interact with Alice and learn for polynomial number of queries, before launching the attack. During the learning, the adversary may store whatever it has seen or leant in a state variable.

DEFINITION 2 ($\mathcal{PRC}$). *A $\mathcal{RC}$ protocol* $(\mathsf{KGen}, \mathsf{DEnc}, \langle \mathsf{Eval}, \mathsf{Ext} \rangle)$ *w.r.t. function* $F : \mathbb{D} \times \mathbb{Q} \to \{0,1\}^*$, *is called* $\mathcal{PRC}$ (Provable Remote Computing) *protocol, if the following two conditions hold: Let* κ *be the security parameter.*

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

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

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

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

The probability is taken over all random coins used by related algorithms, $poly(\cdot)$ *is an arbitrary but fixed polynomial function,* $negl(\cdot)$ *is some negligible function, and* $\mathsf{view}_{\mathcal{A}}^{\mathsf{Ext}}$ *is a state variable*[1] *describing all random coins chosen by* $\mathcal{A}$ *and all messages* $\mathcal{A}$ *received from* Ext *during previous interactions.*

We remark that the security model is similar to the formulation of $\mathcal{POR}$ (Proof of Retrievability) [18], and it is not surprising that our scheme MAIA implies a (ρ, λ)-valid $\mathcal{POR}$ system, with some parameters ρ and λ.

3.4 Assumptions

In this subsection, we state the assumptions that our proposed scheme relies on.

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,*

- $\mathcal{X} \stackrel{def}{=} \{\mathcal{X}_\kappa \stackrel{\$}{\leftarrow} G\}$, *i.e.* $\mathcal{X}_\kappa$ *is uniformly randomly distributed over* G;
- $\mathcal{Y} \stackrel{def}{=} \{\mathcal{Y}_\kappa \stackrel{\$}{\leftarrow} G_r\}$, *i.e.* $\mathcal{Y}_\kappa$ *is uniformly randomly distributed over* G_r.

[1]The adversary $\mathcal{A}$ may keep updating this state variable.

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* $(x, y, z) \in \mathbb{Z}_p \times \mathbb{Z}_q \times \mathbb{Z}_r$, $\mathcal{R}(g_p^x g_q^y g_r^z) \stackrel{def}{=} z$. *Let* Δ *be a set* $\{g_p^x g_q^y g_r^z \in G : z \stackrel{\$}{\leftarrow} \mathbb{Z}_r$ *and* (x, y) *are chosen from* $\mathbb{Z}_p \times \mathbb{Z}_q$ *under* any *distributions*$\}$. *Given only* (n, Δ), *it is hard to compute* (A, B), *such that*

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

Assumption 1 can be considered as a variant version of the p-Subgroup Assumption [39], and also a variant version of the Subgroup Membership Problem ($\mathcal{SMP}$) Assumption [9, 40, 41, 42, 43, 44, 45, 46], and Assumption 2 is closely related to the Projection Problem ($\mathcal{PP}$) Assumption [40, 42].

4. AUTHENTICATION TAG

In this section, we first construct an authentication tag function Tag (Section 4.1), which consists of three components. Next, we give an algorithm to decompose an output of the tag function into components (Section 4.2). At the end of this section, we show that the tag function is unforgeable (Section 4.3).

4.1 Construction of tag functions

We construct several tag functions dtag (stands for "tag of data"), itag (stands for "tag of index"[2]), and ctag (stands for "tag for completeness"). We combine these functions together to obtain tag function tag.

Let $\mathcal{G}$ be as in Section 3.4.

DEFINITION 3. *Let* $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$ *be an output of* $\mathcal{G}$. *Let* $(s_1, s_2, s_3) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^* \times \{0,1\}^\kappa$. *Let key* $k = (s_1, s_2, s_3, n, p, q, r, g_p, g_q, g_r)$. *Define functions* $\mathsf{dtag}, \mathsf{itag}, \mathsf{ctag}$ *and* tag *as follows:*

$$\begin{aligned} \mathsf{dtag}_k(x; \xi) &= g_p^x g_q^{x + s_2 \xi} \pmod{n}, \\ \mathsf{itag}_k(i) &= g_p^{\frac{s_1}{h(i)}} \pmod{n}, \\ \mathsf{ctag}_k(i) &= g_r^{f_{s_3}(i)} \pmod{n}, \\ \mathsf{tag}_k(x, i; \xi) &= \mathsf{dtag}_k(x; \xi) \times \mathsf{itag}_k(i) \times \mathsf{ctag}_k(i) \pmod{n}, \end{aligned}$$

where the random coin $\xi \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$, $h(\cdot) : \mathbb{N} \to \mathsf{Prime}$ *is a collision-resistant*[3] *hash function, and* $\{f_s\}_{s \in \{0,1\}^\kappa}$ *is* PRF [48].

Note that itag is a variant of signature scheme [47, 49].

The data is in multidimensional vector form. For the convenience of presentation, we also define similar tag functions in vector forms.

DEFINITION 4. *Let* $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$ *be an output of* $\mathcal{G}$. *Let* $(s_{\ell,1}, s_{\ell,2}, s_{\ell,3}) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^* \times \{0,1\}^\kappa$ *for each* $\ell \in [d]$. *Let* $\boldsymbol{s}_1 = (s_{1,1}, s_{2,1}, \ldots, s_{d,1})$, $\boldsymbol{s}_2 = (s_{1,2}, s_{2,2}, \ldots, s_{d,2})$ *and* $\boldsymbol{s}_3 = (s_{1,3}, s_{2,3}, \ldots, s_{d,3})$. *Let* $\mathcal{K} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, n, p, q, r, g_p, g_q, g_r)$. *Define vector-valued functions* DTag,ITag, CTag *and* Tag *as follows:*

$$\begin{aligned} \mathsf{DTag}_{\mathcal{K}}(\boldsymbol{x}; \xi) &= \left(\mathsf{dtag}_{k_1}(\boldsymbol{x}[1]; \xi), \ldots, \mathsf{dtag}_{k_d}(\boldsymbol{x}[d]; \xi)\right), \\ \mathsf{ITag}_{\mathcal{K}}(\boldsymbol{i}) &= (\mathsf{itag}_{k_1}(\boldsymbol{i}[1]), \ldots, \mathsf{itag}_{k_d}(\boldsymbol{i}[d])), \\ \mathsf{CTag}_{\mathcal{K}}(\boldsymbol{i}) &= (\mathsf{ctag}_{k_1}(\boldsymbol{i}[1]), \ldots, \mathsf{ctag}_{k_d}(\boldsymbol{i}[d])), \\ \mathsf{Tag}_{\mathcal{K}}(\boldsymbol{x}, \boldsymbol{i}; \xi) &= (\mathsf{tag}_{k_1}(\boldsymbol{x}[1], \boldsymbol{i}[1]; \xi), \ldots, \mathsf{tag}_{k_d}(\boldsymbol{x}[d], \boldsymbol{i}[d]; \xi)), \end{aligned}$$

[2]Here "index" means the normalized value $\boldsymbol{i}$ of data $\boldsymbol{x}$. See the extended data model in Section 6.2.2.

[3]In other words, h should be *division intractable*. See Gennaro *et al.* [47] for the definition of "division intractable".

where for each $\ell \in [d]$, $k_\ell = (\boldsymbol{s}_1[\ell], \boldsymbol{s}_2[\ell], \boldsymbol{s}_3[\ell], n, p, q, r, g_p, g_q, g_r)$, *and the random coin* $\xi \in \mathbb{Z}_q^*$.

In another equivalent expression,

$$\begin{aligned}&\mathsf{Tag}_{\mathcal{K}}(\boldsymbol{x}, \boldsymbol{i}; \xi) = \mathsf{DTag}_{\mathcal{K}}(\boldsymbol{x}; \xi) \otimes \mathsf{ITag}_{\mathcal{K}}(\boldsymbol{i}) \otimes \mathsf{CTag}_{\mathcal{K}}(\boldsymbol{i})\\ &= ((g_p g_q \boldsymbol{e}) \circledast \boldsymbol{x}) \otimes \mathsf{ITag}_{\mathcal{K}}(\boldsymbol{i}) \otimes \mathsf{DTag}_{\mathcal{K}}(\boldsymbol{0}; \xi) \otimes \mathsf{CTag}_{\mathcal{K}}(\boldsymbol{i}). \quad (4)\end{aligned}$$

Note: (1) We call g_q^{ξ} *binding factor* among d dimensions of $\mathsf{Tag}_k(\boldsymbol{x}, \boldsymbol{i}; \xi)$. Given a candidate Tag value $\boldsymbol{t}$, we can extract the binding factor from $\boldsymbol{t}[j]$ for each dimension $j \in [d]$, and check whether they are equal. (2) We do not store the values of ξ's when we use the tag function Tag in our scheme. Sometimes we may ignore ξ, and just write $\mathsf{Tag}(\boldsymbol{x}, \boldsymbol{i})$.

4.2 Decompose a tag value

As mentioned in the introduction, Alice is able to extract an ITag (or CTag, DTag) value from a Tag value, with the private key. Such extraction or decomposition play an important role in the verification of our scheme (See Step A2 of $\mathsf{CollRes}$ in Section 5.3). We construct a pair of encoding and decoding algorithms $(\mathcal{E}, \mathcal{D})$ to reflect the idea of decomposition: (1) $\mathcal{E}$ encodes outputs of DTag, ITag and CTag into an output of Tag; (2) $\mathcal{D}$ decodes an output of Tag into outputs of DTag, ITag and CTag. The rest of this subsection provides the details.

4.2.1 In Scalar Form

The output of function tag is an element from G. Recall that $n = (2p+1)(2q+1)(2r+1)$. With the factorization of n, i.e. p, q, r, one can extract out the G_p, G_q or G_r component from an element from G (or say, project a G element onto subgroup G_p, G_q and G_r [40]).

Let $\lambda_p = (qr)^{-1} \mod p, \lambda_q = (pr)^{-1} \mod q, \lambda_r = (pq)^{-1} \mod r$. Let $t = g_p^x g_q^y g_r^z \mod n$. We have

$$\begin{aligned}\Upsilon_p(t) &\stackrel{\text{def}}{=} (t^{qr})^{\lambda_p} \equiv g_p^x \pmod n;\\ \Upsilon_q(t) &\stackrel{\text{def}}{=} (t^{pr})^{\lambda_q} \equiv g_q^y \pmod n;\\ \Upsilon_r(t) &\stackrel{\text{def}}{=} (t^{pq})^{\lambda_r} \equiv g_r^z \pmod n.\end{aligned}$$

Based on these, one can also extract out dtag, itag, or ctag component from an output of function tag. Let $t = \mathsf{tag}_k(x, i; \xi)$. Note $s_1, s_2, p, g, r, g_p, g_q$ and g_r are parts of the key k defined in Definition 3. We define

$$\begin{aligned}\gamma_1(t, g_p^x) &\stackrel{\text{def}}{=} \Upsilon_p(t) \left(g_p^x\right)^{-1} \equiv \mathsf{itag}_k(i) \pmod n;\\ \gamma_2(t, g_q^x) &\stackrel{\text{def}}{=} \Upsilon_q(t) \left(g_q^x\right)^{-1} \equiv \mathsf{dtag}_k(0; \xi) \pmod n;\\ \gamma_3(t) &\stackrel{\text{def}}{=} \Upsilon_r(t) \equiv \mathsf{ctag}_k(i) \pmod n.\end{aligned}$$

4.2.2 In Vector Form

Here we present the same decomposition in vector form. We may consider Tag as a homomorphic encoding algorithm $\mathcal{E}$ (See equation (4)):

$$\begin{aligned}&\mathsf{Tag}_{\mathcal{K}}(\boldsymbol{x}, \boldsymbol{i}; \xi) = \mathcal{E}_{\mathcal{K}}(\mathsf{ITag}_{\mathcal{K}}(\boldsymbol{i}), \mathsf{DTag}_{\mathcal{K}}(\boldsymbol{0}; \xi), \mathsf{CTag}_{\mathcal{K}}(\boldsymbol{i}), \boldsymbol{x});\\ &\mathcal{E}_{\mathcal{K}}(\boldsymbol{u}_1, \boldsymbol{v}_1, \boldsymbol{w}_1, \boldsymbol{x}_1) \otimes \mathcal{E}_{\mathcal{K}}(\boldsymbol{u}_2, \boldsymbol{v}_2, \boldsymbol{w}_2, \boldsymbol{x}_2) =\\ &\mathcal{E}_{\mathcal{K}}(\boldsymbol{u}_1 \otimes \boldsymbol{u}_2, \boldsymbol{v}_1 \otimes \boldsymbol{v}_2, \boldsymbol{w}_1 \otimes \boldsymbol{w}_2, \boldsymbol{x}_1 \oplus \boldsymbol{x}_2) \quad (5)\end{aligned}$$

The corresponding decoding algorithm $\mathcal{D}$ is: Let $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ are the outputs of $\mathsf{ITag}, \mathsf{DTag}, \mathsf{CTag}$, respectively, and $\boldsymbol{\psi} = \mathcal{E}_{\mathcal{K}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{x})$.

$$\mathcal{D}_{\mathcal{K}}(\boldsymbol{\psi}; \boldsymbol{x}) \stackrel{\text{def}}{=} (\boldsymbol{\Gamma}_1(\boldsymbol{\psi}, \boldsymbol{x}), \boldsymbol{\Gamma}_2(\boldsymbol{\psi}, \boldsymbol{x}), \boldsymbol{\Gamma}_3(\boldsymbol{\psi})) = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \quad (6)$$

where $\boldsymbol{\Gamma}_1, \boldsymbol{\Gamma}_2$ and $\boldsymbol{\Gamma}_3$ are defined below,

$$\begin{aligned}\boldsymbol{\Gamma}_1(\boldsymbol{t}, \boldsymbol{u}) &\stackrel{\text{def}}{=} (\gamma_1(\boldsymbol{t}[1], g_p^{\boldsymbol{u}[1]}), \ldots, \gamma_1(\boldsymbol{t}[d], g_p^{\boldsymbol{u}[d]}));\\ \boldsymbol{\Gamma}_2(\boldsymbol{t}, \boldsymbol{u}) &\stackrel{\text{def}}{=} (\gamma_2(\boldsymbol{t}[1], g_q^{\boldsymbol{u}[1]}), \ldots, \gamma_2(\boldsymbol{t}[d], g_q^{\boldsymbol{u}[d]}));\\ \boldsymbol{\Gamma}_3(\boldsymbol{t}) &\stackrel{\text{def}}{=} (\gamma_3(\boldsymbol{t}[1]), \ldots, \gamma_3(\boldsymbol{t}[d])).\end{aligned}$$

4.3 Security

LEMMA 1. *Let* k *be a key as in Definition 3. Given* $S = \{(x_i, i, t_i) : \xi \xleftarrow{\$} \mathbb{Z}_q^*, t_i \leftarrow \mathsf{tag}_k(x_i, i; \xi))\}_{1 \le i \le N}$, *it is computationally hard to forge a tuple* $(y, j, t) \notin S$, *such that* $\gamma_1(t, g_p^y) = \mathsf{itag}_k(j)$ *and* $\gamma_3(t) = \mathsf{ctag}_k(j)$, *under Assumption 1 and Assumption 2.*

The proof is given in Appendix B.1.

5. MAIA: MAIN CONSTRUCTION

Our scheme MAIA (Multidimensional Aggregate query Integrity Authentication) is a $\mathcal{RC}$ protocol for aggregate range query, and consists of algorithms $(\mathsf{KGen}, \mathsf{DEnc}, \langle \mathsf{Eval}, \mathsf{Ext} \rangle)$. The protocol consists of setup phase (Section 5.2) and query phase (Section 5.3). The setup phase is associated with algorithms KGen and DEnc, and the query phase is associated with interactive algorithm $\langle \mathsf{Eval}, \mathsf{Ext} \rangle$, which calls subroutine $\mathsf{CollRes}$.

Our scheme achieves efficient communication complexity, due to algorithms $\mathsf{Compress}$ and $\mathsf{Uncompress}$. We defer the constructions of $\mathsf{Compress}$ and $\mathsf{Uncompress}$ to Section 5.4.

Basically, the authentication consists of two parts. First, Alice has a mechanism, i.e. the algorithm $\mathsf{CollRes}$, to ensure that Bob produces the result and proof *solely* using data points within the query range $\mathbf{R}$. Next, to verify completeness, Alice also runs $\mathsf{CollRes}$ on aggregate query with the complement range $\mathbf{R}^c$, and checks whether the two query results and proofs "sum" to the whole dataset. Since the range $\mathbf{R}^c$ is not in the form that $\mathsf{CollRes}$ can handle, Alice has to divide it into $O(d)$ rectangular ranges, and runs $\mathsf{CollRes}$ on each of them.

First of all, let us review the authentication tag function Tag, which serves as the basis of our scheme.

5.1 Summary of tag function

Recall that, in section 4, we construct a key-ed tag function Tag, which consists of functions DTag, ITag and CTag as components. We also wrap Tag as a homomorphic encoding algorithm $\mathcal{E}$: Let $\mathcal{K}$ be the key as in Definition 4.

$$\begin{aligned}&\mathsf{Tag}_{\mathcal{K}}(\boldsymbol{x}, \boldsymbol{i}; \xi) = \mathcal{E}_{\mathcal{K}}(\mathsf{ITag}_{\mathcal{K}}(\boldsymbol{i}), \mathsf{DTag}_{\mathcal{K}}(\boldsymbol{0}; \xi), \mathsf{CTag}_{\mathcal{K}}(\boldsymbol{i}), \boldsymbol{x});\\ &\mathcal{E}_{\mathcal{K}}(\boldsymbol{u}_1, \boldsymbol{v}_1, \boldsymbol{w}_1, \boldsymbol{x}_1) \otimes \mathcal{E}_{\mathcal{K}}(\boldsymbol{u}_2, \boldsymbol{v}_2, \boldsymbol{w}_2, \boldsymbol{x}_2) =\\ &\mathcal{E}_{\mathcal{K}}(\boldsymbol{u}_1 \otimes \boldsymbol{u}_2, \boldsymbol{v}_1 \otimes \boldsymbol{v}_2, \boldsymbol{w}_1 \otimes \boldsymbol{w}_2, \boldsymbol{x}_1 \oplus \boldsymbol{x}_2).\end{aligned}$$

The corresponding decoding algorithm $\mathcal{D}$ is: Let $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ are the outputs of $\mathsf{ITag}, \mathsf{DTag}, \mathsf{CTag}$, respectively, and $\boldsymbol{\psi} = \mathcal{E}_{\mathcal{K}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{x})$.

$$\mathcal{D}_{\mathcal{K}}(\boldsymbol{\psi}, \boldsymbol{x}) = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}).$$

5.2 Setup phase

Alice has a dataset $\mathtt{D}$ associated with attribute function Att (See data model in Section 3.2). In the setup phase, Alice generates a private key $\mathcal{K}$ by running key generating function KGen, and preprocesses the dataset $\mathtt{D}$ using data encoding algorithm DEnc with key $\mathcal{K}$, in order to produce an authentication tag $\mathtt{T}$. At the end of setup phase, Alice removes $\mathtt{D}$ and $\mathtt{T}$ from her storage after sending them to Bob.

$\mathsf{KGen}(1^\kappa)$

Alice

1. runs algorithm $\mathcal{G}(1^\kappa)$ to generate $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$, where $\mathcal{G}$ is defined in Section 3.4;
2. chooses $(s_{\ell,1}, s_{\ell,2}, s_{\ell,3})$ at random from $\mathbb{Z}_p^* \times \mathbb{Z}_q^* \times \{0,1\}^\kappa$ for each $\ell \in [d]$.

Let $\boldsymbol{s}_1 = (s_{1,1}, s_{2,1}, \ldots, s_{d,1})$, $\boldsymbol{s}_2 = (s_{1,2}, s_{2,2}, \ldots, s_{d,2})$ and $\boldsymbol{s}_3 = (s_{1,3}, s_{2,3}, \ldots, s_{d,3})$. Let $\mathcal{K} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, n, p, q, r, g_p, g_q, g_r)$. Output $\mathcal{K}$.

$\mathsf{DEnc}(\mathtt{D}, \mathcal{K})$

For each data point $\boldsymbol{i} \in \mathtt{D}$, Alice

1. chooses a number $\xi_{\boldsymbol{i}}$ at random from $\mathbb{Z}_q^*$;
2. computes a tag $\boldsymbol{t}_{\boldsymbol{i}}$ using the private key $\mathcal{K}$:

$$\boldsymbol{t}_{\boldsymbol{i}} \leftarrow \mathsf{Tag}_{\mathcal{K}}(\mathsf{Att}(\boldsymbol{i}), \boldsymbol{i}; \xi_{\boldsymbol{i}}). \tag{7}$$

Next, Alice sends dataset $\mathtt{D}$, tag values $\mathtt{T} = \{\boldsymbol{t}_{\boldsymbol{i}} : \boldsymbol{i} \in \mathtt{D}\}$ and modulus n to Bob, deletes the local copy of $\mathtt{D}$ and $\mathtt{T}$ from her storage, and keeps $(\mathcal{K}, n, N)$, where $N = |\mathtt{D}|$ is the size of the dataset.

5.3 Query phase

During the query phase, Alice has private key $\mathcal{K}$; Bob has dataset $\mathtt{D}$ and tag values $\mathtt{T}$. Alice and Bob share the modulus n and the size $N = |\mathtt{D}|$ of the dataset $\mathtt{D}$. The query phase consists of multiple query sessions. In a query session, Alice has a query $\textsc{Sum}(\mathbf{R})$, where $\mathbf{R} = [a_1, b_1] \times [a_2, b_2] \times \ldots [a_d, b_d] \subseteq [N]^d$. The result to the query w.r.t. dataset $\mathtt{D}$ is $\bigoplus_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}} \mathsf{Att}(\boldsymbol{i})$.

To process this query, Alice and Bob run an interactive algorithm ProVer (stands for "Prover-Verifier"), i.e. Alice running Ext interacts with Bob running Eval, where $\mathsf{ProVer} = \langle \mathsf{Eval}, \mathsf{Ext} \rangle$. The algorithm ProVer calls algorithm $\mathsf{CollRes}$ (stands for "Collusion-Resistant") as subroutine. Note that $\mathsf{CollRes}$ is a strengthened version of the plausible scheme presented in Section 1.

$\mathsf{CollRes}(\textsc{Sum}(\mathbf{R}))$

Precondition. Range $\mathbf{R} = \mathbf{A}_1 \times \mathbf{A}_2 \times \ldots \times \mathbf{A}_d$, where $\mathbf{A}_j \subseteq [N], j \in [d]$ (This is required by the subroutine $\mathsf{Compress}$).

A1: (Alice's first step) Alice

(a) chooses $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_d)$ at random from $\left(\mathbb{Z}_p^*\right)^d$;

(b) computes $\boldsymbol{\delta}$ using $sk = (n, g_p, \phi(n), \boldsymbol{s}_1)$:

$$\boldsymbol{\delta} \leftarrow \mathsf{Compress}_{sk}(\mathbf{R}, \boldsymbol{\alpha}). \tag{8}$$

Next, Alice sends $\mathbf{R}$ and $\boldsymbol{\delta}$ to Bob.

Note: (1) From $\boldsymbol{\delta}$, Bob is able to recover $\boldsymbol{\varpi} = \{\mathcal{E}(\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}, \boldsymbol{v}_{\boldsymbol{i}}, \boldsymbol{w}_{\boldsymbol{i}}, \mathbf{0}) : \boldsymbol{i} \in \mathbf{R}\}$ for some $\boldsymbol{v}_{\boldsymbol{i}} \in (G_q)^d$ and $\boldsymbol{w}_{\boldsymbol{i}} \in (G_r)^d$. We call $\boldsymbol{\varpi}$ *Help-Info*, since it will help Bob to produce a proof. Without considering randomness $\boldsymbol{v}_{\boldsymbol{i}}$'s and $\boldsymbol{w}_{\boldsymbol{i}}$'s, one may view $\boldsymbol{\varpi}$ as $\{\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha} : \boldsymbol{i} \in \mathbf{R}\}$. See Section 5.4 for details of function $\mathsf{Compress}$. (2) The secret random nonce $\boldsymbol{\alpha}$ prevents Bob from abusing this information for other queries, i.e. prevent collusion attack. (3) $\boldsymbol{s}_1$ is part of key $\mathcal{K}$.

B1: (Bob's first step) Bob

(a) computes the sum $\boldsymbol{X}$:

$$\boldsymbol{X} = \bigoplus_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}} \mathsf{Att}(\boldsymbol{i}); \tag{9}$$

(b) aggregates all Tag values for $\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}$:

$$\boldsymbol{\Psi}_1 \leftarrow \bigotimes_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}} \boldsymbol{t}_{\boldsymbol{i}} \mod n; \tag{10}$$

(c) uncompresses $\boldsymbol{\delta}$ with $pk = n$: for each $\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}$,

$$\boldsymbol{u}_{\boldsymbol{i}} \leftarrow \mathsf{Uncompress}_{pk}(\mathbf{R}, \boldsymbol{\delta}, \boldsymbol{i}); \tag{11}$$

(d) aggregates all $\boldsymbol{u}_{\boldsymbol{i}}$'s for $\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}$:

$$\boldsymbol{\Psi}_2 \leftarrow \bigotimes_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}} \boldsymbol{u}_{\boldsymbol{i}} \mod n. \tag{12}$$

Bob sends back to Alice $\boldsymbol{X}$ as query result, and $(\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2)$ as the proof.

Note: In Step B1(c), $\boldsymbol{u}_{\boldsymbol{i}} = \mathcal{E}(\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}, \boldsymbol{v}_{\boldsymbol{i}}, \boldsymbol{w}_{\boldsymbol{i}}, \mathbf{0})$ for some $\boldsymbol{v}_{\boldsymbol{i}} \in (G_q)^d$ and $\boldsymbol{w}_{\boldsymbol{i}} \in (G_r)^d$.

A2: (Alice's second step) Alice verifies the query result in the following way.

(a) Decode $\boldsymbol{\Psi}_1$ using $(\mathcal{K}, \boldsymbol{X})$, and decode $\boldsymbol{\Psi}_2$ using $(\mathcal{K}, \mathbf{0})$:

$$(\boldsymbol{U}_1, \boldsymbol{V}, \boldsymbol{W}) \leftarrow \mathcal{D}_{\mathcal{K}}(\boldsymbol{\Psi}_1, \boldsymbol{X})$$
$$(\boldsymbol{U}_2, \hat{\boldsymbol{V}}, \hat{\boldsymbol{W}}) \leftarrow \mathcal{D}_{\mathcal{K}}(\boldsymbol{\Psi}_2, \mathbf{0})$$

(b) Check whether the ITag component of $\boldsymbol{\Psi}_1$ is consistent with $\boldsymbol{\Psi}_2$ using the secret random nonce $\boldsymbol{\alpha}$:

$$\boldsymbol{U}_1 \circledast \boldsymbol{\alpha} \stackrel{?}{=} \boldsymbol{U}_2 \mod n; \tag{13}$$

(c) Check whether the binding factor g_q^{ξ}'s among different dimensions are consistent: Is there some $a \in \mathbb{Z}_n$, such that

$$\boldsymbol{V} \circledast \boldsymbol{s}_2^{-1} = a\boldsymbol{e} \mod n? \tag{14}$$

Note: (1) $\boldsymbol{e}$ is the d-dimensional vector with each dimension equal to 1. (2) $\boldsymbol{s}_2$ is part of key $\mathcal{K}$ and $\boldsymbol{s}_2^{-1} \otimes \boldsymbol{s}_2 = \boldsymbol{e} \mod q$.

(d) Alice extracts the CTag component from $\boldsymbol{\Psi}_1$:

$$\pi \leftarrow \boldsymbol{W}[1].$$

Note: This π is required by algorithm ProVer to prevent double counting or undercounting.

If all verifications succeed, Alice outputs $(\boldsymbol{X}, \pi)$. Otherwise, Alice output $\perp$.

Remark.

1. If Bob follows the protocol honestly, we have

$$\begin{aligned}
\boldsymbol{\Psi}_1 &= \mathcal{E}(\bigotimes_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{ITag}(\boldsymbol{i}), \bigotimes_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{DTag}(\mathbf{0};\xi_{\boldsymbol{i}}), \\
&\quad \bigotimes_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{CTag}(\boldsymbol{i}), \bigoplus_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{Att}(\boldsymbol{i})\boldsymbol{e}\ ); \\
\boldsymbol{U}_1 &= \bigotimes_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{ITag}(\boldsymbol{i}) \mod n; \\
\boldsymbol{U}_2 &= \left(\bigotimes_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{ITag}(\boldsymbol{i})\right) \circledast \boldsymbol{\alpha} \mod n; \\
\boldsymbol{V} &= \bigotimes_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{DTag}(\mathbf{0};\xi_{\boldsymbol{i}}) = \left(\left(\prod_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} g_q^{\xi_{\boldsymbol{i}}}\right)\boldsymbol{e}\right) \circledast \boldsymbol{s}_2 \mod n.
\end{aligned}$$

2. The equality test in equation (13) prevents Bob from bringing in points that do not satisfy the query condition into the computation of query result and its proof. For each point $\boldsymbol{i} \in \mathtt{D}\cap\mathbf{R}$, Bob has both $\boldsymbol{t}_{\boldsymbol{i}}$ (provided by Alice during setup) and[4] $\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}$ (provided by Alice during the current query session). For any point $\boldsymbol{i} \notin \mathtt{D}\cap\mathbf{R}$, it is infeasible for Bob to forge $\boldsymbol{t}_{\boldsymbol{i}}$ or $\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}$, without the private key $\mathcal{K}$ and secret random nonce $\boldsymbol{\alpha}$.
3. The equality test in equation (14) prevents Bob from forging a new point by mixing and combining different dimensions of different points. For example, without this verification, Bob may fool Alice with a faked point $(\boldsymbol{i}_2[1], \boldsymbol{i}_1[2], \boldsymbol{i}_1[3], \ldots, \boldsymbol{i}_1[d])$ with tag $(\boldsymbol{t}_2[1], \boldsymbol{t}_1[2], \boldsymbol{t}_1[3], \ldots, \boldsymbol{t}_1[d])$, where points $\boldsymbol{i}_1, \boldsymbol{i}_2 \in \mathtt{D}$ and $\boldsymbol{t}_1, \boldsymbol{t}_2$ are their corresponding Tag values. Furthermore, $\boldsymbol{s}_2$ prevents permutation on dimensions of the same data point.
4. As mentioned in the introduction, the above proof system can not ensure the completeness, due to the homomorphism of $\mathcal{E}$. Bob could cheat by double counting or undercounting some points in $\mathtt{D}\cap\mathbf{R}$. For example, Bob may claim the result is $2\sum_{\boldsymbol{i}\in \mathtt{D}\cap\mathbf{R}} \mathsf{Att}(\boldsymbol{i})$ by sending $(\boldsymbol{\Psi}_1 \otimes \boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2 \otimes \boldsymbol{\Psi}_2)$ as proof. Due to homomorphism of $\mathcal{E}$, this forged proof can pass Alice's verification in equations (13) and (14). Such attacks can be detected by algorithm ProVer.
5. The specially designed tag function Tag has the following properties:
   (a) It allows Bob to do aggregation, since $\mathcal{E}$ is homomorphic w.r.t. its inputs $\mathsf{ITag}(\boldsymbol{i}), \mathsf{DTag}(0,\xi)$, $\mathsf{CTag}(\boldsymbol{i})$ and $\boldsymbol{x}$ (See equation (5)).
   (b) It is unforgeable (See Lemma 1), and $\mathsf{Tag}(\boldsymbol{x}, \boldsymbol{i}; \xi)$ itself is not homomorphic w.r.t. its inputs.
   (c) It binds the point $\boldsymbol{i}$ and its attribute value $\boldsymbol{x}$ together in a secure way.
   (d) It allows compression of *Help-Info*. More precisely, ITag allows compression.

To execute an aggregate query with range $\mathbf{R} = [a_1, b_1] \times [a_2, b_2] \times \ldots [a_d, b_d] \subseteq [N]^d$, Alice runs algorithm Ext and interacts with Bob who runs algorithm Eval. To verify the completeness, Alice will divide the complement range $\mathbf{R}^{\mathsf{c}}$ carefully into many subregions, and invokes $\mathsf{CollRes}$ on each subregion. The details is given below.

[4]Without considering the randomization, one may view *Help-Info* $\boldsymbol{\varpi}$ as $\{\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha} : \boldsymbol{i} \in \mathbf{R}\}$. See Section 5.4.

$\mathsf{ProVer}(\textsc{Sum}(\mathbf{R})) = \langle \mathsf{Eval}(\mathtt{D}, \mathtt{T}, n), \mathsf{Ext}(N, \textsc{Sum}(\mathbf{R}), \mathcal{K})\rangle$

Precondition: Alice knows the value[5] $\pi^* = \prod_{\boldsymbol{i}\in \mathtt{D}} \mathsf{ctag}(\boldsymbol{i}[1])$. For each $\ell \in [d]$, let $\mathbf{R}_\ell = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_\ell, b_\ell] \times [N]^{d-\ell} \subseteq [N]^d$ be an ℓ-dimensional range. Let $\mathbf{R}_0$ be the universal set $[N]^d$.

A1: Alice interacts with Bob to simulate $\mathsf{CollRes}$ on $(d+1)$ queries $\textsc{Sum}(\mathbf{R}_d)$, and $\textsc{Sum}(\mathbf{R}_{\ell-1} \setminus \mathbf{R}_\ell)$, $\ell \in [d]$.

B1: Bob interacts with Alice to simulate $\mathsf{CollRes}$ on $(d+1)$ queries $\textsc{Sum}(\mathbf{R}_d)$, and $\textsc{Sum}(\mathbf{R}_{\ell-1} \setminus \mathbf{R}_\ell)$, $\ell \in [d]$.

A2: Alice obtains output $(\boldsymbol{X}_{d+1}, \pi_{d+1})$ of $\mathsf{CollRes}$ for query $\textsc{Sum}(\mathbf{R}_d)$, and $(\boldsymbol{X}_\ell, \pi_\ell)$ for query $\textsc{Sum}(\mathbf{R}_{\ell-1} \setminus \mathbf{R}_\ell)$, $\ell \in [d]$. Alice accepts Bob's reply and output $\boldsymbol{X}$, if

$$\prod_{1\le\ell\le d+1} \pi_\ell \stackrel{?}{\equiv} \pi^* \mod n. \tag{15}$$

Otherwise, Alice outputs $\perp$.

Note: The $(d+1)$ ranges $\mathbf{R}_{\ell-1} \setminus \mathbf{R}_\ell, \ell \in [d]$, and $\mathbf{R}_d$ forms a partition of the universal set $[N]^d$.

Remark. Simply, Alice may precompute and store π^*. We will explain more in Section 6.1.1.

Such division of the complement range $\mathbf{R}^{\mathsf{c}}$ is essential, since our compression algorithm $\mathsf{Compress}$ only works on range of form $\mathbf{A}_1 \times \mathbf{A}_2 \times \ldots \times \mathbf{A}_d$, where $\mathbf{A}_j \subseteq [N], j \in [d]$. Note that both $\mathbf{R}_\ell$ and $\mathbf{R}_{\ell-1} \setminus \mathbf{R}_\ell$ are of this form.

The main reasons that our scheme achieves efficient communication complexity are: (1) Alice can compress *Help-Info* sent from Alice to Bob, using an algorithm $\mathsf{Compress}$, and Bob can uncompress and recover *Help-Info*, using an algorithm $\mathsf{Uncompress}$; (2) The specially designed authentication tag functions Tag allows aggregation, so Bob just sends back some aggregated values as proof.

5.4 Compress and Uncompress

Now we construct the algorithms $\mathsf{Compress}$ and $\mathsf{Uncompress}$. Johnson *et al.* [49] (in Section 5 "Set Homomorphic Signatures" of that paper) proposed a redactable signature scheme, based on the signature scheme in Gennaro *et al.* [47]. We wrap this redactable signature scheme as two algorithms $\mathsf{compress}$ and $\mathsf{uncompress}$: Let $(n, g_p, \boldsymbol{s}_1)$ be as in Definition 4 and hash function $h(\cdot)$ be as in Definition 3, Let $pk \leftarrow n$; $sk \leftarrow (n, g_p, \phi(n), \boldsymbol{s}_1)$, for a set $\mathcal{I}$ of integers,

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

Then we define the compress/uncompress algorithms in vector form. Let $\mathbf{R} \subset [N]^d$. Let $\mathsf{Proj}_j(\mathbf{R})$ denote the projection of $\mathbf{R}$ on j-th dimension, i.e. $\mathsf{Proj}_j(\mathbf{R}) = \{\boldsymbol{i}[j] : \boldsymbol{i} \in \mathbf{R}\}$, $j \in [d]$.

- $\mathsf{Compress}_{sk}(\mathbf{R}, \boldsymbol{\alpha})$: for each $1 \le j \le d$, let $\delta_j \leftarrow \mathsf{compress}_{sk}(\mathsf{Proj}_j(\mathbf{R}))$. Let $\boldsymbol{v} \leftarrow (\delta_1, \delta_2, \ldots, \delta_d) \circledast \boldsymbol{s}_1$, $\boldsymbol{u}_1 \stackrel{\$}{\leftarrow} (G_q)^d$ and $\boldsymbol{u}_2 \stackrel{\$}{\leftarrow} (G_r)^d$. $\boldsymbol{\delta} \leftarrow (\boldsymbol{v} \circledast \boldsymbol{\alpha}) \otimes \boldsymbol{u}_1 \otimes \boldsymbol{u}_2$. Output $\boldsymbol{\delta}$.

[5]The dimension 1 as in expression $\boldsymbol{i}[1]$ is an arbitrary choice in $[d]$, and this choice will influence how to divide $\mathbf{R}^{\mathsf{c}}$.

- $\mathsf{Uncompress}_{pk}(\mathbf{R}, \boldsymbol{\delta}, \boldsymbol{i})$: for each $1 \le j \le d$, let set $\hat{\mathcal{I}}_j = \{\boldsymbol{i}[j]\}$ and let $\hat{\delta}_j \leftarrow \mathsf{uncompress}_{pk}(\mathsf{Proj}_j(\mathbf{R}), \boldsymbol{\delta}[j], \hat{\mathcal{I}}_j)$. Output $\hat{\boldsymbol{\delta}} \leftarrow (\hat{\delta}_1, \hat{\delta}_2, \ldots, \hat{\delta}_d)$.

Remark.

1. The operation $\boldsymbol{v} \circledast \boldsymbol{\alpha}$ aims to prevent reuse of $\boldsymbol{\delta}$, and permutation on dimensions of $\boldsymbol{\delta}$ (So does $\boldsymbol{s}_1$).
2. The operation $(\boldsymbol{v} \circledast \boldsymbol{\alpha}) \otimes \boldsymbol{u}_1 \otimes \boldsymbol{u}_2$ aims to randomize $\boldsymbol{v} \circledast \boldsymbol{\alpha}$ with a random vector $\boldsymbol{u}_1$ from $(G_q)^d$ and a random vector $\boldsymbol{u}_2$ from $(G_r)^d$.
3. The output $\hat{\boldsymbol{\delta}}$ of $\mathsf{Uncompress}_{pk}(\mathbf{R}, \boldsymbol{\delta}, \boldsymbol{i})$ equals to $\mathcal{E}(\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}, \boldsymbol{v}, \boldsymbol{w}, \mathbf{0})$ for some $\boldsymbol{v} \in (G_q)^d$ and $\boldsymbol{w} \in (G_r)^d$. From $\hat{\boldsymbol{\delta}}$, Alice can extract $\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}$ with private key.
4. Without considering those randomization factors, one may view the output $\boldsymbol{\delta}$ of $\mathsf{Compress}_{sk}(\mathbf{R}, \boldsymbol{\alpha})$ as the compression of $\{\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha} : \boldsymbol{i} \in \mathbf{R}\}$.
5. The range $\mathbf{R}$ has to be of form $\mathbf{A}_1 \times \mathbf{A}_2 \times \ldots \times \mathbf{A}_d$, where $\mathbf{A}_j \subseteq [N], j \in [N]$. Here $\mathbf{A}_j$ is not necessarily continuous.

We also provide an alternative constructions of compress and uncompress in Section 6.2.1.

5.5 Security

Our scheme is secure, under Definition 2. The proof is in Appendix C.

THEOREM 2. MAIA *is a $\mathcal{PRC}$ (Provable Remote Computing) protocol, w.r.t. d-dimensional aggregate* SUM *query, under Assumption 1 and Assumption 2.*

We defer the complexity analysis to the next section.

6. PERFORMANCE AND EXTENSIONS

In Section 6.1, we discuss some speedup methods and analyze the complexity of our scheme MAIA. In Section 6.2, we propose an alternative compression method, generalize the dataset and query, and extend our scheme to authenticate COUNT, MIN queries, and range selection query.

6.1 Performance

6.1.1 1D Aggregate Range Query

Suppose the range $\mathbf{R}$ is 1D, i.e. $\mathbf{R} = [a_1, b_1] \times [N]^{d-1}$. Due to normalization, the first dimension of points within 1D range $\mathbf{R}$ is predictable and continuous: $\{\boldsymbol{i}[1] : \boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}\} = [a_1, b_1]$. Hence, when running ProVer on query $\textsc{Sum}(\mathbf{R})$, Alice can set $\pi^* = \prod_{i \in [a_1, b_1]} \mathsf{ctag}(i)$, which she can compute by herself, and invoke only one instance of CollRes on $\textsc{Sum}(\mathbf{R})$. Furthermore, Alice does not need to provide *Help-Info* to Bob in CollRes, since Alice can generate the right hand side of equation (13) by herself. As a result, ProVer is much more efficient when the query range is 1D, compared with higher dimensional case.

6.1.2 ℓ-Dimensional Aggregate Range Query

The algorithms ProVer and CollRes can also handle query like $\sum_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}_\ell}^{\boldsymbol{x} = \mathsf{Att}(\boldsymbol{i})} \boldsymbol{x}[1]$, where $\mathbf{R}_\ell = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_\ell, b_\ell] \times [N]^{d-\ell}$ is an ℓ-dimensional range and $\ell \in [d]$. For such queries, we can do some optimizations: (1) Treat all vectors $\mathsf{Att}(\boldsymbol{i})$, $\boldsymbol{t}_{\boldsymbol{i}}, \boldsymbol{\delta}, \boldsymbol{X}, \boldsymbol{\Psi}_1$ and $\boldsymbol{\Psi}_2$ etc. as ℓ-dimensional vectors, and ignore the remaining $(d-\ell)$ dimensions. So the communication cost of CollRes will be $O(\ell)$ instead of $O(d)$. (2) To ensure completeness, check the relative complement $\mathbf{R}_1 \setminus \mathbf{R}_d$, instead of the absolute complement. So we save one function call $\mathsf{CollRes}(\mathbf{R}_0 \setminus \mathbf{R}_1)$. (3) Run all ℓ calls to CollRes simultaneously in parallel.

The parallel execution of ℓ instances of CollRes can save us a lot: (1) Alice only sends values of $a_1, b_1, \ldots, a_\ell, b_\ell$ to represent all of ℓ ranges $\mathbf{R}_\ell$ and $\mathbf{R}_{j-1} \setminus \mathbf{R}_j$, $j \in [2, \ell]$. (2) Recall that, on input $(\mathbf{R}, \boldsymbol{\alpha})$, Compress will separate each dimension of $\mathbf{R}$, and generate a *Help-Info* for each 1D range $[a_j, b_j]$. For the above ℓ ranges, Alice can only generate compressed *Help-Info* for 1D range $[a_j, b_j]$ and $[N] \setminus [a_j, b_j]$, $j \in [\ell]$, from which Bob is able to reconstruct *Help-Info* for each of ℓ ranges.

Note that the query only asks for the sum of the first dimension. For dimension $j \in [2, \ell]$, Bob may send back $g^{\boldsymbol{X}[j]} \mod n$ instead of the sum $\boldsymbol{X}[j]$ for the j-th dimension, when $\boldsymbol{X}[j]$ is too large, so that the communication cost is always independent of the size of dataset. Here g is some generator of group G provided by Alice.

6.1.3 Other speedup techniques

There are some other tradeoff or speedup techniques, including the following. (1) Batch executions of multiple queries: Alice may find the smallest union set $\mathbf{U}$ of all query ranges, and issue a new query with range $\mathbf{U}$ at the very beginning. Next, Alice authenticates other queries by checking complement w.r.t $\mathbf{U}$. (2) Precomputation: Partition the space into regions, and store necessary information for proof of completeness inside each region, so that Alice only need check the complement w.r.t. some union of regions that fully cover the query range. (3) For any rectangle range $\mathbf{R}$, Alice can find the smallest rectangle range $\bar{\mathbf{R}}$ with help of Bob, such that $\bar{\mathbf{R}} \subseteq \mathbf{R}$ and $\mathtt{D} \cap \mathbf{R} = \mathtt{D} \cap \bar{\mathbf{R}}$. Then Alice can generate *Help-Info* for $\bar{\mathbf{R}}$, instead of $\mathbf{R}$. In many scenarios, $\bar{\mathbf{R}}$ is much smaller than $\mathbf{R}$. For example, in Figure 2, such smallest rectangle range for $\mathbf{R}_1 = [3, 8] \times [2, 7]$ is $[4, 6] \times [4, 7]$. (4) Offline verifications with multiple queriers: Charlie sends query Q to Bob, and Bob returns $(\boldsymbol{X}, \sigma)$ to Charlie, where $\boldsymbol{X}$ is the result to query Q and σ is the signature of $(Q, \boldsymbol{X})$ under Bob's private key. Charlie may decide when (or whether) to ask Alice to do the verification.

6.1.4 Complexity Analysis

Taking the modifications in Section 6.1.1 and Section 6.1.2 into account, we analyze the complexity of our scheme. Let us consider query with range $\mathbf{R} \subseteq [N]^d$, which asks for the sum $\sum_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}} \mathsf{Att}(\boldsymbol{i})[1]$, where $\mathbf{R}$ is an ℓ-dimensional rectangular range, $\ell \in [d]$. To process this query, ProVer requires $\sum_{j \in [\ell]} O(j) = O(\ell^2)$ communication bits and $O(1)$ rounds.

When $\ell = 1$, the computation complexity per query is very efficient: $O(|S|)$ on both sides. Here S denotes the set of points satisfying the query condition. For $\ell \in [2, d]$, the computation complexity per query on Bob's side is $O(\ell N^2)$ (modular exponentiations). The dominant computation step is Uncompress in Step B1(c) of CollRes in Section 5.3. 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) of CollRes in Section 5.3. 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 without sacrificing communication complexity, 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 storage overhead on Bob's side, is $O(dN)$. The storage overhead on Alice's side, i.e. the key size, is $O(d)$.

The summary of complexities of MAIA compared with related schemes is given in Figure 1 in Section 1. Note that when $d > \log(dN)/\log\log N$, we have $dN \log N \leq \log^d N$.

6.2 Extensions

6.2.1 Alternative Compression and Decompression

Here we present an alternative constructions of ITag, Compress and Uncompress, using a binary interval tree [1].

Assume $N = 2^H$ for some integer H. Build a complete binary tree with N leaves, such that: (1) Each tree node $\mathtt{t}$ is associated with an interval, denoted as $\mathtt{t.int}$. For the j-th leaf $\mathtt{t}$ (counting from the left to right), $\mathtt{t.int} = \{j\}$, and for an internal node $\mathtt{t}$ with left child $\mathtt{t_l}$ and right child $\mathtt{t_r}$, $\mathtt{t.int} = \mathtt{t_l.int} \cup \mathtt{t_r.int}$. (2) Each tree edge $\mathtt{e}$ is associated with a public random prime, denoted as $\mathtt{e.p}$. (3) Each tree node $\mathtt{t}$ is associated with an attribute, denoted as $\mathtt{t.val}$. Let $(\mathtt{e_1}, \mathtt{e_2}, \ldots, \mathtt{e_u})$ be the path from the root to a node $\mathtt{t}$. Then $\mathtt{t.val} = \mathtt{r.val}^{\prod_{1 \leq j \leq u} \mathtt{e_j.p}} \mod n$, where $\mathtt{r}$ denotes the root of the tree.

Note ITag and Compress are just vector form of itag and compress. We construct itag, compress as below. (1) $\mathsf{itag}(i)$: Set $\mathtt{r.val}$ to be some generator of a subgroup of $\mathbb{Z}_n^*$. $\mathsf{itag}(i) = \mathtt{t.val}$, where $\mathtt{t}$ is the i-th leaf. (2) $\mathsf{compress}([a,b],\alpha)$: Given an interval $[a,b] \subset [N]$, find the minimum set $\{\mathtt{t_1}, \mathtt{t_2}, \ldots, \mathtt{t_v}\}$ of tree nodes, where $\mathtt{t_j.int}$'s, $j \in [v]$, form a partition of $[a,b]$. Output $\{\mathtt{t_j.val}^\alpha, j \in [v]\}$. Then a corresponding uncompress algorithm will recover $\{\ell_j.\mathtt{val}^\alpha : \ell_j$ is the j-th leaf, $j \in [a,b]\}$, using the public primes associated to edges.

This alternative approach reduces the complexity of uncompress from $O(L)$ to $O(\log L)$, where L is the length of the interval $[a,b]$, at the cost of growth of output size (i.e. communication bandwidth of MAIA) by a factor of $O(\log N)$. Furthermore, the dynamic nature of tree provides us a clue to support dynamic operations on the dataset, for example, adding or deleting a point. We will look into this direction in the future.

6.2.2 Generalization of Dataset and Query

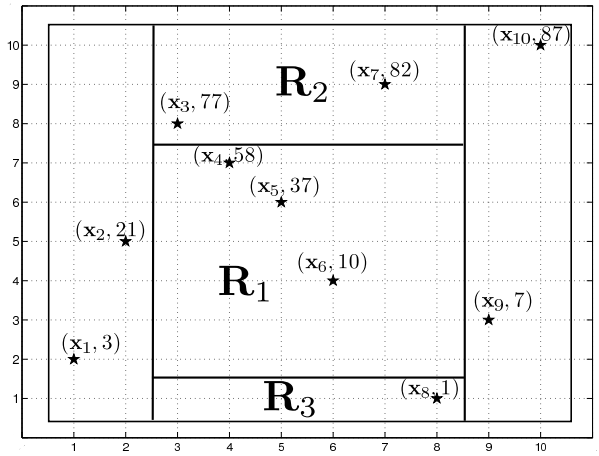
Table 3: The dataset $\hat{\mathtt{D}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_{10}\}$. For each $1 \leq m \leq 10$, we write $\boldsymbol{x}_m = (x_{m,1}, x_{m,2})$, and $(i_{m,1}, i_{m,2})$ is the rank vector of $\boldsymbol{x}_m$, i.e. $i_{m,1}$ is the rank of $x_{m,1}$ among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

	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$	$\boldsymbol{x}_4$	$\boldsymbol{x}_5$	$\boldsymbol{x}_6$	$\boldsymbol{x}_7$	$\boldsymbol{x}_8$	$\boldsymbol{x}_9$	$\boldsymbol{x}_{10}$
$x_{m,1}$	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

As mentioned previously, the dataset and query described in Section 3.2 are simplified for the convenience of presentation, and our scheme MAIA can handle more general case.

Let $\hat{\mathtt{D}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \subset \mathbb{N}^d$ be a dataset. An aggregate Sum query with range $\mathbf{B} \subset \mathbb{N}^d$ over dataset $\hat{\mathtt{D}}$ asks for the sum $\bigoplus_{\boldsymbol{x} \in \hat{\mathtt{D}} \cap \mathbf{B}} \boldsymbol{x}$.

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



We can authenticate such query in the following way. (1) Normalize [38] the dataset $\hat{\mathtt{D}}$: for each dimension $\ell \in [d]$, sort all $\boldsymbol{x}_j$'s along the ℓ-th dimension. Then associate a d-dimensional rank vector $\boldsymbol{i}_j$ for each point $\boldsymbol{x}_j$, $j \in [N]$. Let $\mathtt{D} = \{\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_N\}$. Table 3 and Figure 2 show a normalized 2D dataset. (2) Define a vector-valued attribute function $\mathsf{Att} : \mathtt{D} \to \hat{\mathtt{D}}$, such that $\mathsf{Att}(\boldsymbol{i}) = \boldsymbol{x}$, where $\boldsymbol{i} \in \mathtt{D}$ is the rank vector of $\boldsymbol{x} \in \hat{\mathtt{D}}$. (3) Set up for dataset $\mathtt{D}$ with attribute function Att, by running algorithms KGen and DEnc. (4) Translate query w.r.t. $\hat{\mathtt{D}}$ to query w.r.t. $\mathtt{D}$: find a range $\mathbf{R} \subseteq [N]^d$, such that $\bigoplus_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}} \mathsf{Att}(\boldsymbol{i}) = \bigoplus_{\boldsymbol{x} \in \hat{\mathtt{D}} \cap \mathbf{B}} \boldsymbol{x}$. Alice can do this translation online with help of Bob and is able to verify the correctness of the translation by authenticating some tag values returned by Bob. (5) Run interactive algorithm ProVer to authenticate query $\textsc{Sum}(\mathbf{R})$ w.r.t. $\mathtt{D}$.

We remark that the translation of queries can be done with $O(d)$ communication bits and in $O(d)$ computation time, and this generalization will not change the complexity of our scheme asymptotically.

6.2.3 Authentication of Count, Min Queries

Besides Sum query, we can generalize our scheme to authenticate aggregate Count query and Min query. Count is just a special case of Sum: summing a constant attribute. Min query can be converted to Count query: For any set S of numbers, $c = \min S \Leftrightarrow c \in S \wedge |S| = |\{x : x \in S \wedge x \geq c\}|$. To process a Min query with range $\mathbf{B}$, which asks for $\min_{\boldsymbol{x} \in \hat{\mathtt{D}} \cap \mathbf{B}} \boldsymbol{x}[1]$, Alice and Bob interact in this way: (1) Bob sends back the query result c with the proof that $c \in S$, where $S = \{\boldsymbol{x}[1] : \boldsymbol{x} \in \hat{\mathtt{D}} \cap \mathbf{B}\}$; (2) Alice generates two queries[6] $\textsc{Count}(\mathbf{B})$ and $\textsc{Count}(\mathbf{B} \cap [c, +\infty) \times \mathbb{N}^{d-1})$ w.r.t. $\hat{\mathtt{D}}$, which ask for the sizes of set S and set $\{x : x \in S \wedge x \geq c\}$, respectively. Next, Alice issues the two queries to Bob and

[6]Note that with the generalization in Section 6.2.2, our scheme can authenticate aggregate such queries.

gets two count numbers with proofs. If all proofs are valid and the two count numbers are equal, Alice believes c is the minimum value.

Similarly, MAX, MEDIAN, QUANTILE queries can be authenticated by converting to COUNT query. So we conclude that: our scheme MAIA can authenticate all of SUM, COUNT, MIN, MAX, MEDIAN and QUANTILE aggregate queries.

6.2.4 Multidimensional Range Selection Query

It is straightforward to extend our scheme to authenticate range selection query with range $\mathbf{R}$, which asks for the set $\{\mathsf{Att}(\boldsymbol{i}) : \boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}\}$, with linear communication overhead: (1) Bob sends back $\{(\mathsf{Att}(\boldsymbol{i}), \boldsymbol{i}, \boldsymbol{t_i}) : \boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}\}$; (2) Authenticate a COUNT query with range $\mathbf{R}$ to ensure completeness.

Next, we brief a method to authenticate range selection query with $O(d^2)$ communication overhead. The key idea is that we can extend CollRes to authenticate weighted sum.

Suppose Alice wants to authenticate a range selection query with range $\mathbf{R}$, which asks for $S = \{\boldsymbol{x}[1] : \boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}, \boldsymbol{x} = \mathsf{Att}(\boldsymbol{i})\}$. She does it in three steps: (1) Issue the range selection query to Bob, and Bob returns the set S without any proof; (2) Issue a COUNT query with range $\mathbf{R}$ and verify the result; (3) Simulate (modified) CollRes with Bob to authenticate a weighted sum query with range $\mathbf{R}$. More precisely, Alice asks for the weighted sum $\boldsymbol{X}[j] \leftarrow \sum_{\boldsymbol{i} \in \mathtt{D} \cap \mathbf{R}}^{\boldsymbol{x} = \mathsf{Att}(\boldsymbol{i})} \boldsymbol{x}[j]\ \mathcal{H}(\boldsymbol{x}[1])$, $j \in [d]$, with $\mathcal{H}(\boldsymbol{x}[1])$ as weights, where $\mathcal{H}$ is some collision-resistant hash function. The modification to CollRes is: (a) In equation (9), replace $\mathsf{Att}(\boldsymbol{i})$ with $\mathsf{Att}(\boldsymbol{i}) \otimes (\mathcal{H}(\mathsf{Att}(\boldsymbol{i})[1])\boldsymbol{e})$, where $\boldsymbol{e}$ is the d-dimensional vector with each dimension equal to 1; (b) Modify equations (10) and (12) accordingly, by "raising" $\boldsymbol{t_x}$ and the output of Uncompress to power $\mathcal{H}(\mathsf{Att}(\boldsymbol{i})[1])\boldsymbol{e}$ (See equation (3) for vector exponentiation). (c) Modify Alice's verification steps accordingly, and add one more test[7] $\boldsymbol{X}[1] \stackrel{?}{=} \sum_{x \in S} x\ \mathcal{H}(x)$. We save the details.

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

We remark that the generalization in Section 6.2.2 can be applied here, so that Alice can authenticate range selection query w.r.t. $\hat{\mathtt{D}} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, which asks for $\{\boldsymbol{x}[1] : \boldsymbol{x} \in \hat{\mathtt{D}} \cap \mathbf{B}\}$. The complexity of authentication of range selection query using our modified scheme is shown in Table 1 in Section 1.

7. CONCLUSION

We proposed a scheme to authenticate aggregate range query over static multidimensional dataset, and the communication complexity (in term of bits) is $O(d^2)$ (d is the dimension) and independent of the query range size and the dataset size. Aggregate operations that our scheme can support 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. We will look into the possibility to support dynamic operations on the dataset, like adding or deleting a point, using a tree based compression method.

[7] If $\boldsymbol{X}$ is too large, Bob can just send back $(g\boldsymbol{e}) \circledast \boldsymbol{X}$ for some generator g.

8. 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] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC '09: Annual ACM symposium on Theory of computing. (2009) 169–178

[12] 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

[13] 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

[14] 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

[15] Bowers, K.D., Juels, A., Oprea, A.: Hail: a high-availability and integrity layer for cloud storage. In: CCS '09: ACM Conference on Computer and Communications Security. (2009) 187–198

[16] Dodis, Y., Vadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification. In: TCC '09: Theory of Cryptography. (2009) 109–127

[17] Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In: ASIACRYPT '09: Annual International Conference on the Theory and Application of Cryptology and Information Security. (2009) 319–333

[18] Juels, A., Kaliski, Jr. B. S.: Pors: proofs of retrievability for large files. In: CCS '07: ACM conference on Computer and Communications Security. (2007) 584–597

[19] 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
[20] 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
[21] 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
[22] Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.G.: Authentic data publication over the internet. J. Comput. Secur. **11**(3) (2003) 291–314
[23] 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
[24] Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. Trans. Storage **2**(2) (2006) 107–138
[25] Pang, H.H., Tan, K.L.: Verifying completeness of relational query answers from online servers. ACM Trans. Inf. Syst. Secur. **11**(2) (2008) 1–50
[26] Sion, R.: Query execution assurance for outsourced databases. In: VLDB '05: International conference on Very large data bases. (2005) 601–612
[27] Cheng, W., Tan, K.L.: Query assurance verification for outsourced multi-dimensional databases. J. Comput. Secur. **17**(1) (2009) 101–126
[28] 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
[29] 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
[30] 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
[31] 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
[32] 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
[33] Pang, H.H., Zhang, K.M.: Scalable verification for outsourced dynamic databases. In: VLDB '09: International Conference on Very Large Data Bases. (2009)
[34] 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
[35] 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.
[36] Shamir, A.: How to share a secret. Commun. ACM **22**(11) (1979) 612–613
[37] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO '91: Advances in Cryptology. (1991) 129–140
[38] Preparata, F.P., Shamos, M.I.: Computational geometry: an introduction. (1985)
[39] 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
[40] González 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
[41] Yamamura, A., Saito, T.: Private information retrieval based on the subgroup membership problem. In: ACISP '01: Australasian Conference on Information Security and Privacy. (2001) 206–220
[42] Gjøsteen, K.: Subgoup membership problems and public key cryptosystems. PhD thesis, NTNU (2004)
[43] Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: TCC '05: Theory of Cryptography. (2005) 325–341
[44] Brown, J., Nieto, J.M.G., Boyd, C.: Concrete chosen-ciphertext secure encryption from subgroup membership problems. In: CANS '06: Cryptology and Network Security. (2006) 1–18
[45] Gjøsteen, K.: Symmetric subgroup membership problems. In: PKC '05: Public Key Cryptography. (2005) 104–119
[46] Hofheinz, D., Kiltz, E.: The group of signed quadratic residues and applications. In: CRYPTO '09: Advances in Cryptology. (2009) 637–653
[47] 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
[48] Goldreich, O.: Foundations of Cryptography: Volume 1. (2006)
[49] 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
[50] Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptol. **17**(4) (2004) 297–319
[51] 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
[52] Hacigümüs, H., Iyer, B.R., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: DASFAA. (2004) 125–136
[53] 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
[54] 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
[55] Hinek, M.J.: On the security of multi-prime rsa. Journal of Mathematical Cryptology **2**(2) (2008) 117–147
[56] Miller, G.L.: Riemann's hypothesis and tests for primality. In: STOC '75: ACM Symposium on the Theory of Computing. (1975) 234–239
[57] Xu, J., Chang, E.C.: Authenticating Aggregate Range Queries over Multidimensional Dataset. Cryptology ePrint Archive, Report 2010/050 `http://eprint.iacr.org/`.

APPENDIX

We remark that **the design of Tag function is conservative and there may be some redundancy in our construction**.

A. IDEA OF PROOF OF THEOREM 2

Here we brief the idea to prove Theorem 2. The security model is given in Definition 2. The correctness part is straightforward. We focus on the soundness part.

Let $\mathcal{R}$ be the variant logarithm function as defined in Assumption 2. We define $\mathcal{P}$ and $\mathcal{Q}$ similar to $\mathcal{R}$. Define function $\mathcal{P} : G \to \mathbb{Z}_p$, such that for any $(x, y, z) \in \mathbb{Z}_p \times \mathbb{Z}_q \times \mathbb{Z}_r$,

$\mathcal{P}(g_p^x g_q^y g_r^z) \stackrel{\text{def}}{=} x$. Define function $\mathcal{Q}: G \to \mathbb{Z}_q$, such that for any $(x,y,z) \in \mathbb{Z}_p \times \mathbb{Z}_q \times \mathbb{Z}_r$, $\mathcal{Q}(g_p^x g_q^y g_r^z) \stackrel{\text{def}}{=} y$.

CLAIM 1. *Assumption 2 holds for $\mathcal{Q}$ and $\mathcal{R}$*

During all interactions with Alice (i.e. adversary's learning phase), Bob receives $\boldsymbol{i}$'s, $\mathsf{Att}(\boldsymbol{i})$, values a_j's,b_j's which specify query range, and a lot of elements from group G, including tag values $\boldsymbol{t_i}$ and compressed *Help-Info* $\boldsymbol{\delta}$. For any element $g_p^u g_q^v g_r^w \in G$ Bob receives, both v and w are (cryptographicly secure) pseudo-random: (1) In a tag value $\boldsymbol{t_i}$, ξ and $s_{\ell,2}$, $\ell \in [d]$, are true random and $f_s(\cdot)$ is *PRF*; (2) In each compressed *Help-Info* $\boldsymbol{\delta} = (\boldsymbol{v} \circledast \boldsymbol{\alpha}) \otimes \boldsymbol{u}_1 \otimes \boldsymbol{u}_2$, where $\boldsymbol{v} \in (G_p)^d$, $\boldsymbol{u}_1 \in (G_q)^d$ and $\boldsymbol{u}_2 \in (G_r)^d$, both $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are true random. See Section 5.4 for $\mathsf{Compress}$. Hence, the prerequisite of Assumption 2 is satisfied for both $\mathcal{Q}$ and $\mathcal{R}$.

CLAIM 2. *It is hard for a dishonest Bob to pass tests in equations* (13)(14)*, beyond homomorphism of $\mathcal{E}$.*

1. tag and itag are unforgeable, by Lemma 1 and [47]. The details of the proof will require Assumption 1.
2. One may view $\mathsf{itag}(i)^\alpha$ as a variant version of multi-prime RSA signature[8] [55] on message $\mathsf{itag}(i)$, with private key α and public key $\alpha^{-1} \mod p$, where both public key and private key are kept securely from adversaries. The definition of security of homomorphic signature scheme (intuitively, unforgeability except for homomorphism) is given in [49]. For simplicity, we ignore the requirement of security of multi-prime RSA signature in the statement of the main theorem.
3. Due to the binding factor g_q^ξ among different dimensions (Equation (14) checks the binding factor), $\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}$ can not be forged beyond homomorphism. This can be proved by applying Assumption 2 on $\mathcal{Q}$. Note this homomorphism corresponds to double counting or undercounting points that satisfy the query condition.

CLAIM 3. *If an adversary cheats by double counting and undercounting, and passes the verification in* ProVer*, then he can break Assumption 2.*

Suppose the adversary (i.e. malicious Bob) double counts points in the multiset $S_1 = \{\boldsymbol{i}_{u,j} : \boldsymbol{i}_{u,j} \in \mathtt{D}, j \in [m]\}$ and undercounts points in set $S_2 \subseteq \mathtt{D}$, where $S_1 \neq S_2$, and passes the verification. That means, $\eta_1 = \eta_2$, where

$$\eta_1 = \prod_{\boldsymbol{i} \in S_1} \mathsf{ctag}(\boldsymbol{i}[1]) \mod n; \quad \eta_2 = \prod_{\boldsymbol{i} \in S_2} \mathsf{ctag}(\boldsymbol{i}[1]) \mod n.$$

According to the definition of tag, itag and $\mathcal{R}$, $\log_{g_r} \eta_1 = \mathcal{R}(\prod_{\boldsymbol{i} \in S_1} \boldsymbol{t_i}[1])$ and $\log_{g_r} \eta_2 = \mathcal{R}(\prod_{\boldsymbol{i} \in S_2} \boldsymbol{t_i}[1])$. Hence,

$$\mathcal{R}(\frac{\prod_{\boldsymbol{i} \in S_1} \boldsymbol{t_i}[1]}{\prod_{\boldsymbol{i} \in S_2} \boldsymbol{t_i}[1]}) = \mathcal{R}(\prod_{\boldsymbol{i} \in S_1} \boldsymbol{t_i}[1]) - \mathcal{R}(\prod_{\boldsymbol{i} \in S_2} \boldsymbol{t_i}[1]) = 0.$$

Let $A = \frac{\prod_{\boldsymbol{i} \in S_1} \boldsymbol{t_i}[1]}{\prod_{\boldsymbol{i} \in S_2} \boldsymbol{t_i}[1]} \mod n$. If $A = 1$, this will lead to an efficient method to factorize n. See Lemma 6 in Appendix C. If $A \neq 1$, then $(A, 0)$ is a contradiction to Assumption 2. Note that the Assumption 2 implies that it is hard to factorize n.

[8]Or a symmetric key version of BLS signature [50].

B. SECURITY OF AUTHENTICATION TAG FUNCTION

B.1 Proof of Lemma 1

LEMMA 1. *Let k be a key as in Definition 3. Given $S = \{(x_i, i, t_i) : \xi \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*, t_i \leftarrow \mathsf{tag}_k(x_i, i; \xi))\}_{1 \le i \le N}$ and the modulus n, it is computationally hard to forge a tuple $(y, j, t) \notin S$, such that $\gamma_1(t, g_p^y) = \mathsf{itag}_k(j)$ and $\gamma_3(t) = \mathsf{ctag}_k(j)$, under Assumption 1 and Assumption 2.*

Let $X \sim Y$ denote that X and Y are computationally indistinguishable.

PROOF. Suppose that some PPT adversary $\mathcal{A}$ takes as input a set (S, n), and outputs $(y, j, t) \notin S$, such that $\gamma_1(t, g_p^y) = \mathsf{itag}_k(j)$ and $\gamma_3(t) = \mathsf{ctag}_k(j)$ with non-negligible probability. Note Assumption 2 is applicable for $\mathcal{R}$.

Case 1: $j \in [N]$. Let $(x_j, j, t_j) \in S$ be a tuple in the set S. Since $(y, j, t) \notin S$, we have $(y, t) \neq (x_j, t_j)$.

Case 1.1: $t_j \neq t$. From $\gamma_3(t) = \mathsf{ctag}_k(j) = \gamma_3(t_j)$, we have

$$\mathcal{R}(t) = \log_{g_r} \gamma_3(t) = \log_{g_r} \gamma_3(t_j) = \mathcal{R}(t_j).$$

Let $A = t_j\ t^{-1} \mod n$. Let $A \not\equiv 1 \pmod n$, and $B = \mathcal{R}(A) = 0$. The tupe (A, B) is a contradiction with Assumption 2.

Case 1.2: $t_j = t$ and $x_j \neq y$. From $t_j = t$, we have,

$$\begin{aligned} & \mathcal{P}(t_j) = \mathcal{P}(t) \\ \Rightarrow\ & x_j + \frac{s_1}{h(j)} \equiv y + \frac{s_1}{h(j)} \pmod p \\ \Rightarrow\ & x_j \equiv y \pmod p. \end{aligned}$$

Hence, $p | (x_j - y)$. Simulate $\mathcal{A}$ again, we can get another multiple of p, then apply the GCD algorithm to obtain p. Next, we show that with the knowledge of p, the adversary can break Assumption 1.

Assumption 1 can also be applied on group G_p or G_q. Let $Y_p = \{y_i : y_i \stackrel{\$}{\leftarrow} G_p\}_{1 \le i \le N}$, $Y_q = \{y_i : y_i \stackrel{\$}{\leftarrow} G_q\}_{1 \le i \le N}$, and $\mathcal{X} = \{y_i : y_i \stackrel{\$}{\leftarrow} G\}_{1 \le i \le N}$. Let $S_t = \{t_i : \exists x, i, (x, i, t_i) \in S\}$. Since ξ's are true random, by Assumption 1 with multiple sampling (**Theorem 3.2.6** [48]), $S_t \sim Y_q \sim \mathcal{X} \sim Y_p$. Let $\widetilde{Y}_q = \{(x_i, i, y) : y \in Y_q\}$, $\widetilde{Y}_p = \{(x_i, i, y) : y \in Y_p\}$, and $\widetilde{\mathcal{X}} = \{(x_i, i, y) : y \in \mathcal{X}\}$. Note that $S = \{(x_i, i, t_i) : t_i \in S_t\}$. We have $S \sim \widetilde{Y}_q \sim \widetilde{\mathcal{X}} \sim \widetilde{Y}_p$, since ξ's in t_i's are true random independent of x_i's and i's.

However, the adversary can run $\mathcal{A}$ with input (S, n) and apply GCD algorithm to find p. With the value of p, the adversary can distinguish S and $\widetilde{Y}_p$. Contradiction!

Case 2: $j \notin [N]$. Note that in our use of tag in this paper, this case will not be considered as a valid forgery, since we are only interested in normalized value $i \in [N]$, and (y, j, t) with $j \notin [N]$ will be rejected by Alice immediately.

Note that in the above proof, we do not utilize the condition $\gamma_1(t, g_p^y) = \mathsf{itag}_k(j)$. As mentioned previously, our design is conservative and may contain some redundancy. □

B.2 Security of tag function Tag

Let $\mathtt{D} = \{\boldsymbol{i}_1, \boldsymbol{i}_1, \ldots, \boldsymbol{i}_N\} \subset [N]^d$ be a dataset as in Section 3.2. That is, $\mathtt{D}$ satisfies property: For any $j \in [d]$, $\{\boldsymbol{i}[j] : \boldsymbol{i} \in \mathtt{D}\} = [1, N]$.

Let $\mathcal{K}$ be a key as in Definition 4. We construct a set S as below: For any $\boldsymbol{i} \in \mathtt{D}$,

1. let $\boldsymbol{x_i}$ be any element in $\mathbb{N}^d$;
2. choose $\xi_{\boldsymbol{i}}$ at random from $\mathbb{Z}_q^*$;
3. compute $\boldsymbol{t_i} \leftarrow \mathsf{Tag}_{\mathcal{K}}(\boldsymbol{x}, \boldsymbol{i}; \xi)$.

Let $S = \{(\boldsymbol{x_i}, \boldsymbol{i}, \boldsymbol{t_i}) : \boldsymbol{i} \in \mathtt{D}\}$.

LEMMA 3. *Given (S, n), it is hard to forge $(\boldsymbol{y}, \boldsymbol{j}, \boldsymbol{t}) \notin S$, such that (1) $\boldsymbol{\Gamma}_1(\boldsymbol{t}, \boldsymbol{y}) = \mathsf{ITag}_{\mathcal{K}}(\boldsymbol{j})$, (2) there exists $a \in \mathbb{Z}_n$, $\boldsymbol{\Gamma}_2(\boldsymbol{t}, \boldsymbol{y}) \circledast \boldsymbol{s}_2 = a\boldsymbol{e} \mod n$, (3) $\boldsymbol{\Gamma}_3(\boldsymbol{t}) = \mathsf{CTag}_{\mathcal{K}}(\boldsymbol{j})$, (4) $\boldsymbol{j} \in [N]^d$, under Assumption 1 and Assumption 2.*

PROOF. (Sketch) Without considering the binding factor g_q^{ξ}'s, this problem can be considered as d instances of the problemin Lemma 1.

The adversary either (1) forges a tuple (y, j, t) as in Lemma 1, or (2) permutate on dimensions of a valid tuple in S; (3) mix and combine different dimensions of different valid tuples in S.

Case (1) is prevented by Lemma 1; Case (2) is prevented by $\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3$, which are parts of key $\mathcal{K}$; Case (3) is prevented by the binding factor. □

C. PROOF OF THEOREM 2

The framework of proof is given in Appendix A. Here we fill in the missing parts.

C.1 For Claim 2

We extract out the verification in equation (13) and treat as a private key signature scheme.

LEMMA 4. *Let $S = \{(X_i Y_{i,1} Z_{i,1} \mod n, X_i^{\alpha} Y_{i,2} Z_{i,2} \mod n) : X_i \in G_p, Y_{i,j} \in G_q, Z_{i,j} \in G_r, j \in \{1,2\}\}_{1 \le i \le m}$, where m is some polynomial of the seucrity parameter κ, i.e. the bit length of the modulus n. Given (S, n), it is hard to forge a valid pair $(XY_1Z_1 \mod n, X^{\alpha}Y_2Z_2 \mod n) \notin S$, $X \in G_p, Y_1, Y_2 \in G_q, Z_1, Z_2 \in G_r$, beyond multiplicative homomorphism, if Multiple-Prime RSA [55] is unforgeable beyond multiplicative homomorphism.*

PROOF. Let $(X, Y, Z, U) \xleftarrow{\$} (G_p, G_q, G_r, G)$. By Assumption 1, $X \sim U, Y \sim U, Z \sim U$. Then $XYZ \sim XU \sim U \sim X$ (Here XYZ means $X \times Y \times Z \mod n$.).

By multiple sampling (**Theorem 3.2.6** [48]), we have

$$(X, X^{\alpha}) \sim (XY_1Z_1, X^{\alpha}Y_2Z_2) \text{ and } (X, X^{\alpha}) \sim (U, U^{\alpha}).$$

Hence,

$$(XY_1Z_1, X^{\alpha}Y_2Z_2) \sim (U, U^{\alpha}).$$

Note that $(X, X^{\alpha}) \sim (U, U^{\alpha})$. (U, U^{α}) is the multiple-Prime RSA signature scheme without revealing the public key. □

Let $\mathcal{E}$ and $\mathcal{K}$ be as in Section 4. Let $\mathtt{D}$ and S be the as in Section B.2. Note that $\boldsymbol{t_i} = \mathcal{E}_{\mathcal{K}}(\mathsf{ITag}_{\mathcal{K}}(\boldsymbol{i}), \boldsymbol{v_i}, \boldsymbol{w_i}, \boldsymbol{x_i})$, where $\boldsymbol{v_i} \in (G_q)^d$ and $\boldsymbol{w_i} \in (G_r)^d$.

Let $\mathbf{R} \subseteq [N]^d$ be range. For each $\boldsymbol{i} \in \mathbf{R}$,

1. choose $\boldsymbol{\alpha}$ at random from $(\mathbb{Z}_p^*)^d$;
2. let $\boldsymbol{u_i} \leftarrow \mathcal{E}_{\mathcal{K}}(\mathsf{ITag}_{\mathcal{K}}, \boldsymbol{v_i}, \boldsymbol{w_i}, \boldsymbol{0})$, where $\boldsymbol{v_i} \in (G_q)^d$ and $\boldsymbol{w_i} \in (G_r)^d$.

Let $\bar{S} = \{\boldsymbol{u_i} : \boldsymbol{i} \in \mathbf{R}\}$. The following lemma is for the CLAIM 2.

LEMMA 5. *Given $(S, \bar{S}, n)$, it is hard to forge a tuple $(\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2, \boldsymbol{X})$ beyond homomorphism of $\mathcal{E}$, where $\boldsymbol{\Psi}_1 = \mathcal{E}_{\mathcal{K}}(\boldsymbol{U}, \boldsymbol{V}_1, \boldsymbol{W}_1, \boldsymbol{X})$, $\boldsymbol{\Psi}_2 = \mathcal{E}_{\mathcal{K}}(\boldsymbol{U} \circledast \boldsymbol{\alpha}, \boldsymbol{V}_2, \boldsymbol{W}_2, \boldsymbol{0})$, and $\boldsymbol{U} \in (G_p)^d$, $\boldsymbol{V}_1, \boldsymbol{V}_2 \in (G_q)^d, \boldsymbol{W}_1, \boldsymbol{W}_2 \in (G_r)^d$, such that $(\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2, \boldsymbol{X})$ passes verification in equations* (13) (14).

PROOF. (Sketch) Without considering the binding factor g_q^{ξ}'s, this problem can be considered as d instances of the problem in Lemma 4.

The adversary either (1) forges a Tag value for some $\boldsymbol{i} \in \mathbf{R} \setminus \mathtt{D}$ or $\mathcal{E}_{\mathcal{K}}(\mathsf{ITag}(\boldsymbol{i}) \circledast \boldsymbol{\alpha}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{0})$ for some $\boldsymbol{i} \in \mathtt{D} \setminus \mathbf{R}$, or (2) forges a tuple as in Lemma 4, or (3) permutate on dimensions of a valid tuple, or; (4) mix and combine different dimensions of different valid tuples.

Case (1) is prevented since Tag and (loosely speaking) ITag are unforgeable; Case (1) is prevented by Lemma 4; Case (2) is prevented by $\boldsymbol{s}_1, \boldsymbol{s}_2$ (Note Alice does not have enough information to check CTag with $\boldsymbol{s}_3$, so there is no $\boldsymbol{s}_3$ here), which are parts of key $\mathcal{K}$; Case (3) is prevented by the binding factor, which is checked in equation (14). □

C.2 For Claim 3

Let group G be as in Assumption 1.

LEMMA 6. *Given a set $S = \{y_i : y_i \xleftarrow{\$} G\}_{1 \le i \le m}$ and the modulus n, it is hard to find a set $S_1 \subset S$ and a multiset $S_2 \subset S$, such that (1) $S_1 \neq S_2$ and (2) $\prod_{y \in S_1} y = \prod_{y \in S_2} y \mod n$, assuming factoriztion of n is hard.*

Here multiset $S_2 \subset S$, means any element of S_2 is in S, without considering the number of duplications of elements in S_2.

Note that by Assumption 1, $\{\boldsymbol{t_i}[j] : \boldsymbol{t_i} \in \mathtt{T}, j \in [d]\}$ is indistinguishable from uniform distribution over group G, wher $\mathtt{T}$ is the set of tag values generated by DEnc (See Section 5.2).

PROOF. Assume that an algorithm $\mathcal{A}$ with inputs (S, n) can output such two sets with non-negligible probability.

1. Choose g at random from $\mathbb{Z}_n$. With high chance (non-negligible, more presicely, about 1/8), g is a generator of G.
2. For $1 \le i \le m$, choose x_i at random from $[1, n^2]$ and compute $y_i \leftarrow g^{x_i} \mod n$ (Here we purposely choose the domain with size much larger than $\lambda(n)$. So y_i only carries partial information of x_i).
3. Let $S = \{y_i : 1 \le i \le m\}$. Run $\mathcal{A}(S, n)$ to get sets S_1 and S_2.
4. Compute $W_1 \leftarrow \sum_{y_i \in S_1} x_i$ and $W_2 \leftarrow \sum_{y_i \in S_2} x_i$.
5. We have $\lambda(n) | (W_1 - W_2)$ and with high chance $W_1 - W_2$ is not 0 (This is because we choose a domain of size larger than n, and $\mathcal{A}$ has access to only a partial information of x_i's). From $\lambda(n)$, we can factorize n [56].

□