# Authenticating Aggregate Range Queries over Dynamic Multidimensional Dataset

Jia XU
jiaxu2001@gmail.com

National University of Singapore

**Abstract.** We are interested in the integrity of the query results from an outsourced database service provider. Alice passes a set $\mathbf{D}$ of $d$-dimensional points, together with some authentication tag $\mathbf{T}$, to an untrusted service provider Bob. Later, Alice issues some query over $\mathbf{D}$ to Bob, and Bob should produce a query result and a proof based on $\mathbf{D}$ and $\mathbf{T}$. Alice wants to verify the integrity of the query result with the help of the proof, using only the private key. Xu J. *et al.* [25] proposed an authentication scheme to solve this problem for multidimensional aggregate range query, including SUM, COUNT, MIN, MAX and MEDIAN, and multidimensional range selection query, with $O(d^2)$ communication overhead. However, their scheme only applys to *static* database. This paper extends their method to support *dynamic* operations on the dataset, including inserting or deleting a point. The communication overhead of our scheme is $O(d^2 \log N)$, where $N$ is the number of data points in the dataset.

**Key words:** Authentication, Multidimensional Aggregate Query, Dynamic Database, 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 [6, 11], 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 [17].

Xu J. *et al.* [25] proposed a scheme, called MAIA, to authenticate aggregate range query, including SUM, COUNT, MIN, MAX and MEDIAN, over a static $d$-dimensional dataset, with

$O(d^2)$ communication overhead. Since MAIA indexs each point in the dataset using integers and intergers are not densely ordered, MAIA cannot handle with dynamic operations, like inserting to or deleting a point from the dataset.

In this paper, we propose a method to index each point in the dataset with a rational number in the interval $[0, 1)$. Since rational number is densely ordered, i.e. there is at least one rational number in-between any two unequal rational numbers, our indexing method support dynamic operations. The new indexing method utilizes a special ternary tree, which we call Doit tree— Dynamic Ordered Indexing Ternary Tree.

Based on the new indexing scheme, we propose a scheme, called dynamic MAIA, to authenticate the same types of aggregate range queires as MAIA, over dynamic multidimensional dataset. The communication overhead is $O(d^2 \log N)$, where $N$ is the number of points in the dataset.

Table 1: Performance of our scheme *Dynamic* MAIA, compared with MAIA proposed by Xu J. *et al.* [25]. Note both schemes are much more efficient in computation cost in 1D case, compared with high dimensional case (See annotation $\star$). We point out that the high computation cost on prover can be mitigated with horizontal partition of the dataset and parallel execution on each partition.

| Scheme | Communication overhead (bits) | Key Size | Storage overhead | Computation (Verifier Alice) | Computation (Prover Bob) | Query |
|---|---|---|---|---|---|---|
| MAIA ( [25]) | $O(d^2)$ | $O(d)$ | $O(dN)$ | $O(d \log N)\dagger\star$ | $O(dN \log N)\ddagger\star$ | SUM,COUNT,MIN,MAX, MEDIAN, QUANTILE, Range Selection |
| Dynamic MAIA (This paper) | $O(d^2 \log N)$ | $O(d)$ | $O(dN)$ | $O(d \log N)\dagger\star$ | $O(dN \log N)\ddagger\star$ | INSERT, DELETE, SUM,COUNT,MIN,MAX, MEDIAN, QUANTILE, Range Selection |

$d$: The dimension of data point in the dataset.
$\dagger$: $O(d \log N)$ modular multiplications.
$\star$: If the query range is 1D, the cost is $O(|S|)$.
$N$: The number of tuples in the dataset.
$\ddagger$: $O(dN \log N)$ modular exponentiations.

## 1.1 Contribution

Our main contributions can be summarized as follows:

1. We propose a Dynamic Ordered Indexing Ternatry Tree, called as Doit tree, which maps a datapoint to a rational number in $[0, 1)$ and preserves the order, and supports insertion or deletion of datapoints.
2. Based on Doit tree, we propose dynamic MAIA, to authenticate aggregate range query over dynamic multidimensional dataset.
3. Our scheme dynamic MAIA is efficient (See Table 1) and secure (See Theorem 1).

### 1.2 Organization

The rest of this paper is organized as follows: Section 2 briefs the related works, Section 3 reviews the scheme MAIA propsed in [25], Section 4 proposes the Doit tree, and Section 5 constructs our authentication scheme dynamic MAIA.

## 2 Related work

Researches [6, 11] in secure outsourced database emerged and quickly developed since early 2000's . Most papers focus on privacy-preserving SQL query execution over encrypted datasets [11, 12, 18, 7], and authentication of query results [6, 15, 5, 19, 17, 20, 23, 4, 14, 24, 1, 26, 16, 13, 9].

There are roughly four categories of approaches for outsourced database authentication in the literatures [6,15,5,19,17,20,23,4,14,24,1,26,16,13,9]. (1) (Homomorphic, or aggregatable) Cryptographic primitives, like collision-resistant hash, digital signature, commitment [17, 10, 3]. (2) Merkle Hash Tree and variants [16, 14, 4]. (3) Computational geometry approach [15, 4, 1]. (4) Inserting and auditing fake tuples [24].

In the "integrity authentication" track, most papers are working on authentication of simple range selection queries [15, 19, 17, 20, 1, 4]. [19, 20] authenticated 1D range selection queries, with linear (in the number of tuples selected by the query condition) communication cost and storage overhead. [17] verified range selection queries using aggregated signatures (like RSA [22], BLS [2]). [4] proposed a linear (or superlinear) scheme, using chained signatures over a "verification R-Tree" with partitions of tuples as tree nodes, to authenticate windows query, range query, kNN query, and RNN query. To the best of our knowledge, the current most efficient authentication scheme for range selection queries is [1], which proposed an efficient authentication scheme for 1D and 2D range selection queries over a grid dataset (e.g. GIS or image data) with $O(1)$ communication cost and linear storage overhead. [23] claimed to authenticate arbitrary queries, but their security model is too weak: a playful adversary can easily break their scheme.

## 3 Review of MAIA

In this section, we review the scheme MAIA proposed by Xu J. *et al.* [25]. For the sake of presentation of this paper, we will re-organize the presentation of [25].

Let $\mathbf{D}_{\boldsymbol{x}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be a set of $N$ $d$-dimensional points. In the setup phase, Alice *normalizes* [21] dataset $\mathbf{D}_{\boldsymbol{x}}$, to obtain dataset $\mathbf{D} = \{\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_N\}$. Next, Alice runs key generating algorithm KGen to generate a private key $\mathcal{K}$, and encodes the dataset $\mathbf{D}$ using data encoding algorithm DEnc with key $\mathcal{K}$, to obtain tags $\mathbf{T}$. At the end of setup phase, Alice deletes $\mathbf{D}_{\boldsymbol{x}}, \mathbf{D}, \mathbf{T}$ from her storage after sending them to Bob, and keeps key $\mathcal{K}$ private.

The query phase consists of multiple query sessions. In each session, Alice issues an aggregate range query to Bob. Bob generates the query result, and produces a proof, with help of Alice, by running the interactive algorithm ProVer with Alice. After receiving Bob's query result and proof, Alice verifies authenticity of the query reuslt using the private key $\mathcal{K}$. The type of queries include Sum, Count, Min, Max and Median.

Specially, among all components of MAIA, we take out all algorithms related to precess of index and encapsulate them as an indexing scheme (Idx, ITag, Translate, Compress, Uncompress). Idx maps a datapoint $\boldsymbol{x} \in \mathbf{D}_{\boldsymbol{x}}$ to an integer vector $\boldsymbol{i}$ in $[N]^d$ and preverses order for each dimension. ITag produces an authentication tag for index $\boldsymbol{i}$. Translate converts a query w.r.t. datapoint $\boldsymbol{x}$'s to a query w.r.t. indices $\boldsymbol{i}$'s, which has the same query result as the previous query. For any query range $\mathbf{R}$, Compress can convert the set $\{\mathsf{ITag}(\boldsymbol{i}) : \boldsymbol{i} \in \mathbf{R}\}$ (called as *Help-Info* in [25]) into a $d$-dimensional vector of small size, and Uncompress can reverse the process.

In this paper, we will propose an alternative indexing scheme to replace the one in MAIA [25], and keep the other parts of MAIA unchanged. The resulting scheme is called dynamic MAIA, which can support dynamic operations, like inserting or deleting a datapoint.

## 4 Dynamic Ordered Indexing Ternary Tree

We first propose a *Dynamic Ordered Indexing Ternary* tree, called as Doit tree in Section 4.1, and then present an indexing scheme based on the Doit tree in Section 4.2.

### 4.1 Doit Tree

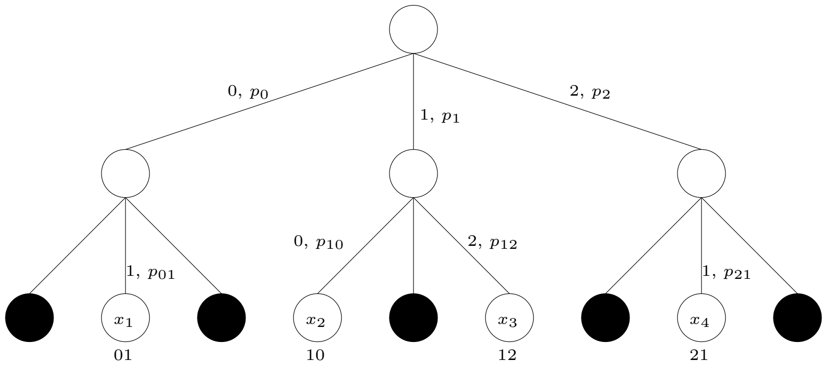

Fig. 1: Doit Tree: Dynamic Ordered Indexing Ternary Tree for dataset $\{x_1, x_2, x_3, x_4\}$.

Let $\mathsf{S2R} : \{0,1,2\}^* \to [0,1)$ be function that converts a string in $\{0,1,2\}^*$ to a real value in $[0,1)$. On input string $s$ in $\{0,1,2\}^*$, S2R treats the string "0."$\|s$ as a rational number in ternary numeral system, and converts it to $y \in [0,1)$. As a result, $\mathsf{S2R}(s) = y$.

We build a complete ternary tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Let $\mathcal{L} \subset \mathcal{V}$ be the set of all leaf nodes. We associate each edge $\mathtt{e} \in \mathcal{E}$ with attributes `e.trit` and `e.p`, and each vertex $\mathtt{v} \in \mathcal{V}$ with attributes `v.lab`, `v.idx`, `v.range` and `v.tag`. These attributes are defined as follows.

1. Edge

(a) Each internal node has three child nodes: *left child*, *middle child*, *right child*. For any edge $\mathtt{e_l} \in \mathcal{E}$ pointing to a left child, $\mathtt{e_l.trit} \leftarrow 0$; for any edge $\mathtt{e_m} \in \mathcal{E}$ pointing to a middle child, $\mathtt{e_m.trit} \leftarrow 1$; for any edge $\mathtt{e_r} \in \mathcal{E}$ pointing to a right child, $\mathtt{e_r.trit} \leftarrow 2$.

(b) Each tree edge $\mathtt{e} \in \mathcal{E}$ is associated with a public random prime, denoted as $\mathtt{e.prime}$.

2. Vertex

(a) For any node $\mathtt{v} \in \mathcal{V}$, let $(\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3, \ldots, \mathtt{e}_w)$ be the unique simple path from the root $\mathtt{r}$ to node $\mathtt{v}$, where $\mathtt{e}_1$ is adjacent to the root and $\mathtt{e}_w$ is adjacent to node $\mathtt{v}$. Then $\mathtt{v.lab} \leftarrow \mathtt{e_1.trit} \| \mathtt{e_2.trit} \ldots \| \mathtt{e_w.trit}$, where $\|$ is the string concatenation operator. Specially, for the root $\mathtt{r}$, $\mathtt{r.lab} \leftarrow \epsilon$, where $\epsilon$ is the empty string.

(b) For each node $\mathtt{v} \in \mathcal{V}$, $\mathtt{v.idx} \leftarrow \mathsf{S2R}(\mathtt{v.lab})$.

(c) For each node $\mathtt{v} \in \mathcal{V}$, $\mathtt{v.range} \leftarrow [\mathtt{v.idx},\ \mathtt{v.idx} + 3^{-|\mathtt{v.lab}|}) \subseteq [0, 1)$, where $|\mathtt{v.lab}|$ denotes the length of trit-string $\mathtt{v.lab}$. Particularly, for root $\mathtt{r}$, $\mathtt{r.range} = [0, 1)$, since $\mathtt{r.lab} = \epsilon$, and $\mathsf{S2R}(\epsilon) = 0$, $|\epsilon| = 0$.

(d) For any node $\mathtt{v} \in \mathcal{V}$, let $(\mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_u)$ be the unique simple path from the root to $\mathtt{v}$. Then $\mathtt{v.tag} = \mathtt{r.tag}^{\prod_{1 \le j \le u} \mathtt{e_j.prime}} \mod n$, where $\mathtt{r}$ denotes the root of the tree.

**4.1.1 Indexing a dataset.** Now, we use the newly constructed ternary tree to index a 1D dataset.

Let dataset $\mathbf{D}_x = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{N}$ and $x_1 < x_2 < x_3 \ldots < x_N$. For simplicity of presentation, assume $N = (3^H - 1)/2$ for some integer $H$. Build a complete ternary tree with $(2N + 1)$ leaves as above.

For the convenience of presentation, we name each vertex and edge.

1. For any $\mathtt{v} \in \mathcal{V}$ with $\mathtt{v.lab} = s$, we name $\mathtt{v}$ with $V_s$. For example, the root node $\mathtt{r} = V_\epsilon$.
2. For any edge $\mathtt{e} = (\mathtt{u}, \mathtt{v}) \in \mathcal{E}$, where $\mathtt{u}$ is the parent node of $\mathtt{v}$ and $\mathtt{v.lab} = s$, we name $\mathtt{e}$ with $E_s$, and name the prime $\mathtt{e.prime}$ with $p_s$ (See Figure 1).

Next, we associate an attribute $\mathtt{val}$ to some leaf nodes and define the mapping $\mathsf{idx} : \mathbf{D}_x \to [0, 1)$ as follows.

1. From left to right, number all leaf nodes with 1, 2, 3, ..., $(2N + 1)$. So the leftmost leaf node is the 1st leaf, and the rightmost leaf node is the $(2N + 1)$-th leaf.
2. For each leaf node $\mathtt{v}$, associate an attribute, denoted as $\mathtt{v.val}$. For the $(2j)$-th leaf $\mathtt{v}$, $j \in [N]$, $\mathtt{v.val} \leftarrow x_j$ and $\mathsf{idx}(x_j) \leftarrow \mathtt{v.idx}$; for any other leaf $\mathtt{v}$ , $\mathtt{v.val} \leftarrow \perp$. We call a leaf $\mathtt{v}$ as *unassigned leaf*, if $\mathtt{v.val} = \perp$. Otherwise, we call it *assigned leaf*. Let $\mathcal{L}^-$ be the set of all assigned leaf nodes and $\mathcal{L}^+$ be the set of all unassigned leaf nodes. $(\mathcal{L}^+, \mathcal{L}^-)$ forms a partition of the set $\mathcal{L}$ of all leaf nodes.

We call the construted ternary tree Doit Tree for dataset $\mathbf{D}_x$, which stands for "*Dynamic Ordered Indexing Ternary*" Tree. Figure 1 shows an example Doit tree for a dataset $\{x_1, x_2, x_3, x_4\}$.

A Doit tree has the following properties.

**Property 1** (Doit **Tree**)

1. *(Ternary Tree) Every internal node has exactly three children, called left child, middle child and right child.*
2. *(Partition) For an internal node* $\mathtt{v}$ *with left child* $\mathtt{t_l}$*, middle child* $\mathtt{v_m}$ *and right child* $\mathtt{t_r}$*,* $(\mathtt{v_l.range}, \mathtt{v_m.range}, \mathtt{v_r.range})$ *form a partition of* $\mathtt{v.range}$*. Additionally, for root node* $\mathtt{r}$*,* $\mathtt{r.range} = [0, 1)$.
3. *(Ordered) For any two nodes* $\mathtt{v_i}, \mathtt{v_j} \in \mathcal{V}$*, if* $\mathtt{v}_i$ *appears before* $\mathtt{v}_j$ *in the preorder traversal, then* $\mathtt{v_i.idx} \leq \mathtt{v_j.idx}$.
4. *(Dynamic) For any* $x_j, x_{j+1} \in \mathbf{D}$*, let* $\mathtt{v}_j, \mathtt{v}_{j+1} \in \mathcal{L}^-$ *be their corresponding leaf nodes, i.e.* $\mathtt{v_j.idx} = \mathsf{idx}(x_j)$ *and* $\mathtt{v_{j+1}} = \mathsf{idx}(x_{j+1})$*, there exists one and only one leaf node between* $\mathtt{v}_j$ *and* $\mathtt{v}_{j+1}$ *and this leaf node is unassigned. Additionally, both the leftmost leaf and the rightmost leaf node are unassigned. Those unassigned leaf nodes in set* $\mathcal{L}^+$ *allow the insertions of new values. After insertions of new nodes, these four properties still hold.*

## 4.2 Indexing Scheme for 1D Dataset

Now we show an indexing scheme based on the Doit tree. As menstion previously, an indexing scheme consists of Idx, Translate, ITag, Compress and Uncompress.

**4.2.1 itag.** At first, we define function $\mathsf{R2S} : [0, 1) \to \{0, 1, 2\}^*$ to convert a real value in $[0, 1)$ to a trit-string, which is the inverse function of S2R. Given $i \in [0, 1)$, let $s_i \in$ "0."$\{0, 1, 2\}^*$ be the expression of $i$ in ternary numeral system. If $|s_i| < H + 2$, then append $(H + 2 - |s_i|)$ 0's to $s_i$: $s_i \leftarrow s_i \| \underbrace{00\ldots0}_{H+2-|s_i|\ 0\text{'s}}$ . Remove the first two symbols from $s_i$. Output $s_i$.

Given $i \in [0, 1)$, we convert it into a trit-string $\mathsf{R2S}(i) \in \{0, 1, 2\}^*$. We define

$$\mathsf{itag}(i) \overset{\text{def}}{=} V_{\mathsf{R2S}(i)}.\mathtt{tag}.$$

Let $\mathsf{prefix}(s, j)$ be the prefix of length $j$ of string $s$ and let $h : \{0, 1, 2\}^* \to \mathsf{Prime}$ be a hash function [8]. In another equivalent expression,

$$\mathsf{itag}(i) \overset{\text{def}}{=} g_p^{\prod_{1\leq j\leq |s|}^{s=\mathsf{R2S}(i)} h(\mathsf{prefix}(s,j))} \mod n.$$

**4.2.2 insert.** The input is $(\mathcal{T}, x)$, where $\mathcal{T}$ is a Doit tree and $x \in \mathbb{N}$.

1. Number all leaf nodes in $\mathcal{L}$ from left to right with $1, 2, 3, \ldots, |\mathcal{L}|$. Denote the $i$-th leaf node with $\ell_i$. Find $\zeta$ such that $\ell_\zeta, \ell_{\zeta+2} \in \mathcal{L}^-$ and $\ell_\zeta.\mathtt{val} \leq x < \ell_{\zeta+2}.\mathtt{val}$.
2. Expand node $\ell_{\zeta+1}$ and let $\mathtt{t_l}$, $\mathtt{t_m}$ and $\mathtt{t_r}$ denote its left child, middle child and right child, respectively. *Comment: As a result,* $\ell_{i+1}$ *becomes an internal node and* $\mathtt{t_l}$*,* $\mathtt{t_m}$ *and* $\mathtt{t_r}$ *are three leaf nodes.*
3. Update the tree $\mathcal{T}$:
   (a) Let $\mathtt{t_m.val} \leftarrow x$;
   (b) Remove $\ell_{\zeta+1}$ from $\mathcal{L}$ and $\mathcal{L}^+$;
   (c) Add $\mathtt{t_l}$, $\mathtt{t_m}$ and $\mathtt{t_r}$ into $\mathcal{L}$;
   (d) Add $\mathtt{t_l}$ and $\mathtt{t_r}$ into $\mathcal{L}^+$;

(e) Add $\mathtt{t_m}$ into $\mathcal{L}^-$, etc.

Figure 2 shows an example of insertion. Note that the tree $\mathcal{T}$, set $\mathcal{L}$ of leaves, set $\mathcal{L}^+$ of unassigned leaves, set $\mathcal{L}^-$ of assigned leaves are dynamic, and will be updated during each insertion. However, for each node `v` and each edge `e`, the attributes `v.lab`, `v.idx`, `v.range`, `e.trit`, `e.prime` are static.

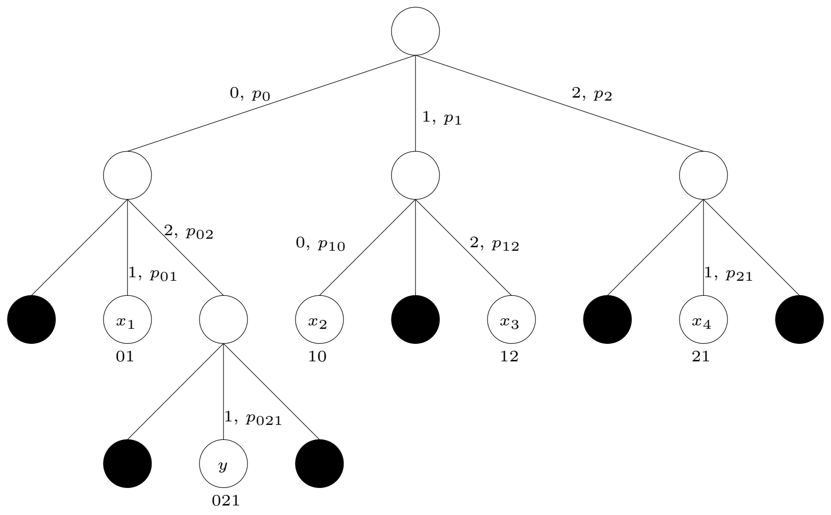

Fig. 2: Doit Tree: Insert a new value $y$, where $x_1 < y < x_2$.

**4.2.3 MinCover.** The input of algorithm **MinCover** is $(\mathtt{v}_1, \mathtt{v}_2)$, where $\mathtt{v}_1$ and $\mathtt{v}_2$ are two assigned leaf nodes of a Doit tree $\mathcal{T}$, and $\mathtt{v}_1.\mathtt{id} \le \mathtt{v}_2.\mathtt{id}$. The equality is for special case $\mathtt{v}_1 = \mathtt{v}_2$. The output is a collection of tree nodes $\{\mathtt{r}_i\}$ such that $\{\mathtt{r}_i.\mathtt{range}\}$ forms a partition of interval $[\mathtt{v}_1.\mathtt{id}, \mathtt{v}_2.\mathtt{id}]$.

1. From $\mathcal{T}$, remove all tree nodes `v` and their ajacent edges, such that $\mathtt{v.id} < \mathtt{v}_1.\mathtt{id}$.
2. From $\mathcal{T}$, remove all tree nodes `v` and their ajacent edges, such that $\mathtt{v.id} > \mathtt{v}_2.\mathtt{id}$.
3. From $\mathcal{T}$, remove all tree nodes, except $\mathtt{v}_1$, along the unique simple path from the root of $\mathcal{T}$ to leaf $\mathtt{v_1}$ and their ajacent edges.
4. From $\mathcal{T}$, remove all tree nodes, except $\mathtt{v}_2$, along the unique simple path from the root of $\mathcal{T}$ to leaf $\mathtt{v_2}$ and their ajacent edges.
5. Let $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m\}$ denote the collection of remaining subtrees and each $\mathcal{T}_i$ is a subtree. Let $\mathtt{r}_i$ be the root node of tree $\mathcal{T}_i$. Note $\{\mathcal{T}_i\}$ is the minimum cover for interval $[\mathtt{v}_1.\mathtt{id}, \mathtt{v}_2.\mathtt{id}]$.
6. Output $(\mathtt{r}_1, \mathtt{r}_2, \ldots, \mathtt{r}_m)$.

Figure 3 shows an example with input $(V_{021}, V_{21})$.

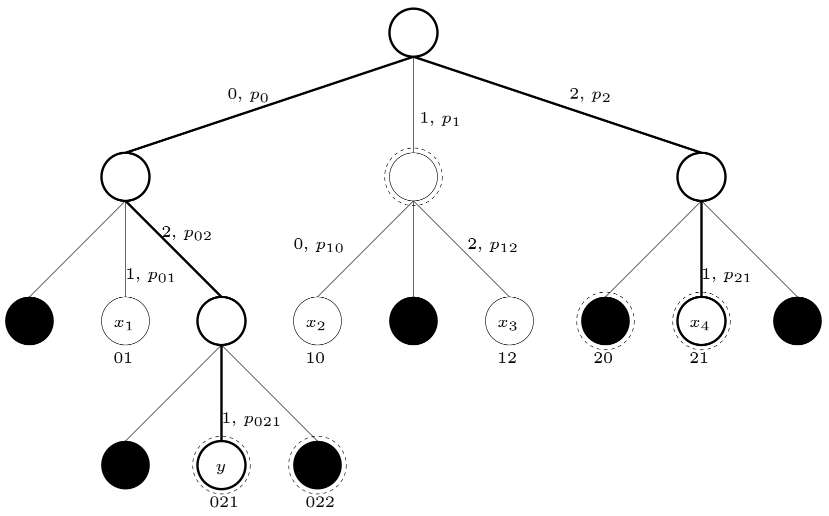

Fig. 3: Doit Tree: Compression of $[0.021, 0.21] = [0.021, 0.022) \cup [0.022, 0.1) \cup [0.1, 0.2) \cup [0.20, 0.21) \cup [0.21, 0.22)$.

### 4.2.4 compress. The input is $(\mathtt{v}_1, \mathtt{v}_2, \alpha)$.

1. $(\mathtt{r}_1, \mathtt{r}_2, \ldots, \mathtt{r}_m) \leftarrow$ **MinCover**$(\mathtt{v}_1, \mathtt{v}_2)$.
2. For each $i \in [m]$, $v_i \overset{\$}{\leftarrow} G_q, w_i \overset{\$}{\leftarrow} G_r$ and $\delta_i \leftarrow (\mathtt{r}_i.\mathtt{tag})^{\alpha} \times v_i \times w_i \mod n$.
3. Output $\{\delta_1, \delta_2, \ldots, \delta_m\}$.

### 4.2.5 uncompress. The input is $(\delta_1, \delta_2, \ldots, \delta_m, \mathtt{v})$.

1. Let $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m\}$ denote the corresponding minimum cover and $\mathtt{r}_i$ be the root node of tree $\mathcal{T}_i$.
2. Find $j \in [m]$, such that $\mathtt{v.id} \in \mathtt{r}_j.\mathtt{range}$. Note $\mathtt{v}$ should be a leaf node of the subtree $\mathcal{T}_j$.
3. Set $\mathtt{r}_j.\mathtt{tag} \leftarrow \delta_j$ and compute $\mathtt{v.tag}$ as described previously.
4. Output $\mathtt{v.tag}$.

## 4.3 Indexing Scheme for *d*-Dimensional Dataset

Let $\mathbf{D}_{\boldsymbol{x}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \subset \mathbb{N}^d$ be a set of $N$ $d$-dimensional data points, where $N = (3^H - 1)/2$ for some integer $H$.

For each dimension $j \in [d]$, let $\mathbf{D}_{\boldsymbol{x}}^{(j)} = \{\boldsymbol{x}[j] : \boldsymbol{x} \in \mathbf{D}_{\boldsymbol{x}}\}$, build a Doit tree $\mathcal{T}^{(j)}$ for 1D dataset $\mathbf{D}_{\boldsymbol{x}}^{(j)}$ and define $\mathsf{idx}^{(j)}, \mathsf{compress}^{(j)}, \mathsf{uncompress}^{(j)}$ correspondingly. Now we define corresponding functions in vector forms.

$$\begin{aligned}
&\mathsf{Idx}(\boldsymbol{x}) = (\mathsf{idx}^{(1)}(\boldsymbol{x}[1]), \mathsf{idx}^{(2)}(\boldsymbol{x}[2]), \ldots, \mathsf{idx}^{(d)}(\boldsymbol{x}[d]))\\
&\mathsf{ITag}(\boldsymbol{x}) = (\left(\mathsf{itag}^{(1)}(\boldsymbol{x}[1])\right)^{\boldsymbol{s}_1[1]}, \left(\mathsf{itag}^{(2)}(\boldsymbol{x}[2])\right)^{\boldsymbol{s}_1[2]}, \ldots, \left(\mathsf{itag}^{(d)}(\boldsymbol{x}[d])\right)^{\boldsymbol{s}_1[d]})\\
&\mathsf{Compress}(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{\alpha}) = (\mathsf{compress}^{(1)}(\boldsymbol{v}_1[1], \boldsymbol{v}_2[1], \boldsymbol{\alpha}[1]), \ldots, \mathsf{compress}^{(d)}(\boldsymbol{v}_1[d], \boldsymbol{v}_2[d], \boldsymbol{\alpha}[d]));\\
&\mathsf{Uncompress}(\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_\mathsf{d}, \boldsymbol{v}) = (\mathsf{uncompress}^{(1)}(\boldsymbol{\delta}_1, \boldsymbol{v}[1]), \ldots, \mathsf{uncompress}^{(d)}(\boldsymbol{\delta}_d, \boldsymbol{v}[d])).
\end{aligned}$$

#### 4.3.1 Translate.

Let $\mathbf{D}_{\boldsymbol{i}} = \{\mathsf{Idx}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbf{D}_{\boldsymbol{x}}\}$ be the set of indices of points in dataset $\mathbf{D}_{\boldsymbol{x}}$. Given a query with range $\mathbf{B} = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d] \subset \mathbb{N}^d$, which asks for the sum $\bigoplus_{\boldsymbol{x} \in \mathbf{D}_{\boldsymbol{x}} \cap \mathbf{B}} \boldsymbol{x}$, algorithm Translate output a query with range $\mathbf{R} = [\beta_1, \gamma_1] \times [\beta_2, \gamma_2] \times \ldots \times [\beta_d, \gamma_d] \subset [0, 1)^d$, which asks for the sum $\bigoplus_{\boldsymbol{i} \in \mathbf{D}_{\boldsymbol{i}} \cap \mathbf{R}} \mathsf{Att}(\boldsymbol{i})$, such that (1) the two queries are equivalent, that is, $\bigoplus_{\boldsymbol{x} \in \mathbf{D}_{\boldsymbol{x}} \cap \mathbf{B}} \boldsymbol{x} = \bigoplus_{\boldsymbol{i} \in \mathbf{D}_{\boldsymbol{i}} \cap \mathbf{R}} \mathsf{Att}(\boldsymbol{i})$; (2) Alice can verify the correctness of the translation.

Translate.

---

1. Alice sends to Bob $\{(a_i, b_i) : i \in [d]\}$.
2. Bob returns to Alice $\{(\bar{\beta}_i, \bar{x}_i, t_{\bar{\beta}_i};\ \beta_i, x_i, t_{\beta_i};\ \gamma_i, y_i, t_{\gamma_i};\ \bar{\gamma}_i, \bar{y}_i, t_{\bar{\gamma}_i}) : i \in [d]\}$.
3. Alice outputs $\{(\beta_i, \gamma_i) : i \in [d]\}$, if
   (a) for each $i \in [d]$, $\bar{x}_i < a_i \le x_i$ and $y_i \le b_i < \bar{y}_i$;
   (b) $t_{\bar{\beta}_i}$, $t_{\beta_i}$, $t_{\gamma_i}$ and $t_{\bar{\gamma}_i}$, are valid itag values of $(\bar{\beta}_i, \bar{x}_i)$, $(\beta_i, x_i)$, $(\gamma_i, y_i)$ and $(\bar{\gamma}_i, \bar{y}_i)$, respectively.

   Otherwise, Alice output $\perp$.

---

## 5 Dynamic MAIA

The dynamic MAIA is a $\mathcal{RC}$ protocol [25] between Alice and Bob, and consists of setup phase and query phase. In setup phase, Alice runs the key generating algorihtm KGen to generate a private key, then encodes the dataset using data encoding algorithm DEnc with the private key. The query phase consists of multiple query sessions. In each query session, Alice either issue a query or update the dataset by inserting or deleting a point from the dataset.

### 5.1 Setup phase

#### 5.1.1 KGen.

At first, we define a subroutine KeyGen. KeyGen$(1^\kappa)$:

---

Alice

1. runs algorithm $\mathcal{G}(1^\kappa)$ to generate $(n, p, q, r, G_p, G_q, G_r, g_p, g_q, g_r)$.
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 $\boldsymbol{s}_1 = (s_{1,1}, s_{2,1}, \dots, s_{d,1})$, $\boldsymbol{s}_2 = (s_{1,2}, s_{2,2}, \dots, s_{d,2})$ and $\boldsymbol{s}_3 = (s_{1,3}, s_{2,3}, \dots, s_{d,3})$. Let $\mathcal{K} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, n, p, q, r, g_p, g_q, g_r)$. Output $\mathcal{K}$.

---

Next, we define the algorithm KGen based on KeyGen.

KGen:

1. Run $\mathsf{KeyGen}(1^\kappa)$ and obtains output $\mathcal{K}$.
2. Run $\mathsf{KeyGen}(1^\kappa)$ and obtains output $\bar{\mathcal{K}}$.

Output $(\mathcal{K}, \bar{\mathcal{K}})$.

---

### 5.2 DEnc.

1. For each dimension $j \in [d]$, construct a Doit tree $\mathcal{T}^{(j)}$ for $\mathbf{D}_{\boldsymbol{x}}^{(j)} = \{\boldsymbol{x}[j] : \boldsymbol{x} \in \mathbf{D}\}$ and define Att, Idx, ITag, Compress and Uncompress;
2. For each $\boldsymbol{i} \in \mathbf{D}_{\boldsymbol{i}}$, Alice
   (a) chooses a number $\xi_{\boldsymbol{i}}$ at random from $\mathbb{Z}_q^*$;
   (b) computes a tag $\boldsymbol{t}_{\boldsymbol{i}}$ using the private key $\mathcal{K}$:
$$\boldsymbol{t}_{\boldsymbol{i}} \leftarrow \mathsf{Tag}_{\mathcal{K}}(\mathsf{Att}(\boldsymbol{i}), \boldsymbol{i}; \xi_{\boldsymbol{i}}). \quad (1)$$
3. Let $\mathbf{T} = \{\boldsymbol{t}_{\boldsymbol{i}} : \boldsymbol{i} \in \mathbf{D}_i\}$, and $\bar{\mathbf{D}} = \bar{\mathbf{T}} = \emptyset$. Alice sends $\mathbf{D}, \bar{\mathbf{D}}$, $\mathbf{T}$ and $\bar{\mathbf{T}}$ to Bob, removes local copy of $\mathbf{D}, \bar{\mathbf{D}}$ and $\mathbf{T}$ from her storage, and keeps $(\mathcal{K}, \bar{\mathcal{K}})$.

---

### 5.3 Query phase

#### 5.3.1 Query. $\langle \mathsf{Eval}, \mathsf{Ext} \rangle(\textsc{Sum}(\mathbf{R}))$. Let ProVer be as in scheme MAIA [25].

1. Simulate ProVer on input $\textsc{Sum}(\mathbf{R})$ w.r.t. dataset $\mathbf{D}$ using private key $\mathcal{K}$, and obatin result $\boldsymbol{X}$.
2. Simulate ProVer on input $\textsc{Sum}(\mathbf{R})$ w.r.t. dataset $\bar{\mathbf{D}}$ using private key $\bar{\mathcal{K}}$, and obatin result $\bar{\boldsymbol{X}}$.
3. Output $\boldsymbol{X} - \bar{\boldsymbol{X}}$.

---

#### 5.3.2 Insert.

$\mathsf{Insert}(\boldsymbol{x}, \mathbf{D}, \mathbf{T}, \mathcal{K}, \pi^*)$: Insert $\boldsymbol{x}$ into $\mathbf{D}$.

1. For each $j \in [d]$, $\mathsf{insert}(\mathcal{T}^{(j)}, \boldsymbol{x}[j])$.
2. Let $\boldsymbol{i} \leftarrow \mathsf{Idx}(\boldsymbol{x})$.
3. Choose $\xi$ from $\mathbb{Z}_q^*$ at random, and compute tag: $\boldsymbol{t}_{\boldsymbol{i}} \leftarrow \mathsf{Tag}_{\mathcal{K}}(\boldsymbol{x}, \boldsymbol{i}; \xi)$.
4. Add $\boldsymbol{x}$ into set $\mathbf{D}$, and add $\boldsymbol{t}_{\boldsymbol{i}}$ into set $\mathbf{T}$.
5. Update $\pi^*$: $\pi^* \leftarrow \pi^* \times \mathsf{CTag}_{\mathcal{K}}(\boldsymbol{i})[1] \mod n$.

---

#### 5.3.3 Delete.

$\mathsf{Delete}(\boldsymbol{x})$: Delete $\boldsymbol{x}$ from the dataset $\mathbf{D}_{\boldsymbol{x}}$.

1. If $\boldsymbol{x} \in \mathbf{D}_{\boldsymbol{x}}$, then $\mathsf{Insert}(\boldsymbol{x}, \bar{\mathbf{D}}, \bar{\mathbf{T}}, \bar{\mathcal{K}}, \bar{\pi}^*)$. Otherwise, do nothing.

Editing can be implemented with combination of deleting and inserting. We save the details.

### 5.4 Complexity Analysis

Without surprise, due the use of tree, the communication complexity of our scheme is $O(d^2 \log N)$. The computation and storage overhead remain the same as MAIA in Xu J. *et al.* [25]. The details are showed in Table 1 in Section 1.

### 5.5 Security

Xu J. *et al.* [25] gave a security model for outsourced computing, called *Provable Remote Computing* ($\mathcal{PRC}$) protocol.

**Theorem 1** *Dynamic* MAIA *as constructed above is a $\mathcal{PRC}$ w.r.t. aggregate* Sum *query, and insertion/deletion operations, if the Assumption 1 and Assumption 2 in Xu J.* et al. *[25] hold, and RSA signature is a secure multiplicative homomorphic signature scheme.*

## 6 Conclusion

In this paper, we proposed a dynamic ordered indexing ternary tree, and based it we proposed a method to authenticate aggregate range query over dynamic multidimensional dataset. The supported query types include Sum, Count, Min, Max and Median. The communication overhead is $O(d^2 \log N)$, where $d$ is the dimension and $N$ is the number of points in the dataset.

## References

1. M. J. Atallah, Y. Cho, and A. Kundu. Efficient data authentication in an environment of untrusted third-party distributors. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 696–704, Washington, DC, USA, 2008. IEEE Computer Society.
2. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.
3. S. H. W. G. H. Brian Thompson, Danfeng Yao and T. Sander. Privacy-preserving computation and verification of aggregate queries on outsourced databases. In *Privacy Enhancing Technologies Symposium (PETS)*, Aug 2009.
4. W. Cheng and K.-L. Tan. Query assurance verification for outsourced multi-dimensional databases. *J. Comput. Secur.*, 17(1):101–126, 2009.
5. P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine. Authentic data publication over the internet. *J. Comput. Secur.*, 11(3):291–314, 2003.

6. P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine. Authentic third-party data publication. In *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security*, pages 101–112, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.
7. T. Ge and S. B. Zdonik. Answering aggregation queries in a secure system model. In *VLDB*, pages 519–530, 2007.
8. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *EUROCRYPT*, pages 123–139, 1999.
9. M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *CT-RSA*, pages 407–424, 2008.
10. S. Haber, W. Horne, T. Sander, and D. Yao. Privacy-preserving verification of aggregate queries on outsourced databases. Technical report, HP Laboratories, 2006. HPL-2006-128.
11. H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227, New York, NY, USA, 2002. ACM.
12. H. Hacigümüs, B. R. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *DASFAA*, pages 125–136, 2004.
13. K. M. HweeHwa PANG, Jilian ZHANG. Scalable verification for outsourced dynamic databases. In *Will appear in 35th International Conference on Very Large Data Bases (VLDB09)*.
14. F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD Conference*, pages 121–132, 2006.
15. C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.
16. K. Mouratidis, D. Sacharidis, and H. Pang. Partially materialized digest scheme: an efficient verification method for outsourced databases. *The VLDB Journal*, 18(1):363–381, 2009.
17. E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2):107–138, 2006.
18. E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In *IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 89–103, July 2006.
19. H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 407–418, New York, NY, USA, 2005. ACM.
20. H. Pang and K.-L. Tan. Verifying completeness of relational query answers from online servers. *ACM Trans. Inf. Syst. Secur.*, 11(2):1–50, 2008.
21. F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.
22. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
23. R. Sion. Query execution assurance for outsourced databases. In *VLDB*, pages 601–612, 2005.
24. M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 782–793. VLDB Endowment, 2007.
25. J. XU and E.-C. CHANG. Authenticating aggregate range queries over multidimensional dataset. Cryptology ePrint Archive, Report 2010/050, 2010. `http://eprint.iacr.org/`.
26. Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. Authenticated join processing in outsourced databases. In *SIGMOD Conference*, pages 5–18, 2009.