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 \( D \) 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. Xu J. et al. [25] proposed an authentication scheme to solve this problem for multidimensional aggregate range query, including \textit{Sum}, \textit{Count}, \textit{Min}, \textit{Max} and \textit{Median}, and multidimensional range selection query, with \( O(d^2 \log N) \) communication overhead. However, their scheme only applies 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 \textit{Sum}, \textit{Count}, \textit{Min}, \textit{Max} and \textit{Median}, over a static \( d \)-dimensional dataset, with
$O(d^2)$ communication overhead. Since MAIA indexes each point in the dataset using integers and integers 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 queries 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 $\ast$). 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.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Communication overhead (bits)</th>
<th>Key Size</th>
<th>Storage overhead</th>
<th>Computation (Verifier Alice)</th>
<th>Computation (Prover Bob)</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIA ([25])</td>
<td>$O(d^2)$</td>
<td>$O(d)$</td>
<td>$O(dN)$</td>
<td>$O(d \log N)$†$\ast$</td>
<td>$O(dN \log N)$†$\ast$</td>
<td>SUM, COUNT, MIN, MAX, MEDIAN, QUANTILE, Range Selection</td>
</tr>
<tr>
<td>Dynamic MAIA (This paper)</td>
<td>$O(d^2 \log N)$</td>
<td>$O(d)$</td>
<td>$O(dN)$</td>
<td>$O(d \log N)$†$\ast$</td>
<td>$O(dN \log N)$†$\ast$</td>
<td>INSERT, DELETE, SUM, COUNT, MIN, MAX, MEDIAN, QUANTILE, Range Selection</td>
</tr>
</tbody>
</table>

$d$: The dimension of data point in the dataset. $N$: The number of tuples in the dataset.

†: $O(d \log N)$ modular multiplications. †: $O(dN \log N)$ modular exponentiations.

$\ast$: If the query range is 1D, the cost is $O(|S|)$.

### 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 proposed in [25], Section 4 proposes the Doit tree, and Section 5 constructs our authentication scheme dynamic MAIA.

2 Related work

Researches in secure outsourced database emerged and quickly developed since early 2000’s. Most papers focus on privacy-preserving SQL query execution over encrypted datasets, and authentication of query results. There are roughly four categories of approaches for outsourced database authentication in the literatures. (1) (Homomorphic, or aggregatable) Cryptographic primitives, like collision-resistant hash, digital signature, commitment. (2) Merkle Hash Tree and variants. (3) Computational geometry approach. (4) Inserting and auditing fake tuples.

In the “integrity authentication” track, most papers are working on authentication of simple range selection queries, with linear (in the number of tuples selected by the query condition) communication cost and storage overhead. verified range selection queries using aggregated signatures (like RSA, BLS). 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. 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 $D_x = \{x_1, x_2, \ldots, x_N\}$ be a set of $N d$-dimensional points. In the setup phase, Alice normalizes $D_x$ to obtain dataset $D = \{i_1, i_2, \ldots, i_N\}$. Next, Alice runs key generating algorithm $KGen$ to generate a private key $K$, and encodes the dataset $D$ using data encoding algorithm $DEnc$ with key $K$, to obtain tags $T$. At the end of setup phase, Alice deletes $D_x, D, T$ from her storage after sending them to Bob, and keeps key $K$ 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 result using the private key $K$. 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 $x \in D_x$ to an integer vector $i$ in $[N]^d$ and preserves order for each dimension. ITag produces an authentication tag for index $i$. Translate converts a query w.r.t. datapoint $x$'s to a query w.r.t. indices $i$'s, which has the same query result as the previous query. For any query range $R$, Compress can convert the set $\{ITag(i) : i \in R\}$ (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

Let $S2R : \{0, 1, 2\}^* \rightarrow [0, 1)$ 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, $S2R(s) = y$.

We build a complete ternary tree $T = (V, E)$ with vertex set $V$ and edge set $E$. Let $L \subset V$ be the set of all leaf nodes. We associate each edge $e \in E$ with attributes $e.trit$ and $e.p$, and each vertex $v \in V$ 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 \( e_i \in E \) pointing to a left child, \( e_i.trit \leftarrow 0 \); for any edge \( e_m \in E \) pointing to a middle child, \( e_m.trit \leftarrow 1 \); for any edge \( e_r \in E \) pointing to a right child, \( e_r.trit \leftarrow 2 \).
(b) Each tree edge \( e \in E \) is associated with a public random prime, denoted as \( e.prime \).

2. Vertex

(a) For any node \( v \in V \), let \((e_1,e_2,e_3,\ldots,e_u)\) be the unique simple path from the root \( r \) to node \( v \), where \( e_1 \) is adjacent to the root and \( e_u \) is adjacent to node \( v \). Then \( v.lab \leftarrow e_1.trit\|e_2.trit\|\ldots\|e_u.trit \), where \( \| \) is the string concatenation operator. Specially, for the root \( r \), \( r.lab \leftarrow \epsilon \), where \( \epsilon \) is the empty string.
(b) For each node \( v \in V \), \( v.idx \leftarrow S2R(v.lab) \).
(c) For each node \( v \in V \), \( v.range \leftarrow [v.idx, v.idx + 3^{-|v.lab|}) \subseteq [0,1) \), where \(|v.lab|\) denotes the length of trit-string \( v.lab \). Particularly, for root \( r \), \( r.range = [0,1) \), since \( r.lab = \epsilon \), and \( S2R(\epsilon) = 0 \), \(|\epsilon| = 0 \).
(d) For any node \( v \in V \), let \((e_1,e_2,\ldots,e_u)\) be the unique simple path from the root to \( v \). Then \( v.tag = r.tag \prod_{1 \leq j \leq u} e_j.prime \mod n \), where \( r \) 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 \( D_x = \{x_1,x_2,\ldots,x_N\} \subseteq \mathbb{N} \) 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 \( v \in V \) with \( v.lab = s \), we name \( v \) with \( V_s \). For example, the root node \( r = V_s \).
2. For any edge \( e = (u,v) \in E \), where \( u \) is the parent node of \( v \) and \( v.lab = s \), we name \( e \) with \( E_s \), and name the prime \( e.prime \) with \( p_s \) (See Figure 1).

Next, we associate an attribute \( val \) to some leaf nodes and define the mapping \( \text{id}x : D_x \rightarrow [0,1) \) as follows.

1. From left to right, number all leaf nodes with 1, 2, 3, \ldots, \((2N + 1)\). 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 \( v \), associate an attribute, denoted as \( v.val \). For the \((2j)\)-th leaf \( v \), \( j \in [N] \), \( v.val \leftarrow x_j \) and \( \text{id}x(x_j) \leftarrow v.idx \); for any other leaf \( v \), \( v.val \leftarrow \perp \). We call a leaf \( v \) as unassigned leaf, if \( v.val = \perp \). 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 constructed ternary tree Doit Tree for dataset \( D_x \), 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 \( v \) with left child \( t_1 \), middle child \( v_m \) and right child \( t_r \), \((v_1.\text{range},v_m.\text{range},v_r.\text{range})\) form a partition of \( v.\text{range} \). Additionally, for root node \( r \), \( r.\text{range} = [0,1) \).

3. (Ordered) For any two nodes \( v_i,v_j \in V \), if \( v_i \) appears before \( v_j \) in the preorder traversal, then \( v_i.\text{idx} \leq v_j.\text{idx} \).

4. (Dynamic) For any \( x_j,x_{j+1} \in D \), let \( v_j,v_{j+1} \in \mathcal{L}^- \) be their corresponding leaf nodes, i.e. \( v_j.\text{idx} = \text{idx}(x_j) \) and \( v_{j+1} = \text{idx}(x_{j+1}) \), there exists one and only one leaf node between \( v_j \) and \( v_{j+1} \) 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 mentioned previously, an indexing scheme consists of \( \text{Id}_{x} \), \( \text{Trans}_{it} \), \( \text{ITag} \), \( \text{Compress} \) and \( \text{Uncompress} \).

4.2.1 \( \text{ITag} \). At first, we define function \( R2S : [0,1) \rightarrow \{0,1,2\}^* \) 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 \( s_i \in \{0,1,2\}^* \) be the expression of \( i \) in ternary numeral system. If \(|s_i| < H + 2\), then append \((H + 2 - |s_i|)\) 0's to \( s_i \): \( s_i = \langle 00 \ldots 0 \rangle \). Remove the first two symbols from \( s_i \). Output \( s_i \).

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

\[
\text{itag}(i) \overset{\text{def}}{=} R2S(i).\text{tag}.
\]

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

\[
\text{itag}(i) \overset{\text{def}}{=} \prod_{1 \leq j \leq |s_i|} h(\text{prefix}(s,i)) \mod n.
\]

4.2.2 \( \text{Insert} \). The input is \((T,x)\), where \( T \) 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 \( \ell_\zeta.\text{val} \leq x < \ell_{\zeta+2}.\text{val} \).

2. Expand node \( \ell_{\zeta+1} \) and let \( t_1, t_m \) and \( t_r \) denote its left child, middle child and right child, respectively. \( \text{Comment: As a result, } \ell_{\zeta+1} \text{ becomes an internal node and } t_1, t_m \text{ and } t_r \text{ are three leaf nodes.} \)

3. Update the tree \( T \):
   (a) Let \( t_m.\text{val} \leftarrow x \);
   (b) Remove \( \ell_{\zeta+1} \) from \( \mathcal{L} \) and \( \mathcal{L}^+ \);
   (c) Add \( t_1, t_m \) and \( t_r \) into \( \mathcal{L} \);
   (d) Add \( t_1 \) and \( t_r \) into \( \mathcal{L}^+ \);
(e) Add \( t_m \) into \( \mathcal{L}^- \), etc.

Figure 2 shows an example of insertion. Note that the tree \( T \), 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 \( x_1 < y < x_2 \).](image)

4.2.3 MinCover. The input of algorithm MinCover is \((v_1, v_2)\), where \( v_1 \) and \( v_2 \) are two assigned leaf nodes of a Doit tree \( T \), and \( v_1\.id \leq v_2\.id \). The equality is for special case \( v_1 = v_2 \). The output is a collection of tree nodes \( \{r_i\} \) such that \( \{r_i\.range\} \) forms a partition of interval \([v_1\.id, v_2\.id]\).

1. From \( T \), remove all tree nodes \( v \) and their adjacent edges, such that \( v\.id < v_1\.id \).
2. From \( T \), remove all tree nodes \( v \) and their adjacent edges, such that \( v\.id > v_2\.id \).
3. From \( T \), remove all tree nodes, except \( v_1 \), along the unique simple path from the root of \( T \) to leaf \( v_1 \) and their adjacent edges.
4. From \( T \), remove all tree nodes, except \( v_2 \), along the unique simple path from the root of \( T \) to leaf \( v_2 \) and their adjacent edges.
5. Let \( \{T_1, T_2, \ldots, T_m\} \) denote the collection of remaining subtrees and each \( T_i \) is a sub-tree. Let \( r_i \) be the root node of tree \( T_i \). Note \( \{T_i\} \) is the minimum cover for interval \([v_1\.id, v_2\.id]\).
6. Output \((r_1, r_2, \ldots, r_m)\).

Figure 3 shows an example with input \((V_{021}, V_{21})\).
4.2.4 compress. The input is \((v_1, v_2, \alpha)\).

1. \((r_1, r_2, \ldots, r_m) \leftarrow \text{MinCover}(v_1, v_2)\).
2. For each \(i \in [m]\), \(v_i \overset{\delta_i}{\leftarrow} G_q, w_i \overset{\delta_i}{\leftarrow} G_r\) and \(\delta_i \leftarrow (r_i.tag)^\alpha \times v_i \times w_i \mod n\).
3. Output \(\{\delta_1, \delta_2, \ldots, \delta_m\}\).

4.2.5 uncompress. The input is \((\delta_1, \delta_2, \ldots, \delta_m, v)\).

1. Let \(\{T_1, T_2, \ldots, T_m\}\) denote the corresponding minimum cover and \(r_i\) be the root node of tree \(T_i\).
2. Find \(j \in [m]\), such that \(v.id \in r_j.range\). Note \(v\) should be a leaf node of the subtree \(T_j\).
3. Set \(r_j.tag \leftarrow \delta_j\) and compute \(v.tag\) as described previously.
4. Output \(v.tag\).

4.3 Indexing Scheme for \(d\)-Dimensional Dataset

Let \(D_x = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{N}^d\) 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 \(D_x^{(j)} = \{x[j] : x \in D_x\}\), build a Doit tree \(T^{(j)}\) for 1D dataset \(D_x^{(j)}\) and define \(\text{idx}^{(j)}, \text{compress}^{(j)}, \text{uncompress}^{(j)}\) correspondingly. Now we define

Fig. 3: Doit Tree: Compression of \([0.021, 0.21] = [0.021, 0.022) \cup [0.022, 0.1) \cup [0.1, 0.2) \cup [0.20, 0.21) \cup [0.21, 0.22)\).
corresponding functions in vector forms.

\[ \text{Idx}(x) = (\text{idx}^{(1)}(x[1]), \text{idx}^{(2)}(x[2]), \ldots, \text{idx}^{(d)}(x[d])) \]

\[ \text{I Tag}(x) = (\{\text{itag}^{(1)}(x[1])\}^{s_1[1]}, \{\text{itag}^{(2)}(x[2])\}^{s_1[2]}, \ldots, \{\text{itag}^{(d)}(x[d])\}^{s_1[d]}) \]

\[ \text{Compress}(v_1, v_2, \alpha) = (\text{compress}^{(1)}(v_1[1], v_2[1], \alpha[1]), \ldots, \text{compress}^{(d)}(v_1[d], v_2[d], \alpha[d])); \]

\[ \text{Uncompress}(\delta_1, \ldots, \delta_d, v) = (\text{uncompress}^{(1)}(\delta_1, v[1]), \ldots, \text{uncompress}^{(d)}(\delta_d, v[d])). \]

### 4.3.1 Translate.

Let \( D_i = \{\text{Idx}(x) : x \in D_x\} \) be the set of indices of points in dataset \( D_x \). Given a query with range \( B = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d] \subset \mathbb{N}^d \), which asks for the sum \( \bigoplus_{x \in D_x \cap B} x \), algorithm Translate output a query with range \( R = [\beta_1, \gamma_1] \times [\beta_2, \gamma_2] \times \ldots \times [\beta_d, \gamma_d] \subset [0, 1]^d \), which asks for the sum \( \bigoplus_{i \in D_i \cap R} \text{Att}(i) \), such that (1) the two queries are equivalent, that is, \( \bigoplus_{x \in D_x \cap B} x = \bigoplus_{i \in D_i \cap R} \text{Att}(i) \); (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 \( \{(\beta_i, x_i, t_{\beta_i}; \beta_i, x_i, t_{\beta_i} ; \gamma_i, y_i, t_{\gamma_i}; \bar{\gamma}_i, \bar{y}_i, t_{\bar{\gamma}_i}) : i \in [d]\} \).
3. Alice outputs \( \{(\beta_i, \gamma_i) : i \in [d]\} \), if
   a. for each \( i \in [d] \), \( x_i < a_i \leq x_i \) and \( y_i \leq b_i < \bar{y}_i \);
   b. \( t_{\beta_i}, t_{\beta_i}, t_{\gamma_i} \) and \( t_{\bar{\gamma}_i} \) are valid itag values of \( (\tilde{\beta}_i, \tilde{x}_i) \), \( (\beta_i, x_i) \), \( (\gamma_i, y_i) \) and \( (\bar{\gamma}_i, \bar{y}_i) \), respectively.
   Otherwise, Alice output \( \perp \).

### 5 Dynamic MAIA

The dynamic MAIA is a \( RC \) protocol \([25]\) between Alice and Bob, and consists of setup phase and query phase. In setup phase, Alice runs the key generating algorithm \texttt{KGen} to generate a private key, then encodes the dataset using data encoding algorithm \texttt{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 \texttt{KeyGen}. \texttt{KeyGen}(1\(^n\)):

Alice

1. runs algorithm \( \mathcal{G}(1^n) \) 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 \( \mathbb{Z}_p^* \times \mathbb{Z}_q^* \times \{0, 1\}^n \) for each \( \ell \in [d] \).
Let $s_1 = (s_{1,1}, s_{2,1}, \ldots, s_{d,1})$, $s_2 = (s_{1,2}, s_{2,2}, \ldots, s_{d,2})$ and $s_3 = (s_{1,3}, s_{2,3}, \ldots, s_{d,3})$. Let $K = (s_1, s_2, s_3, n, p, q, r, g_p, g_q, g_r)$. Output $K$.

Next, we define the algorithm $KGen$ based on $KeyGen$.

**KGen:**

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

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

**5.2 DEnc.**

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

**5.3 Query phase**

**5.3.1 Query.** $\langle \text{Eval}, \text{Ext} \rangle(\text{SUM}(R))$. Let $ProVer$ be as in scheme MAIA [25].

1. Simulate $ProVer$ on input $\text{SUM}(R)$ w.r.t. dataset $D$ using private key $K$, and obtain result $X$.
2. Simulate $ProVer$ on input $\text{SUM}(R)$ w.r.t. dataset $\bar{D}$ using private key $\bar{K}$, and obtain result $\bar{X}$.
3. Output $X - \bar{X}$.

**5.3.2 Insert.**

$Insert(x, D, T, K, \pi^*)$: Insert $x$ into $D$.

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

Delete($x$): Delete $x$ from the dataset $D_x$.

1. If $x \in D_x$, then $\text{Insert}(x, \overline{D}, \overline{T}, \overline{K}, \pi^*)$. 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

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)*.