Secure Outsourced Top-k Selection Queries against Untrusted Cloud Service Providers

Xixun Yu*, Yidan Hu†, Rui Zhang†, Zheng Yan*‡, and Yanchao Zhang§
*State Key Lab on Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an, 710071, China
†Department of Computer and Information Sciences, University of Delaware, Newark, DE, 19716, USA
‡Department of Communications and Networking, Aalto University, Espoo, 02150, Finland
§School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, 85287, USA
xxyu@stu.xidian.edu.cn,{yidanhu,ruizhang}@udel.edu,zyan@xidian.edu.cn,yczhang@asu.edu

Abstract—**As cloud computing reshapes the global IT industry, an increasing number of business owners have outsourced their datasets to third-party cloud service providers (CSP), which in turn answer data queries from end users on their behalf. A well known security challenge in data outsourcing is that the CSP cannot be fully trusted, which may return inauthentic or unsound query results for various reasons. This paper considers top-k selection queries, an important type of queries widely used in practice. In a top-k selection query, a user specifies a scoring function and asks for the k objects with the highest scores. Despite several recent efforts, existing solutions can only support a limited range of scoring functions with explicit forms known in advance. This paper presents three novel schemes that allow a user to verify the integrity and soundness of any top-k selection query result returned by an untrusted CSP. The first two schemes support monotone scoring functions, and the third scheme supports scoring functions comprised of both monotonically non-decreasing and non-increasing subscoring functions. Detailed simulation studies using a real dataset confirm the efficacy and efficiency of the proposed schemes and their significant advantages over prior solutions.**

I. Introduction

As cloud computing revolutionizing the global IT industry, data outsourcing has gained increasing popularity among business owners. In a typical data outsourcing system, a data owner outsources its dataset to a third-party cloud service provider (CSP), which in turn answers various types of data queries from many users on behalf of the data owner. In comparison with hosting the dataset and query services over dedicated infrastructure, data outsourcing offers significant benefits to data owners such as high elasticity, easy management and maintenance, and cost saving.

This paper considers top-k selection queries [1], an important type of queries which ask for the top k objects among a dataset ranked by a user-defined scoring function. In a typical top-k selection query, a user specifies a scoring function over multiple attributes of the object, and the CSP needs to return the k objects of which the scores are the highest. As an example, consider a dataset of used cars with two attributes price and mileage shown in Table 1. A user interested in buying a cheap car with low mileage may issue a top-3 selection query with a scoring function $\texttt{SCORE} = 0.8{*}\texttt{PRICE}{+}0.5{*}\texttt{MILEAGE}$. The scores of the four records are 56,000, 50,000, 49,000, and 47,000, respectively.

TABLE I
Used Car

ID	PRICE	MILEAGE
1	\$ 20,000	80,000
2	\$ 25,000	60,000
3	\$ 30,000	50,000
4	\$ 40,000	30,000

The top 3 objects with respect to the scoring function is then the fourth, third, and second cars. Top-k selection queries have many real-world applications and have attracted significant attentions from both academia and industry.

A well known security challenge in data outsourcing is that the CSP cannot be fully trusted, which may return forged or unsound query result in response to users' queries [2]. In the context of top-k selection queries, the CSP may return tampered or forged objects or replace one or multiple top-k objects with the ones that are authentic but not among the top k. For example, the CSP may return some objects to prompt associated businesses with financial interests. These situations call for effective mechanisms to allow users to verify both the authenticity and soundness of any top-k selection query result returned by the CSP. A query result is considered *authentic* if all the returned objects are among the data owner's dataset and have not been tampered with and *sound* if it contains the k objects of which the scores are highest with respect to the user's scoring function.

Despite significant efforts on authenticating outsourced query processing, authenticating outsourced top-k selection queries poses unique challenges and has so far received very limited attention. In particular, the ranking of the objects with respect to a top-k selection query is determined by the user-defined scoring function, which cannot be predicted in advance. In addition, any practical system needs to support a wide range of scoring functions that are difficult to enumerate in advance. These challenges make it difficult to predetermine the ranking of the objects and also render existing solutions on authenticating outsourced top-k queries over a single attribute [3]–[10] inapplicable. To the best of our knowledge, Ref. [12], [15] are the only two existing solutions for authenticating outsourced top-k selection queries that support user defined scoring functions over multiple attributes. They, however, require that the scoring functions to have explicit forms known in advance so that the order of the objects can be precomputed for every possible query. As this assumption may not hold in practice, there is thus a pressing need for developing effective mechanisms for authenticating outsourced top-k selection queries that support a broader range of scoring functions.

In this paper, we tackle this challenge by introducing three novel schemes for authenticating outsourced top-k selection queries with user-defined scoring functions over multiple attributes against an untrusted CSP. All three schemes explore a partial preference relationship among objects by having the CSP return some additional objects in addition to the top k objects to prove that there is no other object with a score higher than the lowest score among the returned top k objects. The first two schemes support monotone scoring functions but differ in which additional objects need be returned for query result verification. The third scheme further supports scoring functions comprised of both monotonically non-decreasing and non-increasing subscoring functions. Our contributions in this paper can be summarized as follows.

- We introduce three novel schemes for authenticating outsourced top-k selection queries that allow the user to verify both the integrity and soundness of any query result returned by an untrusted CSP.
- We confirm the efficacy and efficiency of the proposed schemes and their advantages over prior solutions via detailed simulation studies using a real dataset.

The rest of the paper is structured as follows. Section II discusses the related work. Section III presents the system, adversary, and query models and our design goals. We present two basic schemes in Section IV and an advanced scheme in Section V. We report the simulation results of the proposed schemes in Section VI and finally conclude this paper in Section VII.

II. Related Work

Our work is mostly related to the studies on authenticating outsourced top-k queries. Yu *et al.* [5], [6] introduced a verifiable top-k query scheme for tiered sensor networks by introducing encrypted dummy readings to ensure query-result completeness. In [3], [8], Zhang *et al.* introduced several techniques for verifiable top-k queries in two-tiered sensor networks by chaining adjacent objects using cryptographic techniques. This technique was later used in [4], [9] to authenticate outsourced spatial top-k queries against an untrusted location service provider, in which the top-k queries are over the records within a user-defined geographic region. Similar problems have also been studied in [7], [10], [13]. Common to all these solutions [3]–[10], [13] is that they target top-k queries over a single attribute with no explicit scoring function.

There have been very limited efforts on authenticating top-k queries with user-defined scoring functions over multiple attributes. Tsou *et al.* [14] introduced SFTopk, a solution for verifiable top-k selection queries over multiple attributes but only support scoring functions as the conjunction or sum of the attributes. Yang *et al.* [15] introduced a scheme for authenticating outsourced function queries that can support top-k selection queries with a limited scoring functions. Their scheme treats objects as function templates and user inputs as variables and precomputes all the subspaces in which the objects exhibit distinct orders. In contrast, we do not assume the scoring function to have any explicit form but consists of monotonically non-increasing or non-decreasing subscoring functions, for which the technique [15] is inapplicable. More recently, Zhu *et al.* [11], [12] proposed a solution that supports the same scoring functions as [15] by partitioning the domain of a dataset into grids with each containing at most one real or dummy record and chaining adjacent grids using cryptographic techniques to enable query result verification. Unfortunately, the total number of grids under their solution is n^d, where n is the number of records and d is the number of attributes. This leads to a high computation complexity of $O(n^d)$.

Authenticating outsourced queries has received significant attentions over the past decade. Many different types of queries have been investigated, including range queries [16], [17], skyline queries [9], [18], [19], kNN queries [20]–[22], shortest-path queries [23], SQL queries [24]–[26], the most recent data queries [27], [28], sliding windows queries [29], grouped aggregation queries [30], etc. None of these works consider the top-k selection query over multiple attributes, and they are orthogonal to our work in this paper.

III. Problem Formulation

In this section, we introduce the system, query, and adversary models, as well as our design goals.

A. System Model

We consider a data outsourcing system comprising a data owner, a third-party CSP, and many users. The data owner has a dataset D consisting of n objects $\{o_i | 1 \leq i \leq n\}$. Each object o_i has d attributes and is denoted by $o_i = (a_{i,1}, \ldots, a_{i,d})$, where $a_{i,j}$ is the jth attribute for all $1 \leq j \leq d$. We assume that the range of every attribute is known in advance and denote the range of the jth attribute by $R_j = [r^j_{\min}, r^j_{\max}]$. Equivalently, we can view the dataset D as a set of n points in the d-dimensional Euclidian space. The data owner outsources the dataset D to the CSP, which in turn answers top-k selection queries from the users on behalf of the data owner.

B. Top-k Selection Query Model

We consider top-k selection query in this paper. Specifically, any user can issue a top-k selection query by specifying $\langle F, k \rangle$, where k is the number of objects requested and F is a scoring function that maps every object in D to a real value in the range $(-\infty, \infty)$. The user may ask for the objects with the k highest or lowest scores under F. Our subsequent discussion focuses on one user and assumes that the user is requesting the highest ranked objects. More specifically, the user requests for a set of k objects $O \subseteq D$ such that for all $o_i \in O$ and

$o_j \in D \setminus O$, $F(o_i) \geq F(o_j)$. Note that the query result for a top-k selection query may not be unique as there could be multiple objects sharing the same score.

In this paper, we assume that the scoring function is a separable function with the following form

$$F(o_i) = \sum_{j=1}^{d} f_j(a_{i,j}) \ , \quad (1)$$

where $f_j(\cdot)$ is the subscoring function defined over the jth attribute for all $1 \leq j \leq d$. In addition, we assume that each subscoring function f_j is either monotonically non-decreasing or non-increasing within the range R_j. We assume that the scoring function F cannot be predicted in advance.

C. Adversary Model

We assume that the communications among the three parties are protected by proper cryptographic techniques and that the data owner is trusted to follow all system operations. In contrast, the CSP may return tampered objects or objects with scores not among the top k with respect to scoring function F in response to the user's queries. We further assume that the user has no prior knowledge about dataset D but knows the data owner's public key.

D. Design Goals

We aim to enable the end users to verify the correctness of any top-k selection query result returned by the CSP. Let O be the set of k objects returned by the CSP in response to the user's query. The user needs to verify whether the query result satisfies the following two conditions.

- *Query-result integrity*: Every returned object must belong to D and have not be tampered with, i.e., $O \subseteq D$.
- *Query-result soundness*: Every returned object must have a score no lower than every other object with respect to F, i.e., for all $o_x \in O$ and $o_y \in D \setminus O$, $F(o_x) \geq F(o_y)$.

A query result is considered correct if and only if it passes both verifications.

IV. Two Basic Schemes

In this section, we introduce two basic schemes that support monotone scoring functions.

A. Overview

The two basic schemes both ensure the integrity of every object returned by the CSP using efficient cryptographic primitives and require the CSP to return some additional information to prove the soundness of the query result.

Suppose that the user issues a top-k selection query with scoring function F. Without loss of generality, suppose that $F(o_1) \geq F(o_2) \geq \cdots \geq F(o_n)$. Since there may be multiple objects that share the same score with respect to F, a sound query result must include all the objects with scores higher than $F(o_k)$ and one or multiple objects with scores equal to $F(o_k)$. Equivalently, a sound query result must contain all the objects in D with scores higher than the lowest score among the top k objects. Based on this idea, both basic schemes require the CSP to return some additional objects to prove that there is no object in $D \setminus O$ with a score higher than the lowest score among the top k objects. They, however, differ in which objects besides the top k need be returned for query-result verification.

B. Basic Scheme 1

Let $J = \{1, \ldots, d\}$ be the set of attribute indexes. When all the subscoring functions $f_1, \ldots, f_d$ are monotonically non-decreasing (or non-increasing), the scoring function F is *monotone*. Scheme 1 explores a fundamental property of monotone scoring function.

Lemma 1. *Assume that all subscoring functions $f_1, \ldots, f_d$ are monotonically non-decreasing. For any two objects o_x and o_y, if $F(o_x) > F(o_y)$, then there must exist $j \in J$ such that $a_{x,j} > a_{y,j}$.*

Proof: We prove this lemma by contradiction. Let $o_x = (a_{x,1}, \ldots, a_{x,d})$ and $o_y = (a_{y,1}, \ldots, a_{y,d})$ be two arbitrary objects. Assume that f_j is monotonically non-decreasing for all $j \in J$ and that $F(o_x) > F(o_y)$. By definition of the scoring function, we have $\sum_{j \in J} f_j(a_{x,j}) > \sum_{j \in J} f_j(a_{y,j})$. Now suppose that $a_{x,j} \leq a_{y,j}$ for all $j \in J$. It follows that $\sum_{j=1}^{d} f_j(a_{x,j}) \leq \sum_{j=1}^{d} f_j(a_{y,j})$ and thus $F(o_x) \leq F(o_y)$, which leads to a contradiction. The lemma is therefore proved. ■

Scheme 1 determines the additional objects that need be returned for soundness verification based on Lemma 1. Let $O = \{o_{\lambda_1}, \ldots, o_{\lambda_k}\}$ be the set of top k objects with respect to the scoring function F chosen by the CSP, where $F(o_{\lambda_1}) \geq \cdots \geq F(o_{\lambda_k})$. We hereafter refer to o_{λ_k} as the *critical object* with respect to F. Based on Lemma 1, Scheme 1 requires the CSP to return every object that has at least one attribute larger than the corresponding attribute of the critical object o_{λ_k} so that the user is able to verify that there is no object in $D \setminus O$ with score higher than $F(o_{\lambda_k})$.

In what follows, we detail the three phases of Scheme 1, including data preprocessing at the data owner, query processing at the CSP, and query-result verification at the user.

1) Data Preprocessing: Assume that the data owner has a dataset $D = \{o_i\}_{i=1}^{n}$, where $o_i = (a_{i,1}, \ldots, a_{i,d})$. The data owner processes the dataset D as follows.

First, the data owner sorts the objects according to their jth attribute values in a descending order for each $j \in J$ to create d sorted lists $L_1, \ldots, L_d$. Consider the jth attribute as an example. Let $(\pi_j(1), \ldots, \pi_j(n))$ be a permutation of $(1, \ldots, n)$ such that $a_{\pi_j(1),j} \geq a_{\pi_j(2),j} \geq \cdots \geq a_{\pi_j(n),j}$ for all $1 \leq j \leq d$. The jth sorted list is then given by

$$L_j = \langle o_{\pi_j(1)}, o_{\pi_j(2)}, \ldots, o_{\pi_j(n)} \rangle.$$

Further denote by $\pi_j^{-1}(\cdot)$ the inverse permutation of $\pi_j(\cdot)$. It follows that object $o_{\pi_j(i)}$ is ranked ith and object o_i is ranked $\pi_j^{-1}(i)$th in list L_j for all $1 \leq i \leq n$.

The data owner then creates a one-way hash chain for each list L_j. Specifically, the data owner first computes $h_i = H(o_i)$

for all $1 \leq i \leq n$, where $H(\cdot)$ denotes a cryptographic hash function such as SHA-256. For each list $L_j, j \in J$, the data owner constructs a one-way hash chain by recursively computing

$$\sigma_{j,x} = \begin{cases} H(h_{\pi_j(x)} || \sigma_{j,x+1}) & \text{if } 1 \leq x \leq n, \\ h^* & \text{if } x = n+1 \,, \end{cases} \tag{2}$$

where "$||$" denotes concatenation and h^* is a publicly known special string.

The data owner further constructs a Merkle hash tree over $\sigma_{1,1}, \sigma_{2,1}, \ldots, \sigma_{d,1}$ and signs the root of the Merkle hash tree using its private key. Finally, the data owner sends the CSP the dataset D, the d lists $L_1, \ldots, L_d$, and its signature on the Merkle hash tree root. On receiving the information, the CSP can compute all the non-root nodes of the Merkle hash tree.

2) Query Processing: Assume that the user issues a top-k selection query with a scoring function F. On receiving the query, the CSP first computes the top k objects with respect to F using an efficient algorithm such as [31]. As discussed earlier, the top k objects may not be unique if multiple objects share the same score. Let the top k objects selected by the CSP be $O = \{o_{\lambda_1}, \ldots, o_{\lambda_k}\}$, where $F(o_{\lambda_1}) \geq \cdots \geq F(o_{\lambda_k})$ and o_{λ_k} is the critical object. Further let $J^+ = \{j | f_j \text{ is not a constant function}, j \in J\}$ be the set of indexes of subscoring functions that are not a constant function. To prove the soundness of the query result to the user, i.e., $o_{\lambda_1}, \ldots, o_{\lambda_k}$ are indeed the top k objects with respect to F, we require the CSP to additionally return all the objects with at least one attribute $j \in J^+$ larger than the corresponding attribute of the critical object o_{λ_k}.

Specifically, for every attribute $j \in J^+$, the CSP finds all the objects that are ranked higher than the critical object o_{λ_k} in the list L_j. Since object o_{λ_k} is ranked $\pi_j^{-1}(\lambda_k)$th in the list L_j, the subset of objects ranked no lower than the critical object o_{λ_k} in list L_j is given by $V_j = \{o_i | 1 \leq \pi_j^{-1}(i) \leq \pi_j^{-1}(\lambda_k)\}$. Further let $V = \bigcup_{j \in J^+} V_j$. It follows that $O \subseteq V$, because for each object $o_{\lambda_x}, 1 \leq x \leq k$, there must exist at least one list $L_j, j \in J^+$ such that o_{λ_x} is ranked higher than o_{λ_k}.

Denote by $A(\sigma_{j,1})$ the subset of internal nodes of the Merkle hash tree needed for computing the root of the Merkle hash tree from leaf node $\sigma_{j,1}$ for all $j \in J$. The CSP returns the following information as the query result.

- Every object in V along with its ID $\{\langle i, o_i \rangle | o_i \in V\}$.
- The IDs of the top k objects, $\lambda_1, \ldots, \lambda_k$.
- For each $j \in J^+$, the ID of every object $o_i \in V_j$, i.e., $\{i | 1 \leq \pi_j^{-1}(i) \leq \pi_j^{-1}(\lambda_k)\}$, along with element $\sigma_{j, \pi_j^{-1}(\lambda_k)+1}$.
- $\bigcup_{j \in J^+} A(\sigma_{j,1})$, which is the union of internal nodes in the Merkle hash tree needed for computing the root from every leaf node in $\{\sigma_{j,1} | j \in J^+\}$ and the data owner's signature on the root of the Merkle hash tree.

3) Query-result Verification: On receiving the query result, the user verifies its integrity and soundness as follows.

Integrity verification. First, for every attribute $j \in J^+$, the user computes the $\sigma_{j,1}$ from the returned information. Consider attribute j as an example. For each $\pi_j(x)$, $1 \leq x \leq \pi_j^{-1}(\lambda_k)$, the user computes $h_{\pi_j(x)} = H(o_{\pi_j(x)})$. With $h_{\pi_j(1)}, h_{\pi_j(2)}, \ldots, h_{\lambda_k}, \sigma_{j, \pi_j^{-1}(\lambda_k)+1}$, the user further computes $\sigma_{j,1}$ according to Eq. (2). Next, for each $\sigma_{j,1}, j \in J^+$, the user computes the root of the Merkle hash tree using the $A(\sigma_{j,1})$. If all the $\{\sigma_{j,1} | j \in J^+\}$ lead to the same root, the user further verifies the data owner's digital signature on the root of the tree. If all the verifications succeed, the user considers the query result authentic and proceeds to verify its soundness.

Soundness verification. Given IDs $\lambda_1, \ldots, \lambda_k$, the user identifies the top k objects as $O = \{o_{\lambda_1}, \ldots, o_{\lambda_k}\}$. The user then verifies whether $F(o_{\lambda_1}) \geq \cdots \geq F(o_{\lambda_k})$. If so, the user identifies o_{λ_k} as the critical object and proceeds to check whether there exists any object in $V \setminus O$ that has a higher score than $F(o_{\lambda_k})$. Let $V'_j = \{o_i | 1 \leq i \leq n, 1 \leq \pi_j^{-1}(i) < \pi_j^{-1}(\lambda_k)\}$ for all $j \in J^+$ and $V' = \bigcup_{j \in J^+} V'_j$. The user further verifies if $F(o_i) \leq F(o_{\lambda_k})$ for all $o_i \in V' \setminus O$. If all the verifications succeed, the user considers the query result sound.

C. Basic Scheme 2

Different from Scheme 1 that requires the CSP to return every object with at least one attribute higher than the corresponding attribute of the critical object, Scheme 2 explores a novel preference tree structure to determine the additional objects that need be returned for query result verification. We first define a preference relationship among objects.

Definition 1. *For any two objects o_x and o_y, object o_x is* preferable *to object o_y, denoted by $o_x \succeq o_y$, if and only if $a_{x,j} \geq a_{y,j}$ for all $j \in J$.*

We then have the following lemma.

Lemma 2. *For any two objects o_x and o_y, if $o_x \succeq o_y$, then $F(o_x) \geq F(o_y)$ for any non-negative monotone scoring function.*

The proof is straightforward and omitted due to the space constraints.

We now introduce a novel *preference tree* structure to represent a dataset $D = \{o_i\}_{i=1}^n$. Specifically, we introduce a special object o_0 with $a_{0,j} = r_{\max}^j$ for all $j \in J$. Let $D' = D \bigcup \{o_0\}$. A preference tree T consists of $n+1$ nodes with each corresponding to one unique object in D', and every parent object is preferable to all of its children objects. It is easy to see that the object o_0 would be the root of any preference tree representation of D as $o_0 \succeq o_i$ for all $1 \leq i \leq n$. Note that the preference tree representation of the dataset D may not be unique, as the same object could be the children of different objects. We will hereafter denote by o_V the object corresponding to node V and use o_V and V interchangeably when no confusion arises.

Given a preference tree T representing the dataset D, we can convert any top-k selection query over D into a top-k selection query over T by requiring the query result to exclude the special object o_0. We then have the following lemma.

Lemma 3. *Let T be a preference tree corresponding to a dataset D and F a monotone scoring function. For any object*

$o_x \in D$, the subset of objects with scores no lower than $F(o_x)$ form a subtree rooted at o_0.

Proof: Let $O_x^+ = \{o_i | 0 \le i \le n, F(o_i) \ge F(o_x)\}$. It suffices to prove that for any node $o_y \in O_x^+$ its parent node also belongs to O_x^+, which is apparent as the parent node must have a score no lower than o_y. ■

Based on the above property, we can further define two subtrees of T with respect to an object o_x. Specifically, we define a *preference subtree* with respect to object o_x, denoted by T_{o_x}, as a subtree of T formed by $O_x^+ = \{o_i | 0 \le i \le n, F(o_i) \ge F(o_x)\}$. We also define the *expanded preference subtree* with respect to object o_x, denoted by $T^e_{o_x}$, as the subtree of T formed by all the nodes in T_{o_x} in addition to the nodes of which the parent has a score higher than $F(o_x)$.

Scheme 2 explores the above property to determine the additional objects that needs to be returned for query-result verification. Suppose that $o_{\lambda_1}, \dots, o_{\lambda_k}$ are the top k objects selected by the CSP with respect to F, where $F(o_{\lambda_1}) \ge \cdots \ge F(o_{\lambda_k})$. Scheme 2 requires the CSP to return all the objects in the expanded preference subtree $T^e_{o_{\lambda_k}}$ whereby the user can verify that all the objects in the preference subtree $T_{o_{\lambda_k}}$ have been returned and that no other object besides $o_{\lambda_1}, \dots, o_{\lambda_k}$ has a score higher than $F(o_{\lambda_k})$. In what follows, we detail the three phases of Scheme 2.

1) Data Preprocessing: First, the data owner constructs a preference tree T from the dataset D. As discussed earlier, the preference tree constructed from the dataset D may not be unique. Since we require the CSP to return all the objects in expanded preference subtree $T^e_{o_{\lambda_k}}$ for query result verification, it is necessary to minimize the number of children nodes that each node has to reduce the communication cost.

Based on this observation, we introduce a divide-and-conquer algorithm for constructing a preference tree T from the dataset D. Since the special object o_0 should be the root of T, the algorithm recursively constructs a preference tree T rooted at o_0. In each subproblem, we take a parent node o_x (i.e., o_0 in the initial call) and a set of descendant nodes E (i.e., D in the initial call) as input where $o_x \succeq o_y$ for all $o_y \in E$ and performs the following two tasks.

- **Task 1**: Find a subset of objects $C \subseteq E$ to be the immediate children of o_x.
- **Task 2**: Partition the remaining objects $E \setminus C$ into $|C|$ possibly empty subsets and assign one subset to each object in C as its descendants.

Let us take a look at the first task. Given a parent object o_x and a set of descendant objects E, we select the subset of objects in E to each of which no other object is preferable as the immediate children of o_x. Doing so can minimize the number of children nodes of o_x. Special attention need be given to the case of multiple objects having exactly same attributes. In particular, for any two objects o_y and o_z, if $o_y = o_z$, then we have both $o_y \succeq o_z$ and $o_z \preceq o_y$. In this case, we can select either one of o_y and o_z as o_x's immediate children node, as the other identical object can be the children of the selected object.

Now we discuss how to assign the remaining objects $E \setminus C$ to each of the children objects C. Our intuition is that the deeper the tree is, the fewer children nodes each non-leaf node has, the fewer nodes in any expanded preference subtree on average, and vice versa. Based on this idea, we iteratively assign children objects to each object in C by maximizing the number of objects assigned in each iteration. Specifically, we first construct a bipartite graph from C and $E \setminus C$ where each vertex corresponds to a unique object in E. There is an edge between object $o_x \in C$ and $o_y \in E \setminus C$ if and only if $o_x \succeq o_y$. We then find the object in C with the largest degree and assign all the objects incident to C as its decedents. We then remove objects that have been assigned from the bipartite graph and proceed to the next iteration. This process continues until all the objects in $E \setminus C$ have been assigned.

Once the two tasks are accomplished, we recursively construct a subtree rooted at each node in C with the assigned descendant nodes until the entire tree is constructed.

Given the preference tree T, the data owner further constructs a *preference hash tree* to encode the preference tree structure using cryptographic hash functions.

Definition 2. ***(Preference Hash Tree)*** *A preference hash tree* T *is a tree constructed from a preference tree T with the same structure, where every node* v *corresponds to one unique node in T and consists of the following fields.*

- $\mathsf{v.id}$*: the ID of object o_v.*
- $\mathsf{v.hash} = H(o_\mathsf{v})$*: the hash of object o_v.*
- $\mathsf{v.cid} = \langle \mathsf{u.id} | \mathsf{u} \in \mathsf{v.children} \rangle$*: The list of the IDs of the children objects.*
- $\mathsf{v.chash} = H(||_{\mathsf{u} \in \mathsf{v.children}} \mathsf{u})$*: The hash of the concatenation of its children objects.*

The preference hash tree T can be easily constructed from the preference tree T in a bottom-up fashion. Specifically, for every leaf node v that corresponds to object o_v, we set $\mathsf{v.id} = \emptyset$ and $\mathsf{v.chash} = \emptyset$, where $\emptyset$ is a publicly known special string indicating that v is a leaf node. We then set $\mathsf{v.id}$ and $\mathsf{v.hash}$ as the ID and the hash of o_v, respectively. Now consider a non-leaf node u that corresponds to object o_u with a set of children nodes $\mathsf{u.children}$. Similar to the leaf node, we first compute $\mathsf{u.chash} = H(||_{\mathsf{v} \in \mathsf{u.children}} \mathsf{v})$.

Once the preference hash tree T is constructed, the data owner sends all the objects $\{o_i\}_{i=1}^n$, the preference hash tree T, and its signature over the root of T to the CSP.

2) Query Processing: Assume that the data user submits a top-k selection query with scoring function F. On receiving the query, the SP constructs the query result as follows.

The CSP first finds a set of k objects with the highest scores $O = \{o_{\lambda_1}, \dots, o_{\lambda_k}\}$, where $F(o_{\lambda_1}) \ge F(o_{\lambda_2}) \ge \cdots \ge F(o_{\lambda_k})$ and o_{λ_k} is the critical object. We define the *preference hash subtree* $\mathsf{T}_{o_{\lambda_k}}$ with respect to object o_{λ_k} as the subtree of T formed by objects $O_x^+ = \{o_i | 0 \le i \le n, F(o_i) \ge F(o_{\lambda_k})\}$, and the *expanded preference hash subtree* $\mathsf{T}^e_{o_{\lambda_k}}$ with respect to object o_{λ_k} as the subtree of T formed by $\mathsf{T}_{o_{\lambda_k}}$ in addition to the nodes of which the parent has a score higher than $F(o_{\lambda_k})$.

The CSP then constructs the query result based on the expanded preference hash subtree $\mathsf{T}^e_{o_{\lambda_k}}$. For every node $\mathsf{v} \in \mathsf{T}^e_{o_{\lambda_k}}$, the CSP returns

$$R_{\mathsf{v}} = \langle o_{\mathsf{v}}, \mathsf{v.id}, \mathsf{v.hash}, \mathsf{v.cid}, \mathsf{v.chash} \rangle \ . \tag{3}$$

Moreover, the CSP also returns the IDs of the top k objects $\lambda_1, \ldots, \lambda_k$ and the data owner's signature on the root of T.

3) Query-Result Verification: On receiving the query result, the user first reconstructs expanded preference hash subtree $\mathsf{T}^e_{o_{\lambda_k}}$ based on the received $\{R_{\mathsf{v}} | \mathsf{v} \in \mathsf{T}^e_{o_{\lambda_k}}\}$ and then verifies its integrity and soundness as follows.

Integrity verification. First, for every received $R_{\mathsf{v}}, \mathsf{v} \in \mathsf{T}^e_{o_{\lambda_k}}$, the user verifies if $\mathsf{v.hash} = H(o_{\mathsf{v}})$. Second, for every non-leaf node v of $\mathsf{T}^e_{o_{\lambda_k}}$, the user obtains the list of its children from $\mathsf{v.cid}$ and checks if $\mathsf{v.chash} = H(||_{\mathsf{u} \in \mathsf{v.children}} \mathsf{u})$. Finally, the user verifies the data owner's signature on the root of the T. If all the verifications succeed, the user considers the query result authentic.

Soundness verification. The user identifies the top k objects $o_{\lambda_1}, \ldots, o_{\lambda_k}$ based on the received IDs $\lambda_1, \ldots, \lambda_k$. The user then checks if $F(o_{\lambda_1}) \geq \cdots \geq F(o_{\lambda_k})$ and identifies the critical object o_{λ_k}. The user further checks if there is any other object with score higher than o_{λ_k} among all the returned objects other than $o_{\lambda_1}, \ldots, o_{\lambda_k}$. If not, the user checks if all the nodes in expanded preference subtree $\mathsf{T}^e_{o_{\lambda_k}}$ have been returned. Specifically, for every leaf node $\mathsf{v} \in \mathsf{T}^e_{o_{\lambda_k}}$, the user checks whether either one of following two conditions holds

- **Condition 1**: The corresponding object has a score equal or lower than $F(o_{\lambda_k})$.
- **Condition 2**: The object has a score higher than $F(o_{\lambda_k})$ but has no child, i.e., $\mathsf{v.cid} = \emptyset$.

If all the verifications succeed, the user considers the query result sound.

V. An Advanced Scheme

In this section, we introduce an advanced scheme that supports any scoring function F comprised of both monotonically non-decreasing and non-increasing subscoring functions.

A. Overview

The advanced scheme is designed by generalizing the property characterized by Lemma 1. Let $F = (f_1, \ldots, f_d)$ be a scoring function comprised of monotonically non-decreasing and non-increasing subscoring functions. Further let $J_\uparrow$ and $J_\downarrow$ be the sets of indexes of non-decreasing and non-increasing subscoring functions, respectively, where $J_\uparrow \bigcup J_\downarrow = J$ and $J_\uparrow \bigcap J_\downarrow = \emptyset$. We have the following Lemma as a generalization of Lemma 1.

Lemma 4. *Let F be a scoring function comprised of monotonically non-decreasing subscoring functions $\{f_j | j \in J_\uparrow\}$ and monotonically non-increasing subscoring functions $\{f_j | j \in J_\downarrow\}$. Let $J_1, \ldots, J_s$ be a family of subsets of $J_\uparrow$ such that $\bigcup_{i=1}^s J_i = J_\uparrow$ and $J'_1, \ldots, J'_t$ be a family of subsets of $J_\downarrow$ such that $\bigcup_{i=1}^s J_i = J_\downarrow$. For any two objects o_x and o_y, if $F(o_x) > F(o_y)$, then there must exist either $i \in \{1, \ldots, s\}$ such that*

$$\sum_{j \in J_i} f_j(a_{x,j}) > \sum_{j \in J_i} f_j(a_{y,j}) \ ,$$

or $i \in \{1, \ldots, t\}$ such that

$$\sum_{j \in J'_i} f_j(a_{x,j}) < \sum_{j \in J'_i} f_j(a_{y,j}) \ .$$

The proof is similar to that of Lemma 1 and omitted here due to space limitations. Based on Lemma 4, we define a partial preference relationship as follows.

Definition 3. *Let $J' \subseteq J$ be a subset of attributes. For any two objects o_x and o_y, object o_x is preferable to object o_y* with respect to attributes J', *denoted by $o_x \succeq_{J'} o_y$, if and only if $a_{x,j} \geq a_{y,j}$ for all $j \in J'$.*

For any subset of attributes $J_x \in J$, we can define both a *partial preference tree* T_x and an *inverse partial preference tree* T^-_x. Specifically, a partial preference tree T_x with respect to J_x is similar to a preference tree with the exception that for every parent node o_x and children node o_y, $o_x \succeq_{J_x} o_y$. Similarly, an inverse partial preference tree T^-_x is a tree in which for every parent node o_x and children node o_y, $o_x \preceq_{J_x} o_y$. Given a partial preference tree T_x and an inverse partial preference tree T^-_x, we can construct the corresponding partial preference hash tree T_x and the inverse partial preference hash tree T^-_x accordingly.

Under the advance scheme, the data owner constructs a partial preference hash tree and an inverse partial preference hash tree for every non-empty subset $J_x \subseteq J$. On receiving a top-k selection query, the CSP divides the scoring function $F = (f_1, \ldots, f_d)$ into the subset of monotonically non-decreasing subscoring functions $\{f_j | j \in J_\uparrow\}$ and the subset of monotonically non-increasing subscoring functions $\{f_j | j \in J_\downarrow\}$ and constructs one partial query result for each of them. On receiving the query result, the user verifies the soundness of the query result according to Lemma 4. In what follows, we details its operations.

B. Data Preprocessing

Given a dataset D, the data owner first generates the family of non-empty subsets of index set J as $\mathcal{P} = \{J_x | J_x \subseteq 2^J \setminus \{\emptyset\}\}$. It follows that $|\mathcal{P}| = 2^d - 1$.

For each subset $J_x \in \mathcal{P}$, the data owner constructs a partial preference hash tree T_x as in Scheme 2 with the exception that every parent node is preferable to all of its children nodes with respect to J_x. The data owner also constructs an inverse partial preference hash tree T^-_x in which every children node is preferable to its parent node with respect to J_x and the root node is another special object $o^-_0 = (a^1_{\min}, \ldots, a^d_{\min})$. The data owner then sends $\{o_i | 1 \leq i \leq n\}$ and $\{\langle \mathsf{T}_x, \mathsf{T}^-_x \rangle | J_x \in \mathcal{P}\}$ along with its signature on the root of every partial preference hash tree and inverse partial preference hash tree to the CSP.

C. Query Processing

Assume that the user issues a top-k selection query with a scoring function F. The CSP first finds the top k objects with respect to F as $O = \{o_{\lambda_1}, \ldots, o_{\lambda_k}\}$ where $F(o_{\lambda_1}) \geq \cdots \geq F(o_{\lambda_k})$ and o_{λ_k} is the critical object.

The CSP then separately constructs one partial query result for the subset of non-decreasing subscoring functions and another for the subset of non-increasing subscoring functions. Recall that $J_{\uparrow}$ and $J_{\downarrow}$ are the sets of indexes of non-decreasing and non-increasing subscoring functions, respectively. It is easy to see that $J_{\uparrow} \in \mathcal{P}$ and $J_{\downarrow} \in \mathcal{P}$. Without loss of generality, assume that $J_x = J_{\uparrow}$ and $J_y = J_{\downarrow}$.

For $J_x \in P$ and the corresponding partial preference tree T_x, we define the critical partial preference hash subtree $\mathsf{T}_x(o_\lambda)$ as the subtree of T_x formed by nodes with partial score no lower than o_{λ_k}, i.e., $\{o_i | 0 \leq i \leq n, \sum_{j \in J_x} f_j(a_{i,j}) \geq \sum_{j \in J_x} f_j(a_{\lambda_k,j})\}$. The expanded partial preference hash subtree $\mathsf{T}^e_x(o_\lambda)$ can be defined accordingly. For every node $\mathsf{v} \in \mathsf{T}^e_x(o_\lambda)$, the partial query result R_v is given by

$$R_\mathsf{v} = \langle \mathsf{v.id}, \mathsf{v.hash}, \mathsf{v.cid}, \mathsf{v.chash} \rangle \ . \tag{4}$$

The partial query result for J_x is then $R_{J_x} = \{R_\mathsf{v} | \mathsf{v} \in \mathsf{T}^e_x(o_\lambda)\}$.

Similarly, for $J_y \in P$ and corresponding partial preference hash tree T^-_y, we define the critical partial preference hash subtree $\mathsf{T}^-_y(o_\lambda)$ as the subtree of T^-_y formed by nodes with partial score no higher than o_{λ_k}, i.e., $\{o_i | 0 \leq i \leq n, \sum_{j \in J_y} f_j(a_{i,j}) \leq \sum_{j \in J_y} f_j(a_{\lambda_k,j})\}$. The expanded critical partial preference hash subtree $\mathsf{T}^{-,e}_x(o_\lambda)$ can be defined accordingly. For every node $\mathsf{v} \in \mathsf{T}^{-,e}_x(o_\lambda)$, the partial query result R_v is given by Eq. (4). The partial query result for J_y is then $R_{J_y} = \{R_\mathsf{v} | \mathsf{v} \in \mathsf{T}^{-,e}_x(o_\lambda)\}$.

The complete query result in response to the top-k selection query consists of the following pieces of information.

- All the involved objects $\{o_\mathsf{v} | \mathsf{v} \in \mathsf{T}^e_x(o_\lambda) \bigcup \mathsf{T}^{-,e}_y(o_\lambda)\}$.
- The IDs of the top k objects $\lambda_1, \ldots, \lambda_k$.
- The two partial query results R_{J_x} and R_{J_y}.
- The data owner's signature on the roots of T_x and T^-_y.

D. Query Result Verification

On receiving the query result, the user first verifies the integrity of all the returned information based on the corresponding partial preference hash trees and the data owner's signatures on the roots as in Scheme 2.

The user then checks if $F(o_{\lambda_1}) \geq \cdots \geq F(o_{\lambda_k})$ and identifies the critical object as o_{λ_k} whereby to verify the soundness of the query result based on Lemma 4. Specifically, for each o_v where $\mathsf{v} \in \mathsf{T}^e_x(o_\lambda)$ and o_v is not among $o_{\lambda_1}, \ldots, o_{\lambda_k}$, the user checks whether it has a partial score higher than $\sum_{j \in J_x} f_j(a_{\lambda_k,j})$. Similarly, for each o_v where $\mathsf{v} \in \mathsf{T}^{-,e}_y(o_\lambda)$ and o_v is not among $o_{\lambda_1}, \ldots, o_{\lambda_k}$, the user checks whether it has a partial score lower than $\sum_{j \in J_y} f_j(a_{\lambda_k,j})$. Finally, the user checks whether all the nodes in $\mathsf{T}^e_x(o_\lambda)$ and $\mathsf{T}^{-,e}_x(o_\lambda)$ have been returned. Specifically, for each leaf node of $\mathsf{T}^e_x(o_\lambda)$, the user checks whether either it has no children or it has a partial score no lower than $\sum_{j \in J_x} f_j(a_{\lambda_k,j})$. For each leaf node of $\mathsf{T}^{-,e}_y(o_\lambda)$, the user checks whether either it has no children or it has a partial score no higher than $\sum_{j \in J_x} f_j(a_{\lambda_k,j})$. If no violation is found, the user considers the query result sound.

TABLE II
DEFAULT SIMULATION SETTINGS

Para.	Value	Description
n	8,000	The number of objects
d	5	The number of attributes
k	250	The number of objects requested
$\lvert H(\cdot)\rvert$	256	The length of hash in bit
	1024	The length of data owner's signature in bit

E. Discussions

While our primary motivation behind the advanced scheme is to support scoring functions comprised of both monotonically non-decreasing and non-increasing functions, the advanced scheme can also be adopted to minimize the communication and computation costs. Given $\{\langle \mathsf{T}_x, \mathsf{T}^-_x \rangle | J_x \in \mathcal{P}\}$, the CSP can construct multiple authentic and sound valid query results in response to a top-k selection query. Suppose that all the subscoring functions are monotonically non-decreasing. We call a subset $P \subset \mathcal{P}$ a *cover* of J^+ if the following two conditions hold.

- **Condition 1**: $\bigcup_{J_x \in P} J_x = J^+$, i.e., the union of the subsets covers J^+.
- **Condition 2**: $J_x \bigcap J_y = \emptyset$ for any $J_x, J_y \in P$, i.e., there is no overlap between any two subsets.

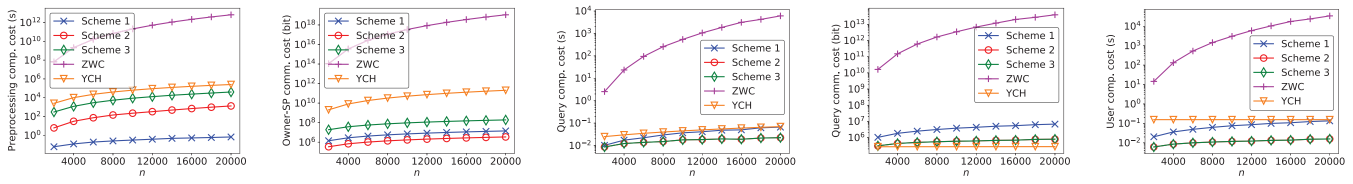
For every valid cover P, the CSP can construct a query result consisting of a partial query result R_x for every $R_x \in P$. Since different covers would have different sets of expanded preference subtrees, they would require different numbers of additional objects to be returned and thus incur different communication and computation costs. Intuitively, the CSP may examine all the covers to find the optimal query result with the minimal number of additional objects. Moreover, the data owner may choose to construct preference subtrees only for some of the subsets of J to reduce the computation and communication cost as long as a valid cover can be found for any J. There is thus a trade-off among the computation costs of the three phases. We leave the investigation of these issues as our future work due to space constraints.

VI. Performance Evaluation

In this section, we evaluate the performance of the proposed schemes using detailed simulation studies.

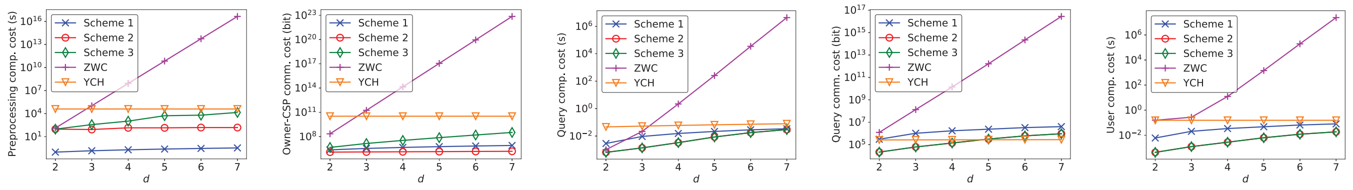
A. Simulation Setting

We implemented the three proposed schemes in Python and tested them on a desktop equipped with an i7-8700 CPU, 16GB RAM, and 64-bit Win10 OS. We choose SHA-256 for the cryptographic hash function and 1024-bit RSA as the digital signature scheme. We used a public dataset [32] for our



(a) Preprocessing comp. cost (b) Owner-CSP comm. cost (c) CSP comp. cost (d) Query comm. cost (e) User comp. cost

Fig. 1. Comparison of Scheme 1, Scheme 2, Scheme 3, ZWC, and YCH with n varying from 2,000 to 20,000.



(a) Preprocessing comp. cost (b) Owner-CSP comm. cost (c) CSP comp. cost (d) Query comm. cost (e) User comp. cost

Fig. 2. Comparison of Scheme 1, Scheme 2, Scheme 3, ZWC, and YCH with d varying from 2 to 7.

simulation, which consists of 22,365 records of NBA players, and each record is comprised of 13 numeric attributes about different performance statistics of the player, from which we randomly select 2 to 7 attributes. We also generates top-k selection queries with random linear scoring functions. Table II summarizes our default settings unless mentioned otherwise.

We compare the three schemes with the two solutions proposed in [11], [12] (refer to as ZWC hereafter) and [15] (referred to as YCH hereafter). Despite their assumptions of scoring functions having explicit forms, they are the only two that can support scoring functions over multiple attributes. Also note that the performance of ZWC [11], [12] is based on our best-effort estimation due to its high computation and communication costs.

We use five metrics in our simulation studies, including (1) *preprocessing computation cost*, which is the amount of time incurred by preprocessing the dataset D at the data owner, (2) *owner-CSP communication cost*, which is the amount of extra information in bits transmitted from the data owner to the CSP, (3) *query computation cost*, which is amount of time incurred by the CSP for processing a top-k selection query, (4) *CSP communication cost*, which is the amount of extra information in bits transmitted from the the CSP to the user in response to a top-k selection query, and (5) *user computation cost*, which is the amount of time needed for verifying a query result.

B. Simulation Results

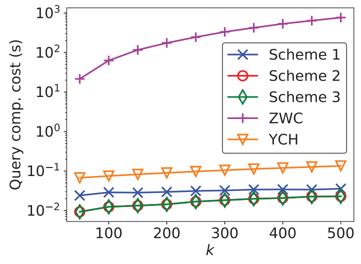
In this subsection, we report out simulation results. We assume that all subscoring functions are monotonically non-decreasing in Sections VI-B1 to VI-B3 to allow comparison among the three proposed schemes and postpone to evaluation of mixed subscoring functions to Section VI-B4.

1) Impact of n: Figs.1(a) to 1(e) compare the performance of Schemes 1 to 3, ZWC, and YCH with the number of objects n varying from 2000 to 20000.

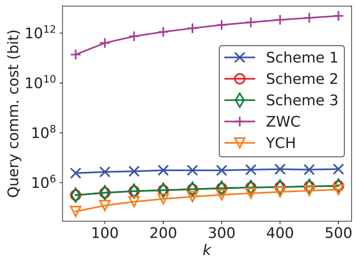
Fig. 1(a) compares the preprocessing computation cost under the five schemes. As we can see, the preprocessing computation cost increases as n increases under all five schemes as expected. Among them, Schemes 1 has the lowest preprocessing computation cost, followed by Schemes 2, Scheme 3, YCH, and ZWC. The reason is that the data owner only needs to sort d lists under Scheme 1, which is very efficient. In contrast, the data owner needs to construct one and $2^{d+1}-2$ preference hash trees under Scheme 2 and Scheme 3, respectively. YCH incurs higher preprocessing computation cost than all three proposed schemes, as the data owner needs to precompute all subspaces where the objects exhibit distinct orders, and the total number of subspace is $O(n^2)$ assuming linear scoring functions. All four schemes incur significantly lower computation cost than ZWC, as ZWC uses digital signature to chain every pair of adjacent grids, and the number of grids is $O(n^d)$.

Fig. 1(b) compares the owner-CSP communication cost of the five schemes. Similar to Fig. 1(a), the owner-CSP communication cost increase as n increases under all five schemes as expected. Among them, Scheme 2 incurs the lowest owner-CSP communication cost, followed by Scheme 1, Scheme 3, and YCH and ZWC is the highest. The reason is that approximately nd, n, and $n2^{d+1}$ hashes need be transmitted from the data owner to the CSP under Schemes 1 to 3, respectively, whereas $O(n^2)$ and $O(n^d)$ digital signatures are needed under YCH and ZWC, respectively.

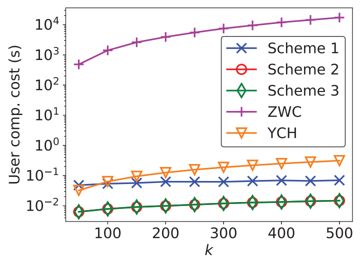
Fig. 1(c) compares the CSP computation cost of the five schemes. Note that Schemes 2 and 3 are equivalent when all subscoring functions are monotonically non-decreasing as the CSP only needs to construct the query result based on one preference hash tree. We can see that the CSP computation cost increases as n increases under all five schemes, which is expected. The CSP computation cost under YCH is initially higher than Schemes 1 to 3 but grows slower as n increases. The reason is that the CSP needs to search a space-partition



(a) CSP computation cost

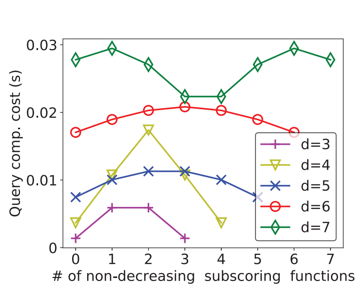


(b) Query communication cost

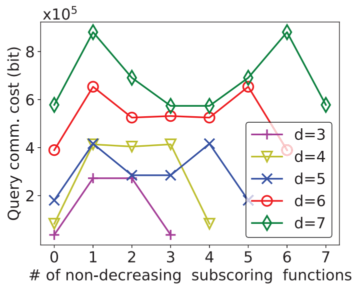


(c) User computation cost

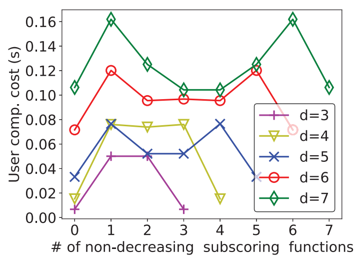
Fig. 3. Comparison of Scheme 1, Scheme 2, Scheme 3, ZWC, and YCH with k varying from 50 to 500.



(a) CSP computation cost



(b) Query communication cost



(c) User computation cost

Fig. 4. The impact of the number of non-decreasing subscoring functions on Scheme 3.

tree of depth $O(\log n)$ under YCH. Moreover, Schemes 2 and 3 incur lower CSP computation cost than Scheme 1, as the CSP only needs to examine one preference subtree under Schemes 2 and 3 but d sorted list under Scheme 1. Both schemes 2 and 3 outperform YCH a large margin and ZWC by orders of magnitude under the default setting.

Figs. 1(d) and 1(e) compare the query communication cost and user computation cost of the five schemes. We can see that the query communication cost and user computation cost both increase as n increases under Schemes 1 to 3 and ZWC, which is expected. YCH incurs the lowest query communication cost as the CSP only needs to return k digital signatures independent of n. Schemes 2 and 3 incur lower CSP computation cost and user computation cost than Scheme 1 as the CSP only needs to return one hash for every node in one expanded preference hash tree under Schemes 2 and 3 and one hash for every node ranked higher than the critical object in each of the d lists under Scheme 1. Moreover, YCH incurs higher user computation cost than Schemes 2 and 3 as it requires k expensive signature verifications. Moreover, all three proposed schemes and YCH outperform ZWC by orders of magnitude.

2) Impact of d: Figs. 2(a) to 2(e) compare the performance of Schemes 1 to 3, ZWC, and YCH with d, i.e., the number of attributes, varying from 2 to 7. As we can see, the preprocessing computation cost, owner-CSP communication cost, CSP computation cost, query communication cost, and user computation cost of increase as the d increases under all five schemes, which is anticipated. In contrast, the costs under YCH is relatively insensitive to the change in d, as they are mainly affected by n. Under the default setting, ZWC incurs significantly higher costs than the other four schemes except when $d = 2$. Among Schemes 1 to 3, Scheme 3 incurs the highest preprocessing computation cost, owner-CSP communication cost, and CSP computation cost, due to the $2^{d+1} - 2$ preference subtrees involved. Scheme 1 incurs the lowest preprocessing computation cost, while Schemes 2 and 3 have the lowest CSP computation cost, query communication cost, and user computation cost.

3) Impact of k: Figs. 3(a) to 3(c) compare the query computation cost, query communication cost, and user computation cost of the five schemes with k varying from 50 to 500. Generally speaking, the larger k, the higher CSP computation cost, query communication cost, and user computation cost for all five schemes, and vice versa, which is expected. ZWC always incurs the highest costs among the five. YCH incurs higher CSP computation cost and user computation cost but lower query communication cost than Schemes 1 to 3, as it requires the CSP to locate the subspace containing the query point and the query result contains k digital signatures. In contrast, the CSP only needs to examine d sorted lists and one preference hash tree under Scheme 1 and Schemes 2 and 3, respectively, resulting in lower CSP computation cost and user computation cost. In addition, all three schemes primarily use cryptographic hash functions for integrity verification, which are more efficient than digital signatures.

4) Impact of The Number of Non-decreasing Subscoring Functions: Figs. 4(a) to 4(c) show the query computation cost, query communication cost, and user computation cost of Scheme 3 with the number of monotonically non-decreasing subscoring functions, i.e., $|J_{\uparrow}|$, varying from 0 to d for $d = 3$ to 7. An interesting phenomenon we can observe is that the the query computation cost, query communication cost, and user computation cost sometimes fluctuate as $|J_{\uparrow}|$ increases from 0 to d, which is particularly obvious when $d = 7$. The reason is that the number of additional objects that need be returned for query-result verification under Scheme 3 depends on the two expanded partial preference hash trees. Specifically, an object needs be returned if it belongs to either $\mathsf{T}_x^e(o_\lambda)$ or $\mathsf{T}_y^{-,e}(o_\lambda)$. However, the number of nodes in an (inverse) expanded partial preference hash tree is affected by the number of children nodes each node has on average. When $|J_{\uparrow}|$ is small, it is more likely for one object to be preferable to another, which leads to a deeper preference hash tree, fewer children nodes each node has on average, and a smaller expanded partial preference tree. As $|J_{\uparrow}|$ increases, there will be fewer preference relationship among objects, which leads to a wider preference hash tree and more children nodes each node has on average. It is thus not surprising to see that the query computation cost, query communication cost, and user computation cost are the lowest when $|J_{\uparrow}| \approx d/2$, i.e., $\mathsf{T}_x^e(o_\lambda)$ are $\mathsf{T}_y^{-,e}(o_\lambda)$ have similar sizes.

VII. Conclusion

In this paper, we have introduced three novel schemes for authenticating outsourced top-k selection queries against an untrusted CSP. Exploring a partial preference relationship among objects, the three schemes allow the user to verify both the integrity and soundness of any top-k selection query result returned by the CSP. Detailed simulation studies using a real dataset confirm the efficacy of efficiency of the proposed schemes and their significant advantages over prior solutions.

Acknowledgement

The authors would like to thank the anonymous reviewers for their constructive comments and helpful advice. This work was partially supported by National Natural Science Foundation of China under Grant 62072351, Academy of Finland under Grant 308087 and Grant 335262, US National Science Foundation through grants CNS-1933069, CNS-1824355, CNS-1651954 (CAREER), CNS-1718078 and CNS-1933047.

References

[1] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, Oct. 2008.
[2] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *ACM Trans. Storage*, vol. 2, no. 2, p. 107–138, May 2006.
[3] R. Zhang, J. Shi, Y. Liu, and Y. Zhang, "Verifiable fine-grained top-k queries in tiered sensor networks," in *IEEE INFOCOM'10*, San Diego, CA, USA, March 2010, pp. 1–9.
[4] R. Zhang, Y. Zhang, and C. Zhang, "Secure top-k query processing via untrusted location-based service providers," in *IEEE INFOCOM*, Orlando, FL, USA, March 2012, pp. 1170–1178.
[5] Chia-Mu Yu, Guo-Kai Ni, Ing-Yi Chen, E. Gelenbe, and Sy-Yen Kuo, "Top-k query result completeness verification in sensor networks," in *IEEE ICC'13*, Budapest, Hungary, June 2013, pp. 1026–1030.
[6] C. Yu, G. Ni, I. Chen, E. Gelenbe, and S. Kuo, "Top-k query result completeness verification in tiered sensor networks," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 109–124, 2014.
[7] D. Hua, Y. Geng, X. Fu, and Z. Qiang, "Evtq: An efficient verifiable top-k query processing in two-tiered wireless sensor networks," in *IEEE MSN'13*, Dalian, China, Dec. 2013, pp. 206–211.
[8] R. Zhang, J. Shi, Y. Zhang, and X. Huang, "Secure top-k query processing in unattended tiered sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 9, pp. 4681–4693, 2014.
[9] R. Zhang, J. Sun, Y. Zhang, and C. Zhang, "Secure spatial top-k query processing via untrusted location-based service providers," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 1, pp. 111–124, Jan. 2015.
[10] R. Li, A. X. Liu, S. Xiao, H. Xu, B. Bruhadeshwar, and A. L. Wang, "Privacy and integrity preserving top-k query processing for two-tiered sensor networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2334–2346, 2017.
[11] X. Zhu, J. Wu, W. Chang, G. Wang, and Q. Liu, "Authentication of multi-dimensional top-k query on untrusted server," in *IEEE/ACM IWQoS*, 2018, pp. 1–6.
[12] X. Zhu, J. Wu, W. Chang, G. Wang, and Q. Liu, "Efficient authentication of multi-dimensional top-k queries," *IEEE Access*, vol. 7, pp. 4748–4762, 2019.
[13] X. Ma, J. Liang, S. Yang, Y. Li, Y. Li, W. Ma, and T. Wang, "Sls-stq: A novel scheme for securing spatial–temporal top-k queries in twsns-based edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 093–10 104, 2019.
[14] Y.-T. Tsou, Y.-L. Hu, Y. Huang, and S.-Y. Kuo, "SFTopk: Secure functional top-k query via untrusted data storage," *IEEE Access*, vol. 3, pp. 2875–2890, 2015.
[15] G. Yang, Y. Cai, and Z. Hu, "Authentication of function queries," in *IEEE ICDE'16*, Helsinki, Finland, May 2016, pp. 337–348.
[16] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," *The VLDB Journal*, vol. 18, no. 3, pp. 631–648, Jun 2009.
[17] D. Yung, E. Lo, and M. L. Yiu, "Authentication of moving range queries," in *CIKM'12*, Maui, HI, 2012, pp. 1372–1381.
[18] Q. Chen, H. Hu, and J. Xu, "Authenticating top-k queries in location-based services with confidentiality," *Proc. VLDB Endow.*, vol. 7, no. 1, pp. 49–60, 2013.
[19] W. Chen, M. Liu, R. Zhang, Y. Zhang, and S. Liu, "Secure outsourced skyline query processing via untrusted cloud service providers," in *IEEE INFOCOM'16*, San Francisco, CA, USA, April 2016, pp. 1–9.
[20] M. L. Yiu, E. Lo, and D. Yung, "Authentication of moving knn queries," in *IEEE ICDE'11*, Hannover, April 2011, pp. 565–576.
[21] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi, "Spatial query integrity with voronoi neighbors," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 4, pp. 863–876, April 2013.
[22] L. Hu, Y. Jing, W.-S. Ku, and C. Shahabi, "Enforcing k nearest neighbor query integrity on road networks," in *ACM SIGSPATIAL'12*, Redondo Beach, CA, Nov. 2012, pp. 422–425.
[23] M. L. Yiu, Y. Lin, and K. Mouratidis, "Efficient verification of shortest path search via authenticated hints," in *IEEE ICDE'10*, Long Beach, CA, March 2010, pp. 237–248.
[24] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan, "Verifying completeness of relational query results in data publishing," in *ACM SIGMOD'05*, 2005, p. 407–418.
[25] Y. Zhang, J. Katz, and J. Katzs, "Integridb: Verifiable SQL for outsourced databases," in *ACM CCS'15*, Denver, CO, 2015, pp. 1480–1491.
[26] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases," in *IEEE Symposium on Security and Privacy*, San Jose, CA, May 2017, pp. 863–880.
[27] Y. Tang, T. Wang, L. Liu, X. Hu, and J. Jang, "Lightweight authentication of freshness in outsourced key-value stores," in *ACM ACSAC'14*, New Orleans, Louisiana, USA, 2014, p. 176–185.
[28] Y. Hu, R. Zhang, and Y. Zhang, "KV-Fresh: Freshness authentication for outsourced multi-version key-value stores," in *IEEE INFOCOM'20*, Toronto, ON, Canada, July 2020, pp. 1638–1647.
[29] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios, "Proof-infused streams: Enabling authentication of sliding window queries on streams," in *VLDB'07*, Vienna, Austria, Sept. 2007, pp. 147–158.
[30] S. Nath and R. Venkatesan, "Publicly verifiable grouped aggregation queries on outsourced data streams," in *IEEE ICDE'13*, Brisbane, Australia, April 2013, pp. 517–528.
[31] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *ACM PODS'01*, Santa Barbara, CA, 2001, p. 102–113.
[32] "Available at," https://slamdunk.sports.sina.com.cn.