AuthPDB: Query Authentication for Outsourced Probabilistic Databases

Bo Zhang  
Department of Computer Science  
Stevens Institute of Technology  
Hoboken, NJ, 07030  
bzhang41@stevens.edu

Boxiang Dong  
Department of Computer Science  
Montclair State University  
Montclair, NJ, 07043  
dongb@montclair.edu

Wendy Hui Wang  
Department of Computer Science  
Stevens Institute of Technology  
Hoboken, NJ, 07030  
hwang4@stevens.edu

Abstract—Spurred by developments such as cloud computing, there are increasing efforts for outsourcing of data management. A company (data owner) who lacks expertise and computational resources can outsource his data to a third-party service provider (server) for query evaluation. One of the security concerns of the outsourcing paradigm is the integrity of the returned query results on the outsourced data. In this paper, we consider the outsourcing of probabilistic databases, on which query evaluation results on the outsourced data. In this paper, we consider the outsourcing of probabilistic databases, on which query evaluation is of high complexity. A dishonest server may return cheap (and incorrect) query answers, hoping that the client who has weak computational power cannot catch the incorrect results. To address this issue, we design efficient integrity verification methods for both all-answer and top-k query evaluation on outsourced probabilistic databases. Our empirical results demonstrate the effectiveness and efficiency of our verification methods.

I. INTRODUCTION

Recently there has been an increasing demand for managing incomplete and uncertain data that emerges from scientific data management, sensor data management, data cleaning [1], [2], and information extraction [3]. Probabilistic database systems have been considered as a successful tool for managing uncertain data. A large number of systems (e.g., [4], [5], [6]) have been developed for performing efficient query processing on probabilistic databases. One major challenge of query evaluation on probabilistic databases is its high complexity; the evaluation of some certain types of queries is of #P-complete complexity [5]. The high complexity of query evaluation and complex semantics behind the probabilistic databases hinder common users to establish the probabilistic database management for their own business and research use.

A cost-effective solution is to outsource the probabilistic database to a third-party service provider (e.g., the Cloud). In the outsourcing paradigm, the data owner who has large volume of data but limited knowledge and resources for data analysis can send his data to the service provider. There exists quite a few outsourcing services that support the applications based on probabilistic databases, for example, Nogamy [7] for data integration, Flatworld [8] for data cleaning, and Informatics [9] for scientific data management.

In this paper, we consider two types of queries on probabilistic databases, namely all-answer queries that return all possible answers from the probabilistic database, and top-k queries that return k possible answers of the largest probability. Example 1.1 gives the examples of both types of queries.

Example 1.1: Consider a probabilistic data instance D in Table I (a), in which each record corresponds to an estate property listed for sale, collected from the Internet. Each record is associated with a tuple probability that describes its reliability. Consider a query $Q: \sigma_{\text{Price} \geq 20k}(D)$ (i.e., return all property records whose price is greater than 20k. Table I (d) shows the all-answer results of $Q$, which include four possible answers $\{t_2\}, \{t_3\}, \{t_2, t_3\}$, and $\emptyset$. Each possible answer is associated with an answer probability, calculated from the tuple probability. How to compute the answer probability will be in Section II. The top-3 answers are $\{(t_2, t_3), 0.48\}, \{(t_2), 0.32\}, \{(t_2), 0.12\}$ (i.e., the answers of top-3 probability).

The complexity of all-answer and top-k query evaluation on probabilistic databases is exponential to the size of the dataset. An untrusted service provider (server) is incentivized to improve its revenue, e.g. by computing with less resource while charging for more, especially when it believes that the client cannot easily re-compute the results given his limited computational resources. Therefore, the server can forge the probability of possible answers with a random value. It also can cheat on the top-k results by randomly choosing k possible answers. Furthermore, a malicious server may return wrong query answers intentionally (e.g., return $\{t_3\}$ as the top-1 query result instead of $\{t_2, t_3\}$), if the server is hired to promote the estate properties in Fonda area and demote those in Cave Junction area.

The problem of authenticating query evaluation over outsourced deterministic databases have been well investigated in the literature (e.g., [10], [11], [12]). These existing solutions can verify whether the records (named the hit set) in the returned possible answers satisfy the selection constraint of the query (e.g., $\text{Price} \geq 20k$ in Example 1.1). However, they cannot verify whether the probability of each answer (e.g., the probability 0.32 of the answer $\{t_2\}$) is correct. Apparently, re-computing the probability of each possible answer is prohibitively expensive, as the number of possible answers is exponential to the size of the hit set. For example, the hit set of the query $Q$ contains two records $\{t_2, t_3\}$. There are $2^2 = 4$ possible answers in the all-answer results of $Q$. In
In this paper, we design AuthPDB, a framework that supports efficient result integrity verification of all-answer and top-k query evaluation on outsourced probabilistic databases. Figure 1 illustrates the framework of AuthPDB in a nutshell. Before outsourcing, the data owner constructs an authenticated data structure (ADS) $T$ of his probabilistic dataset $D$. He transmits both $D$ and $T$ to the server, and sends the root hash $h_{root}$ of $T$ to the legitimate clients. For a given query, the server evaluates the query on $D$ and obtains the answers $R$. To serve the purpose of integrity verification, the server constructs a proof of $R$, which takes the form of a verification object (VO). The server returns both $R$ and VO to the client. The client verifies if $R$ is correct based on VO. In particular, our contributions include the following.

First, we design a new ADS named aggregated probability B-tree (APB-tree). APB-tree enables to verify the probability of the query answers directly on the tree by integrating the tuple probability information in the outsourced database with the state-of-the-art Merkle hash tree [13].

Second, we design an authentication method that verifies the authenticity, soundness and completeness of all-answer query results. The verification method enables the server to construct a VO of the returned answers from the APB-tree. To avoid re-computing the probability of each possible answer, the client partitions the answers into several groups. The probability of possible answers in the same group satisfies a certain group property. Based on grouping, the client re-calculates the probability of only one possible answer per group. The probability of the other possible answers in the same group is verified based on the group property, whose cost is much cheaper than probability re-computation.

Third, we design an efficient verification method to authenticate the top-k answers (i.e., if the returned k possible answers are indeed associated with the top-k probability). Instead of re-computing the probability of all possible answers (as the naive method), our authentication method minimizes the re-computation of the probability that is not top-k. In particular, our method only needs to compute at most $\lceil k \ln k \rceil + k$ probabilities to find the top-k probability. In other words, our method only has to compute at most $\lceil k \ln k \rceil$ probabilities besides the top-k ones.

Fourth, we formally prove the security of our verification methods for both all-answer and top-k queries with the presence of the malicious adversary who has full knowledge of the verification methods.

Last but not least, we perform an extensive set of experiments on both real-world and synthetic datasets to evaluate the performance of our verification approaches. Our experimental results show that our verification approaches are efficient.

The paper is organized as following. Section II explains the preliminaries. Section III discusses our ADS structure. Section IV presents our verification methods for all-answers and top-k answers. Section V and VI present the security and complexity analysis respectively. Section VII discusses our experimental results. Section VIII presents the related work. Section IX concludes the paper.

II. PRELIMINARIES

A. Probabilistic Databases

In this paper, we consider the tuple uncertainty probabilistic database model [14], [5] that is widely used for probabilistic database management. In this model, each probabilistic relational instance $D$ consists of a set of basic attributes that describe the data. Each tuple $t_i$ is associated with a probability $p_i \in (0, 1]$ that denotes the existence probability of $t_i$. We call the pair $(t_i, p_i)$ the tuple-probability pair (TP-pair). We follow the same tuple-independence assumption as [5], [14] that the
existence of a tuple is independent of the existence of the other tuples in the database. Table I (a) shows an example of the probabilistic database.

We consider the possible world semantics [15] that is widely used for probabilistic databases. It models the probabilistic database as a probability distribution over all deterministic versions of the database [16, 17]. Following this model, for any probabilistic database \( D \), its possible worlds \( \mathcal{PWD}(D) \) is defined as a set of possible database instances of \( D \), each instance \( W \) associated with a probability \( Pr(W) \). Formally,

\[
\mathcal{PWD}(D) = \{(W, Pr(W)|W \subseteq D)\},
\]

(1)

where \( Pr(W) = \prod_{t \in W} p_t \prod_{t \notin W}(1 - p_t) \).

In general, given a probabilistic database that consists of \( n \) tuples, there are \( 2^n \) possible worlds. For example, the probabilistic database in Table I (a) has eight possible worlds. Table I (b) and (c) display two of these possible worlds.

In this paper, we consider two types of query evaluation over the probabilistic database, namely the all-answer and top-\( k \) queries. For the sake of simplicity, we only consider single-dimensional selection queries, denoted as \( \sigma_{A \in [l, u]}(D) \), where \( A \) is an indexing attribute, and \([l, u]\) is the selection range. Our approach can be extended to multi-dimensional selection queries. Next, we formally define the two types of queries and present their evaluation process.

**All-answer queries.** Given a probabilistic database instance \( D \) and its possible worlds \( \mathcal{PWD}(D) \), for a given query \( Q \) on \( D \), its answer \( Q(D) \) is defined as a set of possible answers, each associated with a probability. The result is obtained by applying \( Q \) to each deterministic instance in \( \mathcal{PWD}(D) \). The probability of each unique answer is calculated as the sum of all instances that return the same answer. Formally,

\[
\mathbb{R} = \{(A, P)|P = \sum_{W \in \mathcal{PWD}(D), Q(W) = A} Pr(W)\}. \quad \text{(2)}
\]

We define the hit set \( \mathcal{H} \) as the set of unique tuples in \( \mathbb{R} \) (i.e., the unique tuples in \( D \) that satisfy the selection range of \( Q \)). We call each pair \((A, P) \in \mathbb{R}\) an answer-probability pair (AP-pair). A more efficient way to compute the probability \( P \) of each unique answer \( A \) is following (we use \( \setminus \) to denote set difference):

\[
P = Pr(A) = \prod_{t_i \in A} p_i \prod_{t_i \notin \mathcal{H}\setminus A}(1 - p_i). \quad \text{(3)}
\]

**Example 2.1:** Continue with the probabilistic database \( D \) in Table I (a) and the query \( Q : \sigma_{price \geq 200k}(D) \) in Example 1.1. The hit set of \( Q \) is \( \mathcal{H} = \{t_2, t_3\} \). There are four possible answers \( \{t_2\}, \{t_3\}, \{t_2, t_3\}, \) and \( \emptyset \). Table I (d) shows all four possible answers of \( Q \) and their probability. Consider the possible answer \( \{t_2\} \), whose probability is computed as \( p_2(1 - p_3) = 0.32 \).

**Top-\( k \) queries.** For those queries of large hit set, enumerating all possible answers is prohibitively expensive. Therefore, instead of all answers, a top-\( k \) query \( Q \) on a probabilistic database instance \( D \), denoted as \( Q^k(D) \), returns the \( k \) possible answers of the highest probability. Continue with the query in Example 2.1 The top-3 answers include \( Q^3(D) = \{\{t_2, t_3\}, 0.48\}, \{\{t_2\}, 0.32\}, \{\{t_3\}, 0.12\} \).

**B. Authenticated Data Structure (ADS)**

One of the widely-used authenticated data structures (ADS) is Merkle hash tree (MHT) [13]. A MHT \( T \) is a tree in which each leaf node \( N \) stores the digest of a tuple \( t: h_N = H(t) \), where \( H() \) is a one-way, collision-resistant hash function (e.g. SHA-1 [18]). For each internal node \( N \) of \( T \), it is assigned the value \( h_N = H(h_{N_1}|| \ldots ||h_{N_f}) \), where \( N_1, \ldots, N_f \) are the children of \( N \), and \( || \) is the concatenation operator. The hash value \( h_{root} \) of the root node is used as the digest of the tree.

Before outsourcing, the data owner constructs a MHT of the relation \( D \), and keeps \( h_{root} \) locally. Then he sends the MHT to the server, and \( h_{root} \) to the client. When the server returns the query result to the client, it searches through the MHT, and constructs a verification object (VO) of the query results by including the information in MHT regarding the query results. The client can verify the query results by recomputing the root hash \( h'_{root} \) from VO and the query results. The query results are considered as correct if \( h'_{root} = h_{root} \).

**C. Condensed RSA**

RSA [19] is a classic public key encryption scheme. The scheme generates two \( \lambda/2 \)-bits random prime numbers \( p \) and \( q \), where \( \lambda \) is the security parameter. It computes \( B = pq \), and finds a pair of integers \((e, d)\) such that \( ed \equiv 1 \mod \phi(B) \), where \( \phi(B) = (p - 1)(q - 1) \). The public key \( pk = (B, e) \) is released to the public, while the private key \( sk = d \) is kept secret. Given an message \( m \), its signature \( \sigma \) is generated as:

\[
\sigma = H(m)^d \mod B, \quad \text{(4)}
\]

where \( H() \) is a full-domain cryptographic hash function that converts a message into a value in \( \mathbb{Z}_B \). To enable authentication of a sequence of messages, a signature aggregation scheme named Condensed-RSA [20] compresses a set of RSA signatures into a single signature. To prove the authenticity of \( m_1, \ldots, m_n \), the prover computes an single aggregate signature \( \sigma_{1,n} = \prod_{i=1}^{n} \sigma_i (mod B) \). Only \( \sigma_{1,n} \) is sent to the verifier, who verifies the authenticity of \( m_1, \ldots, m_n \) by checking if \( (\sigma_{1,n})^e \equiv \prod_{i=1}^{n} H(m_i) (mod B) \).

**D. Authentication Goal**

**All-answer queries.** Given a probabilistic database instance \( D \) and an all-answer query \( Q \), let \( R \) be the answer of \( Q \) that is returned by the server. The client verifies authenticity, soundness, and completeness of \( R \).

- **Authenticity:** for each AP-pair \((A_i, P_i) \in R\), it verifies if all tuples in \( A_i \) exist in \( D \) and are not tampered with.
- **Soundness:** for each AP-pair \((A_i, P_i) \in R\), it verifies two types of soundness: (1) result soundness (R-soundness) verifies if \( A_i \) satisfies the selection condition of \( Q \); and (2) probability soundness (P-soundness) verifies if \( P_i \) is correct (according to Equation (3)).
• Completeness: it verifies if \( \mathbb{R} \) includes all the AP-pairs that satisfy \( Q \).

**Top-k queries.** Given a top-k query \( Q^k \), the client verifies authenticity, soundness, and completeness of the top-k results.

- **Authenticity:** the same as that of the all-answer queries.
- **Soundness:** For each AP-pair in \( \mathbb{R} \), it verifies three types of soundness, result soundness (R-soundness), probability soundness (P-soundness) and top-k soundness (TopK-soundness). R-soundness and P-soundness are the same as that of the all-answer queries. Top-k soundness verifies if the AP-pairs of \( \mathbb{R} \) are indeed of the \( k \) largest probability. It is worth noting that Top-k soundness naturally implies completeness. Thus, we do not deliberate over the completeness verification of top-k queries.

The problem of verifying authenticity, soundness, and completeness of selection queries over deterministic databases have been well investigated in the literature (e.g., \[10\], \[11\], \[12\]). However, these works only can authenticate the hit set. They cannot verify P-soundness and TopK-soundness. The challenge to address is to enable the client who has limited computational resources (e.g., on mobile devices) to verify the query results that are potentially large, as the number of possible answers is exponential to the hit set size.

### E. Authentication Protocol and Security Model

In this section, we first define the authentication protocol. Then we define the security of the authentication protocol. We adapt the definition of authentication protocols in \[21\] to our setting. Formally,

**Definition 2.1 (Authentication protocol):** Let \( D \) be any probabilistic database. Let \( Q \) be an (all-answer/top-k) query on \( D \), \( \text{auth}(D) \) be the authenticated data structure (ADS) constructed from \( D \), and \( \Pi \) be the proof of the query result \( \mathbb{R} \). The authentication protocol is a collection of the following four polynomial-time algorithms:

- \( \{s_k, p_k\} \leftarrow \text{genkey}(\lambda) \): Given the security parameter \( \lambda \), it outputs a secret key \( s_k \) and a public key \( p_k \);
- \( \{\text{auth}(D), \delta\} \leftarrow \text{setup}(D, s_k, p_k) \): On input of a probabilistic database \( D \), the secret key \( s_k \), and the public key \( p_k \), it computes the authenticated data structure \( \text{auth}(D) \) and its digest \( \delta \);
- \( \{\Pi, \mathbb{R}\} \leftarrow \text{certify}(Q, \text{auth}(D), p_k) \): Given a query \( Q \), the authenticated data structure \( \text{auth}(D) \) and the public key \( p_k \), it returns the result \( \mathbb{R} \), along with its proof \( \Pi \);
- \( \{\text{accept}, \text{reject}\} \leftarrow \text{verify}(Q, \mathbb{R}, \delta, \Pi, p_k) \): Given a query \( Q \), the result \( \mathbb{R} \), the digest \( \delta \) of the ADS \( \text{auth}(D) \), the proof \( \Pi \), and the public key \( p_k \), it outputs either accept or reject.

The \text{genkey} protocol is straightforward. Given the security parameter \( \lambda \), the data owner picks a collision-resistant hash function \( H \) whose output length is \( \lambda \)-bits. Then the data owner generates the keys of RSA signatures (i.e., \( B, e \), and \( d \)) with respect to \( \lambda \) (Section \[13\]). The \text{genkey} protocol outputs a pair of secret and public keys, where \( s_k = d \), and \( p_k = \{H, B, e\} \). In the following sections, we mainly focus on the design of \text{setup}, \text{certify}, and \text{verify} algorithms.

In this paper, we consider the malicious adversary who has full knowledge of the authentication protocol. Next, we define the security of the authentication protocol against such malicious adversary.

**Definition 2.2 (Security):** Let \( \text{Auth} \) be an authentication scheme \( \{\text{genkey}, \text{setup}, \text{certify}, \text{verify}\} \), \( \lambda \) be the security parameter, \( e(\lambda) \) be a negligible function, and \( \{s_k, p_k\} \leftarrow \text{genkey}(\lambda) \). Let also \( \text{Adv} \) be a probabilistic polynomial-time adversary that is only given \( p_k \). The adversary has unlimited access to all algorithms of \( \text{Auth} \), except for the algorithm \text{setup} to which he has only oracle access. Then, for the query \( Q \), \( \text{Adv} \) returns a wrong result \( \mathbb{R} \neq Q(D) \), and a proof \( \Pi \). The authentication scheme \( \text{Auth} \) is secure if for all \( \lambda \in \mathbb{N} \), for all \( \{s_k, p_k\} \) pairs generated by the \text{genkey} scheme, and for any probabilistic polynomial-time adversary \( \text{Adv} \), it holds that

\[
\Pr(\{\mathbb{R}, \Pi\} \leftarrow \text{Adv}(\lambda, Q, p_k); \text{accept} \leftarrow \text{verify}(\mathbb{R}, \delta, \Pi, p_k)) \leq e(\lambda).
\]

Intuitively, the authentication protocol is secure if the probability that the wrong query results can be accepted is negligible.

### III. Aggregated Probability B-tree (APB-tree)

To facilitate efficient query authentication on probabilistic databases, we design a new authenticated data structure named aggregated probability B-tree (APB-tree). In this section, we first describe the APB-tree structure. Then we present the \text{setup} protocol that constructs the APB-tree.

**Fig. 2. An example of APB-tree**

APB-tree is built on top of the Merkle hash tree (MHT) \[13\]. It stores an aggregated probability in each tree node. In particular, given a probabilistic database instance \( D \), as well as the secret key \( s_k \) and the public key \( p_k \), its TP-pairs are sorted by the canonical order of their values on the indexing attribute \( A \). Each TP-pair \( (t_i, p_i) \) corresponds to a leaf node in the APB-tree, whose value takes the format of \( (t_i, p_i, \sigma_i, h_i) \), where \( \sigma_i \) is the RSA signature of \( t_i | p_i \) by using \( s_k \) (Equation \( 4 \)), \( h_i = H(t_i | p_i | \sigma_i) \), and \( H() \) is a full-domain cryptographic hash function and is part of \( p_k \). For any internal node \( N \) of the APB-tree, let \( \text{cov}(N) \) be the set of tuples enclosed in the leaf nodes of the subtree rooted at \( N \), and \( C = \{N_1, \ldots, N_f\} \) be the children nodes of \( N \), where \( f \) is the fanout of \( N \). Apparently, \( \text{cov}(N) = \bigcup_{f=1}^{f} \text{cov}(N_i) \). The internal node \( N \) takes the format \( (t_{\text{min}}, t_{\text{max}}, p, \sigma, h) \), where: \( t_{\text{min}} = \min(\text{cov}(N)) \) and \( t_{\text{max}} = \max(\text{cov}(N)) \) are the minimum and maximum values of tuples in \( \text{cov}(N) \)
respectively; the probability \( p = \prod_{1 \leq i \leq f} N_{i,p} \) (i.e. the multiplication of the probability of the leaf tuples in the subtree rooted at \( N \)); the signature \( \sigma = \prod_{1 \leq i \leq f} N_{i,\sigma} \) (mod \( B \)) and the hash value \( h = H(t_{\min}[t_{\max}[p][\sigma][h^{\rightarrow f}]) \), where \( h^{\rightarrow f} = H(h_1[\ldots][h_y]) \). Figure 2 shows an example of the APB-tree. It is worth noting that the APB-tree only supports one-dimensional query authentication. It can be extended to facilitate multi-dimensional query selections by switching the underlying indexing structure to R-tree [22]. Before outsourcing the original database instance \( D \) to the server, the data owner constructs the APB-tree \( T \) from \( D \), and sends both \( D \) and \( T \) to the server. The data owner only sends the root hash \( h_{root} \) to legitimate clients.

### IV. AUTHENTICATION METHODS

We observe that authenticity and R-soundness of both all-answers and top-k-answers can be verified by the same process that verifies the soundness and completeness of the hit set. Therefore, we design a verification method that consists of two steps: (1) verification of hit set; and (2) verification of all-answers/topk-answers based on the authenticated hit set. Note that the hit set can be generated from the AP-pairs in the query results; the server does not need to return it separately from the query results. Next, we explain the details of these two steps. The verification of hit set is discussed in Section IV.A. The verification of all-answers and top-k answers is explained in two separate subsections (Sec IV.B and IV.C) respectively.

#### A. Authentication of Hit Set

There are quite a few existing solutions to verify authenticity, soundness and completeness of the selection query execution for outsourced deterministic databases (e.g., [23], [24]). The common idea of these solutions is the following: the server traverses the ADS and visits the essential nodes to construct the VO. The VO is sent back to the client together with the query results. From the query results and the VO, the client re-constructs the traversal path used in query execution and verifies that it is indeed authentic. We adapt the same idea to this paper to verify the correctness (i.e., authenticity, soundness, and completeness) of the hit set.

**Certify Protocol:** We first introduce a number of definitions before the discussion of Certify protocol.

**Definition 4.1:** Given an APB-tree \( T \) and a selection query \( Q \) of the range \([l, u]\) on \( N \), we say an internal node \( N(t_{\min}, t_{\max}, p, \sigma, h) \) of \( T \) is a maximum false hit node (MF-node) if both of following conditions are satisfied: **Condition (1):** \( t_{\min} > u \) or \( t_{\max} < l \) (i.e., the tuples of \( N \) are false hits of \( Q \)); and **Condition (2):** the parent of \( N \) does not satisfy Condition (1) (i.e., \( N \) is maximum).

Given a selection query \( Q \), we categorize a tuple \( t \) into one of the three types:

- **M-tuple** (in short for matching tuple): if \( t \) is located in the query range;
- **NC-tuple** (in short for non-candidate tuple): if \( t \) is the descendant of an MF-node;
- **C-tuple** (in short for candidate tuple): if \( t \) does not satisfy the selection condition, and is not covered by any MF-node.

Obviously, the hit set \( H \) is the set of \( M \)-tuples.

**Definition 4.2:** Given a probabilistic data instance \( D \), its APB-tree \( T \), and a query \( Q \), let \( N \) be a set of nodes of \( T \). We say \( N \) is the minimum coverage set (MCS) of \( Q \) if (1) \( \bigcup_{N \in \mathcal{N}} cov(N) = H \), where \( H \) is the hit set of \( Q \) (i.e., \( N \) covers all hit tuples); (2) for any pair of nodes \( N_i, N_j \in \mathcal{N} \) (i \( \neq j \)), \( cov(N_i) \cap cov(N_j) = \emptyset \) (i.e., \( N_i \) and \( N_j \) cover non-overlapping tuples); and (3) \( \mathcal{N} \) contains the minimum number of nodes.

In the rest of our paper, we use \( M(Q) \) to denote the MCS of a query \( Q \). Take Figure 2 as an example. Consider a query \( Q \) whose hit set \( H = \{t_3, t_4, t_5, t_6\} \). The MCS of the query \( Q \) is \( M(Q) = \{N_3, N_46\} \). Now we are ready to describe how to construct the VO.

**Definition 4.3:** Given a probabilistic data instance \( D \) and its APB-tree \( T \), consider a query \( Q \) on \( T \), let \( C \) and \( \mathcal{M}F \) be the set of nodes of \( C \)-tuples and MF-nodes respectively. The VO of \( Q \) includes the following information:

- For each leaf node \( N \) in \( M(Q) \cup C \), the pair \((t, p, \sigma)\) is stored in VO, where \( t \), \( p \), and \( \sigma \) are the tuple, probability, and the RSA signature stored in \( N \);
- For each internal node \( N \) of \( M(Q) \cup \mathcal{M}F \), it is stored in VO in the format of \((t_{\min}, t_{\max}, p, \sigma, h^{\rightarrow f})\);
- For each M-tuple \( t_i \in H \), the TP-pair \((t_i, p_i)\) is stored in VO.

A pair of brackets is injected before and after the objects that locate in the same tree node to denote the structure information.

**Verify Protocol:** Given a query \( Q \) and the returned results \( \mathcal{R} \), the protocol is to verify the correctness of hit set \( H \). This is achieved by re-constructing the root hash from VO and comparing it against the root digest received from the data owner. In particular, for each leaf node \( N \) in \( VO \), the client computes \( h_N = H(t_{i_1}[p_{i_1}][\sigma_{i_1}]) \), where \( t_{i_1}, p_{i_1} \), and \( \sigma_{i_1} \) are obtained from the VO directly. For each internal node \( N \) in \( VO \) that are not included in the VO, the client can easily calculate \( h_N = H(t_{\min}[t_{\max}[p][\sigma][h^{\rightarrow f}]) \). For those internal nodes that are not included in the VO, the client recovers \( h_N \) from their children nodes. The client repeats this process until it obtains the root hash value, \( h'_{root} \). The client compares \( h'_{root} \) with the local copy to verify the correctness of hit set \( H \).
\{t_3, t_4, t_5, t_6\}\). Apparently \(\mathcal{MF} = \{N_{79}\}\), \(\mathcal{C} = \{N_1, N_2\}\), and \(\mathcal{M}(Q) = \{N_3, N_{46}\}\). The VO is constructed as following:

\[
V_O = (((\{(t_1, p_1, \sigma_1), (t_2, p_2, \sigma_2), (t_3, p_3, \sigma_3)\}, (t_4, t_6, p_{46}, \sigma_{46}, \sigma_{46}^{-1})), (\tau, 3, p_{46}, \sigma_{46}, h^{-1})), ((t_3, p_{46}), (t_4, p_{46}), (t_5, p_{46}), (t_6, p_{46})))
\]

where \(h^{-1} = H(h_4||h_5||h_6)\), and \(h^7 = H(h_7||h_8||h_9)\).

In the verification process, the client first re-calculates the root hash value of the APB-tree. Next, the VO, the client constructs \(\mathcal{M}(Q) = \{N_3, N_{46}\}\) and checks if \(\sigma_3 \ast \sigma_{46} = H(t_{3||p_{36}}) \ast H(t_{4||p_{46}}) \ast H(t_{5||p_{56}}) \ast H(t_{6||p_{66}}) (\mod B)\). This verifies the authenticity, soundness, and completeness of \(\mathcal{H}\).

### B. Authentication of All-answer Queries

Given a query \(Q\) and its all-answer result \(R\) returned by the server, a naive verification method is that the client re-calculates the probability of all AP-pairs in \(R\). Apparently the naive method is prohibitively expensive, since the number of AP-pairs is exponential to the size of hit set of \(Q\). Therefore, we design an efficient verify protocol that does not need to calculate the probability of all AP-pairs for the verification of P-soundness. Instead, the verify protocol partitions the AP-pairs into several groups. For all the AP-pairs in the same group, the verify protocol only re-calculates the probability of one single AP-pair, while the probability of the remaining AP-pairs is verified by checking if the AP-pairs in the same group have a certain property. This enables efficient verification of P-soundness. Apparently, for each AP-pair, the probability calculation takes \(O(h)\) complexity, where \(h\) is the size of the hit set, while our grouping-based approach verifies it with \(O(1)\) complexity. Therefore, our grouping-based approach saves the verification cost by a factor of \(h\).

Because P-soundness can be verified by using the same VO constructed by the certify protocol for the hit set, we omit the details of the certify protocol and only discuss the details of the verify protocol.

**Verify Protocol:** In the verification process, the client partitions the AP-pairs into several groups for verification. We first explain how the AP-pairs are grouped. The grouping is based on the concept of AP-lattice. We define the AP-lattice formally below. We use \(S(v)\) to denote the tuples that the vertex \(v\) corresponds to.

**Definition 4.4:** Given a query \(Q\) and its all-answer result \(R = \{(A, P)\}\), let \(\mathcal{H}\) be the hit set of \(Q\). An AP-lattice \(\mathcal{L}\) of \(R\) consists of \(h + 1\) levels, where \(h\) is the size of the hit set \(\mathcal{H}\). The bottom (top, resp.) of \(\mathcal{L}\) consists of one node, which corresponds to \(\mathcal{H}\) (\(\emptyset\), resp.). Each AP-pair \((A, P) \in R\) corresponds to a node \(v\) at the \(i\)-th level of \(\mathcal{L}\), where \(i = |A| + 1\) (i.e., the number of tuples in \(A\)). For any two nodes \(v_1, v_2 \in \mathcal{L}\), there is an edge between \(v_1\) and \(v_2\) if it satisfies two conditions: (1) \(S(v_1) \subset S(v_2)\), and (2) \(|S(v_2)| - |S(v_1)| = 1\) (i.e., only differ at one tuple). The edge \(\epsilon(v_1, v_2)\) is labeled with the tuple in \(S(v_2)\setminus S(v_1)\). Note that \(S(v_1)/S(v_2)\) always contains a single tuple.

Figure 3 displays an example of AP-lattice constructed from a hit set of four tuples \(\{t_1, t_2, t_3, t_4\}\). We have the following theorem to show an important property of AP-lattice.

**Theorem 4.1:** Given a probabilistic database \(D\), a query \(Q\), and the AP-lattice \(\mathcal{L}\) of \(Q\), for any edge \(\epsilon(v_i, v_j) \in \mathcal{L}\), let \(t\) be the tuple in \(D\) that corresponds to the label of \(\epsilon(v_i, v_j)\), then \(t\) is the AP-pair that \(v_i\) and \(v_j\) correspond to respectively. Then it always holds that

\[
\frac{P_{A_i}}{P_{A_j}} = \frac{p_t}{1 - p_t},
\]

where \(p_t\) is the probability associated with the tuple \(t\) in \(D\).

**Proof.** As \(v_i\) and \(v_j\) are connected by an edge whose label is \(t\) in the AP-lattice, it must be true that \(R_j = R_i \cup \{t\}\). According to Equation (3), we have

\[
P_{R_j} = \Pi_{x \in R_j} p_x \Pi_{x \notin \mathcal{H} \setminus \{t\}} (1 - p_x) = (\Pi_{x \in R_i} p_x) \ast p_t \ast (\Pi_{x \notin \mathcal{H} \setminus \{t\}} (1 - p_x))/(1 - p_t) = P_{R_i} \ast P_{t},
\]

where \(p_t\) is the probability of \(t\) in \(D\). Therefore, it is easy to see that \(\frac{P_{A_i}}{P_{A_j}} = \frac{p_t}{1 - p_t}\). \(\square\)

To have a better understanding of Theorem 4.1 consider the AP-lattice in Figure 3 and two pairs of answer pairs \(<\{t_1\}, \{t_1, t_2\}>\) and \(<\{t_3\}, \{t_2, t_3\}>\) in the lattice. Since their corresponding edges in the AP-lattice are labeled with the same tuple \(t_2\), it must be true that \(\frac{P_{\{t_1, t_2\}}}{P_{\{t_1\}}} = \frac{P_{\{t_2\}}}{P_{\{t_1\}}} = \frac{p_2}{1 - p_2}\), where \(p_2\) is the probability of \(t_2\). Based on this property of the AP-lattice, the client groups the AP-pairs by the following procedure: for any two pairs of AP-pairs, \(PA < (A_i, P_i), (A_j, P_j) >\) and \(PA’ < (A_i’, P_i’), (A_j’, P_j’)>\), \(PA\) and \(PA’\) are assigned to the same group if their corresponding edges in the AP-lattice are assigned with the same label. Figure 3 uses colors to show partial grouping results. For simplicity, we only show one group, with the edges colored red. The grouping is constructed before the client performs the verification procedure.

Given a query \(Q\) and its returned all-answer results \(\mathcal{R} = \{\mathcal{H}, P_{\mathcal{H}}\}\) to be the seed AP-pair (i.e., the answer of the seed AP-pair contains all M-tuples), then the verification follows the 2-step procedure: (1) verify the authenticity, R-soundness and completeness of \(\mathcal{R}\); (2) verify P-soundness of \(\mathcal{R}\). Next, we explain the details of these two steps.

**Step 1:** Verification of authenticity, R-soundness, completeness.

After the hit set \(\mathcal{H}\) passes the verification, the authenticity of \(\mathcal{R}\) naturally follows. R-soundness of \(\mathcal{R}\) is authenticated by
First, the client verifies P-soundness of the seed AP-pair by checking if Π_{j \in M(Q)} P_j = P_{\mathcal{H}} because the minimum coverage set of M(Q) exactly covers the hit set \mathcal{H} (Def. 4.2). If the result P_{\mathcal{H}} passes the verification, the client is assured of P-soundness of the seed AP-pair, and continues to verify P-soundness of non-seed AP-pairs based on the grouping of AP-pairs constructed from the AP-lattice \mathcal{L}. In particular, for each group G, and for each pair of AP-pairs \((A_i, P_i), (A_j, P_j)\) in G, WLOG we assume \(A_i \subset A_j\). Then the client verifies if \(p_i \geq \frac{p_j}{1 - m}\), where \(p_i\) is the probability of the tuple \(t = A_j \setminus A_i\).

C. Authentication of Top-K Queries

In this section, we discuss the details of certify and verify protocols for top-k queries. To verify authenticity, R-soundness and P-soundness of the returned top-k AP-pairs, we follow the same VO construction strategy for all-answer queries. However, the VO construction procedure for the top-k answers is different from that of the all-answer results due to two reasons. First, the seed AP-pair that is used for all-answer query authentication may not be included in the top-k result. Second, the returned top-k results may not include all the AP-pairs that satisfy the query range. This enables to verify the P-soundness of the top-k AP-pairs by using grouping (Section IV-B). Next, we discuss how to resolve these two challenges.

Certify Protocol: To overcome the first challenge, we require the server to include the seed AP-pair \((\mathcal{H}, P_{\mathcal{H}})\) and the proof of \(\mathcal{H}\) (Section IV-A) in the VO, regardless whether the seed is included in the top-k result. To overcome the second challenge, we require that the VO should include some additional AP-pairs (called witness AP-pairs) to ensure that with these witness AP-pairs, each top-k AP-pair can be reachable by the seed AP-pair in the AP-lattice. There may exist more than one set of witness AP-pairs. To reduce the VO size, we will pick the minimum set of the witness AP-pairs. For example, consider the AP-lattice in Figure 4 in which the top-k AP-pairs are colored blue. Given two sets of witness AP-pairs: \(S_1 = \{(t_1, t_3, t_4), (t_2, t_3, t_4)\}\) and \(S_2 = \{(t_1, t_2, t_4)\}\), we pick \(S_2\) as the minimum witness set that is added to the VO in addition to the top-k AP-pairs. Next, we formally define the minimum witness set (MWS).

Fig. 4. An example of the MWS for top-k queries (the nodes of blue color are the top-k results. The nodes in red color construct the minimum witness set (MWS)).

Definition 4.5: Given a top-k query \(Q^k\) and its returned results \(\mathbb{R} = \{(A_i, P_i)\}_{1 \leq i \leq k}\), let \(\mathcal{L}\) be the AP-lattice of \(\mathbb{R}\), and \((\mathcal{H}, P_{\mathcal{H}})\) be the seed AP-pair of \(\mathbb{R}\). A set of AP-pairs \(X\) is the minimum witness set (MWS) of \(\mathbb{R}\) if \(X\) satisfies the following two conditions: (1) for each AP-pair \((A, P)\in \mathbb{R}\), there is a path \(\{v_0, v_1, \ldots, v_j, v_{\text{seed}}\}\in \mathcal{L}\), where \(v_0\) corresponds to the AP-pair \((A, P)\), \(v_{\text{seed}}\) corresponds to the seed AP-pair, and \(v_i (1 \leq i \leq j)\) corresponds to an AP-pair in \(\mathbb{R} \cup X\); and (2) the size of \(X\) is minimum.

The problem of discovering the minimum witness set (MWS) is equivalent to the classic minimum Steiner tree (MST) problem [25]. It is well-known that the MST problem is NP-hard. Therefore, the MWS problem is NP-hard too. We first adapt the 2-approximation algorithm [25] to discover the MST. Then we obtain MWS by extending the path between the node pairs in the MST. The time complexity is \(O(k^2 + L)\), where \(k\) is the size of top-k results in \(\mathbb{R}\), and \(L\) is the total number of edges in the MST. Based on the definition of MWS, now we are ready to define the VO of top-k query evaluation.

Definition 4.6: Given a probabilistic database \(D\), a top-k selection query \(Q^k\) of \(D\) and its results \(\mathbb{R}\) returned by the server, let \(Q\) be the all-answer version of \(Q^k\) (i.e., returns all AP-pairs instead of top-k ones). The VO of \(\mathbb{R}\) includes: (1) the seed AP-pair \((\mathcal{H}, P_{\mathcal{H}})\) of \(Q\); (2) VO of \(Q\) (Definition 4.3); and (3) the MWS of \(\mathbb{R}\).

Example 4.2: Consider the top-k query \(Q^k\) \((k = 4)\) and its hit set \(\mathcal{H} = \{t_1, t_2, t_3, t_4\}\). Assume the query result is \(\mathbb{R} = \{\{t_4\}, \{t_1, t_4\}, \{t_2, t_3\}, \{t_1, t_2, t_3\}\}\). The VO includes (1) the seed AP-pair \((\mathcal{H}, P_{\mathcal{H}})\), (2) the VO of the all-answer query version (as shown in Example 4.1), and (3) the MWS \{\{t_3, t_2, t_4\}\}, which is shown in red color in Figure 4.

Verify Protocol: The verification of a top-k query result \(\mathbb{R}\) consists of two steps: (1) verify authenticity, R-soundness, completeness, and P-soundness of \(\mathbb{R}\) by inspecting the VO; and (2) verify TopK-soundness of \(\mathbb{R}\). This is remarkably challenging, since the client has to check if the returned AP-pairs have the top-k probability without re-computing the probability of all AP-pairs. To address this challenge, we design an efficient verification method that only has to compute at most \(|klnk| + k\) probabilities to find the top-k ones. Next, we explain the details of the two steps of our Verify protocol.

Step 1: Verification of authenticity, R-soundness, P-soundness, completeness. Similar to all-answer verification in section IV-B once the hit set \(\mathcal{H}\) pass the verification, the authenticity of \(\mathbb{R}\) naturally follows. R-soundness of \(\mathbb{R}\) is authenticated by checking if for each AP-pair \((A, P)\) in \(\mathbb{R}\), whether \(A \subseteq \mathcal{H}\). The completeness of \(\mathbb{R}\) is verified by checking if the number of AP-pairs of \(\mathbb{R}\) equals to \(k\). P-soundness verification is slightly different from all-answer query authentication, since the client may not access all the AP-pairs in the AP-lattice \(\mathcal{L}\). The client first checks for every AP-pair \((A, P)\) in \(\mathbb{R}\), whether there exists a path: (1) that consists of nodes corresponding to the AP-pairs in \(\mathbb{R} \cup X\) only, where \(X\) is the MWS of \(\mathbb{R}\), and (2) that connects the AP-pair \((A, P)\) to the seed node in the AP-lattice. Next, the client groups the AP-pairs in \(\mathbb{R} \cup X\) based on the edge labels in the AP-lattice \(\mathcal{L}\). For each pair
The challenge is to minimize the probability computation of any non top-k AP-pair. To address this challenge, we design an algorithm based on the divide-and-conquer strategy that enables the client to catch any TopK-soundness violation. Next we discuss the details of our approach.

**Algorithm 1 Combine**\((R_{1,h/2}, R_{h/2+1,h})\)

Require: The top-k AP-pairs \(R_{1,h/2}\) and \(R_{h/2+1,h}\) of \(\mathcal{H}[1, \ldots, h/2]\) and \(\mathcal{H}[h/2+1, \ldots, h]\), where \(h\) is the size of the hit set \(\mathcal{H}\)

Ensure: The top-k AP-pairs of the hit set \(\mathcal{H}\).

1: \(R = \emptyset\)
2: for \(i = 1\) to \(\min\{k, |R_{1,h/2}|\}\) do
3: \(\quad (A_i, P_i)\) be the AP-pair of the \(i\)-th largest probability in \(R_{1,h/2}\)
4: \(\quad \text{for } j = 1\) to \(\min\{\lfloor\frac{h}{2}\rfloor, |R_{h/2+1,h}|\}\) do
5: \(\quad \quad (A_j, P_j)\) be the AP-pair of the \(j\)-th largest probability in \(R_{h/2+1,h}\)
6: \(\quad (A, P)\) be a new AP-pair, where \(A = A_i \cup A_j, P = P_i \times P_j\)
7: \(\quad \text{Add}(A, P)\) to \(R\)
8: end for
9: end for
10: Sort the AP-pairs in \(R\) by their probability
11: Keep the AP-pairs of top-k probability in \(R\)
12: return \(R\)

The key idea of our verification approach is that the client generates the top-k AP-pairs from the hit set \(\mathcal{H}\) in polynomial time, and compares them with \(\mathbb{R}\). Note that the hit set \(\mathcal{H}\) is always included in the VO of top-k answers. In the literature, the divide-and-conquer (DC) strategy has been widely applied to top-k search \([27], [28]\). We adapt the same strategy to our verification method. Before we explain the details of the verification method, first we present Lemma 4.1 which shows that the top-k AP-pairs of a large hit set \(\mathcal{H}\) can be computed from the top-k AP-pairs of the disjoint subsets.

**Lemma 4.1:** Given a top-k query \(Q^k\) and its hit set \(\mathcal{H}\) = \(\{t_1, \ldots, t_h\}\), let \(\mathcal{H}[1, \ldots, h/2]\) and \(\mathcal{H}[h/2+1, \ldots, h]\) be the first and second half of \(\mathcal{H}\). Let \(R_{1,h/2}\) and \(R_{h/2+1,h}\) be the set of top-k AP-pairs that only consist of the tuples in \(\mathcal{H}[1, \ldots, h/2]\) and \(\mathcal{H}[h/2+1, \ldots, h]\) respectively. Let \((A_i, P_i)\) \((A_j, P_j)\), resp. be the top-i (top-j, resp.) AP-pair in \(R_{1,h/2}\) \((R_{h/2+1,h}, \text{resp.})\). Then the AP-pair \((A, P)\) is a top-k answer of \(Q^k\) if \(i \times j \leq k\), where \(A = A_i \cup A_j, P = P_i \times P_j\).

**Proof.** First, it is straightforward that \(A\) is a possible answer for the hit set \(\mathcal{H}\), i.e., \(A \subseteq \mathcal{H}\), since \(A_i \subseteq \mathcal{H}[1, \ldots, h/2]\) and \(A_j \subseteq \mathcal{H}[h/2+1, \ldots, h]\). Also, it is easy to see that \(Pr(A) = Pr(A_i) \times Pr(A_j) = P_i \times P_j\).

Next, there exist at least \(i \times j\) AP-pairs on \(\mathcal{H}\) whose probability is no smaller than \(P\). For any answer \(A' = A_x \cup A_y\), where \((A_x, P_x) \in R_{1,h/2}\) for any \(1 \leq x \leq i\), and \((A_y, P_y) \in R_{h/2+1,h}\) for any \(1 \leq y \leq j\), it must be that \(P^x = P_x \times P_y \geq P\). This is because \(P_x \geq P_i\) and \(P_y \geq P_j\). So among all the possible AP-pairs for the hit set \(\mathcal{H}\), the highest rank of \((A, P)\) is \(i \times j\). Therefore, in order for \((A, P)\) to be included in \(R_{1,h/2}\), it must be that \(i \times j \leq k\).

Based on Lemma 4.1, we design the divide-and-conquer (DC) method that combines the top-k AP-pairs of \(\mathcal{H}[1, \ldots, h/2]\) and \(\mathcal{H}[h/2+1, \ldots, h]\). Algorithm 2 shows the pseudo code. From Line 3 to 9, for each AP-pair \((A_i, P_i) \in R_{1,h/2}\), we only consider its combination of an AP-pair \((A_j, P_j) \in R_{h/2+1,h}\) if \(j \leq \lfloor\frac{h}{2}\rfloor\), i.e., \(i \times j \leq k\). Among all the generated AP-pairs, we keep k AP-pairs of the highest probability and arrange them in descending order according to their probability. Based on Algorithm 1 we design the verification method that generates the top-k AP-pairs from the hit set \(\mathcal{H}\). Algorithm 2 shows the pseudo code. At high level, we keep dividing \(\mathcal{H}\) until it only includes a single tuple. If \(t\) is the only tuple in \(\mathcal{H}\), there exists only two AP-pairs, i.e., \((\{t\}, p)\) and \((\emptyset, 1-p)\), where \(p\) is the probability associated with \(t\). After that, we keep combining the solutions from the subset of \(\mathcal{H}\) to generate the top-k AP-pairs of \(\mathcal{H}\). The total number of AP-pairs whose probability needs to be recomputed by Algorithm 1 & 2 is 

\[N = k + \frac{k}{2} + \cdots + \frac{k}{k} = k(1 + \frac{1}{2} + \cdots + \frac{1}{k})\]

Since \(\sum_{i=1}^{k} \frac{1}{i} \leq \ln k + 1\) (a property of harmonic series), it follows that \(N \leq k\ln k + k\). In other words, at most \(k\ln k\) probabilities have to be computed to generate the top-k answers.

**Algorithm 2 DC-TopK(\(\mathcal{H}\))**

Require: The hit set \(\mathcal{H}\) = \(\{t_1, \ldots, t_h\}\)

Ensure: The top-k AP-pairs of the hit set \(\mathcal{H}\).

1: if \(h > 1\) then
2: \(R_l = DC-TopK(\mathcal{H}[1, \ldots, h/2])\)
3: \(R_v = DC-TopK(\mathcal{H}[h/2+1, \ldots, h])\)
4: \(R = Combine(R_l, R_v)\)
5: else
6: \(\mathcal{H} = \{t\},\) where \(p\) is the probability of tuple \(t\)
7: \(R = \{\{(t), p\}, (\emptyset, 1-p)\}\)
8: end if
9: return \(R\)

Based on the generated top-k answers, TopK-soundness can be easily verified by comparing the output of Algorithm 1 with the result \(\mathbb{R}\) returned by the server.

V. SECURITY ANALYSIS

In this section, we prove that our verification methods are secure (Def. \([22]\)). Our security analysis first shows the security of the authentication procedure of the hit set, as it is the common procedure for the authentication of both all-answers and top-k answers. Then we discuss the security of authentication protocols of all-answer and top-k queries respectively.
A. Authentication of Hit Set

Theorem 5.1: Given a probabilistic data instance \(D\), an all-answer/top-k query \(Q\) on \(D\), let \(H\) be the hit set of \(Q\), our authentication protocol of hit set \(H\) is secure (Def. 2.2) under the RSA assumption and the collision resistant hash function.

Proof. Given a query \(Q\), consider a probabilistic polynomial-time adversary \(\text{Adv}\) that generates an incorrect result \(\mathbb{R}'\) with incorrect hit set \(H'\) and proof \(\Pi'\) of \(H'\), and tries to pass the verification routine of hit set \(H'\). The incorrect hit set \(H'\) must fall into one of the following cases:

- Case 1. Authenticity violation The hit set is not included in the dataset, i.e., \(H \not\subseteq D\).
- Case 2. R-soundness violation \(H\) does not satisfy the selection range in \(Q\).
- Case 3. Completeness violation There is at least an M-tuple missing in \(H'\).

Next, we prove that the probability that \(\text{Adv}\) can pass the verification is negligible. For Case 1, let \(H' = \{t\}\) be the only tuple that exists in \(H'\) but not in \(D\). In other word, \(H' = H \cup \{t\}\). In order to pass the verification, the signature of MCS \(M'(Q)\) constructed from \(H'\) must pass the root hash re-constructing procedure in the protocol. The probability that \(\text{Adv}\) can pass the verification is the same probability of forging the condensed-RSA signature of \(M'(Q)\) (i.e., the correct answer), which is negligible [20]. Condensed-RSA is proved to be existentially unforgeable against chosen plaintext attack for any probabilistic polynomial-time adversary under the RSA assumption [20].

For Case 2, without loss of generality, we assume that \(t \in D\) and \(t \not\in [l, u]\) is the only incorrect tuple that exists in the hit set \(H'\). In other words, \(H' = H \cup \{t\}\). In order to make the incorrect result to pass verification, \(t\) must not be covered by a leaf node in \(M'(Q)\). Otherwise, the client can catch \(t\) with 100\% certainty according to our certify protocol. Thus, \(t\) must be covered by an internal node in \(M'(Q)\). Let \(N(t_{min})[t_{max}]\sigma(h)\) be the internal node that covers \(t\). Obviously, either \(t_{min} > u\) or \(t_{max} < l\). To let it pass the verification, \(\text{Adv}\) must substitute \(N\) with \(N'(t'_{min}, t'_{max})[p']\sigma'[h']\) s.t. \(t'_{min} > l, t'_{max} < u\) and \(h' = h\). In other words, \(N'\) is an internal node in the selection range of \(Q\) and the hash of \(N'\) matches that of \(N\). Thus the probability that \(\text{Adv}\) passes the verification is the same as the collision probability of the hash function \(H\), which is negligible. In other words, the security against Case 2 follows the security of collision-resistant hash functions [29].

For Case 3, let \(H\) be the correct hit set for the query \(Q\), and \(H'\) be the incorrect hit set. Without loss of generality, let \(t'\) be the only M-tuple in \(H\) that is missing in \(H'\). In other word, \(H' = H \cup \{t'\}\). In order to pass the verification, the signature of MCS \(M'(Q)\) constructed from \(H'\) must pass the root hash re-constructing procedure in the protocol. The probability that \(\text{Adv}\) can pass the verification is the same probability of forging the condensed-RSA signature of \(M(Q)\), which is similar to case 1. \(\square\)

B. Authentication of All-answer Queries

In this section, we prove our authentication protocol of all-answer queries meets the security definition (Def. 2.2).

Theorem 5.2: Given a probabilistic data instance \(D\), and an all-answer query \(Q\) on \(D\), our authentication protocol of all-answer query evaluation is secure under the RSA assumption and the collision resistant hash function.

Proof. Consider a probabilistic polynomial-time adversary \(\text{Adv}\) that generates the incorrect result \(\mathbb{R}'\) which must fall into one of the following cases:

- Case 1. Authenticity violation There exists an AP-pair \((A_i, P_i) \in \mathbb{R}'\) s.t. \(A_i \not\subseteq D\).
- Case 2. R-soundness violation There exists an AP-pair \((A_i, P_i) \in \mathbb{R}'\) s.t. \(A_i\) does not satisfy the selection range in \(Q\).
- Case 3. P-soundness violation There exists an AP-pair \((A_i, P_i) \in \mathbb{R}'\) s.t. \(P_i \neq Pr(A_i)\).
- Case 4. Completeness There is at least a pair \((A_i, P_i)\) missing in \(\mathbb{R}'\).

With the full knowledge of the authentication protocol, \(\text{Adv}\) generates the proof \(\Pi'\) of \(\mathbb{R}'\), aiming to let \(\mathbb{R}'\) being accepted by using \(\Pi'\). Next, we prove that the probability that \(\text{Adv}\) can pass verification is negligible. First, consider Case 1-2. Given the correct hit set \(H\), if there exists an AP-pair \((A_i, P_i) \in \mathbb{R}'\) s.t. \(A_i \not\subseteq D\) or \(A_i\) does not satisfy the selection range in \(Q\), it must be true that \(A_i \not\subseteq H\). The probability that \(\text{Adv}\) can pass the verification is 0. Next, let's consider Case 3. To pass P-soundness verification, \(\text{Adv}\) must make sure that \((\Pi_{N_j \in M'(Q)} \sigma_j)^c = \Pi_{t_i \in H} H(t_i||p_i) \mod B \) and \(\Pi_{N_j \in M'(Q)} P_j = P_H\). To achieve this, \(\text{Adv}\) must return at least one tuple \(t_i \in H\) s.t. \(p_i \neq p_i\). Besides, the proof must pass the authentication based on the Condensed-RSA signature, i.e., \(\text{Adv}\) generates \(M'(Q)\) s.t. \(\Pi_{N_j \in M'(Q)} \sigma_j' = \Pi_{t_i \in H} H(t_i||p_i') \mod B\). Similar to Case 1 in the proof of Theorem 5.1 of hit set authentication (Section V-A), for both scenarios, the security follows the RSA assumption. For Case 4, given the correct hit set \(H\), if a pair \((A_i, P_i)\) missing in \(\mathbb{R}'\), it must be true that \(2|H| \neq |\mathbb{R}'|\). The probability that \(\text{Adv}\) can pass the verification is 0. \(\square\)

C. Authentication of Top-k Queries

In this section, we prove our verification approach for top-k answers is secure (Def. 2.2).

Theorem 5.3: Given a probabilistic dataset \(D\), and a top-k query \(Q^k\) on \(D\), our authentication scheme of top-k answer query evaluation is secure under the RSA assumption and collision hash function.

Proof. Given a query \(Q^k\), consider a probabilistic polynomial-time adversary \(\text{Adv}\) that generates the incorrect top-k result \(\mathbb{R}'\) which must fall into one of the following cases:

- Case 1-4. Violation of authenticity, R-soundness, P-soundness and completeness. Case 1-4 is similar to Case 1-4 for all-answer queries (Section V-B).
• Case 5. TopK-soundness violation. There is a pair $(A_i, P_i)$ in $\mathbb{R}^+$ s.t. $P_i$ is the smaller than the k-th probability.

$\text{Adv}$ generates the proof $\Pi'$ of $\mathbb{R}'$, and tries to pass the verification routine by utilizing $\Pi'$. Case 1-4 is similar to the proof of Theorem 5.2 of all-answer query evaluation (Section V-B). The only difference is that the verification methods also check the AP-pairs in MWS in Case 3 and check if $|R| = k$ in Case 4. For Case 5, the probability that $\text{Adv}$ can pass the verification is negligible. This is straightforward, as the verification approach generates the correct top-k results. □

VI. COMPLEXITY ANALYSIS

For verification preparation, the data owner constructs the APB-tree with $O(n(C_r + C_H + C_P))$ time complexity. It is worth noting that this is a one-time process, and its cost can be amortized over subsequent query verification. Moreover, in our experiments, we find that $C_P$ is cheaper than $C_H$ and $C_r$ by three orders of magnitude. In the VO verification process of all-answer query, the main complexity arises from hash and signature computation of MCS, MF-nodes and C-tuples, which is $O(n_{MCS} + n_{MF} + n_C)(C_H + C_r)$. In comparison, the server has to traverse the APB-tree to construct VO, whose complexity is linear to $n$. Since in practice, $n_{MCS}$, $n_{MF}$ and $n_C$ are significantly smaller than $n$, the verification complexity at the client side is substantially smaller than the VO construction complexity at the server side. To verify top-k queries, the client generates the top-k answers from the hit set by using our Algorithm 2, where the complexity is only $O(hklogk)$. While the server needs to discover the MWS, it is shown in Figure 5 (a). The main observation is that the VO construction time increases linearly with the growth of selection ratio. This is because the server traverses the APB-tree to find the hit set. Consequently, the larger the hit set is, the more APB-tree nodes the server visits to construct VO. Nevertheless, the VO construction process is extremely fast (it never exceeds 2.5 seconds), even for the query of large selection ratio such as 30%. Furthermore, the VO construction time is insensitive to the probability distribution of the data, as the tuple probability does not change the number of APB-tree nodes that are visited to construct VO. We have similar observation on the IIP dataset. We omit the discussion due to the limited space.

We summarize the complexity analysis of the all-answer and top-k query verification approaches in Table II. The verification complexity of our approach is greatly cheaper than that of the client side measured on the Uservisit dataset. We have similar results shown later in this section are the average results of all the queries of the same selection ratio, with 20 times executions per query. For the top-k queries, we vary $k$ from 10 to 300, and the hit set size from 14 to 24.

Compared method To our best knowledge, this is the first work on query authentication for probabilistic databases. Thus we do not have any state-of-the-art work to compare with. Therefore, we consider the baseline approach by which the client first verifies the correctness (soundness and completeness) of the hit set by using the verification approach for deterministic databases [23], then computes the answer probability of each possible answer. The top-k result is obtained by returning $k$ AP-pairs of the highest probability.

B. All-answer Query Verification

First, we measure the VO construction time. The result is shown in Figure 5 (a). The main observation is that the VO construction time increases linearly with the growth of selection ratio. This is because the server traverses the APB-tree to find the hit set. Consequently, the larger the hit set is, the more APB-tree nodes the server visits to construct VO. Nevertheless, the VO construction process is extremely fast (it never exceeds 2.5 seconds), even for the query of large selection ratio such as 30%. Furthermore, the VO construction time is insensitive to the probability distribution of the data, as the tuple probability does not change the number of APB-tree nodes that are visited to construct VO. We have similar observation on the IIP dataset. We omit the discussion due to the limited space.

Second, we measure how the VO size changes with regard to various query selection ratios. The results are shown in Figure 5 (b). First, we observe that the VO size is always relatively small (around 6KB, 0.002% of the data size). We also observe that the VO size increases with the growth of query selection ratio, then drops. This is because when the query selection ratio rises from 0.3% to 20%, the number of C-tuples, MF-nodes and the size of the minimum coverage set (MCS) that covers the M-tuples grows slightly. After that, the number of C-tuples and the size of MCS drops slightly, since an internal node in the APB-tree can cover more tuples in the hit set. Overall, the change in the VO size is substantial (within 10%) with the increase of query selection ratio.

Third, we compare the VO size $|VO|$ with the size of query result $|R|$, and define ratio of VO size $r_s = \frac{|VO|}{|R|}$, where $R$ does not include the VO. We display the ratio of VO size with various query selection ratios in Figure 5 (c). Overall, the VO size is negligible compared with the query result size (the ratio never exceeds 0.7%). Furthermore, we observe that with the growth of the query selection ratio, the ratio of VO size decreases dramatically. The reason is that while the size of query results grows exponentially to the hit set size, the VO size is relatively stable (has been shown in Figure 5 (b)).

In Figure 8 (a), we display the verification time at the client side measured on the Uservisit dataset. We have similar

https://nsidc.org/data/g00807
https://github.com/intel-hadoop/HiBench
observation for the IIP dataset in Figure 7. We only show the result when \( h \leq 24 \), since we are not able to generate all answers in the memory when \( h \) is larger than 24. The results are consistent with our theoretical analysis - the verification time increases exponentially with the size of hit set. We also observe that the verification time is insensitive to the probability distribution of the data. This is not surprising as the verification time is determined by the hit set size, not tuple probability.

**C. Top-k Query Verification**

**VO construction time.** The VO construction time is very small for both datasets. As show in Table III it never exceeds 0.3 second. The fast VO construction is due to the fact that small hit set (i.e. 14-24) leads to fast APB-tree traversal for VO construction.

**VO size.** We measure the VO size and show the results in Figure 6. We can see that the VO size is very small and stable. It never exceeds 5KB when the size of hit set varies from 16 to 24. We also notice that the VO size is insensitive to the choice of \( k \), since the MWS only takes a very small fraction of the VO.

**VO verification time.** We evaluate the verification time of Step 1 (verification of authenticity, R-soundness and completeness) and Step 2 (verification of TopK-soundness) of our verification approach separately. We measure the verification time for small hit sets. From the results shown in Figure 8(b), we observe that the verification is very fast. It never takes more than 25 milliseconds. We also observe that Step 1 dominates the verification time. Moreover, with the increase of \( k \), the verification time of Step 1 keeps stable, but the verification time of Step 2 grows. This is because Step 1 only verifies the correctness of the hit set, whose time performance is irrelevant to \( k \). On the other hand, the complexity of Step 2 depends on \( k \). Besides, we observe that the verification time of both steps increase linearly with the hit set size. This is consistent with our complexity analysis in Section VII.

**D. Comparison with Baseline**

We measure the verification time \( T \) of our approach, and the verification time \( T_B \) of the baseline approach, and report the ratio of verification time measured as \( \frac{T}{T_B} \). Intuitively, the smaller \( T \), the more efficient of our approach compared with the baseline approach.

**All-answer queries.** We show the ratio of verification time on the Uservisit dataset in Figure 10(a). First, we observe that the ratio of verification time is always no more than 9%. In other words, in most cases, our verification approach is 10 times more efficient than the baseline approach. Second, the ratio of verification time decreases when the hit set grows larger. This is because the P-soundness verification (Step 2
in Section [IV-B] takes the majority of the verification time. With the growth of the hit set size, the generation time of all-answer AP-pairs for the baseline approach increases much faster than the P-soundness verification time, which means $T_B$ grows faster than $T$. Thus, the ratio of verification time decreases. This demonstrates that our verification method is suitable for the verification of all-answer queries that have large hit sets. The observation on the IIP dataset is similar as shown in Figure [9].

![Figure 9. Comparison with baseline (All-answer queries, IIP dataset)](image)

**Top-k queries.** We vary the hit set size from 14 to 24, and report the ratio of the verification time in Figure [10] (b). The results show that in all the cases, the ratio is within 10%, which shows that our verification approach is at least 10 times more efficient than the baseline. Furthermore, with the growth of the hit set size, the ratio of verification time decrease dramatically, because the verification complexity of our approach is polynomial to the size of the hit set, while the baseline is exponential to the size of hit set. Moreover, we observe that smaller $k$ yields smaller ratio of verification time, since the verification time of our method increases with the growth of $k$, while the complexity of the baseline approach is irrelevant to $k$.

![Figure 10. Comparison with baseline approach (Uservist dataset)](image)

## VIII. RELATED WORK

**Authentication of Outsourced SQL Queries.** Authentication of SQL query results has been studied by a large body of literature. A variety of SQL queries have been considered. Due to the space limit, we only discuss the related work on range and aggregate queries, which are most relevant to our work. None of the existing works consider the SQL evaluation on the probabilistic database. First, most of the verification methods of range queries use tree-based authentication data structure (ADS). With certain information stored in the ADS, the ADS can be also used to verify the aggregation queries, like SUM, MIN and MAX. Most tree-based methods [30], [23] handle single-dimensional range queries by constructing VO from Merkle Hash tree. Zhang et al. [12] design a system named IntegrIDB that can handle a rich subset of SQL queries, including multi-dimensional range queries, join, and aggregate queries. Those authentication methods can verify the tuple probability. But they cannot verify P-soundness and TopK-soundness. Li et al. [10] design efficient index structures for authentication of a variety of aggregate queries. Compared with this work, we store the aggregate tuple probability in the ADS instead of the aggregate tuple attribute values.

**Query Evaluation on Probabilistic Database.** Fuhr et al. [31] define the relational algebra for probabilistic databases and introduce the intensional semantics for query evaluation. Dalvi et al. [5] transfer an arbitrarily complex SQL query with uncertain predicates to extensional query semantics. More work on algorithms and applications of probabilistic databases can be found in [32]. Re et al. [16] initialize the research on top-k query evaluation on probabilistic databases. They present the top-k query in DNF and design a multi-simulation based Monte-Carlo algorithm for top-k evaluation. Zhang et al. [33] study the semantics of the top-k query evaluation in probabilistic databases. Soliman et al. [34] define a top-k query model named U-topK query model. This model returns the k-length tuple vector that is of the highest probability. A good survey of top-k query evaluation on probabilistic databases can be found in [35]. Most of these works focus on designing efficient algorithms for query evaluation on probabilistic databases. None of them consider authentication of query evaluation.

## IX. CONCLUSION

In this paper, we study the query authentication problem for outsourced probabilistic databases. We design efficient verification solutions for both all-answer and top-k queries. Empirical studies demonstrate the efficiency of our approaches. In the future, we plan to investigate the query authentication methods that support database updates and other operations (e.g., join and aggregation) on probabilistic databases.

## REFERENCES

[1] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, “Robust and efficient fuzzy match for online data cleaning,” in ACM International Conference on Management of Data (SIGMOD), 2003, pp. 313–324.
[2] D. V. Kalashnikov and S. Mehrotra, “Domain-independent data cleaning via analysis of entity-relationship graph,” ACM Transactions on Database Systems (TODS), vol. 31, no. 2, pp. 716–767, 2006.
[3] S. Satpal and S. Sarawagi, “Domain adaptation of conditional probability models via feature subssetting,” in European Conference on Principles of Data Mining and Knowledge Discovery, 2007, pp. 224–235.
[4] L. Antova, T. Jansen, C. Koch, and D. Olteanu, “Fast and simple relational processing of uncertain data,” in IEEE International Conference on Data Engineering (ICDE), 2008, pp. 983–992.
[5] N. Dalvi and D. Suciu, “Efficient query evaluation on probabilistic databases,” The International Journal on Very Large Data Bases, vol. 16, no. 4, pp. 523–544, Oct. 2007. [Online]. Available: http://dx.doi.org/10.1007/s00778-006-0004-3.
[6] C. Re and D. Suciu, “Efficient evaluation of having queries on a probabilistic database,” in International Conference on Database Programming Languages, ser. DBPL’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 186–200. [Online]. Available: http://dl.acm.org/citation.cfm?id=1783534.1783553.
[7] “Deploy the power of data,” http://www.nogamy.com.
[8] “Flatworld believes that instructors are the key to quality student education,” https://catalog.flatworldknowledge.com.
“Lab data management & analysis software,” [https://www.thermofisher.com/us/en/home.html].

F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, “Authenticated index structures for aggregation queries,” ACM Transactions on Information and System Security (TISSEC), p. 32, 2010.

Q. Zheng, S. Xu, and G. Ateniese, “Efficient query integrity for outsourced dynamic databases,” in Proceedings of the ACM Workshop on Cloud Computing Security, 2012, pp. 71–82.

Y. Zhang, J. Katz, and C. Papamanthou, “Integridb: Verifiable sql for outsourced databases,” in Conference on Computer and Communications Security (CCS), 2015, pp. 1480–1491.

R. C. Merkle, “Protocols for public key cryptosystems,” in Symposium on Security and Privacy (S&P), 1980.

B. Kanagal, J. Li, and A. Deshpande, “Sensitivity analysis and explanations for robust query evaluation in probabilistic databases,” in Proceedings of the International Conference on Management of Data. ACM, 2011, pp. 841–852.

J. Hintikka, “Knowledge and belief,” 1962.

C. Re, N. Dalvi, and D. Suciu, “Efficient top-k query evaluation on probabilistic data,” in IEEE International Conference on Data Engineering (ICDE), April 2007, pp. 886–895.

P. Sen, A. Deshpande, and L. Getoor, “Representing tuple and attribute uncertainty in probabilistic databases,” in IEEE International Conference on Data Mining Workshops. IEEE, 2007, pp. 507–512.

D. Eastlake 3rd and P. Jones, “Us secure hash algorithm 1 (sha1),” Tech. Rep., 2001.

R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” Communications of the ACM, vol. 21, no. 2, pp. 120–126, 1978.

E. Myklethun, M. Narasimha and G. Tsudik, “Authentication and integrity in outsourced databases,” ACM Transactions on Storage (TOS), vol. 2, no. 2, pp. 107–138, 2006.

C. Papamanthou, R. Tamassia, and N. Triandopoulos, “Optimal verification of operations on dynamic sets,” in Annual Cryptology Conference, 2011, pp. 91–110.

A. Guttman, R-trees: A dynamic index structure for spatial searching. ACM, 1984, vol. 14, no. 2.

F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, “Dynamic authenticated index structures for outsourced databases,” in ACM International Conference on Management of Data (SIGMOD), 2006, pp. 121–132.

K. Mouratidis, D. Sacharidis, and H. Pang, “Partially materialized digest scheme: an efficient verification method for outsourced databases,” International Journal on Very Large Data Bases, vol. 18, no. 1, pp. 363–381, 2009.

M. R. Garey and D. S. Johnson, Computers and intractability. wh freeman New York, 2002, vol. 29.

V. V. Vazirani, Approximation algorithms. Springer Science & Business Media, 2013.

Z. Xue, R. Li, H. Zhang, X. Gu, and Z. Xu, “De-top-k: A novel top-k selecting algorithm and its parallelization,” in International Conference on Parallel Processing. IEEE, 2016, pp. 370–379.

Y. Kim and K. Shim, “Parallel top-k similarity join algorithms using mapreduce,” in International Conference on Data Engineering (ICDE). IEEE, 2012, pp. 510–521.

I. B. Damgård, “Collision free hash functions and public key signature schemes,” in Workshop on the Theory and Application of of Cryptographic Techniques. Springer, 1987, pp. 203–216.

P. Devanbu, M. Gerst, C. Martel, and S. G. Stubblebine, “Authentic data publication over the internet,” Journal of Computer Security, vol. 11, no. 3, pp. 291–314, 2003.

N. Fuhr and T. Rölleke, “A probabilistic relational algebra for the integration of information retrieval and database systems,” ACM Transactions on Information Systems, vol. 15, no. 1, pp. 32–66, Jan. 1997. [Online]. Available: [http://doi.acm.org/10.1145/239041.239045]

C. C. Aggarwal and S. Y. Philip, “A survey of uncertain data algorithms and applications,” IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 5, pp. 609–623, 2009.

X. Zhang and J. Chomicki, “Semantics and evaluation of top-k queries in probabilistic databases,” Distributed and Parallel Databases, vol. 26, no. 1, pp. 67–126, 2009.

M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, “Top-k query processing in uncertain databases,” in IEEE International Conference on Data Engineering, 2007, pp. 896–905.