Secure, Available, Verifiable, and Efficient Range Query Processing on Outsourced Datasets

Li, Meng; Gao, Jianbo; Zhang, Zijian; Conti, Mauro; Alazab, Mamoun

DOI
10.1109/ICC51166.2024.10622526

Publication date
2024

Document Version
Final published version

Published in
Proceedings of the ICC 2024 - IEEE International Conference on Communications

Secure, Available, Verifiable, and Efficient Range Query Processing on Outsourced Datasets

Meng Li[1], Jianbo Gao[2], Zijian Zhang[2*], Mauro Conti[3,4], Mamoun Alazab[5]
[1]School of Computer Science and Information Engineering, Hefei University of Technology, China
[2]School of Cyberspace Science and Technology, Beijing Institute of Technology, China
[3]Department of Mathematics and HIT Center, University of Padua, Italy
[4]Department of Intelligent Systems, TU Delft, Netherlands
[5]College of Engineering, IT and Environment Charles Darwin University, Australia

***Abstract*—Range queries allow data users to outsource their data to a Cloud Server (CS) that responds to data users who submit a request with range conditions. However, security concerns hinder the wide-scale adoption. Existing works neglect *item availability*, fail to protect *secure verification* or sacrifice search accuracy for efficiency. In this paper, we propose Secure, Available, Verifiable, and Efficient (SAVE) range query processing, which has three distinctive features. (1) Secure availability checking against a malicious CS: we design a keyed index-based secure verification mechanism to check the availability of matched nodes, including validity and freshness. (2) Secure result verification: we design a targeted verification mechanism for result correctness and completeness while not compromising security. (3) Improved efficiency and accuracy: we design a layered encoding method to improve search efficiency and accuracy. We formally stated and proved the security of SAVE in the random oracle model. We conducted extensive experiments over the Yelp and FourSquare dataset to validate the efficiency, e.g., a query over 10 thousand data items only needs 19.4 ms to get queried results and 3.5 ms for local verification.**

I. Introduction

Range query services enable data owners to outsource their data to a Cloud Server (CS). The CS can respond to data users who submit a range query to look for matching data. Promising as they are, range queries incur serious security concerns. This is because data breach is a common occurrence, the CS is not fully trusted [1]–[3], and the outsourced data (range queries) contain sensitive data. For example, the personal information of over 533 million Facebook users are exposed [4] and sensitive data can be correlated with data owners and data users' privacy [5].To dispel security concerns, immense research efforts have been spawned on secure range query processing (PBTree [6], IBTree [7], ServeDB [8]) and related query processing (SecEQP [9]) while achieving a sublinear search time.

We are motivated by three key observations. (1) *Item availability and security.* Node availability is twofold: validity and freshness. Validity means some data items are valid and others are not. Freshness refers to the available duration of data items. Some items are accessible for some time, e.g., some administrative normative documents are effective within five years. (2) *Malicious CS and secure result verifiability.* Most existing work adopts an honest-but-curious security model for the CS [6], [7], [9], [10]. In reality, this does not always hold. We assume that the CS can tamper with the database and neglect some data items. Therefore, the CS has to provide verification information to data users for result correctness and completeness [8]. Meanwhile, we cannot allow the CS to violate the security of unmatched items. (3) *Efficient and accurate range coding.* Efficiency and accuracy are non-ignorable factors in building user-friendly online services. Although our research focus is security, we cannot abandon efficiency or accuracy. Therefore, it necessitates a proper range coding algorithm beneath the security mechanism.

In this work, we propose *a secure, available, verifiable, and efficient range query scheme named SAVE*. Toward this goal, we need to cope with three technical challenges. (1) *How to keep the malicious CS from replacing the verification information of a matching data item?* Since each data item (a leaf node in IBF) has its own verification information for validity checking, the malicious CS may mismatch the leaf nodes with their verification information to deceive the data user. (2) *How to achieve secure result verification without violating the security of unmatched data items?* Result verification involves matching nodes and unmatched nodes. Intuitively, this process requires the CS to provide partial information of unmatched nodes, which in turn leaks their data items. (3) *How to improve search accuracy while ensuring the search efficiency?* Efficiency and accuracy restrain each other during the search. Existing work has achieved good efficiency, but incurs false positives [8].

To address the three challenges, we make the following efforts. **First, we design a keyed index-based secure validity verification mechanism.** Specifically, we design a keyed index-based non-interactive zero knowledge proof of knowledge for validity. Checking validity is basically a problem of existence, i.e., either the data item is valid (1) or invalid (0). Therefore, we leverage the efficient Zero Knowledge Proof of Knowledge (ZKPK) [11] for the data user to check validity locally. Given the Fiat-Shamir heuristic making the proof non-interactive [12], we are enlightened to use the node's index and a secret key as a part of the random challenge to create joint proofs and complete the proving process, thus binding the verification information to its corresponding node and solving

*Corresponding author.

ICC 2024 - IEEE International Conference on Communications | 978-1-7281-9054-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICC51166.2024.10622526

the first challenge. Checking freshness equals to verifying if current timestamp falls in a predefined range. We utilize the efficient Bulletproof [13] to achieve this and craft a keyed index-based proving process similarly.

Second, we design a secure result verification mechanism for correctness and completeness. It is straightforward to verifying correctness and we realize it by building a Merkle hash tree [14] and returning path proofs of matching nodes. For completeness, we first return only segments of IBF $\mathcal{B}$ to data user for unmatched non-leaf node. We treat unmatched leaf node separately because the previous method enables a curious data user to conduct unlimited queries on $\mathcal{B}$ and break security. Therefore, we compute the complementary IBF $\mathcal{B}'$ of each leaf node and return segments of $\mathcal{B}'$ to data user for unmatched leaf node, thus protecting the index of unmatched nodes and solving the second challenge.

Third, we design a layered encoding method as a foundation for efficient and accurate search. The new algorithm first divides the data ranges into l layers with different number of cubes. From the top layer down to the $l-1$th layer, we transform each data item to a cube code by using Hash-based Message Authentication Code (HMAC) to guarantee efficiency. While in the bottom layer with the finest granularity, we adopt prefix encoding [15] to compute the precise codes and compare them with data user's data range codes, aiming to maintain accuracy. Consequently, we can achieve a sublinear search time while not losing accuracy. Our contributions are briefly summarized as follows.

- To the best of our knowledge, we are the first to study item availability and pertinent security in range query.
- We propose a secure, available, verifiable, and efficient range query scheme SAVE.
- We give formal security analysis. We conduct extensive experiments to demonstrate the efficiency of SAVE.

II. Problem Statement

A. System Model

• **Data Owner** (DO) has a dataset $\mathcal{D}$ with data items $\mathcal{D} = \{D_1, D_2, \cdots, D_N\}$. For each D_i, the DO computes a secure index, i.e., an IBF $\mathcal{B}_i$ by using secret keys and a unique random number, and calculates its verification information vi for availability, correctness, and completeness. The DO takes all indexes as leaf nodes and construct a secure index tree STree $\mathcal{ST}$ from bottom to top. The DO encrypts each D_i by using a Chosen Plaintext Attack (CPA)-secure encryption algorithm Enc Next, the DO outsources the $\mathcal{ST}$ and ciphertexts to CS, and delegates the query service to authenticated data users by sharing secret keys with them.

• **Data User** (DU) is an authorized party who has the shared keys and a range query $\mathcal{RQ}$ with some range conditions. The DU computes a trapdoor $\mathcal{TD}$ of Q and sends $\mathcal{TD}$ to the CS. After receiving the query results from the CS, the DU verifies the availability, correctness and completeness of the results, and decrypt the encrypted data items if all verifications pass.

• **CS** stores the STree and the encrypted data items sent from the DO, and processes range queries from the DU. When it searches a td in the $\mathcal{ST}$, it generates corresponding proofs for correctness and completeness. Finally, it returns query results and proofs to the DU.

B. Security Model

Unlike the semi-honest model [6], [7], [9] or limited security model [9] in existing work, we assume that the adversary can mismatch the IBF of any leaf node in $\mathcal{ST}$ with its verification information, tamper with the query results, and neglect a part of the secure index tree (dataset). The CS is malicious because some rogue employee controls the searching process or it has been compromised by adversaries. We adopt the Adaptive INDistinguishable under Chosen Keyword Attack (A-IND-CKA) threat model [16].

Specifically, the malicious CS can launch three types of attacks. **Tampering attack**: tamper with the returned data items that is stored on the cloud. **Neglecting attack**: ignore some data items during search in the secure index tree. **Mismatching attack**: mismatch some leaf nodes with their verification information.

C. Design Objectives

Security. *Data privacy*. The adversary cannot know anything useful about the data items from the secure index $\mathcal{B}$, encrypted data items $\mathcal{E}$, or the verification information vi. *Query privacy*. The adversary cannot learn anything useful about the range query from the trapdoor $\mathcal{TD}$ or the verification information vi. *Result privacy*. The DU cannot infer anything about the unmatched data items more than the fact that they do not meet the search conditions.

Availability. *Validity*. The DU can verify whether the returned data items are valid. *Freshness*. The DU can verify whether the returned data items are fresh.

Verifiability. *Correctness*. The returned data items are not tampered by the CS, i.e., its $\mathcal{B}$ and $\mathcal{E}$ are intact. *Completeness*. The returned data items are all the answers to DO's range query, i.e., no matching data items are left out.

Efficiency. SAVE should achieve high efficiency, i.e., the index building time, trapdoor generation time, query processing time, and the verification time are acceptable.

III. The Proposed Scheme SAVE

A. Overview

At a high level, SAVE consists of five phases: setup, index building, trapdoor generation, range query processing, and local verification. We give an overview of SAVE in Fig. 1.

B. Setup

The DO randomly chooses four secret keys sk_1, sk_2, sk_3, sk_4 to compute cube code, HMAC for item availability proof, HMAC of segments, and symmetric encryption, respectively.

The DO randomly chooses a set of $t+1$ secret keys $\mathcal{SK} = \{k_1, k_2, \cdots, k_{t+1}\}$, t pseudo-random hash functions h_1, h_2, $\cdots, h_t$, and a hash function $H()$ for constructing IBFs and a STree. Here, $h_i() = \mathsf{HMAC}()\%n$ $(1 \le i \le t)$, $h_{t+1}() = \mathsf{HMAC}_{t+1}()$, and $H() = \mathsf{SHA256}()\%2$. n is the length of an

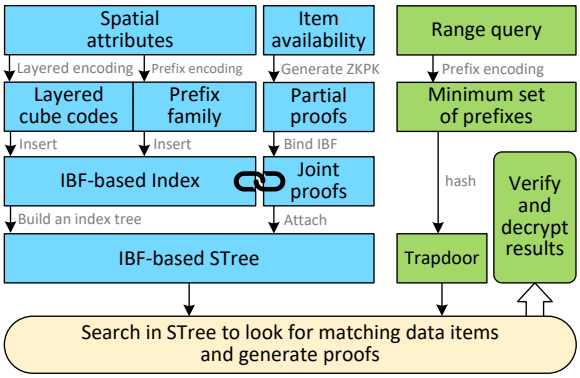


Fig. 1. Overview of SAVE.

IBF. The IBFs on the same layer of the STree share the same length that increases from the bottom up as the high-layer nodes contain more data items.

Assume r is a large prime such that $q = 2r + 1$ and $p = kq + 1$ are prime where k is an even number [11]. The DO initiates a multiplicative group $\mathbb{G}$ with an element g of order r, i.e., $\mathbb{G} = \langle g \rangle$, a fixed element $f \in \mathbb{G}$, and a collision-resistant hash function h. The DO sends the keys and public parameters to qualified DUs.

C. Index Building

The DO, who has a dataset $\mathcal{D}$, encodes the N data items, builds N IBFs, generates proofs of availability, encrypts N data items, and builds an STree as follows.

Layered Encoding. The DO collects the left bound and right bound of each dimension of N data items to form a d-dimensional multi-cube [8]. The DO divides the cube into 2^d equal-size sub-cubes and obtains the first-layer of a layered cube structure. The division continues until the lth layer where the side length reaches a threshold.

Then, the DO encodes each cube from layer 1 to layer $l-1$. Assume a cube cb_i is on layer j, it is encoded to a cube code $c_i = \mathsf{HMAC}(sk_1, j + cp_i.x + cp_i.y)$ where cp_i is the central point of cube cb_i. Each data item D_i will have $l-1$ a cube code set $\mathcal{C}_i = \{c_1, c_2, \cdots, c_{l-1}\}$. For layer l, we transform the value of each dimension into a prefix family by using prefix encoding, thereby avoiding the false positives in existing work [8]. Next, the DO computes codes of the prefixes for D_i and inserts them into $\mathcal{C}_i$.

Note that for the negative numbers, we treat it like positive numbers but with a special character '#'.

Building IBFs. For $\mathcal{C}_i$ of D_i $(1 \le i \le N)$, the DO initiates an IBF $\mathcal{B}_i$, chooses a random number r_i, and inserts the cube codes of $\mathcal{C}_i = \{c_{ij}\}$ into $\mathcal{B}_i$ by setting for $1 \le o \le t$:

$$\mathcal{B}_i[h_o(c_{ij})][H(h_{t+1}(h_o(c_{ij})) \oplus r_i)] = 1, \quad (1)$$

$$\mathcal{B}_i[h_o(c_{ij})][1 - H(h_{t+1}(h_o(c_{ij})) \oplus r_i)] = 0. \quad (2)$$

Generating Proof of Availability (PoA). For $1 \le i \le N$, the DO extracts D_i's auxiliary information $Aux(D_i)$ regarding availability (i.e., validity and freshness), and computes a PoA:

$$\pi_i^{\text{Ava}} = \mathsf{GenProof}(sk_2, Aux(D_i), \mathcal{B}_i). \quad (3)$$

There are two types of π^{Ava}: one for validity and the other one for freshness. For the first type, we design keyed index-based non-interactive zero knowledge proof of knowledge. Instead of proving that DO has a secret s being either 1 (valid, $F = g^s f^z = g f^z$, $z \in Z_q$) or 0 (invalid, $F = g^s f^z = f^z$, $z \in Z_q$), we adopt the proof rationale of the valid case and infuse $(\mathcal{B}_i, sk_2)$ into the process to create a joint proof. For the invalid case, we just use random number to replace the original parameters. We can also swap the two cases. *In this way, the adversary cannot distinguish from the two cases other that a 50% successful probability or recreate the correct proof.* Concretely, if D_i is valid, the DU proceeds as follows:

- randomly choose $e, u_2, v_2 \in_R Z_q$,
- compute $A = f^{u_2} F^{-v_2} \mod p$ and $B = f^e \mod p$,
- compute a random challenge $rc = \mathsf{HMAC}(sk_2, tp, \mathcal{B}_i)$, where tp is the current time period, indicating a regular index update,
- compute $v_1 = rc - v_2 \mod q$ and $u_1 = e + z v_1 \mod q$,
- set as $\pi_i^{\text{Ava}} = (A, B, u_1, u_2, v_1, v_2)$.

For the second type, we design a keyed index-based Bulletproof by using sk_3 to craft two similar random challenges.

Encrypting Data Items. The DO encrypts the N data items to obtain a ciphertext set $\mathcal{E} = \{E_1, E_2, \cdots, E_N\}$ with $E_i = \Omega.\mathsf{Enc}(sk_4, D_i)$. Each ciphertext will correspond to a leaf node during tree construction.

Building STree. Given $\{\mathcal{B}_i, \pi_i^{\text{Ava}}, E_i\}$, the DO constructs a STree $\mathcal{ST}$ as follows.

For $1 \le i \le N$, the DO initiates a leaf node $LN_i = (\mathcal{B}_i, \pi_i^{\text{Ava}})$ and computes the l complementary sets of D_i on l layers. After acquiring all cube codes, the complementary $\mathcal{B}'_i$ is computed. By doing so, when a query does not match a node, instead of strenuously proving such a case, we can prove that the query matches the $\mathcal{B}'_i$, i.e., one cube code of the query is key-hashed into t '1's in $\mathcal{B}'_i$.

Given $\mathcal{B}_i$ and $\mathcal{B}'_i$ $(1 \le i \le N)$, the DO divides each of them into L segments of bits $\{seg_{i1}, seg_{i2}, \cdots, seg_{iL}\}$, computes L HMACs $\{hm_{i1}, hm_{i2}, \cdots, hm_{iL}\}$ corresponding to the segments, and computes a hash value $hv_i = \mathsf{SHA256}(\mathcal{B}_i)$. The segments and their HMACs are to be used for generating a proof for completeness. The hash values are prepared for generating a proof for correctness.

The DO merges the N leaf nodes from the bottom up. For each non-leaf node NN_j, the DO computes a new complementary set as well as a new complementary $\mathcal{B}'$, computes $\mathcal{B}_j$ by using a new random number r_j and the cube codes of the data items from its left child node N_{le} and right child node N_{ri}, and computes $hv_j = \mathsf{hash}(\mathcal{B}_j + hv_{le} + hv_{ri})$.

Finally, a leaf node N_i and a non-leaf node N_j are

$$LN_i = (\mathcal{B}_i, \pi_i^{\text{Ava}}, hv_i, \mathcal{B}'_i, \{seg_{ij}, hm_{ij}\}), \quad (4)$$

$$NN_i = (\mathcal{B}_i, hv_i, \mathcal{B}'_i, \{seg_{ij}, hm_{ij}\}). \quad (5)$$

The DO outsources $(\mathcal{ST}, \mathcal{E})$ to CS and shares hv_{rt} with DUs.

D. Trapdoor Generation

A DU has a range query $Q = [90, 130], [0, 50]$, computes l_1 cube codes for the first $l-1$ layers and a set of cube codes for

the lth layer, and a cube code set $\mathcal{C}_Q$. Then, the DU computes a trapdoor $\mathcal{TD}$ by inserting the cube codes into a sequence of hash pairs. For layer l, the DU transforms $[90, 130]$ and $[0, 50]$ into two minimum set of prefixes MSF and computes two code sets $c_{3i}^x = \mathsf{HMAC}(sk_1, 3 + w_i)$, $c_{3j}^y = \mathsf{HMAC}(sk_1, 3 + w_j)$ where $w_i \in \mathsf{MSF}([90, 130])$ and $w_j \in \mathsf{MSF}([0, 50])$. Finally, the a full set of cube codes of Q is $\mathcal{C}_Q = \{c_1, c_2, \{c_{3i}^x\}, \{c_{3y}^y\}\}$ and the DU computes a trapdoor for $c_i \in \mathcal{C}_Q$, $\mathcal{TD} =$:

$$\{(h_1(c_i), H(h_{t+1}(h_1(c_i))), \cdots, (h_t(c_i), H(h_{t+1}(h_t(c_i))))\}.$$

The DU sends $\mathcal{TD}$ to the CS and waits a result and a proof.

E. Range Query Processing

Searching. The CS initiates a result $\mathcal{R}$ and a proof π, and searches $\mathcal{ST}$ with $\mathcal{TD}$. For a non-leaf node NN_i, the CS maps each element in $\mathcal{ST}$ to $\mathcal{B}_i$. If there exists a cube code in $\mathcal{C}_Q$ that for every $i(0 \le i \le t)$, all the hash values of the mapped positions on $\mathcal{B}_i$ are '1', it means that $\mathcal{TD}$ matches $\mathcal{B}_i$. Therefore, the CS continues to search from the matched non-leaf node MND. Otherwise, the CS stops search from the unmatched non-leaf node (UNN). For a matched leaf node MLN, the CS inserts its E and π^{Ava} into $\mathcal{R}$ and π. For a unmatched non-leaf node ULN, the CS stops branch-search.

Generating Proof of Correctness (PoCor). For MLFs, UNNs, and ULNs, the CS inserts their hash values of $\mathcal{B}$ into π. For MLN, the CS additionally inserts t pairs of segment and HMAC and random number into π.

Generating Proof of Completeness (PoCom). For ULNs and UNNs, the CS inserts one pair of segment and HMAC and a random number of their $\mathcal{B}'$ as PoCom.

Eventually, the CS returns $(\mathcal{R}, \pi)$ to the DU.

F. Local Verification

Given $(\mathcal{R}, \pi)$, the DU verifies the availability, correctness and completeness of $\mathcal{R}$.

Verifying PoA. For validity of a MLN, the DU verifies $rc \stackrel{?}{=} v_1 + v_2 \bmod q$, $f^{u_1} \stackrel{?}{=} B(F/g)^{v_1} \bmod p$, and $f^{u_2} \stackrel{?}{=} AF^{v_2} \bmod p$. For freshness of a MLN, the DU verifies the keyed index-based Bulletproof as in [13]. If either of the two verifications passes, the DU continues to check correctness and completeness.

Verifying PoCor. First, the DU decrypts the ciphertexts and checks whether the data item matches the range condition. Second, the DU recomputes the hash value hv'_{rt} based on the hash values in π and checks if $hv'_{rt} \stackrel{?}{=} hv_{rt}$, the data user acknowledges the correctness of $\mathcal{R}$ and continues.

Verifying PoCom. For UNN and ULN, the DU remaps $\mathcal{TD}$ to each UNN and ULN's segments to see whether the remapped locations include '0'. If so, the DU accepts the $\mathcal{R}$.

IV. Security Analysis

We define two leakage functions as follows. L_1 reveals what is leaked from the index and L_2 reveals what is leaked from the queries and verification operation:

• $\mathsf{L}_1(\mathcal{D}) = \{N, n, \Delta(\mathcal{ST}), |E|\}$: Given a dataset $\mathcal{D}$, L_1 outputs a dataset size N, an IBF length n, an STree structure Δ, and a ciphertext length $|E|$. In SAVE, the STree is constructed as a balanced binary tree, i.e., each non-leaf node is randomly and evenly split into two child nodes.

• $\mathsf{L}_2(\mathcal{D}, Q) = \{\alpha(Q), \beta(Q), \gamma(Q), (\alpha(c_i), \beta(c_i), \gamma(c_i))_{c_i \in Q}\}$: Given a dataset $\mathcal{D}$ and a range query Q, L_2 outputs an access pattern $\alpha(Q)$, a search pattern $\beta(Q)$, and a path pattern $\gamma(Q)$. $\alpha(Q)$ outputs the data item id returned by $\mathcal{Q}$, $\beta(Q)$ outputs the information about whether two queries Q_i and Q_j are different, $\gamma(Q)$ outputs the search path of $\mathcal{TD}$ from $\mathcal{Q}$.

We denote $\mathcal{S}$ as a PPT simulator that can simulate future queries and define two experiments $\mathbf{Real}_{\mathcal{A}}^{\Pi}(1^\lambda)$ and $\mathbf{Ideal}_{\mathcal{S},\mathsf{L}_1,\mathsf{L}_2}^{\Pi}(1^\lambda)$ as follows.

$\mathbf{Real}_{\mathcal{A}}^{\Pi}(1^\lambda)$. A challenger $\mathcal{C}$ executes $\mathsf{Setup}(1^\lambda)$ to generate secret keys $sk_1, sk_2, sk_3, sk_4, \mathcal{SK}$. $\mathcal{A}$ generates a dataset $\mathcal{D}$ and gets a secure index tree $\mathcal{ST} \leftarrow \mathsf{Index}(\mathcal{D}, sk_1, sk_2, sk_3, sk_4, \mathcal{SK})$ from $\mathcal{C}$. $\mathcal{A}$ submits a polynomial number of queries $Q_1, Q_2, \cdots, Q_{num}$ to $\mathcal{C}$. For each Q_i $(1 \le i \le num)$, $\mathcal{A}$ receives a trapdoor $\mathcal{TD} \leftarrow \mathsf{Trapdoor}(\mathcal{SK}, Q)$ from $\mathcal{C}$. Eventually, $\mathcal{A}$ returns a bit b that is output by the experiment.

$\mathbf{Ideal}_{\mathcal{S},\mathsf{L}_1,\mathsf{L}_2}^{\Pi}(1^\lambda)$. $\mathcal{A}$ outputs a dataset $\mathcal{D}'$. Given $\mathsf{L}_1(\mathcal{D}')$, $\mathcal{S}$ produces and sends an index $\mathcal{ST}$ and an encrypted dataset $\mathcal{E}$ to $\mathcal{A}$. $\mathcal{A}$ submits a polynomial number of queries $Q_1, Q_2, \cdots, Q_{num}$ to $\mathcal{S}$. For each Q_i $(1 \le i \le num)$, $\mathcal{S}$ is given $\mathsf{L}_2(\mathcal{D}', Q_i) = \{\alpha(Q_i), \beta(Q_i), \gamma(Q_i), \{\alpha(c_i), \beta(c_i), \gamma(c_i)\}_{c_i \in Q_i}\}$. $\mathcal{S}$ returns a trapdoor $\mathcal{TD}$. Eventually, $\mathcal{A}$ returns a bit b that is output by the experiment.

Formally, a range query processing scheme Π is secure if a PPT adversary $\mathcal{A}$ cannot distinguish the a real index generated by pseudo-random functions from a simulated index generated by truly random functions, with a non-negligible probability:

$$|\Pr[\mathbf{Real}_{\mathcal{A}}^{\Pi}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{S},\mathsf{L}_1,\mathsf{L}_2}^{\Pi}(1^\lambda) = 1]| \le \text{negl}(\lambda),$$

where $\text{negl}(\lambda)$ is a negligible function.

Theorem 1: The range querying processing scheme SAVE is IND-CKA $(\mathsf{L}_1, \mathsf{L}_2)$-secure in the random oracle model against an adaptive adversary $\mathcal{A}$.

Proof: We first construct $\mathcal{S}$ that simulates a view $\mathcal{V}^* = (\mathcal{ST}^*, \mathcal{TD}^*, \mathcal{E}^*)$ based on the information returned by $\mathsf{L}_1(\mathcal{ST}, \mathcal{D})$ and $\mathsf{L}_2(\mathcal{ST}, \mathcal{D}, Q)$. Next, we show that $\mathcal{A}$ cannot distinguish between $\mathcal{V}^*$ and the real adversary view A.

• To simulate $\mathcal{ST}^*$, $\mathcal{S}$ first builds an identically structured STree. $\mathcal{S}$ acquires N from L_1 and sets up an IBF $\mathcal{B}$ for each node in the $\mathcal{ST}$. In the jth cell of $\mathcal{B}_i$, $\mathcal{S}$ sets $\mathcal{B}_i[j][0] = 1$ and $\mathcal{B}_i[j][1] = 0$ or vice versa. The twin is determined by flipping a coin. Next, $\mathcal{S}$ attaches each $\mathcal{B}_i$ with a random number r_i. $\mathcal{S}$ computes $\mathcal{B}'_i$ similarly. For PoA, $\mathcal{S}$ just generates random numbers for proof of validity or proof of freshness. If the PoA is computed to prove validity, then the random numbers will not validate the local verification, i.e., $\mathcal{A}$ successfully guess the underlying data item's validity with probability $1/2$. Otherwise, $\mathcal{A}$ generates a Bulletproof with a fixed range to hide the real freshness. For PoCor, $\mathcal{S}$ computes a hash value hv for each $\mathcal{B}_i$. For PoCom, $\mathcal{S}$ divides each $\mathcal{B}_i$ and $\mathcal{B}'_i$, chooses a random secret key, and computes the HMACs of segments. Finally, $\mathcal{S}$ returns $\mathcal{ST}^*$ to $\mathcal{A}$. Each IBF $\mathcal{B}_i$ in $\mathcal{ST}^*$ and $\mathcal{ST}$

share the same size. Their '0's and '1's are equally distributed. Therefore, $\mathcal{A}$ cannot distinguish $\mathcal{ST}^*$ from the real $\mathcal{ST}$.

• If a received Q has been processed, $\mathcal{S}$ returns the previous trapdoor $\mathcal{TD}$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ generates a new trapdoor $\mathcal{TD}^*$ that is a set of t-pair of hashes. Given the α from L_2, $\mathcal{S}$ knows which data items match $\mathcal{TD}$. For the MLF, $\mathcal{S}$ generates the output by using H to select t-pair of hashes while satisfying that the selected hash pairs match the MLF. For the ULN, $\mathcal{S}$ generates the output by using the random oracle to mismatch the $\mathcal{TD}$ with the leaf node. The t-pair of hashes is $\mathcal{TD}^*$. Since the trapdoor is generated by the random hash functions, $\mathcal{A}$ cannot distinguish $\mathcal{TD}^*$ from the real $\mathcal{TD}$.

• To simulate $\mathcal{E}^*$, $\mathcal{S}$ acquires N and $|E|$ from L_1. Next, $\mathcal{S}$ generates N random plaintexts, simulates N ciphertexts with the random plaintexts and an encryption algorithm $\Omega.\mathsf{Enc}$. Meanwhile, $\mathcal{S}$ ensures that the size of the simulated ciphertext is the same as the one of a real ciphertext. Since $\Omega.\mathsf{Enc}$ is CPA-secure, $\mathcal{A}$ cannot distinguish $\mathcal{E}^*$ from the real $\mathcal{E}$.

• During PoCom generation, only one segment of ULN and UNN's $\mathcal{B}_i$ and its random number are returned to DUs. Hence, a curious DU cannot breach the data privacy of D_i by a brute-force attack, i.e., mapping a new trapdoor into the $\mathcal{B}$.

• To improve security protection of search and access patterns, we can use padding to obfuscate search and access patterns [17], [18]. Furthermore, we can adopt re-encryption to periodically redefine L_1 and L_2 periodically [8].

To sum up, the simulated view and the real view are indistinguishable by $\mathcal{A}$. Therefore, SAVE is adaptive IND-CKA $(\mathsf{L}_1, \mathsf{L}_2)$-secure in the random oracle model against an adaptive adversary. □

V. Performance Analysis

A. Experiment Settings

Dataset and Parameters. We use a Yelp dataset consisting of 150,346 business data. We list the parameter in Table I.

TABLE I
PARAMETER SETTINGS (BOLD: DEFAULT VALUES)

Parameter	Value
Number of data items N	**20000** 40000 60000 80000 100000
Length of keys $\|sk\|$, $\|k\|$	1024
Number of secret keys $t+1$	6
False positive rate (FPR)	1%
Length of an IBF $\|n\|$	$\lceil N * \log(\text{FPR})/\log(1/2^{\log(2)}) \rceil$
Ratio of validity to freshness	**1:1**, 1 : 3, 1 : 2, 2 : 1, 3 : 1.

Metrics. We measure the computational cost (C.C.) and the communication overhead (C.O.) of DO, DU, and CS in the index building, trapdoor generation, range query processing, and result verification.

Baselines. To further evaluate the performance of SAVE, we compare TiveQP with three baselines: (1) PBtree [6] supports privacy-preserving range query on single dimensional data by using Bloom filters. (2) IBtree [7] achieves conjunctive query processing by using IBFs. (3) ServeDB [8] facilitates multi-dimensional and verifiable range query with result verification. Since the first two baselines do not have result verification, we only compare with ServeDB in this regard.

Setup. We implemented SAVE in Java and conducted experiments on a PC server running Windows Server 2021 R2 Datacenter with a 3.7-GHz Intel(R) Core(TM) i7-8770K processor and 32 GB RAM.

B. Index Building

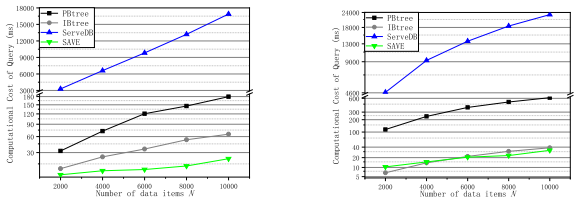
The construction time of STree grows from 33.3 to 166.22 second with N increasing from $2,000$ to $10,000$.The size of STree grows from 15.02 to 75.15 MB.The results indicate a positive correlation between construction cost and N. In STree, due to the need to compute validity and freshness proofs for each leaf, as well as compute hm and hv for each node, more time needs to be spent on these verification information compute operations and more cost to store them. For example, when $N = 2,000$, the tree size is 15.02 MB while the size of verification information is 5.56 MB.

C. Trapdoor Generation

We used a two-dimension query in the experiments, i.e., a query has two ranges corresponding to the locations of data items. The time cost of generating a trapdoor is 0.6 millisecond with a size of 0.48 KB. Since this part is entirely unrelated to N, we do not draw experimental figures.

D. Range Query Processing

Given the structure of the tree index, the time complexity of range query processing is $O(\log(N))$. Fig. 2(a) and Fig. 2(b) shows that the C.C. in different datasets grows sublinearly with N and the time cost is in ms scale. When $N = 2,000$, the processing times are 5.5 and 10.4 ms.

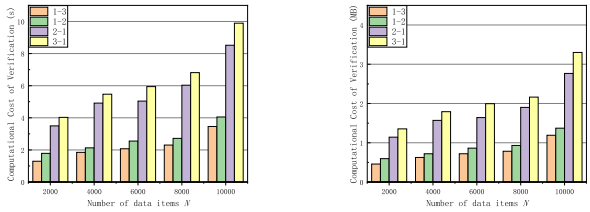


(a) C.C., varying N in Yelp (b) C.C., varying N in FourSquare

Fig. 2. Performance of Range Query Processing.

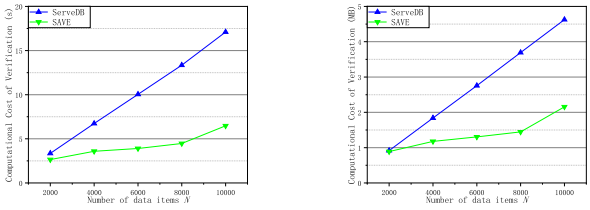
E. Result Verification

Fig. 3(a) and Fig. 3(b) show that the time and size grow with N because the total number of leaf nodes that need to be verified is increasing in different ratio setting. For the same N, the evidence size and validation time for freshness to validity in each ratio are different, and the evidence size generated by a single node is larger for freshness than validity, resulting in a longer validation time.



(a) C.C. with varying N (b) C.O. with varying N

Fig. 3. Performance of Result Verification with Varied Ratio of VP to FP.



(a) C.C. with varying N (b) C.O. with varying N

Fig. 4. Comparison of Result Verification.

F. Comparison

In index construction, the costs of PBtree and IBtree are lower for no verification information. ServeDB costs less than STree for not generating a complementary set of location attribute and then compute more hash, and there is no need to generate extra proof information for validity and freshness for each leaf node. In trapdoor generation, the time and size of all schemes are consistent. In query processing, First, compared to STree, PBtree and IBtree do not use the multi-cube structure to encode location attribute, resulting in lower query efficiency. Second, ServeDB has extra query time for using matched trapdoor set UMT and matched trapdoor set MT that involves more hash checks on Bloom filters. It requires the transmission of Bloom filter and corresponding HMAC of key nodes as proof, which comes at a higher cost. In result verification, ServeDB consumes more time and size than STree (Fig. 4(a) and Fig. 4(b)) because its data user spend more time on verifying the correctness of the entire Bloom filter for each key node and of UMT and MT for each key node. But for STree, it does not have the operations and can accelerate the validation process through validity and freshness verification.

STree has advantages in query processing and result verification. And for the construction, it is an one-time and offline process, which will not impact the efficiency greatly.

VI. Conclusions

In this work, we have proposed a secure, available, verifiable, and efficient range query processing scheme SAVE. In particular, we developed STree in SAVE that organized multi-dimensions data, which enabled security, availability, verifiability, and efficiency of range query. We have formally defined and proved the security and validated the efficiency.

Acknowledgment

This work is supported by National Natural Science Foundation of China (NSFC) under the grant No. 62372149, No. U23A20303, and National Key Research and Development Program of China under the grant No. 2021YFB2701202. It is partially supported by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735.

References

[1] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "Data recovery on encrypted databases with k-nearest neighbor query leakage," *Proc. 40th IEEE Symposium on Security and Privacy (SP)*, 2019: 1033-1050.

[2] M. Li, Y. Shen, G. Ye, J. He, X. Zheng, Z. Zhang, L. Zhu, and M. Conti, "Anonymous, secure, traceable, and efficient decentralized digital forensics," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2023, Online.

[3] Y. Ye, D. Ma, Z. Wen, Y. Ma, G. Wang, and L. Chen, "Efficient graph query processing over geo-distributed datacenters," *Proc. 43rd ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, 2020: 619-628.

[4] A. Holmes, "533 million Facebook users' phone numbers and personal data have been leaked online." Insider, 2021. https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4

[5] Y. Cao, Y. Xiao, L. Xiong, L. Bai, and M. Yoshikawa, "Protecting spatiotemporal event privacy in continuous location-based services," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2021, 33 (8): 3141-3154.

[6] R. Li, A. Liu, A. L. Wang, and B. Bruhadeshwar, "Fast range query processing with strong privacy protection for cloud computing," *Proc. 40th International Conference on Very Large Data Bases (VLDB)*, 2014: 1953-1964.

[7] R. Li and A. X. Liu, "Adaptively secure conjunctive query processing over encrypted data for cloud computing," *Proc. 33rd IEEE International Conference on Data Engineering (ICDE)*, April 2017: 697-708.

[8] S. Wu, Q. Li, G. Li, D. Yuan, X. Yuan, and C. Wang, "ServeDB: Secure, verifiable, and efficient range queries on outsourced database," *Proc. 35th IEEE International Conference on Data Engineering (ICDE)*, 2019: 626-637.

[9] X. Lei, A. X. Liu, R. Li, and G.-H. Tu, "SecEQP: A secure and efficient scheme for SkNN query problem over encrypted geodata on cloud," *Proc. 35th IEEE International Conference on Data Engineering (ICDE)*, 2019: 662-673.

[10] Y. Zhang, X. Jia, B. Pan, J. Shao, L. Fang, R. Lu, and G. Wei, "Anonymous multi-hop payment for payment channel networks," *IEEE Transactions. Depandeble and Secure Compting (TDSC)*, 2023.

[11] W. Mao, "Guaranteed correct of integer factorization with pff-line shareholders," *Workshop of Public Key Cryptography (PKC)*, February 1998, 60-71.

[12] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *Proc. 3rd Advances in Cryptology (CRYPTO)*, January 1987, 186-194.

[13] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," *Proc. 39th IEEE Symposium on Security and Privacy*, 2018: 315-334.

[14] M. Szydlo, "Merkle tree traversal in log space and time," *Proc. 10th International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, May 2004: 541-554.

[15] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC)*, August 2008: 95-104.

[16] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *Proc. 13th ACM Conference on Computer and Communications Security (CCS)*, November 2006: 79-88.

[17] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "BlindBox: Deep packet inspection over encrypted traffic," *SIGCOMM*, 2015: 213-226.

[18] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big data analytics over encrypted datasets with seabed," *OSDI*, November 2016, 587-602.