Private Search on Key-Value Stores with Hierarchical Indexes

Haibo Hu #1, Jianliang Xu #2, Xizhong Xu #*3, Kexin Pei #4, Byron Choi #5, Shuigeng Zhou *6

# *Department of Computer Science, Hong Kong Baptist University, China*
1,2,4,5 {haibo,xujl,kxpei,bchoi}@comp.hkbu.edu.hk

* *School of Computer Science, Fudan University, China*
3,6 {xuxzh,sgzhou}@fudan.edu.cn

***Abstract*—Query processing that preserves both the query privacy at the client and the data privacy at the server is a new research problem. It has many practical applications, especially when the queries are about the sensitive attributes of records. However, most existing studies, including those originating from data outsourcing, address the data privacy and query privacy separately. Although secure multiparty computation (SMC) is a suitable computing paradigm for this problem, it has significant computation and communication overheads, thus unable to scale up to large datasets. Fortunately, recent advances in cryptography bring us two relevant tools — conditional oblivious transfer and homomorphic encryption. In this paper, we integrate database indexing techniques with these tools in the context of private search on key-value stores. We first present an oblivious index traversal framework, in which the server cannot trace the index traversal path of a query during evaluation. The framework is generic and can support a wide range of query types with a suitable homomorphic encryption algorithm in place. Based on this framework, we devise secure protocols for classic key search queries on B$^+$-tree and R-tree indexes. Our approach is verified by both security analysis and performance study.**

I. Introduction

In the information age, organizations, corporations or even individuals collect and own large amounts of data, based on which they provide query services to subscribers. As these data are private assets of the service providers, they should be properly protected against the subscribers, who should get only the query results and nothing beyond. On the other hand, the subscribers have similar privacy requirements — neither their queries nor the results (from which sensitive information might be inferred) should be learnt by the service providers. Protection of mutual privacy fits into many business models, one of which is the NoSQL key-value store on cloud databases (e.g., Amazon's SimpleDB [1]). The most common query in this model is to search a key for its corresponding value. For example, a WiFi service provider stores the WPA2 network security keys of all WiFi hotspots it manages. A subscriber queries for the security key of the hotspot to which he/she is connecting. From the service's perspective, the subscriber should access the key for this hotspot only; whereas from the subscriber's perspective, the service should not learn which hotspot he/she is connecting since this will disclose his/her location information.

Such application scenarios widely exist in commercial, medical, public service and military sectors, where both queries and data can reveal confidential intelligence. As another example, a doctor accesses an electronic health record (eHR) database for his/her patient's recent biological test result. The doctor should receive only this patient's result through a unique identifier, while the database should not know the test result being accessed, as it was protected by the patient-doctor confidentiality. These scenarios become even more prevalent since the emergence of cloud computing and database-as-a-service, where outsourced service providers are generally distrusted by the subscribers.

The above scenarios call for a query processing model for key-value stores that preserves *mutual* privacy of both parties: the service provider can protect its data privacy by revealing no key or value except for the returned one, and the subscriber can protect her query privacy against the service provider by disclosing nothing about the search key or the returned value.

In the literature, there are a lot of research efforts that address data privacy or query privacy separately. For data privacy, generalization [2] has been proposed to protect quasi-identifier attributes and avoid the disclosure of sensitive information. For query privacy, a similar technique called location cloaking has been proposed in location-based services to generalize (i.e., blur) the user locations when they issue queries [3], [4], [5], [6]. However, these techniques still disclose the data or query in a coarser and imprecise form.

More recently, there are two categories of research that aim at preserving strong and *unconditional* privacy. The first category, introduced by data outsourcing, assumes that the user owns and hosts data on the server in an encrypted form [7], [8], [9], [10], [11], [12], [13]. The query from trusted parties is also encrypted in the same form for the outsourcing server to evaluate. Unfortunately, this approach cannot be applied to our problem in this paper as the server still knows the returned value or a superset of it, whether encrypted or not. The second category of research, secure multiparty computation (SMC), originates from cryptography. SMC computes a function from multiple participants in a distributed network, each holding one private input. During the computation, no information is revealed to any participant except those implied by this participant's input and the function output. Some specialized SMC algorithms have been devised for certain query types, such as the nearest neighbor search [14], [15]. However, SMC-based solutions in general are expensive in terms of computation and communication costs, so they cannot scale up to real datasets with millions of records.

These studies, however, inspire us to integrate SMC techniques with database indexes for efficient privacy-preserving query processing. As an index node only consists of a handful of data items, the execution of an SMC algorithm on the index becomes practical. Nevertheless, there are several challenges along this integration. First, a database index, e.g., a B$^+$-tree or R-tree, consists of a hierarchy of nodes, and query processing is essentially a traversal on these nodes. The service provider should not be able to trace this traversal and hence to get any clue of the query. In this paper, we propose an oblivious index traversal framework where only the client keeps track of the traversal through a local "shadow index". While this is a general-purpose framework, we then focus on the key search query in a key-value store, and reduce it to "conditional oblivious transfer" in SMC [16]. To support single- and multi-dimensional key space, we design secure and efficient index-node access protocols for two classic indexes, namely, B$^+$-tree and R-tree. The second challenge is for the client to retrieve the value of the search key obliviously from the server. To this end, we propose an oblivious value transfer protocol based on the shadow index scheme. To summarize, our contributions in this paper are as follows:

- To the best of our knowledge, this is the first work that studies private search over hierarchically indexed key-value stores for **mutual privacy protection**. Combining both security and efficiency, this approach is more practical than existing theoretical solutions.
- We present a general-purpose oblivious index traversal framework that accommodates any multi-level index. The framework can resist traceablility from the service provider during query processing.
- Based on this framework, we present a comprehensive set of protocols for private key search in classic B$^+$-tree and R-tree indexes.
- We thoroughly analyze the security and complexity of the proposed framework and protocols. We also conduct experiments to evaluate the performance on real datasets.

The rest of the paper is organized as follows. Section II reviews existing work on privacy-preserving query processing. Section III formulates the problem and Section V presents the oblivious index traversal framework. Section VI studies private key search on B$^+$-tree index and one-dimensional data. Section VII extends to R-tree index and multi-dimensional data. Section VIII analyzes the security of the framework and protocols, followed by the performance optimization and evaluation in Sections IX and X. Section XI concludes this paper with some future research directions.

II. Related Work

In this section, we review two categories of techniques that aim at preserving unconditional (but not necessarily mutual) privacy in query processing.

Transformation-based Query Processing: The first category transforms both the query and the data into another space for evaluation, using hashing or space filling curves. This technique was first introduced in data outsourcing where an *untrusted* service provider (SP) stores and manages the data on behalf of the data owner, who then invites *trusted* users to query on the data. Agrawal *et al.* proposed an order-preserving encryption scheme (OPES) for 1D numeric values [7]. Yiu *et al.* adopted the transformation technique for 2D spatial data points and proposed hierarchical space-division (HSD) for kNN queries [10]. Alternatively, hashing (bucketization) and encryption schemes have also been proposed for kNN queries [10], [13], similarity queries [12] and range queries [17]. Since data outsourcing does not need to protect data privacy against trusted querying users, all these transformation techniques are not required to accurately preserve the same operation in ciphertext domain; instead, the querying user can get a superset of the result for local refinement. To protect mutual privacy as in this paper, Khoshgozaran and Shahabi studied transformations for nearest neighbor search [9]. They use space filling curves as the transformations, due to the locality and distance-preserving properties of these curves. However, since distance is not completely preserved in the transformed space, the results are only approximate kNNs. Another disadvantage of transformation techniques is the potential disclosure risks from the distance-preserving property [18].

SMC-based Query Processing: The second category of research originates from the theoretical results of secure multiparty computation (SMC). The most fundamental problem in the SMC literature is Yao's Millionaire Problem [19], which compares two private numbers. Theoretically, the Millionaire problem and the secure multi-party computation problem in general can be solved using circuit evaluation protocol. In this protocol, the function is represented by a Boolean circuit, and each party jointly evaluates the circuit without disclosing to the other parties their own inputs that are fed into this circuit. However, the communication cost of such protocols depends on the size of the circuit, which in turn depends on the size of the input domain and on the complexity of function expression [20]. Since then, many research efforts, especially those from privacy-preserving data mining community, have been made to develop more efficient SMC protocols for specialized functions and tasks, including secure sum [21], scalar product [22], [23], add vector [24], [25], and set operations [21].

As for query processing, Qi and Atallah solved NN queries for horizontally-partitioned data from two parties [14]. They apply a blind-and-permute protocol, together with a secure selection protocol. The computation and communication complexity is linear to the dataset size. They also extended it to kNN queries and proposed a multi-step protocol. In location-based services, Ghinita *et al.* proposed a private information retrieval (PIR) based approach for 1NN queries [15]. It partitions the space into grid cells as retrieval units, and the user can privately retrieve the content of his/her residing cell to resolve the 1NN query. Similar to data outsourcing solutions, this approach sends a superset of the result to the user. There is some other research on vertically partitioned data, i.e., attributes of every record are owned by different parties. Vaidya and Clifton studied top-k queries where the sorting metric is a sum of scores computed independently at different parties [26]. Li and Chen proposed a trusted third party to perform joins on the data from mutually distrustful parties, with the aid of a secure coprocessor installed at the third party [27]. Jagannathan and Wright proposed a secure k-means clustering algorithm for either horizontally or vertically partitioned data from two parties [28]. As for query authentication, we have studied privacy-preserving authentication schemes for range [29] and kNN queries [30] in location-based services.

Differences from This Work: Our work distinguishes itself from the others by addressing the mutual privacy protection problem for queries over large-scale, *indexed* data. Unlike our previous work [31] that studied distance-based queries (including range and kNN) on two-dimensional data, this work deals with key search queries on indexed key-value stores. Furthermore, this work features general-purpose oblivious index traversal framework that accommodates any multi-level index. It is noteworthy that privacy-preserving search on tree-structured data has been investigated in some existing studies [8], [32], [33]; however, these studies assumed a three-party data outsourcing computing model, where the owner knows both the data and the query while the outsourcing server knows nothing. This model differs from our two-party one where one side owns the query and the other side owns the data.

III. Problem Formulation

The system model consists of two parties: 1) a server who owns a large dataset $\mathcal{D}$ of key-value pairs and, 2) a client who wants to search for a particular value by providing the correct key. The key ξ is a *sensitive* attribute such as the social security number, credit card number, or location coordinates, so the client wants to keep it anonymous from the server. On the other hand, as the value attribute is its asset, the server wants to return only the value that matches the search key q. This mutual privacy model requires both q and the values (except for the matched one) kept secret to each other during query processing, and in the end, the client gets only the matched value (or nothing if q does not exist) and the server knows nothing about q.

Two apparent solutions that do not work: The first solution applies the same hashing or encryption on both sides and lets the server match the search key in the transformed space and return the matched value. This does not work because the server can learn the matched value (whether it is encrypted or not) and thus infer the search key.[1] The second solution stores the hashed or encrypted keys (together with their values) at the client, who can then match the search key locally. However, in this solution the client can enumerate all valid keys in a brute-force manner without the server's control at all (recall that the dataset is a private asset of the server).

A. Data Model

The keys on attribute ξ can be any data type. Without loss of generality, we assume they are first hashed by a cryptographic hash function such as MD5 or SHA-1. These hash values are the actual keys that are indexed by a multi-level index such as B$^+$-tree and R-tree. Figure 1 illustrates an abstraction of such an index. The index consists of a hierarchy of nodes, each of which comprises many *entries*. Each entry corresponds to a child node — it consists of a pointer p to the child node and a range e of all keys in this node. In particular, in an R-tree the range e is a multi-dimensional box, and in B$^+$-tree the upper bound of e is omitted. That is, the i-th entry in a B$^+$-tree node consists of a single key value k_i and a pointer p_i.

[1]For example, by judging if two search keys return the same value, the server can tell if they are the same. As such, frequency-based attacks [34] can be launched to infer search keys.

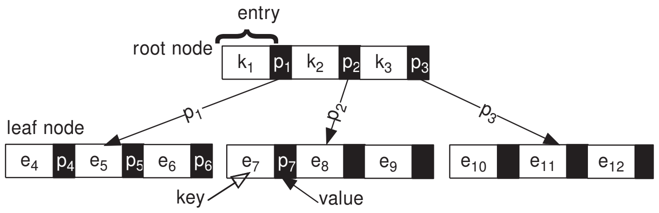


Fig. 1. Multi-Level Index

The key range implied is $[k_{i-1}, k_i)$, and $k_0 = -\infty$, $k_n = +\infty$. The *root* node of a multi-level index is on the topmost level and the *leaf* nodes are on the lowermost level. An entry in a leaf node (called a "leaf entry") stores a key-value pair in its e and p, respectively, for example $\langle e_7, p_7 \rangle$ in Fig. 1.

There are two remarks about the problem definition. First, for simplicity we assume the values are accommodated in the leaf-entry's "pointer" fields. This assumption suffices for many applications where the value size is moderate (e.g., the WPA2 security key or a blood test result in a set of positives or negatives). Nonetheless, if the value size is too large to fit in the pointer field, it will be stored in a separate array of values at the server side and the pointer field stores its index in this array. After obtaining such an index, the client can invoke a standard private information retrieval protocol with the server to retrieve the actual value. Second, our problem setting allows the client to enumerate valid keys to the server in a brute-force manner. While such misbehavior can be audited and prohibited by the server through access control, captcha, or other penalties on heavy users, our problem does not target the applications where the valid key domain is both small and publicly known by the client, such as the 5-digit US zip code.

B. Security Model

We consider two kinds of adversaries: 1) a client adversary attempts to obtain the proprietary values without actually owning their keys; 2) a server adversary attempts to infer the search key of a client. The privacy objective of our problem is that *no keys or values should be disclosed to the client except its search key q and the matched value v, and the server should not learn q*. As such, we define the security model in this paper as *indistinguishability* — the client cannot distinguish a valid key (or value) s from an invalid one s', except for the keys and values it has searched before; the server cannot distinguish a search key q by the client from a random key q'. Formally,

Definition 3.1: **Indistinguishable Key, Value and Search Key**. Given s and s' to the client, a probabilistic polynomial-time adversary distinguishes s as a valid key (or value) with probability

$$Pr[s = valid] \leq \frac{1}{2} + negl,$$

where $negl$ is a negligible function. Similarly, given q and q' to the server, a probabilistic polynomial-time adversary distinguishes q as the search key with probability

$$Pr[q = search\_key] \leq \frac{1}{2} + negl.$$

In this definition, the probabilistic adversary, playing the role of a client or the server, is polynomially bounded by its computational power and storage space. Following cryptography conventions, we further assume an adversary: i) to know all protocols and algorithms except for the secret keys of its counterparts; ii) to follow a semi-honest model;[2] and iii) to possess no a-priori knowledge of the keys.[3]

IV. Preliminary: Cryptographic Building Blocks

In this section, we introduce three secure building blocks that are heavily used in this paper, namely, homomorphic encryption, conditional oblivious transfer and commutative encryption.

A. Homomorphic Encryption

A homomorphic encryption is an encryption scheme that maps an operation on plaintexts to another (probably different) operation on ciphertexts. Formally, let M and C denote the plaintext and ciphertext space, respctively. For any $m_1, m_2 \in M$ and $c_1, c_2 \in C$ where $m_1 = E^{-1}(c_1)$ and $m_2 = E^{-1}(c_2)$, it holds that

$$E^{-1}(c_1 \odot c_2) = m_1 \odot m_2,$$

where $\odot$ is the group operation in M and C [35]. Practical homomorphic encryptions include Paillier, El Gamal, and Goldwasser-Micali (GM), all of which are public-key crytosystems. In particular, Paillier encryption [36] is homomorphic over a large additive group $\mathcal{Z}_N$, where N is its public key:

$$E_N(m_1) \cdot E_N(m_2) = E_N([m_1 + m_2 \mod N]).$$

B. GT-COT: Conditional Oblivious Transfer for "Greater Than" Predicate

Oblivious transfer (OT) is a fundamental operation in many SMC protocols. It helps a sender to transmit one out of a set of data values to a receiver without it knowing which value has been transferred. A similar cryptography notation "private information retrieval" (PIR) relaxes the requirement of OT that the receiver can only receive one value. As this compromises data privacy, it cannot be applied to our problem. Conditional oblivious transfer (COT), proposed by Crescenzo et al. [37], is a variant of OT where the receiver does not know the value to receive in advance, and instead, this value is determined by the result of a predicate on private inputs from both parties. A common predicate is "greater than" and thus leads to the acronym "GT-COT". Formally, let S denote the sender and R the receiver, and let D_I and $D_S = \{s_0, s_1\}$ be the input domain and value domain (both are known to both parties). Let R have input x and S have input y, $x, y \in D_I$ and $x \neq y$. After the GT-COT protocol, R receives s_0 if $x < y$, or receives s_1 if $x > y$. Note that the predicate result, i.e., $x > y$ or $x < y$ is unknown to S but known to R.[4]

There are several GT-COT protocols in the literature and the state-of-the-art one [16] is by Blake et al. It is a round-trip protocol initiated by R, who encrypts each bit x_i of x by a homomorphic encryption $Enc(\cdot)$. These ciphertexts $Enc(x_i)$ are sent to S, together with the public key pk of $Enc(\cdot)$. S then computes a series of variables in the ciphertext domain based on the homomorphic property and sends the final results μ to R. R decrypts them using its private key sk and outputs the only valid $\mu_i \in D_s$ as the received value. We adopt this GT-COT protocol in our framework, but it can be switched to any other GT-COT protocols.

C. Commutative Encryption

A commutative encryption $CE(\cdot)$ satisfies the property that the order of two encryptions is irrelevant. Formally, let e_1 and e_2 denote the encryption keys, and m denote a plaintext message. Then

$$CE_{e2}((CE_{e1}(m)) = CE_{e1}((CE_{e2}(m)).$$

In the literature, the Pohlig-Hellman exponentiation cipher (abbr. PH cipher) is commutative [38]. Its key generation, encryption and decryption can be summarized as follows:

- **Key generation**: Generate a prime z larger than the maximum possible m. Pick a random e as the encryption key, and the decryption key d is:
$$d = e^{-1} \mod z - 1.$$
- **Encryption**: The ciphertext c is computed as:
$$c = m^e \mod z.$$
- **Decryption**: The plaintext m can be recovered from c as:
$$m = c^d \mod z.$$

PH cipher is commutative with regard to the same modulus z. That is, for encryption keys e_1 and e_2, we have

$$CE_{e2}((CE_{e1}(m)) = CE_{e1}((CE_{e2}(m)) = m^{e_1 e_2} \mod z.$$

As for the security, the symmetric PH cipher is proved to sustain known plaintext attacks (KPA) and chosen plaintext attacks (CPA) due to the difficulty of Discrete Logarithm Problem (DLP) [35].

V. Oblivious Index Traversal Framework

The private key search problem can be resolved by applying a commutative encryption alone, as will be presented below. However, directly applying such costly cryptographic tools cannot scale up to large datasets. Therefore, in this paper we integrate them with indexes for more efficient query processing. As a key search query over an index is essentially a traversal on the index nodes, to preserve the privacy of search key, the client must keep the traversal path away from the server lest the latter narrow down its range. In this section, we propose "oblivious index traversal" as the enabling framework, which is illustrated in Figure 2. The key idea is to let the client keep track of the traversal path so that the server does not know the exact node the client is accessing. To achieve this, during initialization (step 0), the server sends to the client a *shadow index* — an encrypted version of the index I (denoted by $E(I)$). $E(I)$ has the same topology as I, with the entries of each node encrypted as follows. For an intermediate node n, the key range of each entry (e.g., e_1, e_2, e_3) is encrypted by an additively homomorphic encryption $E(\cdot)$, while the pointers

[2]That is, it follows the designated protocol properly, but may record intermediate results and try by all means to deduce about the private information of other parties [35].

[3]If there is a-priori knowledge, hashing or other transformations on the key domain can always be applied first.

[4]In [16], Blake and Kolesnikov proposed a stronger COT protocol where R cannot learn the predicate result either.

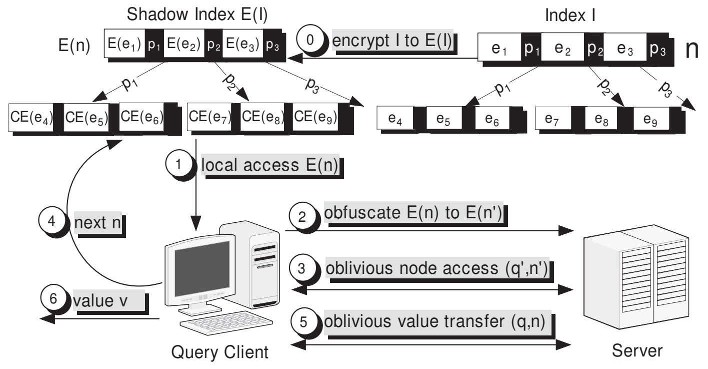


Fig. 2. Oblivious Index Traversal Framework

(e.g., p_1, p_2, p_3) are not encrypted. For a leaf node, both entries and pointers (which store the keys and values) are encrypted by a commutative encryption $CE(\cdot)$. By padding with dummy entries, each leaf and non-leaf node has the same number of entries. The shadow index is then stored at the client side for future access.

Given a search key q, starting from the root node, the client iteratively retrieves the next node $E(n)$ to access from the shadow index (step ①) and repeats the following steps ②③④ until the search terminates. Before sending $E(n)$ to the server, the client obfuscates it to $E(n')$ (step ②) lest the server recognize it. The obfuscation also changes the search key from q to q' accordingly. The server then decrypts $E(n')$ to n' and initiates a two-party oblivious node access (ONA) protocol with q' at the client (step ③). This protocol designates the client the pointer of the next child node to access (step ④). It is noteworthy that this framework works for a broad set of queries besides the key search, e.g., a selection or range query. Algorithm 1 shows the pseudo-code of key search under this framework. In particular, when n reaches a leaf node, to obtain the matched value v, the client will invoke a two-party oblivious value transfer (OVT) protocol with q and the encrypted keys (as well as values) in this shadow node n (step ⑤).

Algorithm 1 Oblivious Index Traversal Framework

```
Input:      $q$: the search key at the client
            $E(I)$: the shadow index at the client
            $E(\cdot)$: additively homomorphic encryption from server
Output:     $v$: the value returned to the client
Procedure:
1:  client initializes $n$, the next node to access, as the root of $E(I)$
2:  while the search is not terminated do
3:    if $n$ is a non-leaf node then
4:      client retrieves the local $E(n)$ // step ①
5:      client obfuscates it into $E(n')$ and sends it to the server //
        step ②
6:      server decrypts it into $n'$
7:      client and server perform $oblivious\_node\_access(q', n')$ //
        step ③
8:    else
9:      // $n$ is a leaf node
10:     client and server perform $oblivious\_value\_transfer(q, n)$
        to obtain the value $v$ // step ⑤
11:   client updates $n$ // step ④
```

Algorithm 1 establishes a unified way of private key search on various types of indexes. While the oblivious value transfer protocol is generic and will be presented in the following subsection, the major difference across various types of indexes lies in the obfuscation and ONA protocols (steps ② and ③). The next two sections will study these protocols on B$^+$-tree and R-tree, respectively.

A. *Oblivious Value Transfer (OVT) Protocol*

This protocol enables the client to test whether its query key q matches any of the (encrypted) keys in a shadow leaf node n. If it does, the matched value v is returned to the client. As for the security requirement, the server is completely oblivious — it does not learn anything about n, q or v, while the client only learns v.

The input of this protocol is the search key q and n at the client-side, in the form $CE_s(e_i)$ and $CE_s(v_i)$ of every entry i, where $CE_s(\cdot)$ is the commutative encryption with the server's key. To prevent them from being recognized by the server, the client re-encrypts each with her own encryption key, denoted by $CE_c(\cdot)$. As such, the server receives $CE_c(q)$, $CE_c(CE_s(e_i))$ and $CE_c(CE_s(v_i))$. The server then re-encrypts the query as $CE_s(CE_c(q))$ and matches it with $CE_c(CE_s(e_i))$ of every entry i. If there is a match, the server decrypts the corresponding $CE_c(CE_s(v_i))$ value as follows:

$$CE_s^{-1}(CE_c(CE_s(v_i))) = CE_s^{-1}(CE_s(CE_c(v_i))) = CE_c(v_i).$$

Once the clients receives $CE_c(v_i)$, it obtains the matched value v_i with decryption:

$$CE_c^{-1}((CE_c(v_i))) = v_i.$$

VI. Privacy-Preserving Key Search over B$^+$-tree Indexed Data

In this section, we propose the obfuscation and oblivious node access protocols (i.e., steps ② and ③) for private key search on B$^+$-tree indexed single-dimensional data.

A. *Shadow Node Obfuscation*

The purpose for the client to obfuscate a shadow node $E(n)$ to $E(n')$ is two-folded. First, it prevents the server from recognizing the node n when decrypted. Second, it paves the way for the ONA protocol — GT-COT requires the inputs on both sides are not equal.

Let $[l_i, u_i)$ denote the key range of each entry i in the shadow node $E(n)$. To obfuscate l_i and u_i, the client chooses two independent and identically distributed random offsets $\delta_i, \Delta_i \in D_I$, and adds them on both sides of the inequalities $q \geq l_i$ and $q < u_i$:

$$q + \delta_i \equiv q_i.l \quad \geq \quad l_i + \delta_i \equiv l'_i \quad \text{and} \tag{1}$$
$$q + \Delta_i \equiv q_i.u \quad < \quad u_i + \Delta_i \equiv u'_i. \tag{2}$$

To ensure $q_i.l \neq l'_i$ and $q_i.u \neq u'_i$ for subsequent GT-COT calls, the client further doubles both sides of the inequalities and increments the left hand side by 1:

$$2q_i.l + 1 \quad > \quad 2l'_i \quad \text{and} \tag{3}$$
$$2q_i.u + 1 \quad < \quad 2u'_i. \tag{4}$$

Since all variables are in the integer domain, Inequalities 1 and 2 are equivalent to Inequalities 3 and 4, and both sides of the latter two cannot be equal. To further obfuscate the node, the resulted entries are then randomly permutated. The complete node obfuscation procedure is summarized in Alg. 2. Note that the client has the public key of $E(\cdot)$, and is able to perform encryptions as well as additions in ciphertext.

Algorithm 2 Shadow Node Obfuscation

Input: q: the query at the client
$E(n)$: the shadow index node at the client
$E(\cdot)$: the encryption from the server
Output: $E(n')$: the obfuscated shadow node
q': the corresponding obfuscated query
Procedure:

```
1: for each entry $[l_i, u_i)$ of $E(n)$ do
2:     generate random offsets $\delta_i$ and $\Delta_i$
3:     add $\delta_i$ to $q_i.l$ and $E(\delta_i)$ to $l_i$
4:     add $\Delta_i$ to $q_i.u$ and $E(\Delta_i)$ to $u_i$
5:     $q_i = 2q_i + 1$, $l_i = 2l_i$ and $u_i = 2u_i$
6: randomly permutate $q$ and $E(n)$ into $q'$ and $E(n')$
```

B. Oblivious Node Access Protocol

Now we present the oblivious node access protocol for private key search on a B$^+$-tree node. As for the input, the client has the set of obfuscated query keys for a shadow index node in the form of $\langle q_i'.l, q_i'.u\rangle$, and the server has the set of corresponding decrypted entries in the form of $\langle l_i', u_i'\rangle$, $1 \leq i \leq m$, where m is the number of entries in this node. After executing this protocol, the client obtains the only entry i that satisfies $l_i' < q_i'.l$ and $q_i'.u < u_i'$.

A straightforward ONA protocol is as follows (let s_0 and s_1 denote the received values by the client after a GT-COT call). For each entry i, the client and server run GT-COT($q_i'.l, l_i'$) and GT-COT($q_i'.u, u_i'$), respectively. If the client receives s_1 and s_0 as the results, the result entry is found. The disadvantage of this approach is that it invokes GT-COT protocols for $2m$ times.

To reduce GT-COT calls, we design a randomized 2-phase ONA protocol as follows. The client first fills all m entries in an $\sqrt{m}$-by-$\sqrt{m}$ matrix in column-first order (see Fig. 3). If m is not a square root, we use $\lceil\sqrt{m}\rceil$ and pad the last column with dummy entries. Each entry is then indexed by its row and column numbers, such as $e(i,j)$. The protocol calls GT-COT in two phases. In the first phase, the client chooses, from each column, a random pivot entry (marked in black), sends them to the server, and invokes GT-COT. After this phase, the client learns the pivot entry $e(i,j)$, the first entry that satisfies $l_i' > q_i'.l$. In the second phase, the client sends all entries between $e(i,j)$ and the pivot entry prior to it (the latter inclusive, marked in grey) to the server.[5] The sequential order of B$^+$-tree entries ensures that the result must come from one of these entries. Then the client invokes GT-COT again to find out the result, which is the first entry that satisfies $q_i'.u < u_i'$.

As for the cost, the first phase calls GT-COT for $\sqrt{m}$ times, and the second phase for $\sqrt{m}$ times on average and $2\sqrt{m}-1$ times in the worst case (when there are $2\sqrt{m}-1$ grey entries).

[5]If no $e(i,j)$ exists in the first phase, the client sends all entries after the last pivot entry.

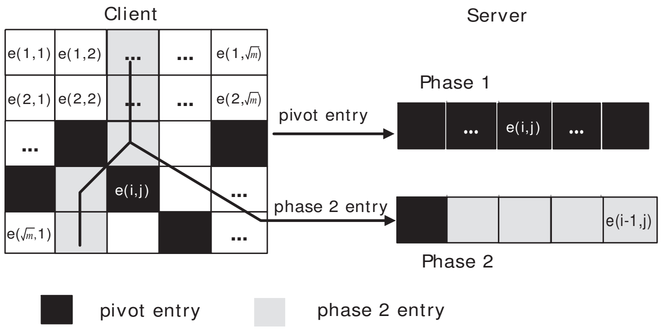


Fig. 3. Randomized 2-Phase ONA

It is noteworthy that the use of random pivot entries prevents the server from narrowing down the position of the result — otherwise if the pivots are fixed, the entries sent in the second phase are fewer than normal when $e(i,j)$ does not exist or is the first pivot entry. Alg. 3 lists the pseudo-code of this protocol.

Algorithm 3 *oblivious_node_access*: Randomized 2-Phase ONA Protocol for Key Search on B$^+$-tree

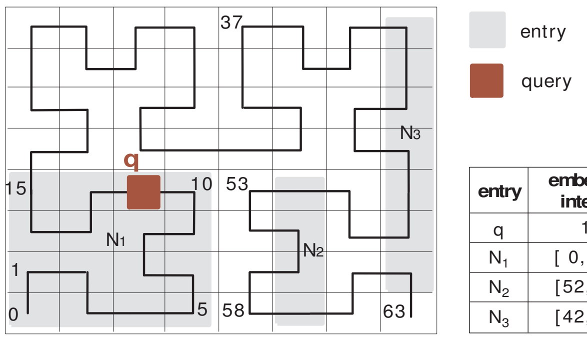
Input: q: the query at the client
$E(n)$: the shadow index node at the client
$E(\cdot)$: the encryption from the server
Output: n: the next node to access
Procedure:

```
1: client randomly chooses $\sqrt{m}$ pivot entries from the matrix of
   $E(n)$ and organizes it into a virtual node $E(n.\sqrt{m})$
2: client obfuscates it to $E(n.\sqrt{m}')$ and sends the latter to server
3: client and server invoke GT-COT($q_i'.l, l_i'$) for $\sqrt{m}$ entries, where
   client learns $e(i,j)$
4: client organizes entries between $e(i,j)$ and the pivot entry prior
   to it into a virtual node $E(n.v)$
5: client obfuscates it to $E(n.v')$ and sends the latter to server
6: client and server invoke GT-COT($q_i'.u, u_i'$) for these entries,
   where client learns $n$
```

VII. Extension to Multi-Dimensional Index

While the same oblivious index traversal framework works for private key search on a multi-dimensional index, such as R-tree, the server should generate different private-public key pairs of encryption $E(\cdot)$ for each dimension in the shadow index. This prevents dimension correlation attacks: even if $E(\cdot)$ in a single dimension is breached (e.g., by background knowledge attacks), the remaining dimensions will not be affected.

The major difference of multi-dimensional index is that the entry values in an R-tree node are no longer sorted as in B$^+$ tree. As such, in the oblivious node access protocol, all entries whose key ranges cover (in all dimensions) the query point should be further accessed. The idea is for the client to adopt the best-first search paradigm, maintain these entries in a priority queue, and always choose the most promising entry to access next (see Section VII-B). Nonetheless, the major challenge is that the matrix-based 2-phase ONA protocol also relies on the linearity of the entry keys, which is no longer available. In the next subsection, we extend the 2-phase ONA protocol with a linear embedding technique.



entry	embedded interval
q	11
N_1	[0, 15]
N_2	[52, 59]
N_3	[42, 62]

Fig. 4. Linear Embedding

A. 3-Phase ONA with Linear Embedding

A linear embedding re-imposes a linear order on the entries so that the original matrix-based protocol can still work. It has the advantage of incurring minimal cost and applicable to arbitrary dimensions — although it is more efficient in lower dimensions such as 2D or 3D space. It works as follows. In the shadow index, besides the key range, each entry e is also embedded with (i.e., mapped to) an interval, denoted by $map(e)$.[6] The mapping scheme is public and known to the client, so the client can obtain $map(q)$ for the query point q and filter out those entries whose $map(e)$ do not cover $map(q)$. To retain as much filtering power as possible, the mapping scheme should preserve most of the locality. There are a lot of mature dimension-reduction mappings, most famous of which are space filling curves. Fig. 4 shows a Hilbert curve of order 3 that partitions the space into 2^3 by 2^3 grid cells. The curve labels each cell with a Hilbert value from 0 to 63. The embedded interval of an entry is the lowest and highest Hilbert values covered by this entry. In the figure, entry N_1 overlaps the lower-left 16 cells, so $map(N_1) = [0, 15]$, while $map(q) = 11$ is covered by this entry.

With this embedding technique in place, the ONA protocol for a multi-dimensional index is extended to a 3-phase one as follows (see Fig. 5 for illustration). Initially, the entries in a shadow index node are sorted in ascending order of the left endpoints of their map intervals and accommodated into the same $\sqrt{m}$-by-$\sqrt{m}$ matrix, in column first order. In the first phase, the client chooses a random pivot entry from each column, sends them to the server, and invokes GT-COT. After this phase, the client learns $e(i,j)$, the first pivot entry that satisfies $map(e).l > map(q)$. In the second phase, the client organizes all entries prior to $e(i,j)$ into a second matrix, in ascending order of the right endpoints of their map intervals.[7] The client then chooses another set of pivots and invokes GT-COT. After this phase, the client learns $e(s,t)$, the first pivot entry that satisfies $map(e).u > map(q)$. In the third phase, the client organizes all entries next to $e(s,t)$ including itself, sends their key ranges, and invokes GT-COT to find all entries that satisfies $q'_i.l < l'_i$ and $q'_i.u < u'_i$.

B. Cost Model for Best-First Search

At any time during the search, the client maintains a priority queue of entries to be further explored, the top of which becomes the next entry n to access. To reduce the

[6]As with the key range, map is encrypted by the server's encryption $E(\cdot)$.
[7]This order should be embedded in the shadow node.

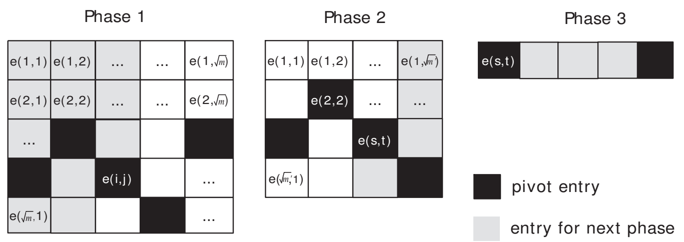


Fig. 5. Randomized 3-Phase ONA Protocol for Key Search on Multi-Dimensional Index

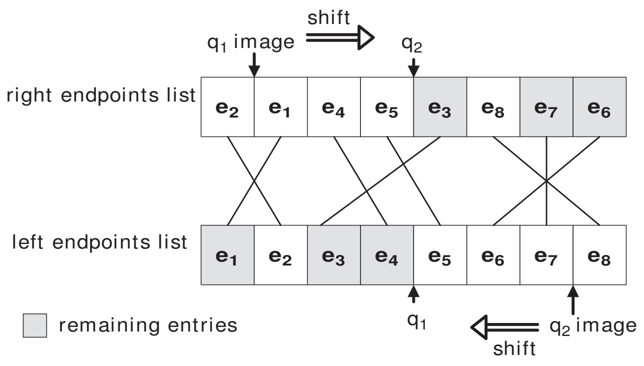


Fig. 6. Estimation on Remaining Entries by Sorted Lists

search cost, the queue should sort the entries by their cost-effectiveness, i.e., the probability of containing the search key divided by the cost to find it. To model the CPU cost, we use the number of modular exponentiations in ONA and OVT for this entry and its subtree. As we assume the keys are distributed uniformly in space after cryptographic hashing, the cost-effectiveness of an entry e can be approximated as

$$cost\_effect(e) \simeq \frac{||keys||}{\frac{1}{2}\sum_{i \in int} ONA(i) + \frac{1}{2}\sum_{i \in leaf} OVT(i)},$$

where $keys$, int, and $leaf$ are the sets of keys, intermediate, and leaf nodes in e, and ONA and OVT are the costs of one protocol invocation. These variables are all accessible to the client from the shadow index except $ONA(i)$, which is dominated by $remain(i)$, the number of entries that are not filtered by the mapping scheme on node i. In what follows, we present a heuristic algorithm for the client to estimate $remain(i)$ based on the two entry lists in shadow node i. Recall that the two lists are sorted in ascending order of their left and right endpoints of map intervals, respectively. Fig. 6 illustrates the two lists, where a line connects the same entry. The algorithm generates two queries q_1 and q_2 at the midway of two lists, and attempts to find their images in the other list. As the keys are uniformly distributed and both queries are in the center of node i, the number of remaining entries in both queries should be (approximately) the same. As such, starting from both ends, this algorithm in turn shifts the images of q_1 and q_2 in opposite directions until their remaining entries are approximately the same. In this figure, after one round of shifting, q_1's image is between e_2 and e_1, and q_2's image is between e_7 and e_8. This leads to $remain(i) = \{e_1, e_3, e_4\}$ for q_1 and $\{e_3, e_7, e_6\}$ for q_2, which terminates the algorithm. It is noteworthy that this estimation algorithm can work offline when the shadow index is first received by the client, so it does not incur any CPU overhead during query processing.

VIII. Security Analysis

In this section, we analyze the security aspect of the proposed framework and protocols. We in turn prove that the building blocks and the oblivious index traversal framework preserve privacy. To demonstrate that a protocol does not leak private information of the client or the server, we adopt the *security proof by simulation* [20]: *A protocol privately computes function $f(\cdot)$ if whatever a semi-honest party can obtain after participating in the protocol, could be essentially obtained from the input and output available to that party.* In simulation terminology, this protocol suffices to "simulate the view" of each semi-honest party, which is formally defined as follows:

Definition 8.1: (Privacy w.r.t semi-honest behavior): Let f: $\{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$ be a functionality where $f_1(x,y)$ (resp. $f_2(x,y)$) denotes the first (resp. second) element of $f(x,y)$, and π be a two-party protocol for computing $f(\cdot)$. The view of the first (resp. second) party during an execution of π on (x,y), denoted by $VIEW_1^{\pi}(x,y)$ (resp. $VIEW_2^{\pi}(x,y)$), is $(x,r,m_1,...,m_t)$ (resp. $(y,r,m_1,...,m_t)$), where r is the outcome of the internal coin tosses, and m_i is the i-th message it has received.

It is noteworthy, however, that a party's view in a protocol (in particular the intermediate messages it receives) sometimes depends on the coin tosses of other parties (e.g., the choice of random numbers). In practice, we only need to simulate the distributions of these random elements in the view, rather than their exact values. Furthermore, sometimes a protocol cannot be simulated without some a-priori knowledge $\mathcal{K}$, besides the party's input and output. In this case, this protocol is no longer completely *private*, and we say it is *private with $\mathcal{K}$ disclosure*.

In the rest of this section, we prove the security of the proposed protocols and framework. As the GT-COT protocol has been proved semantic security in semi-honest model [37], in the following, we first prove the security of the building blocks, i.e., the node obfuscation, OVT and ONA protocols, followed by the entire oblivious index traversal framework.

A. Security Proofs for Building Blocks

Lemma 8.2: The shadow node obfuscation protocol (Algorithm 2) hides the shadow node $E(n)$ from the server S.

Proof: We prove that there is a polynomial-time view simulator $Sim_S(I)$ for server S, which generates a distribution statistically close to what is viewed from S, i.e., n'. Here I is the original index at S. $Sim_S(I)$ works as follows. First, it randomly chooses a node $\hat{n}$ from I. For each entry e_i in $\hat{n}$, it adds random offsets δ_i and $\Delta_i \in D_I$ for lower and upper bounds, respectively. It then doubles both bounds and finally permutates all entries in $\hat{n}$ in a random order . Let $\hat{n}'$ denote the output. It is easy to conclude that $\hat{n}'$ is statistically the same as n', because each value in $\hat{n}'$ or n' is a pure random value in D_I and is independent of all other values. ■

Similarly, we can prove the security of the oblivious value transfer protocol.

Lemma 8.3: The oblivious value transfer protocol hides the shadow leaf node $E(n)$ and query q from the server S, and hides the other contents of n from the client R, except for the returned value v_i.

Proof: We first prove that there is a polynomial-time view simulator $Sim_S(I)$ for server S, which generates a distribution statistically close to what is viewed from S, i.e., $CE_c(q)$, $CE_c(CE_s(e))$ and $CE_c(CE_s(v))$ of all entries. $Sim_S(I)$ works as follows. First, it randomly chooses a leaf node $\hat{n}$ from I and generates a random $\hat{c}$ as the key for commutative encryption $CE_{\hat{c}}(\cdot)$. Due to the semantic security of $CE(\cdot)$, $CE_c(q)$ and $CE_{\hat{c}}(\hat{q})$ are statistically the same, so are $CE_c(e)$ and $CE_{\hat{c}}(\hat{e})$, and $CE_c(v)$ and $CE_{\hat{c}}(\hat{v})$.

We then prove that there is a polynomial-time view simulator $Sim_R(CE_s(n))$ for client R, which generates a distribution statistically close to what is viewed from R, i.e., $CE_c(v_i)$. $SIM_R(CE_s(n))$ works by simply choosing a random $CE_s\hat{v_i}$ from $CE_s(n)$. Due to the semantic security of $CE(\cdot)$, $CE_s(\hat{v_i})$ and $CE_c(v_i)$ are statistically the same. ■

On the other hand, the ONA protocol discloses to the client the GT-COT result between each **non-leaf** entry e and the query q. Specifically, the client has the **boundary knowledge** of whether e is larger or smaller than q. Without this knowledge $\mathcal{K}$, this protocol cannot be simulated. To model it, we define the following boundary knowledge $\mathcal{K}$.

Lemma 8.4: The oblivious node access protocol (Algorithm 3) hides the shadow node $E(n)$ and query q from the server S, and hides the contents of n from the client R, with boundary knowledge $\mathcal{K}$ disclosure.

Proof: The proof has two directions. First, we show there is a polynomial-time view simulator $Sim_S(I)$ for server S, which generates a distribution statistically close to the view of S, i.e., $\{n.\sqrt{m}', n.v'\}$, where $n.\sqrt{m}'$ and $n.v'$ are the obfuscated nodes in the first and second phase, respectively.[8] $Sim_S(I)$ works as follows. In the first phase, it randomly chooses a node $\hat{n}$ from I and maps it into a $\sqrt{m}$-by-$\sqrt{m}$ matrix. It then randomly chooses a pivot entry in each column, denoted by $\hat{e}(i_1,1)$, $\hat{e}(i_2,2)$, $\cdots$. It organizes and obfuscates this virtual node into $\hat{n}.\sqrt{m}'$. Due to the randomness of q and the semantic security of obfuscation, $\hat{n}.\sqrt{m}'$ has the same distribution of values as $n.\sqrt{m}'$. Then in the second phase, it chooses a random column j and organizes entries from $\hat{e}(i_{j-1},j-1)$ to $\hat{e}(i_j,j)$ into another virtual node $\hat{n}.v$. Finally, it obfuscates $\hat{n}.v$ into $\hat{n}.v'$. Due to the randomness of q and the semantic security of obfuscation, $\hat{n}.v'$ has the same distribution of values as $n.v'$.

As for the second direction, we show there is a polynomial-time view simulator $Sim_R(q, E(n), \mathcal{K})$ for the client R, which generates a distribution statistically close to the view of R, i.e., $\{e(i,j), n\}$, where n is the next node to access. First, Sim_R maps the entries in $E(n)$ to a matrix, and then randomly chooses pivot entries in each column. According to the boundary knowledge $\mathcal{K}$, Sim_R can locate the entry $\hat{e}(i,j)$ as the first entry whose lower bound is larger than q. Since both q and the choice of pivot entries are random, $\hat{e}(i,j)$ is statistically the same as $e(i,j)$. Finally, Sim_R can locate $\hat{n}$, the next node to access, as the first entry whose upper bound is larger than q. Obviously, $\hat{n} = n$ since q and $E(n)$ do not change. ■

[8]This proof is for 2-Phase ONA on B$^+$-tree, and the proof for 3-Phase ONA follows and is thus omitted.

B. Security Proof for Oblivious Index Traversal Framework

Now we proceed to prove the security of the oblivious index traversal framework. According to the composition theorem [20], we only need to prove the above building blocks are invoked in a secure manner, which is equivalent to having an oracle read the inputs from both parties and send the outputs to them.

Theorem 8.5: The oblivious index traversal framework (Algorithm 1) securely finds the value to the key q, by hiding q from the server S, and hiding the contents of the original index I from the client R with boundary knowledge $\mathcal{K}$ disclosure.

Proof: In each round of node access, the server S receives an obfuscated node n' from the client R. By Lemma 8.2, the actual accessed node n is indistinguishable from other nodes in I, thus hiding q from S. Then R and S invoke the ONA protocol, which is proved to keep hiding q from S by Lemma 8.4. The iterative rounds follow the same protection of q and eventually when the value is retrieved, the OVT protocol is proved to keep hiding q from S by Lemma 8.3. Therefore, the framework hides q from S.

On the other hand, Lemma 8.4 shows that in each round of node access, the framework hides the contents of I from R with boundary knowledge $\mathcal{K}$ disclosure. The OVT protocol also hides the other contents of n from the client R, except for the returned value v_i. Since v_i is the value for a leaf entry, the boundary knowledge $\mathcal{K}$ disclosure is preserved. ■

Theorem 8.5 proves the framework is secure with $\mathcal{K}$ disclosure. In what follows, we show that such disclosure does not compromise the security model defined in Section III. Recall that in this model, the genuine keys or values are indistinguishable by the client (except for the keys or values it has searched), and the genuine search key is indistinguishable by the server. Note that the following proof assumes that the keys are uniformly distributed in the key domain, as achieved by the preprocessing of cryptographic key hashing.

Corollary 8.6: A probabilistic adversary $\mathcal{A}$ who eavesdrops the entire communication of a protocol run of the oblivious index traversal framework achieves a success probability $Pr[v = genuine]$ bounded by $\frac{1}{2}+negl$, and $Pr[q = genuine]$ bounded by $\frac{1}{2} + negl$, as in Definition 3.1.

Proof: Let $\cup_j q_j$ denote the sorted search keys by the client. By Theorem 8.5, $\mathcal{A}$ only knows $\mathcal{K}$ of non-leaf entries. That is, $\mathcal{A}$ only knows $e(i) < q_j \leq e(i+1)$, $i \in [1, m-1]$, for any accessed non-leaf node n. For a new key $v \notin \cup_j q_j$ from the key domain, what $\mathcal{A}$ can learn from $\mathcal{K}$ is that there exists some j where $q_j < v < q_{j+1}$. The smallest granularity this information can narrow down v to occur when q_j and q_{j+1} belongs to a leaf node n', which imposes v on n', too. However, since all keys in n' are uniformly distributed as all other leaf nodes, knowing $v \in n'$ does not add more to probability $Pr[v = genuine]$ than a random guess, which is bounded by $\frac{1}{2}+negl$. The proof of $Pr[q = genuine]$ bounded by $\frac{1}{2} + negl$ follows. ■

IX. Performance Optimizations

In this section, we propose two optimization techniques. These techniques are orthogonal to the above processing techniques and can thus be used altogether. Note that they both come with security tradeoffs, which are discussed below.

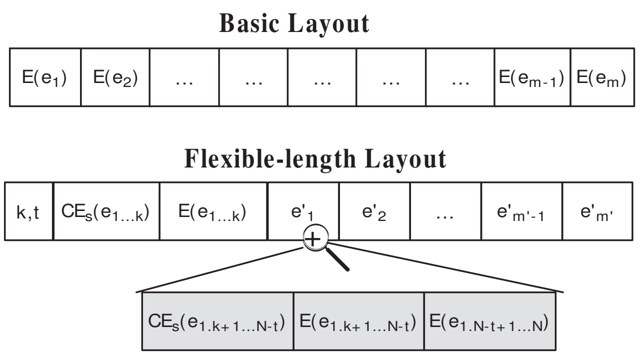


Fig. 7. Shadow Index Node Layout of Keys for Flexible-Length GT-COT

A. Flexible-Length GT-COT

A key observation of this optimization is that the GT-COT protocol compares q and an N-bit key in a bit-wise manner. When this algorithm is invoked by the ONA protocol, the keys to be compared come from the same index node — they share the most significant bits and meanwhile still distinguish each other without the least significant bits. Let k and t denote the numbers of these bits for this node, respectively. Then the ONA protocol can invoke a flexible-length GT-COT protocol instead, whose algorithm is the same as the general GT-COT, but only takes bits $k+1 \cdots N-t$ for comparison.

To support flexible-length GT-COT, in each shadow index node, the k shared most significant bits of all entry keys are encrypted separately, and so are the t least significant bits in each entry key. Fig. 7 compares the flexible-length node layout of keys with the basic one. $e_{1\ldots k}$ is the k shared bits and e'_i are the remaining bits of entry e_i, which are further partitioned into bits $k+1 \cdots N-t$ and bits $N-t+1 \cdots N$. Given this, before invoking the GT-COT for all or a set of entries in a node as in 2-Phase or 3-Phase ONA, some preprocessing steps are added to the protocol to prepare the bits to compare in the GT-COT. First, it tests the equality between the shared k most significant bits and those of q, using $CE_s(e_{1\ldots k})$.[9] If they are not equal, then the bits to compare are from $1 \cdots k$, and q is either larger or smaller than all entries; if they are equal, the remaining $N-k$ bits are the bits to compare. To further narrow down these bits, the protocol runs one more equality test between q and all entries of bits $k+1 \cdots N-t$. If there is no match, the bits to compare are reduced to $k+1 \cdots N-t$; otherwise, some entry e_j matches q, and the bits to compare are $N-t+1 \cdots N$ between q and e_j.

The major security concern of this optimization is that the client learns the two equality test results. If the first test result is "equal", the client learns the first k bits of keys in this node; if the second test is also "equal", the client further learns the next $N-k-t$ bits of some entry. However, by properly

[9]This equality test is similar to the oblivious value transfer (OVT) protocol and works as follows. The client encrypts the k most significant bits of q into $CE_c(q_{1\ldots k})$ and sends it to the server; the client also encrypts $CE_s(e_{1\ldots k})$ into $CE_c(CE_s(e_{1\ldots k}))$. The server further encrypts $CE_c(q_{1\ldots k})$ into $CE_s(CE_e(q_{1\ldots k}))$ and sends it back. The client then tests if $CE_s(CE_e(q_{1\ldots k})) = CE_c(CE_s(e_{1\ldots k}))$.

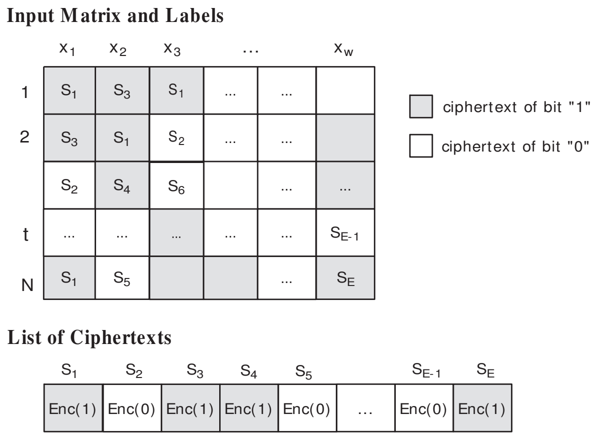


Fig. 8. Randomized Shared Encryption

setting k and t according to the security profile of the system, such security impact can be well controlled. For example, if the security profile requires the current node and each of its children to occupy a portion $1/m$ and $1/r$ of the entire key domain, then setting $k = \log m$ and $t = N - \log r$ will disclose no more information about the node to the client.

B. Shared Encryption

Algorithm 4 *share_enc*: Randomized Shared Encryption

```
Input:      $x_i$: the private values in GT-COT
            $\lambda$: the independency threshold
            $Enc(\cdot)$: the encryption scheme
Output:     $x_i(k)$: labels of ciphertexts
            $s_i$: the list of shared ciphertexts
Procedure:
1: share all bits with a single $Enc(1)$ and $Enc(0)$
2: for each $1 \leq i < w$ do
3:     for each $0 \leq j < i$ do
4:         while $\max|x_i - x_j| < \lambda$ do
5:             randomly selects $k \in [1, N]$ s.t. $T(x_i(k), x_j(k)) = 0$
6:             filps $x_j(k)$ to a new ciphertext
```

This optimization reduces the communication cost in the batch GT-COT calls from the ONA protocol. We observe that: 1) in a GT-COT call, the receiver (i.e., the client) sends the ciphertext of every bit (either 0 or 1) of its private input x and, 2) GT-COT is batch invoked in an ONA protocol, where the client has a set of private inputs x_1, x_2, ..., x_w (for example, l'_i and u'_i). If we allow some "0" bits of x_i to share the same ciphertext $Enc(0)$ and "1" bits to share the same ciphertext $Enc(1)$, the communication cost can be greatly reduced. The consequence, however, is that these private inputs are no longer perfectly independent, thus disclosing their correlations to the server. To model the loss of independency, let $\max|x_i - x_j|$ denote the maximum difference induced by the encryptions on x_i and x_j. It can be calculated as follows:

$$\max|x_i - x_j| = \sum_{k=1}^{N} 2^{N-k} T(x_i(k), x_j(k)),$$

where $T(s,t)$ takes 1 if s and t are both shared ciphertexts, and 0 otherwise. As a special case, $x_0 = 0$ is the origin, so $\max|x_i - x_0|$ is the maximum induced difference on x_i itself. Obviously, without any sharing, $\max|x_i, x_j| = 2^N - 1$ for any $i \neq j$, reaching perfect independency. So the loss of independency by shared encryption can be captured by λ, the minimum of $\max|x_i - x_j|$ between any pair of inputs,

$$\lambda = \min_{i,j} \max|x_i - x_j|, \forall 0 \leq i, j \leq w, i \neq j.$$

In essence, λ is the extra deviation that any two private values can have in the worst case when the server is able to decrypt all shared ciphertexts. Obviously, the larger the λ, the fewer ciphertexts can be shared. In what follows, we propose for the client a randomized shared encryption algorithm that achieves maximal sharing, given a λ threshold. Fig. 8 illustrates the procedure. As for the input, we organize all bits to be encrypted into a matrix, each column corresponding to a private input x_i. The output is a list of ciphertexts (s_1, s_2, ...) and their labels on each matrix element. Initially, all ciphertexts share a single $Enc(1)$ and $Enc(0)$, randomly denoted by s_1 and s_2. Then from left to right we scan each x_i and randomly flip a shared ciphertext to a new non-shared one until x_i no longer violates the λ threshold with any x_j, $0 \leq j < i$. For example, in Fig. 8, when x_1 is scanned, it violates the λ threshold with x_0, so the client chooses $x_1(2)$ and flips it from s_1 to a new ciphertext s_3. As this process continues, all remaining ciphertexts s_4, s_5, s_6, ... in the list are created. Algorithm 4 shows the pseudo-code of this algorithm. Note that as the scan guarantees all $\max|x_i - x_j|$ would increase monotonically, each x_i only needs to be scanned once.

X. Performance Analysis

In this section, we analyze the performance of our proposed framework, protocols and optimization techniques under various datasets and parameter settings. The 1D dataset is a dictionary of 100,000 English words; and the 2D dataset consists of 100,000 points from a real spatial database of 123,593 postal addresses in New York, Philadelphia and Boston [39]. The keys in both datasets are hashed by MD5 before being indexed by the B$^+$-tree and R-tree. As such, each individual key is 128 bits and a value is set to be the same 128-bit length. We set a page size to 4KB, so the resulted fanouts of a B$^+$ tree and R-tree shadow index node are 110 and 50, respectively. The shadow indexes take 9.2 and 25.2 seconds to construct, and their sizes are 3.1 MB and 24.5 MB, respectively. Table I summarizes the parameter settings used in the experiments.

We use two machines with the same configuration for the client and server. Each features Intel Xeon X5650 @ 2.67GHz, 8GB RAM and runs CentOS 5.6 x64. The code is implemented and executed in OpenJDK 1.6. Paillier and Pohlig-Hellman encryptions are used for the shadow indexes and the OVT protocol, all with 1024-bit keys unless otherwise stated. As for the GT-COT protocol, we adopt the same security settings for Paillier encryption as in [40]. The metrics to measure are the server CPU time, client CPU time, and the communication bandwidth. For each measurement, we execute 1,000 queries and report their average costs.

A. B$^+$-tree Search Performance

Fig. 9 shows the performance of successful and unsuccessful search with respect to various dataset sizes. Our oblivious index traversal framework enables a private key search with

Parameter	Sym.	Default	Range
# of B$^+$-tree records	N	100,000	1,000-100,000
# of R-tree records	N	100,000	1,000-100,000
page size	–	4KB	–
key bits	–	128	–
value bits	–	128	–
independency threshold	λ	2^{32}	2^{16}-2^{128}
order of Hilbert curve	—	32	–

TABLE I. PARAMETER SETTINGS

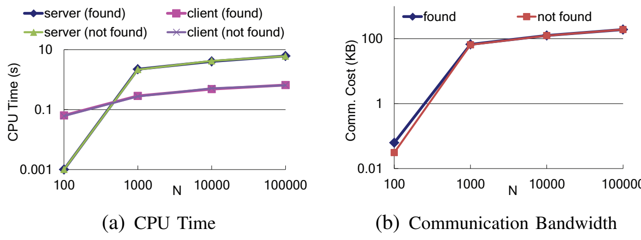


Fig. 9. B$^+$-tree Key Search Performance

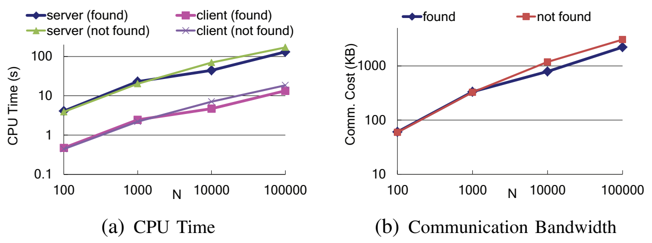


Fig. 10. R-tree Key Search Performance

Dataset (N) \ Strategy	Cost Est. (base)	FIFO	FILO
1,000	1	0.68	1.2
10,000	1	1.44	1.41
100,000	1	1.47	1.43

TABLE II. CLIENT CPU TIME VS. STRATEGIES IN BEST FIRST SEARCH

up to 6 seconds on the server and up to only 0.6 second on the client, whereas applying GT-COT directly on such a large number of keys is infeasible. Furthermore, both the CPU and communication bandwidth increase very slowly as the number of keys increases, thanks to the slow increase of the B^+-tree depth. It is noteworthy that the performance of successful and unsuccessful search is consistent. This is an essential requirement for cryptographic algorithms, which guarantees that no party (or eavesdropper) is able to infer the result from one's CPU or communication bandwidth.

B. R-tree Search Performance

We vary the size of the 2D dataset and plot the performance of successful and unsuccessful search in Fig. 10. We observe that the costs are more significant than those in B$^+$-tree, which is attributed to the loss of linear order in the R-tree nodes. Nonetheless, the increase of costs is much flatter than that of N, partially because of the linear embedding technique that filters the entries whose map intervals do not cover the query. Another contributing technique is the best-first search strategy based on cost-effectiveness estimation. Table II shows the client CPU time when applying a simple first-in-first-out or first-in-last-out strategy, in terms of a multiple of that when applying the estimation. Thanks to both techniques, our framework takes up to about 130 seconds on the server and up to only 13 seconds on the client. The CPU time can be further reduced by standard multithreading as our ONA protocol is highly parallelizable when it boils down to GT-COT calls. The communication bandwidth is up to 2-3 MB, which is also moderate. It is noteworthy that there is some performance discrepancy between successful and unsuccessful search, as the latter has to exhaust all entries in the priority queue. To remedy this without delaying responses to the user, the client can invoke dummy ONA protocols after the result is found. In the sequel, we only show the results of successful search in the interest of space.

C. Effect of Optimization Techniques

In this subsection, we evaluate the performance of the two optimization techniques, namely, flexible-length GT-COT and shared encryption. In flexible-length GT-COT, we allow each node to maximize the bits to share, i.e., there is no constraint on k or t; in shared encryption, we set the independency threshold λ to 2^{32}. Since they come with different security trade-offs, their performance cannot be directly compared. Figs. 11(a) and 11(b) show the performance of all combinations for B$^+$-tree and R-tree, respectively, under the default system settings. Both figures consistently show that: 1) flexible length GT-COT greatly reduces the CPU time of both parties by up to 90%; 2) shared encryption reduces the client CPU time and communication bandwidth by up to 50% and 40%, respectively; 3) by applying both optimizations, the communication bandwidth is about 40% of the original.

To further evaluate the performance of shared encryption, we vary the independency threshold λ from 2^{16} to 2^{128} and plot the CPU and communication bandwidth in Fig. 12. As λ increases, the costs of client CPU and bandwidth increase because fewer "0" and "1" ciphertexts can be shared; meanwhile the server CPU time drops a little, as this technique comes with a small overhead at the server to "decompress" the shared encryption. To conclude, in both B$^+$-tree and R-tree, the choice of λ leads to a steady and robust performance gain in terms of the client CPU time and bandwidth.

XI. CONCLUSIONS

In this paper, we have studied the problem of private key search on hierarchically indexed key-value stores for mutual privacy protection. We have presented an oblivious index traversal framework, based on which secure protocols

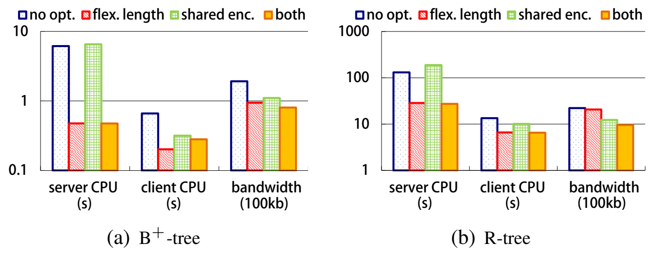
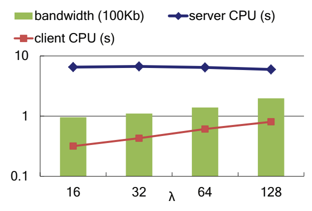
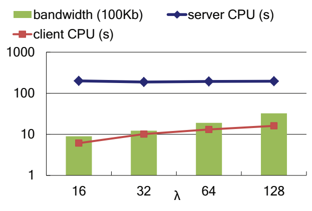


Fig. 11. Performance of Optimization Techniques (N=100,000)



(a) B$^+$-tree



(b) R-tree

Fig. 12. Performance of Shared Encryption vs. λ (N=100,000)

have been devised for private key search on single- and multi-dimensional key spaces. Through theoretical proofs and performance evaluation, this approach has been shown to be secure and efficient under various parameter settings and security threats. We believe this work steps towards practical applications of SMC protocols to large-scale, structured datasets. As for future work, we plan to extend the key search query to complex queries such as selection, semi-join and top-k queries. This calls for not only a new schema design of shadow index nodes but also more sophisticated oblivious node access protocols that address the query semantics.

Acknowledgment

This work was supported by Research Grants Council, Hong Kong SAR, China, under projects HKBU210811, 210612, 211512, 210510, FRG2/12-13/042, and FRG2/11-12/074, and by The Research Innovation Program of Shanghai Municipal Education Commission under grant No. 13ZZ003.

References

[1] "Amazon simpledb (beta)," http://aws.amazon.com/simpledb/.

[2] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 571–588, 2002.

[3] G. Ghinita, P. Kalnis, and S. Skiadopoulos, "Prive: Anonymous location-based queries in distributed mobile systems," in *WWW*, 2007.

[4] C. Chow, M. Mokbel, and W. Aref, "Casper*: Query processing for location services without compromising privacy," *ACM TODS*, vol. 34, no. 4, 2009.

[5] H. Hu and J. Xu, "2pass: Bandwidth-optimized location cloaking for anonymous location-based services," *IEEE TPDS*, vol. 21, no. 10, 2010.

[6] J. Xu, X. Tang, H. Hu, and J. Du, "Privacy-conscious location-based queries in mobile environments," *IEEE TPDS*, vol. 21, no. 3, 2010.

[7] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," in *Proc. SIGMOD*, 2004.

[8] H. Wang and L. V. S. Lakshmanan, "Efficient secure query evaluation over encrypted xml databases," in *Proc. VLDB*, 2006.

[9] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *Proc. SSTD*, 2007.

[10] M. L. Yiu and P. K. G. Ghinita, C. S. Jensen, "Outsourcing search services on private spatial data," in *Proc. ICDE*, 2009.

[11] W. K. Wong, W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proc. SIGMOD*, 2009.

[12] M. Kuzu, M. Islam, and M. Kantarcioglu, "Efcient similarity search over encrypted data," in *Proc. ICDE*, 2012.

[13] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Proceedings of ICDE*, 2013.

[14] Y. Qi and M. J. Atallah, "Efficient privacy-preserving k-nearest neighbor search," in *Proc. ICDCS*, 2008.

[15] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: Anonymizers are not necessary," in *Proc. of SIGMOD*, 2008.

[16] I. F. Blake and V. Kolesnikov, "Strong conditional oblivious transfer and computing on intervals," in *Proceedings of ASIACRYPT*, 2004.

[17] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu, "Secure multidimensional range queries over outsourced data," *The VLDB Journal*, vol. 21, no. 3, 2012.

[18] K. Liu, C. Giannella, and H. Kargupta, "An attacker's view of distance preserving maps for privacy preserving data mining," in *PKDD*, 2006.

[19] A. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, 1982.

[20] O. Goldreich, *The Foundations of Cryptography – Volume 2*. Cambridge University Press, 2004.

[21] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, "Tools for privacy preserving distributed data mining," *ACM SIGKDD Explorations*, vol. 4, no. 2, pp. 28–34, 2002.

[22] J. Vaidya and C. Clifton, "Privacy preserving association rule mining in vertically partitioned data," in *Proc. ACM SIGKDD*, 2002.

[23] W. Du and Z. Zhan, "Building decision tree classifier on private data," in *Proceedings of the IEEE international conference on Privacy, security and data mining*, 2002.

[24] W. Du and M. Atallah, "Privacy-preserving cooperative statistical analysis," in *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001.

[25] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," in *Proc. ACM SIGKDD*, 2003.

[26] ——, "Privacy-preserving top-k queries," in *Proc. ICDE*, 2005.

[27] Y. Li and M. Chen, "Privacy preserving joins," in *Proc. ICDE*, 2008.

[28] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proc. SIGKDD*, 2005.

[29] H. Hu, J. Xu, Q. Chen, and Z. Yang, "Authenticating location-based services without compromising location privacy," in *Proc. SIGMOD*, 2012, pp. 301–312.

[30] Q. Chen, H. Hu, and J. Xu, "Authenticating top-k queries in location-based services with confidentiality," in *Proc. of the VLDB Endowment, Vol. 7, No. 1*, 2014, pp. 49–60.

[31] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *Proc. ICDE*, 2011.

[32] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. CRYPTO*, 2007.

[33] S. Vimercati, S. Foresti, S. Paraboschi, and P. S. G. Pelosi, "Efficient and private access to outsourced data," in *Proc. ICDCS*, 2011.

[34] H. Liu, H. Wang, and Y. Chen, "Ensuring data storage security against frequency-based attacks in wireless networks," in *Proc. DCOSS*, 2010.

[35] J. Katz and Y. Lindell, *Introduction to modern cryptography*. Chapman&Hall/CRC Press, 2008.

[36] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of EUROCRYPT*, 1999.

[37] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional oblivious transfer and timed-release encryption," in *Proc. EUROCRYPT*, 1999.

[38] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over gf(p) and its cryptographic significance," *IEEE Trans. Information and System Security*, vol. 24, no. 1, 1978.

[39] "Postal addresse in north east metropolitan areas (new york, philadelphia and boston)," http://www.chorochronos.org/?q=node/60.

[40] A. Rivera, P. Gowda, N. Vidyadhar, and M. Hall, "Accelerating the greater than-strong conditional oblivious transfer multiparty protocol using graphics processing units," in *GPU Technology Conference*, 2012.