sciendo

Proceedings on Privacy Enhancing Technologies ; 2021 (4):549–574

Alexandra Boldyreva and Tianxin Tang*

Privacy-Preserving Approximate k-Nearest-Neighbors Search that Hides Access, Query and Volume Patterns

Abstract: We study the problem of privacy-preserving approximate kNN search in an outsourced environment — the client sends the encrypted data to an untrusted server and later can perform secure approximate kNN search and updates. We design a security model and propose a generic construction based on locality-sensitive hashing, symmetric encryption, and an oblivious map. The construction provides very strong security guarantees, not only hiding the information about the data, but also the access, query, and volume patterns. We implement, evaluate efficiency, and compare the performance of two concrete schemes based on an oblivious AVL tree and an oblivious BSkiplist.

Keywords: oblivious data structures, locality-sensitive hashing, approximate kNN, search on encrypted data.

DOI 10.2478/popets-2021-0084
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

Background and Motivation. The k-nearest-neighbor (kNN) search problem is defined as follows: given a dataset D of points in a metric space S and a query point $q \in S$, find the top k nearest neighbors to q. kNN is used in computer vision and serves as a basic yet very effective machine learning classification technique, where it is commonly used to classify a point based on the consensus of its neighbors. The numerous applications include content-based image search, automated face recognition in video surveillance, stock market predictions, medical diagnoses, risk assessment, and credit card fraud detection.

While the existing kNN algorithms are not efficient for dense high-dimensional data used by many applications, one approach to deal with kNN's inefficiency is to use a different problem, known as the approximate kNN. It asks to return points whose distance from the query is no more than $(1+\epsilon)$ times the distance of the true k-th nearest-neighbor, and this can be done only with high probability. Approximate kNN (AkNN) is appropriate for many applications that can sacrifice some precision for efficiency.

Nowadays, it is extremely common to outsource data storage, management, and search functionality to a cloud. For the kNN or AkNN search this means that the client's dataset D is outsourced to a cloud server that answers the client's kNN (AkNN) queries. If the data is sensitive and the server is untrusted or subject to compromise, it is desirable for the server to be able to perform the search on encrypted data without knowing the secret key of the client and learning anything about the client's data and queries.

Our Goal. We (quite ambitiously) aim to build a practical searchable encryption scheme for approximate kNN queries that provides very strong security: in addition to provably hiding information about the data and the queries, we also aim to hide the access, query equality and volume patterns, as such information, if leaked, can give rise to attacks. We want our protocol to yield working solutions for privacy-preserving data classification and search by classifiers (labels). We want to avoid relying on multiple non-colluding servers or secure hardware, as these are hard to ensure in practice.

Prior Work and its Limitations. Searchable encryption (SE) is a hot topic in cryptography that seeks protocols permitting efficient search on outsourced encrypted data. While most solutions focus on the basic query types such as exact match and range queries, in recent years, there is a rise of interest in SE schemes for kNN queries, e.g., [5, 9, 23, 29, 30]. However, there are still no solutions for kNN or AkNN search providing strong security guarantees. Even when papers provide formal security notions and proofs of security for their schemes, still the problem is that any practical SE incurs security/efficiency/functionality tradeoffs, and there is no acceptable balance reached for SE for kNN. The cipher-

Alexandra Boldyreva: Georgia Institute of Technology, USA, E-mail: sasha@gatech.edu
***Corresponding Author: Tianxin Tang:** Georgia Institute of Technology, USA, E-mail: ttang@gatech.edu

texts of proposed efficient solutions leak distance and/or closeness pattern between data points for functionality and efficiency.

Recent works by Kornaropoulos et al. [18, 19] presented efficient attacks on SE for kNN queries with such seemingly minimal leakage. More precisely, they show how to attack SE schemes only leaking which k encrypted records are retrieved for a kNN query. While their attacks are for one-dimensional data, the authors argued how the attack also applies to high-dimensional data. The problem with SE that permits such attacks is the access pattern leakage (revealing which encrypted points are queried). An even more recent work by Grubbs et al. [13] observes the relationship between the problem of reconstructing encrypted databases from access pattern leakage and statistical learning theory. This observation allowed the authors to mount effective nearly-optimal approximate database reconstruction attacks for richer queries such as range, prefix and suffix type queries, and could apply to kNN queries. These attack results make a very strong case for targeting SE schemes that hide the access pattern. There are works [9, 25] that target public-key SE for kNN that hides the access pattern, but they require the dataset to be split among two non-colluding cloud servers.

We note that practical SE that hides the access pattern is still a big open problem even for basic query types such as exact-match. A known theoretical approach to ensure access pattern security for SE schemes is to utilize the Oblivious RAM (ORAM) tool. However, this approach is not very efficient since ORAM incurs heavy computational and communication costs. Moreover, as was shown by Naveed in [22], using ORAM to hide the access pattern in SE schemes for exact match is more expensive (bandwidth-wise) than the straightforward solution where the server sends the whole encrypted database to the client as a query response.

Kamara et al. [17] show how to compile an SE (or encrypted search index) for document identifiers, such as the classical symmetric SE scheme by Curtmola et al. [6], in order to suppress leakage of the query equality pattern, namely, of which queries are equal. Their solution could be combined with the ORAM-based stage for retrieving the documents for given identifiers, so that no access pattern is leaked. The main problem with this approach is efficiency: there are no implementations of the compiler and efficient ones seem currently out of reach, plus the aforementioned negative result of Naveed [22] implies that the solution cannot be reasonable communication-wise. Moreover, the solution only provides a static scheme without the possibility of updates.

The Oblix system by Mishra et al. [21] presents a dynamic encrypted search index that hides the access pattern, but it relies on secure hardware such as Intel's SGX.

To make the situation more challenging, Grubbs et al. [12] present attacks that apply even to schemes hiding the access pattern and use only the volume of responses to reconstruct databases. Their attacks are for the case of range queries, but there is no good reason to assume they cannot be extended to the case of other queries.

Our Approach. Our first observation is that the negative results of [22] consider the worst case when a keyword in an exact-match query matches almost all documents in the database. However, for the case of AkNN queries, we can stop after k matches are found. This fact makes the use of ORAM-like solutions promising for our application.

To construct a secure AkNN protocol, we first study the existing approaches to solve AkNN for non-sensitive unencrypted data. Since the naive approach of comparing the pairwise distances between the query and data points is not feasible for large databases, it is common to use locality-sensitive hashes (LSH). LSH encode partial information of locality: the closer the data, the more likely that their hashes collide. Usually, an extended LSH (eLSH) is used. Each eLSH application yields multiple outputs (tags). If at least one tag overlaps between the hashes of the query and the data, the points are close with high probability.

It is known how to use eLSH to solve a decisional approximate near neighbor problem [14], and how a solution to this problem can be used to solve AkNN [11]. We can build on these works, but we design our own eLSH parameter selection since prior works suggest setting up the eLSH parameters empirically, but this approach does not guarantee either the correctness or the upper-bound on the size of returned results. We, on the other hand, aim at having a protocol for which we can assess concrete correctness. Moreover, not having a bound on the returned results prevents us from efficiently using oblivious data structures.

Now, to add security, we employ *oblivious data structures* (ODS) [20, 28]. ODS is an abstraction of data structures that support oblivious access: one can retrieve and update the data, while the data, physical addresses of the components of the data structure, and the type of operation stay hidden. (One can view ORAM

as ODS of an encrypted array.) We will utilize oblivious maps to enable secure search on eLSH tags.

Wang et al.'s ODS framework [28] was inspired by Gentry et al.'s approach in optimizing the tree-based ORAMs for binary search [10]. Their ODS framework utilizes tree-based ORAM (e.g., Path ORAM [27]) as a building block and supports any tree structure with bounded degrees. Using trees with bounded degrees is important for efficiency and this is a good match for us since we select parameters so that the number of returned results for each query is bounded. One could not use the same approach for regular keyword search.

We note that instead of ODS one could use searchable encryption for disjunctive exact-match keyword search, where eLSH tags would serve as keywords. However, there is no immediate choice of SE for disjunctive keyword search because there are no efficient solutions that hide the access (or query or volume) pattern. Encrypted multi-maps [16] can also be used to implement response-hiding secure keyword search, hiding the volume pattern while revealing the query-equality pattern. As we mentioned, the approach to suppress the query-equality pattern using structured encryption with leakage suppression [17] is neither efficient nor dynamic.

Our contributions. We first define the syntax for a *privacy-preserving AkNN* (PP-AkNN) protocol, its correctness and security. Both definitions are inspired by the general definitions for structured encryption with leakage suppression by Kamara et al. [17], but we simplify and customize them for our specific AkNN query type. The security definition is simulation-based. It asks that the protocol could be simulated without the data or secret key and just from the outputs of the leakage function, if any. This captures the intuition that no information other than some possible leakage is revealed by the protocol. As common for the ODS protocols including ORAM, we only treat honest-but-curious adversaries and leave treatments of active adversaries for future work.

Next, we present our generic construction that combines the aforementioned eLSH-based algorithm to solve AkNN, an oblivious map with encryption (OMapE) and the parameters selection that enables concrete correctness assessment. (Unlike most cryptographic schemes, assessing the correctness of our protocol is not straightforward.)

We define a new adaptive security definition for OMapE, addressing the previously omitted data privacy in oblivious map definition. We prove that our generic protocol meets our security definition assuming the building block OMapE is adaptively secure. The only leakage our generic protocol incurs in addition to the leakage of the specific OMapE instantiation (specified below) is the (constant) numbers of data accesses for search and update queries. Despite this minimal leakage, which is independent of the data, our PP-AkNN scheme hides the query-equality pattern for both search and update queries, the volume pattern (since each query yields a constant number of data accesses), and the access pattern. It also hides the operation type of the update query — whether it is an add or a remove operation.

It remains to specify particular OMapE constructions one can rely on. We combine a standard IND-CPA symmetric encryption scheme and an oblivious map candidate, either an AVL tree [28] or a BSkiplist [24]. AVL tree is a self-balancing tree that directly yields an efficient map structure; BSkiplist is a B-tree variant of standard skiplist; both can be used to construct an efficient map.

Both candidates we use are built using a non-recursive position-based ORAM, such as PathORAM. Assuming "adaptive obliviousness" and data privacy held by the underlying ORAM with encrypted blocks, the total leakage of the OMapE comprises the number of nodes (blocks) in the tree, the node (block) size, the bucket size, the tree height, and the branching factor — for AVL tree-based instantiation, the branching factor is 2.

With respect to PP-AkNN, these translate to very small leakage independent of the data content: the number of items stored in the database and public parameters used to initialize the structure.

We are interested in comparing the performance of the two to see whether they are feasible for practice. The authors of [28] initially provided no implementations, but recently they made the oblivious AVL tree implementation available. For BSkiplist, [24] provided implementation results in the paper, but their implementation was not public. Hence, we implemented both schemes, incorporated the eLSH part to complete the implementation of our PP-AkNN protocol, and evaluated the two's performance in searching and updating on multiple benchmark data sets to test their efficiency and scalability. We found that BSkiplist-based implementation performs better in terms of the number of roundtrips incurred compared with the oblivious AVL tree-based construction. Also, compared with the baseline — downloading the whole encrypted database, the BSkiplist-based scheme supports a much more effi-

cient secure AkNN search solution, especially for a small client storage. More details are provided in Section 7.

Applications. Our privacy-preserving protocol PP-AkNN can be adopted in all AkNN applicable settings where k is small and the database is sufficiently large. For instance, consider performing AkNN-based classification on images. Given a ground-truth set — images and associated feature vectors classified into different classes (e.g., by scenes) and attached with the class labels, encrypt and store feature vectors and labels on an untrusted server. On each query image and its feature vector(s), we can first retrieve through PP-AkNN the AkNN of the query feature vector and associated class labels. Then perform AkNN-based classification — assigning the query image to the majority class voted by the returned AkNN.

2 Preliminaries

Notation and Conventions. For some $n \in \mathbb{N}$, we let $[n]$ denote the discrete range $[1, n]$, and let $x[i]$ denote the i-th element for some vector, or ordered set, i.e. list x. We use $(x_1, x_2, \ldots, x_n)$ to denote a vector of elements x_i for all $i \in [n]$. For a set T we write $|T|$ for the size of T. The algorithms are randomized and polynomial-time (in the security parameter) unless otherwise specified. Given any security parameter $\lambda \in \mathbb{N}$, any $x \in \mathbb{R}$, we let $\langle\, x \,\rangle$ denote x in the format of a binary string of length polynomial in the security parameter λ. For the syntax of any interactive protocol (algorithm) $\mathcal{I}$ executed between party A and party B, we use the convention: $(\mathsf{output}_A, \mathsf{output}_B) \leftarrow [\mathcal{I}_A(\mathsf{input}_A), \mathcal{I}_B(\mathsf{input}_B)]$.

Metric Spaces. We adopt the definition from [3]. $(\mathcal{D}, d)$ is a *metric space* if $\mathcal{D}$ is a set and d (the *metric*) is a real-valued function on $\mathcal{D} \times \mathcal{D}$ such that for all $x, y, z \in \mathcal{D}$ the distance function d satisfies the conditions as follows,

$$d(x, y) \geq 0 \qquad d(x, y) = 0 \text{ iff } x = y$$
$$d(x, y) = d(y, x) \qquad d(x, z) \leq d(x, y) + d(y, z).$$

Ball. Over some metric space $(\mathcal{D}, d)$, we define the ball $\mathcal{B}$ by, for any message $Q \in \mathcal{D}$, any radius (distance threshold) $r \in \mathbb{R}$, $\mathcal{B}(Q, r) = \{M : M \in \mathcal{D}, d(M, Q) \leq r\}$.

approximate k-NN. We use the following definition for AkNN throughout the work.

Definition 2.1 ((ϵ, δ)-AkNN)**.** *For metric space $(\mathcal{D}, d)$, database $\mathsf{DB} \subseteq \mathcal{D}$, integer $k \geq 1$, failure probability $\delta > 0$, construct a data structure so that on every query $Q \in \mathcal{D}$, with at least $1 - \delta$ probability, it efficiently returns k points $\{P_i\}_{i \in [k]}$ in the database such that for all $i \in [k]$, $d(Q, P_i) \leq (1 + \epsilon) \cdot d(Q, \mathsf{DB})_i$, where $d(Q, \mathsf{DB})_i$ is the distance from Q to its ith-nearest neighbor in DB and $\epsilon > 0$ is the error factor.*

Locality-sensitive Hashing. Our constructions utilize locality-sensitive hashing (LSH), especially its extension form eLSH, so we start with recalling the LSH primitive introduced in [15]. Below, we give definitions for an arbitrary metric space $(\mathcal{D}, d)$.

Definition 2.2 (Locality-sensitive Hashing)**.** *A family $\mathcal{H}$ is called (r, cr, p_1, p_2)-sensitive if for any two points $x, y \in \mathcal{D}$ [26]:*

- *If $d(x, y) \leq r$ then $\Pr_{\mathcal{H}}[h(x) = h(y)] \geq p_1$;*
- *If $d(x, y) \geq cr$ then $\Pr_{\mathcal{H}}[h(x) = h(y)] \leq p_2$.*

We now recall the definition of extended LSH. Given (r, cr, p_1, p_2)-sensitive $\mathcal{H}$, one can think of (l, s)-eLSH extension of $\mathcal{H}$ as a locality sensitive hashing function with improved sensitivity (r, cr, p_1', p_2'), where $p_1' \geq p_1$ and $p_2' \leq p_2$, formally defined as follows.

Definition 2.3 (eLSH)**.** *Let $\mathcal{H}$ be a (r, cr, p_1, p_2)-sensitive hash family. For positive integers s, l, choose random $h_{i,j} \in \mathcal{H}$ for all $i \in [l]$, all $j \in [s]$ and define the hash functions $g_i(\cdot)$ by*

$$g_i(x) = (h_{i,1}(x), h_{i,2}(x), \ldots, h_{i,s}(x)) \qquad \text{for all } i \in [l].$$

We refer to the set of functions $\{g_i\}_{i \in [l]}$ as the (l, s)-eLSH extension of $\mathcal{H}$:

- *If $d(x, y) \leq r$ then $\Pr\{$ there exists $i \in [l]$ such that $g_i(x) = g_i(y)\} \geq p_1'$, where $p_1' \geq p_1$;*
- *If $d(x, y) \geq cr$ then $\Pr\{$ there exists $i \in [l]$ such that $g_i(x) = g_i(y)\} \leq p_2'$, where $p_2' \leq p_2$.*

3 Secure AkNN Search Protocol

We start with defining syntax and correctness for a *privacy-preserving approximate* kNN (PP-AkNN) protocol primitive.

Over metric space $(\mathcal{D}, d)$, associated domain $\mathcal{D}_{\mathrm{A}}$, let message space $\mathcal{MS}$ be $\mathcal{D} \times \mathcal{D}_{\mathrm{A}}$. A PP-AkNN protocol between two parties: client C and server S, is defined by two algorithms Setup, Dec and two two-party protocols Search, Update as follows:

The client C first runs the following algorithm:

- $(\mathsf{k_{max}}, K, \mathsf{EDS}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \epsilon, \delta, Z, \mathbf{M})$: is a (possibly) randomized algorithm. It takes as input security parameter 1^λ, error factor $\epsilon > 0$, probability of failure $\delta \in [0,1]$, auxiliary information $Z \in \{0,1\}^*$, a set of messages $\mathbf{M} \subseteq \mathcal{MS}$, and outputs a public parameter $\mathsf{k_{max}} \in \mathbb{N}$, a secret key K, and an encrypted data structure EDS.

The following interactive algorithms are between parties C and S:

- $(\mathbf{M}', \mathsf{EDS}') \leftarrow [\mathsf{Search}_C(K, Q_\mathcal{D}, k), \mathsf{Search}_S(\mathsf{EDS})]$: C inputs secret key K, a query message $Q_\mathcal{D} \in \mathcal{D}$, parameter $k \in [\mathsf{k_{max}}]$, and server S inputs EDS. At the end, C gets a set $\mathbf{M}' \subseteq \mathcal{MS}$; S updates the encrypted data structure to EDS'.
- $(\perp, \mathsf{EDS}') \leftarrow [\mathsf{Update}_C(K, U, M^*), \mathsf{Update}_S(\mathsf{EDS})]$: C inputs secret key K, an update operation type $U \in \{\mathsf{add}, \mathsf{remove}\}$, message $M^* \in \mathcal{MS}$, and server inputs EDS; S updates the encrypted data structure to EDS'.

The client C can also run the following decryption algorithm:

- $\mathbf{M} \leftarrow \mathsf{Dec}(K, \mathsf{EDS})$: is a deterministic algorithm. It takes as input secret key K, encrypted data structure EDS, and outputs a set of messages $\mathbf{M}$.

Correctness. For all security parameter $\lambda \in \mathbb{N}$, all $Z \in \{0,1\}^*$, all $\epsilon > 0$, all $\delta \in [0,1]$, all $(\mathsf{k_{max}}, K)$ generated by $\mathsf{KeyGen}(1^\lambda, \epsilon, \delta, Z)$, all $\mathbf{M} \subseteq \mathcal{MS}$, all EDS generated by $\mathsf{Setup}\ (1^\lambda, \epsilon, \delta, Z, \mathbf{M})$, $\mathsf{Dec}(K, \mathsf{EDS}) = \mathbf{M}$:

- Search is (ϵ, δ)-correct if for all query message $Q_\mathcal{D} \in \mathcal{D}$, all $k \in [\mathsf{k_{max}}]$, all set $\mathbf{M}'$ client C receives from running $[\mathsf{Search}_C(K, Q_\mathcal{D}, k), \mathsf{Search}_S(\mathsf{EDS})]$, size of set $\mathbf{M}'$ equals to k, and it holds with at least $(1-\delta)$ probability that set $\mathbf{M}'$ contains $Q_\mathcal{D}$'s approximate i-th nearest neighbor in $\mathbf{M}$ with error factor ϵ (Definition 2.1), for all $i \in [k]$; $\mathsf{Dec}(K, \mathsf{EDS}') = \mathbf{M}$ with no error.
- Update is correct if for all update operation type $U \in \{\mathsf{add}, \mathsf{remove}\}$, all message $M^* \in \mathcal{MS}$, all EDS' updated by the server S from running $[\mathsf{Update}_C(K, U, M^*), \mathsf{Update}_S(\mathsf{EDS})]$, the following conditions hold:
  - If U is add, $\mathsf{Dec}(K, \mathsf{EDS}') = \mathbf{M} \cup \{M^*\}$;
  - If U is remove, $\mathsf{Dec}(K, \mathsf{EDS}') = \mathbf{M} \setminus \{M^*\}$.

Efficiency. We require that the communication bandwidth induced in both Search and Update protocol is $O(\mathsf{polylog}(N))$, where N is number of items in the database, and the computational complexity for Search and Update is $O(\mathsf{polylog}(N))$.

Remark 1. Note that in typical AkNN applications, k is a variable, and can be set by the client in each search query, and hence our scheme also supports different $k \in [\mathsf{k_{max}}]$. We allow the message space to contain the auxiliary data to accommodate common applications. For example, for an image classification application, $\mathcal{D}$ will contain vector(s) extracted from images (e.g, a deep-neural-network feature), d will be the Euclidean distance, and $\mathcal{D}_\mathrm{A}$ will contain class labels. For each image, client first extracts a feature vector (e.g, using a deep neural network), retrieves approximate k nearest neighbors for the feature, and performs kNN-based classification — assigning the image to the labeled class voted by majority of the k returned labels. In some applications, update operations are not important, but we consider dynamic datasets for completeness. Similarly, it may not be necessary to demand decryptability of the encrypted data structure. We consider decryptability because clients may need to extract the data stored on the cloud. Moreover, having decryptability simplifies defining correctness.

Remark 2. Our syntax above is inspired by that for structured encryption with leakage suppression defined by Kamara et al. [17]. Their definition is very general and captures structured and searchable encryption, ORAM and oblivious data structure. We do not need such generality as we target specific AkNN queries only and hence prefer a simpler syntax. In particular, we do not need the Rebuild protocol they have in order to capture suppressing the leakage of existing structured encryption schemes (STE).

4 Security Definition

We present the security definition for PP-AkNN protocols. It is simulation-based and asks that for an adversary who knows the dataset and the encrypted database and can adaptively make queries, the transcript of the protocol can be simulated by a simulator who only has access to some leakage information, which the security statement for each construction has to specify. The definition captures the intuition that the protocol hides all information about the data and queries besides the

specified leakage. For some protocols, leakage may include access, query and/or volume patterns, but for our construction we show that such leakage is absent.

Our definition is similar to that by Kamara et al. [16], though, as typical for ODS and ORAM-like primitives, we only consider passive (honest-but-curious) attackers and leave treatment of active attackers to future works. In applications where tampering with the software is not likely, considering passive attackers may be sufficient.

We provide the formal definition and follow with more discussion.

Definition 4.1 (Adaptive Security for PP-AkNN)**.** *Let* Π *be a* PP-AkNN *protocol. Let* $\mathcal{L}_{\mathsf{PP\text{-}AkNN}} = \{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}}\}$ *be the leakage profile describing leakages of* Π*'s algorithms.*

$\underline{\mathsf{SetupH}_\Pi(1^\lambda, \epsilon, \delta, Z, \mathbf{M}) :}$
($\overline{\mathsf{k_{max}}}, \overline{K}, \overline{\mathsf{EDS}}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \epsilon, \delta, Z, \mathbf{M})$.
Return ($\overline{\mathsf{k_{max}}}, \overline{K}, \overline{\mathsf{EDS}}$).

$\underline{\mathsf{SearchH}_\Pi(Q_t, k_t, \overline{K}, \overline{\mathsf{EDS}}_t) :}$
Run honest execution $(\overline{\mathbf{M}'_t}, \overline{\mathsf{EDS}}_{t+1}) \leftarrow$
$[\mathsf{Search}_C(\overline{K}, Q_t, k_t), \mathsf{Search}_B(\overline{\mathsf{EDS}}_t)]$.
Return $(\overline{\mathbf{M}'_t}, \overline{\mathsf{EDS}}_{t+1})$.

$\underline{\mathsf{UpdateH}_\Pi(U_t, M^*_t, \overline{K}, \overline{\mathsf{EDS}}_t) :}$
Run honest execution $(\perp, \overline{\mathsf{EDS}}_{t+1}) \leftarrow$
$[\mathsf{Update}_C(\overline{K}, U_t, M^*_t), \mathsf{Update}_B(\overline{\mathsf{EDS}}_t)]$
Return $\overline{\mathsf{EDS}}_{t+1}$.

Fig. 1. Helper Functions for Defining PP-AkNN Security.

Consider the probabilistic experiments $\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda)$ *and* $\mathbf{Ideal}_{\Pi,\mathcal{A},\mathcal{S}}(1^\lambda)$ *defined in Figure 2 with helper functions defined in Figure 1, associated with a stateful adversary* $\mathcal{A}$ *and simulator* S*.*

We say that Π *is adaptively* $\mathcal{L}_{\mathsf{PP\text{-}AkNN}}$ *secure if there exists a PPT simulator* $\mathcal{S}$ *such that for all (non-uniform) PPT adversaries* $\mathcal{A}$*, all* $Z \in \{0,1\}^*$*, all* $\epsilon > 0$*, all* $\delta \in [0,1]$*, the following is negligible (in* λ*):*

$$\left|\Pr\left[\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathbf{Ideal}_{\Pi,\mathcal{A},\mathcal{S}}(1^\lambda) = 1\right]\right|.$$

$\underline{\mathbf{Real}_{\Pi,\mathcal{A}}(1^\lambda) :}$
Given $1^\lambda, \epsilon, \delta, Z$, adversary $\mathcal{A}$ outputs a message set $\mathbf{M}_1 \subseteq \mathcal{MS}$.
$(\mathsf{k_{max}}, K, \mathsf{EDS}_1) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, \epsilon, \delta, Z, \mathbf{M}_1)$.
$\mathcal{A}$ is given $\mathsf{k_{max}}$ and EDS_1.
$\mathcal{A}$ adaptively makes q queries in any order, for all $t \in [q]$:
  $\mathsf{Search}(Q_t, k_t)$, where $Q_t \in \mathcal{D}$, $k_t \in [\mathsf{k_{max}}]$
    returns to $\mathcal{A}$ the transcript of honest execution $(\mathbf{M}'_t, \mathsf{EDS}_{t+1}) \leftarrow$
    $[\mathsf{Search}_C(K, Q_t, k_t), \mathsf{Search}_B(\mathsf{EDS}_t)]$,
    C's output $\mathbf{M}'_t$ and B's output EDS_{t+1}.
  $\mathsf{Update}(U_t, M^*_t)$, where $U_t \in \{\mathsf{add}, \mathsf{remove}\}$,
    $M^*_t \in \mathcal{MS}$, returns to $\mathcal{A}$ the transcript of honest execution $(\perp, \mathsf{EDS}_{t+1}) \leftarrow$
    $[\mathsf{Update}_C(K, U_t, M^*_t), \mathsf{Update}_B(\mathsf{EDS}_t)]$
    and B's output EDS_{t+1}.
Finally, $\mathcal{A}$ outputs bit b.
The experiment returns the same bit b.

$\underline{\mathbf{Ideal}_{\Pi,\mathcal{A},\mathcal{S}}(1^\lambda) :}$
Given $1^\lambda, \epsilon, \delta, Z$, adversary $\mathcal{A}$ outputs a message set $\mathbf{M}_1 \subseteq \mathcal{MS}$.
$(\overline{\mathsf{k_{max}}}, \overline{K}, \overline{\mathsf{EDS}}_1) \xleftarrow{\$} \mathsf{SetupH}(1^\lambda, \epsilon, \delta, Z, \mathbf{M}_1)$.
$\mathcal{A}$ is given $\overline{\mathsf{k_{max}}}$.
Given $1^\lambda, \overline{\mathsf{k_{max}}}, \mathcal{L}_{\mathsf{Setup}}(\mathbf{M}_1)$, simulator $\mathcal{S}$ outputs encrypted structure EDS_1 and sends it to $\mathcal{A}$.
Let B be an honest server and given EDS_1.
$\mathcal{A}$ adaptively makes q queries in any order, for all $t \in [q]$:
  $\mathsf{Search}(Q_t, k_t)$, where $Q_t \in \mathcal{D}$, $k_t \in [\overline{\mathsf{k_{max}}}]$:
    $(\overline{\mathbf{M}'_t}, \overline{\mathsf{EDS}}_{t+1}) \leftarrow \mathsf{SearchH}(Q_t, k_t, \overline{K}, \overline{\mathsf{EDS}}_t)$.
    Return to $\mathcal{A}$ the transcript of
    $[\mathcal{S}(\mathcal{L}_{\mathsf{Search}}(\mathbf{M}_t, Q_t, k_t)), \mathsf{Search}_B(\mathsf{EDS}_t)]$,
    SearchH's output $\overline{\mathbf{M}'_t}$,
    and B's output EDS_{t+1}.
    $\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t$.
  $\mathsf{Update}(U_t, M^*_t)$, where $U_t \in \{\mathsf{add}, \mathsf{remove}\}$, $M^*_t \in \mathcal{MS}$:
    $(\overline{\mathbf{M}'_t}, \overline{\mathsf{EDS}}_{t+1}) \leftarrow \mathsf{UpdateH}(U_t, M^*_t, \overline{K}, \overline{\mathsf{EDS}}_t)$.
    Return to $\mathcal{A}$ the transcript of
    $[\mathcal{S}(\mathcal{L}_{\mathsf{Update}}(\mathbf{M}_t, U_t, M^*_t)), \mathsf{Update}_B(\mathsf{EDS}_t)]$
    and B's output EDS_{t+1}.
    If $U_t = \mathsf{add}$ then
      $\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t \cup \{M^*_t\}$;
    Else if $U_t = \mathsf{remove}$ then
      $\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t \setminus \{M^*_t\}$.
Finally, $\mathcal{A}$ outputs bit b.
The experiment returns the same bit b.

Fig. 2. Experiments for Defining PP-AkNN Security.

Remark 3. Our security definition follows the common approach: whatever the attacker can learn from the real protocol's transcript should be simulatable in the ideal world without the secrets, data and the queries. However, there is one important issue we had to take care of. In the searchable/structured encryption literature, the distinguisher between real and ideal worlds is not given the output of the client. This is because the works usually deal with deterministic functionalities where the client's output is determined be the inputs and the transcript. But we deal with the randomized functionality of approximate search. It is known from the MPC literature, that it is necessary to give the distinguisher access to the client's output in this case. And in the ideal world this output is computed using the ideal functionality.

However, we are facing a problem the MPC literature does not address. Namely, our ideal functionality, which is not explicitly defined, but follows from the protocol's correctness requirement, does not specify the exact probabilities of the data appearing or not appearing in the client's output. Instead, it only specifies bounds for such probabilities. Hence, in the ideal world we cannot easily compute the client's output without it being trivially distinguished from that in the real world. We resolve this issue by computing the output by running the protocol. To the best of our knowledge, this problem has not been treated before in the searchable-encryption literature. Note that even though we run the protocol in the ideal world, this is just to ensure correctness, and does not affect security or cause a tautology, as the simulator is not involved.

Remark 4. In our security definition, the leakage profile is abstract. Depending on the protocol, leakage may include identity and/or equality patterns for responses and queries. We refer to [17] for the formal definitions of such patterns. Access/query patterns usually mean the union of the corresponding equality and identity patterns. In ORAM-related community, access pattern is defined differently as the sequence of physical memory addresses accessed. Volume pattern is information about the relation between the queries and the amount of communication and rounds of interactions. We do not provide formal definitions since for our construction it will be immediate that such patterns are not leaked.

5 Solving Approximate k-Nearest Neighbors Problem

Before we present our protocol we need to study known approaches to solving the (un-secured) Approximate k-Nearest Neighbors Problem (AkNN) problem (Definition 2.1).

A straightforward approach is storing the data in an array, then searching the approximate k-NN by comparing the pairwise distances of data points, clearly not feasible for large databases. It is possible to improve efficiency and construct a data structure by partitioning the whole metric space, such as k-d tree, but this approach has severe scalability issues when dealing with high-dimensional data as the size of space grows exponentially in the number of dimensions.

Therefore, a randomized approach is used, usually equipped with an approximation algorithm that allows some error in correctness, but greatly improves the efficiency. First, one uses extended LSH (eLSH Definition 2.3) [7] to solve a decisional approximate near neighbor problem [14]. Next, a solution to this problem is used to solve AkNN, possibly by solving the approximate nearest neighbor 1-NN first.

We note that practitioners sometimes prefer using an eLSH look-up map directly to tackle the AkNN problem. They examine all the points that share overlapped eLSH tags with the query point, specifically, by selecting eLSH parameters by training a subset of the data and choose parameters with reasonable accuracy and a smaller cost (size of matched points divided by the size of data set).

This approach, though, does not provide a guarantee on either the correctness or the upper-bound on the size of returned results. We are interested in solving the AkNN problem without relying heavily on the training procedure — estimating intrinsic parameters of the data distribution. In particular, we focus on providing a theoretical guarantee on setting the parameters to achieve concrete correctness of AkNN with an upper-bound on the returned results. Moreover, not having a bound on the returned results prevents us from efficiently using oblivious data structures.

So we turn back to discuss the aforementioned steps in solving AkNN problem in more detail. But first we recall some related definitions. Given any query point $Q \in \mathcal{D}$, for any data point $P \in \mathcal{D}$, we say P is a "r-

near neighbor" of Q if and only if $P \in \mathcal{B}(Q, r)$[1]. We now formally define the 1-NN problem.

Definition 5.1 ((c, r)-NN)**.** *Fix a set $\mathbf{M} \subseteq \mathcal{D}$, and closeness thresholds $r > 0$, failure probability $\delta > 0$, construct a data structure such that, given any query point $Q \in \mathcal{D}$, with probability at least $1 - \delta$: if $M \cap \mathcal{B}(Q, r)$ is not empty, it reports some cr-near neighbor of Q and outputs "YES" in $\mathbf{M}$ where c is the approximation factor and if $M \cap \mathcal{B}(Q, r)$ is empty, it outputs "NO".*

To solve this problem, one uses an LSH-based algorithm introduced in [7], which we will refer to as eLSH-cRNN. It works as follows. Given a set of locality-sensitive hash families, set eLSH parameters s, ℓ based on approximation factor c, closeness threshold r, and probability of failure δ (E.1, E.2). To ensure the cr-near neighbors of the query point share at least one overlapped eLSH tag. Construct (l, s)-eLSH instances I times, each with fresh independent randomness, which yields (l, s)-eLSH$_i$. Then initialize an empty look-up map; for each (l_i, s_i)-eLSH instance, where $i \in [I]$; for each point P in the data set, compute its eLSH tags; for each eLSH tag T, add (T, P) pair to the map. We may only use a single map by pre-appending $\langle\, i \,\rangle$ to each eLSH tag to indicate that the (l_i, s_i)-eLSH instance is used.

During the search stage, given query point Q, for all $i \in [I]$, if any point share with at least one eLSH tag with the query point, then add the point to a set $\mathbf{S}$. At the end, if $\mathbf{S}$ is not empty, report a point in $\mathbf{S}$ that is a cR-NN of Q, and output "YES"; otherwise output "NO". It is proved that such an algorithm indeed solves the (c, r)-NN problem in [7]. For simplicity, we may refer "eLSH look-up map" to the map used in eLSH-cRNN in later discussion.

Now, as the next step, the approximate 1-NN problem can be reduced to a sequence of (c, r)-NN problems by utilizing eLSH maps with varying closeness thresholds and approximation factors. Usually, theoreticians focus on using as few (c, r)-NN problems as possible [7, 14], even at the cost of increasing the complexity of the data structure — more complicated than a look-up map. And then it is not hard to solve AkNN, though one would need to prove correctness.

Looking ahead, dealing with encrypted data excludes such an option, mainly due to practical issues or security concerns. Also, we focus on finding approximate k nearest neighbors instead of a single approximate nearest neighbor discussed in the prior works. This rules out the possibility of reusing their correctness results.

Therefore, we turn to a simpler but less optimized reduction idea of solving AkNN using (c, r)-NN, proposed without a proof of correctness in [11]. The reduction works as follows: Suppose there are multiple eLSH maps solving a sequence of (c_i, r_i)-NN problems, $i \in [size_r]$ for some parameter $size_r$, with (c_i, r_i) crafted carefully, r_i in increasing order. Start searching from the smallest closeness threshold $r_{\min}$ by returning all the data points (with some upper-bound) sharing at least one eLSH tag overlaps with the query point. If not enough points are found, then next eLSH map of a larger r_{i+1} will be examined in the $(i + 1)$-th iteration.

6 Our PP-AkNN Construction

6.1 Overview

We build a generic PP-AkNN protocol Π using the following key ingredients: eLSH-based algorithm (eLSH-cRNN), utilized to solve the approximate near-neighbor problem, solving AkNN using the approximate near-neighbor problem, and securing AkNN using an oblivious map with encryption (OMapE). We described the first two steps in Section 5 (however, we will still need to provide extra work to enable proofs of correctness).

To add security to the AkNN solution we need a secure data structure with efficient search functionality to store eLSH tags. Because standard SSE schemes leak access and query pattern (aka response equality pattern and query equality pattern, defined in [17]), and also the volume pattern, this potentially may incur substantial security damage, as discussed in Section 1. Therefore we turn to an oblivious map [28] to prevent such leakage.

An oblivious map protocol allows the client to perform a sequence of operations on the oblivious map located on the server, so that the server will know neither the operation types nor the underlying node ids (logical addresses) the operations access. Privacy of data can be achieved by using encryption.

In our construction we use an oblivious map instantiated with an oblivious tree structure of bounded degree. The bounded degree requirement is to ensure $O(\log N)$ communicational blow-up for efficiency (overall bandwidth is $O(\log^2 N)$). A tree structure of bounded degree can be interpreted as a sequence of nodes, where each node contains a node id (logical address) and node

1 Cf. the ball definition in Section 2.

data. The oblivious map can be represented as a sequence of encrypted nodes. For the feasibility of such approach, it is crucial to observe that while solving approximate k-NN search using a sequence of eLSH-cRNN algorithms with eLSH look-up maps, we can bound the number of items associated with each eLSH tag in k instead of the size of the database, and store them in a single node of the oblivious tree implementing the map. This allows us to avoid potential blowup in storage and bandwidth, and thus have a practical solution.

6.2 Building Block: OMapE

Oblivious map with Encryption. We adopt the standard *map* definition: Let $\mathbb{L}$ be a label (keyword) space $\mathbb{V}$ be a value space. A *map* is a set of label (keyword) and value pairs, denoted by $\mathsf{map} = \{(\ell_i, v_i)\}_{i\in\mathbb{N}}$, where $\ell_i \in \mathbb{L}$, $v_i \in \mathbb{V}$. Both maps and oblivious maps support standard operations including Find, Insert, and Delete, specified in the following definition $\tilde{\mathcal{F}}_{\mathsf{OMap}}$ (Definition 6.1). Let $\mathsf{map}[\ell_i] = v_i$ denote the map look-up on label ℓ_i, where v_i is ℓ_i's associated value. We use the latest definition of *oblivious simulation of reactive functionality* as in [1]. The *reactive functionality* is a functionality that keeps an internal state between the executions, defined similarly to $\mathcal{F}_{\mathsf{ORAM}}$ provided in [1] except that we make the state explicit and output by $\tilde{\mathcal{F}}_{\mathsf{OMap}}$. A similar obliviousness definition is used in [24, 28] for their oblivious map constructions. However, both works used a non-reactive definition of the functionality, fall short of capturing the adaptiveness in their obliviousness definition — adversary chooses the input to f adaptively, based on previous returned function output.

Definition 6.1 ($\tilde{\mathcal{F}}_{\mathsf{OMap}}$)**.** *The functionality is reactive, and holds a state — N memory blocks, each of (expected) size w denoted by $\mathbf{X}[1,\dots N]$ storing* $\mathsf{map} \subseteq \mathbb{L}\times\mathbb{V}$.

- $\mathsf{Access}(\mathsf{map}, \mathsf{op}, \ell, v)$*: where* $\mathsf{op} \in \{\mathsf{Find}, \mathsf{Insert}, \mathsf{Delete}\}$*,* $\ell \in \mathbb{L}$*, and* $v \in \mathbb{V}$.
  1. *If* $\mathsf{op} = \mathsf{Find}$*, if* ℓ *is in* map *then* $v^* \leftarrow \mathsf{map}[\ell]$*; otherwise* $v^* \leftarrow \perp$.
  2. *If* $\mathsf{op} = \mathsf{Insert}$*,* $\mathsf{map}[\ell] \leftarrow v$ *and* $v^* \leftarrow v$.
  3. *If* $\mathsf{op} = \mathsf{Delete}$*,* $\mathsf{map}[\ell] \leftarrow \perp$ *and* $v^* \leftarrow \perp$.
  4. *Output* (v^*, map).

Remark 5. For Find, Delete operation, input v is set by $\perp$. We use $\tilde{\mathcal{F}}_{\mathsf{OMap}}(\mathsf{map}, \mathsf{op}, \ell, v)$ as a shorthand notation for $\mathsf{Access}(\mathsf{map}, \mathsf{op}, \ell, v)$ defined in $\tilde{\mathcal{F}}_{\mathsf{OMap}}$.

Moreover, [1] and ODS framework [28] suggest providing data privacy using symmetric encryption (e.g., blockcipher-based mode of operation), while omitting the discussion. Similarly, [24] mentions using AES-based encryption scheme encrypting the blocks stored in the underlying ORAM. We elaborate on the data privacy aspect, providing a unified syntax OMapE combining the oblivious map with standard symmetric encryption, and a security definition in the style of structured encryption capturing adaptive obliviousness and data privacy.

For security parameter $\lambda \in \mathbb{N}$, any map $\mathsf{DS} \subseteq \mathbb{L}\times\mathbb{V}$ implemented using a tree structure of bounded degree, an OMapE protocol between client C and server S, is defined by three algorithms and one two-party protocol as follows:

- $K \xleftarrow{\$} \mathsf{OMapE.KeyGen}(1^\lambda)$: is a randomized algorithm run by C that takes as input a security parameter λ, and outputs a secret key K.
- $\mathsf{EDS} \xleftarrow{\$} \mathsf{OMapE.Setup}(1^\lambda, Z, K, \mathsf{DS})$: is a randomized algorithm run by C that takes as input a security parameter λ, auxiliary information $Z \in \{0,1\}^*$ (e.g., upper-bound N on the number of labels in the map), secret key K, map DS and outputs an encrypted data structure EDS.
- $(v^*, \mathsf{EDS}^*) \leftarrow [\mathsf{OMapE.Access}_C(K, \mathsf{op}, \ell, v), \mathsf{OMapE.Access}_S(\mathsf{EDS})]$: is a two-party protocol executed between client C and server S, where C inputs a secret key K, an operation $\mathsf{op} \in \{\mathsf{Find}, \mathsf{Insert}, \mathsf{Delete}\}$, $\ell \in \mathbb{L}, v \in \mathbb{V}$ and server inputs EDS; at the end, C receives $v^* \in \mathbb{V}$; S updates the encrypted data structure to EDS^*.
- $\mathsf{DS} \leftarrow \mathsf{OMapE.Dec}(K, \mathsf{EDS})$: is a deterministic algorithm that takes as input secret key K, encrypted data structure EDS and outputs DS.

The security experiment in Figure 3 with ideal functionality $\tilde{\mathcal{F}}_{\mathsf{OMap}}$ (Definition 6.1) also captures the correctness requirement of OMapE. We define the adaptive security for OMapE with leakage profiles in the STE style, except we specify the correctness and security in a single definition for compactness. Note that we cannot have a compact definition similarly for PP-AkNN since the protocol does not have perfect correctness. Also, we consider transcripts can be simulated in our definition instead of the sequence of physical addresses Addrs [1]. Our security definition is stronger than ODS's, as it addresses both the adaptive obliviousness and data privacy. Moreover, it is stronger than STE's if all leakage profiles are empty since the map operation type is hidden by our definition. In contrast, STE divides the ac-

cess protocol into *query* and *update* algorithms and thus leaks the operation type by syntax.

Definition 6.2 (Adaptive Security for OMapE)**.** *Let Φ be an* OMapE *protocol. Let $\mathcal{L}_{\mathsf{OMapE}} = (\mathcal{L}_{\mathsf{OMapE.Setup}}, \mathcal{L}_{\mathsf{OMapE.Access}})$ be the leakage profile describing leakages of Φ's algorithms. Let $\lambda \in \mathbb{N}$ be the security parameter. Consider the probabilistic experiments defined in Figure 3. We say that Φ is adaptively $\mathcal{L}_{\mathsf{OMapE}}$-secure if there exists a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$, all $Z \in \{0,1\}^*$, the following is negligible (in λ):*

$$\left|\Pr\left[\mathbf{Real}_{\Phi,\mathcal{A}}(1^\lambda) = 1\right] - \Pr\left[\mathbf{Ideal}_{\Phi,\mathcal{A},\mathcal{S}}(1^\lambda) = 1\right]\right|.$$

6.3 Generic PP-AkNN Construction

We now describe our protocol, providing an outline of each algorithm, and clarify certain parts via bullet points.

Over some metric space $(\mathcal{D}, d)$, associated domain $\mathcal{D}_\mathrm{A}$, let the message space be $\mathcal{MS} = \mathcal{D} \times \mathcal{D}_\mathrm{A}$. Let $\mathbf{M} \subseteq \mathcal{MS}$ and $\mathbf{Q}_\mathcal{D} \subseteq \mathcal{D}$, where $\mathbf{M} = \{(\mathbf{M}_\mathcal{D}[i], \mathbf{M}_{\mathcal{D}_\mathrm{A}}[i])\}_{i\in[n]}$ and $|\mathbf{Q}_\mathcal{D}| = t$. Fix a security parameter $\lambda \in \mathbb{N}$, an error parameter $\epsilon > 0$, and a parameter for probability of failure $\delta \in [0,1]$. Let $\Phi = (\mathsf{KeyGen}, \mathsf{Setup}, \mathsf{Access}, \mathsf{Dec})$ be an OMapE protocol. Given $\mathbf{M}_\mathcal{D}, \mathbf{Q}_\mathcal{D} \subseteq \mathcal{D}$, $|\mathbf{M}_\mathcal{D}| = n$, $|\mathbf{Q}_\mathcal{D}| = t$, $\alpha \in [0,1]$, $\mathsf{k}_{\max} \ll n$, run $Z \xleftarrow{\$} \mathsf{genAuxInfo}(\alpha, \mathbf{M}_\mathcal{D}, \mathbf{Q}_\mathcal{D}, \mathsf{k}_{\max})$, where genAuxInfo is the algorithm outputting the protocol parameters $Z = \mathsf{k}_{\max}\|n\|t\|\alpha\|r_{\min}\|r_{\max}$. In practice, such parameters can be selected empirically, and we refer to Section 7 for more details. But for our formal protocol specification we need to choose the parameters that enable us to formally evaluate correctness. We specify such algorithm genAuxInfo in Section 6.4.

Algorithm $\mathsf{Setup}(1^\lambda, \epsilon, \delta, Z, \mathbf{M})$ (part 1)

1: $K[1] \xleftarrow{\$} \Phi.\mathsf{KeyGen}(1^\lambda)$
2: map $\leftarrow$ empty map $\{\}$
3: Parse Z as $\mathsf{k}_{\max}\|n\|t\|\alpha\|r_{\max}\|r_{\min}$
4: $I \leftarrow \lceil \log_{1/e+1/3} \frac{\xi}{nt} \rceil$,
5: with $0 < \xi \leq \frac{\delta}{2\cdot\mathsf{k}_{\max}\cdot\lceil\log_{1+\epsilon}\frac{r_{\max}}{r_{\min}}\rceil}$
6: $c \leftarrow 1+\epsilon$, $\gamma \leftarrow \sqrt{c}$
7: $size_r \leftarrow \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil$
8: $K[2] \leftarrow n\|t\|r_{\min}\|r_{\max}\|size_r\|I$
9: numAccess $\leftarrow 0$

$\mathbf{Real}_{\Phi,\mathcal{A}}(1^\lambda)$:
Given $1^\lambda, Z$, adversary $\mathcal{A}$ outputs a map DS_1 over $\mathbb{L} \times \mathbb{V}$.
$K \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$.
$\mathsf{EDS}_1 \xleftarrow{\$} \mathsf{Setup}(1^\lambda, Z, K, \mathsf{DS}_1)$.
$\mathcal{A}$ is given EDS_1.
$\mathcal{A}$ adaptively makes q queries in any order, for all $t \in [q]$:
$\mathsf{Access}(\mathsf{op}_t, \ell_t, v_t)$:
where $\mathsf{op}_t \in \{\mathsf{Find}, \mathsf{Insert}, \mathsf{Delete}\}$, $\ell_t \in \mathbb{L}, v_t \in \mathbb{V}$,
returns to $\mathcal{A}$ the transcript of honest execution $(v_t^*, \mathsf{EDS}_{t+1}) \leftarrow$ $[\mathsf{OMapE.Access}_C(K, \mathsf{opt}_t, \ell_t, v_t)$, $\mathsf{OMapE.Access}_B(\mathsf{EDS}_t)]$,
B's output EDS_{t+1},
and C's output v_t^*.
Finally, $\mathcal{A}$ outputs bit b.
The experiment returns the same bit b.

$\mathbf{Ideal}_{\Phi,\mathcal{A},\mathcal{S}}(1^\lambda)$:
Given $1^\lambda, Z$, adversary $\mathcal{A}$ outputs a map DS_1 over $\mathbb{L} \times \mathbb{V}$.
Given $1^\lambda, \mathcal{L}_{\mathsf{Setup}}(\mathsf{DS}_1)$, simulator $\mathcal{S}$ outputs encrypted structure EDS_1 and sends it to $\mathcal{A}$.
Let B be an honest server and given EDS_1.
$\mathcal{A}$ adaptively makes q queries in any order, for all $t \in [q]$:
$\mathsf{Access}(\mathsf{op}_t, \ell_t, v_t)$:
where $\mathsf{op}_t \in \{\mathsf{Find}, \mathsf{Insert}, \mathsf{Delete}\}$, $\ell_t \in \mathbb{L}, v_t \in \mathbb{V}$,
returns to $\mathcal{A}$ the transcript of $[\mathcal{S}(\mathcal{L}_{\mathsf{OMapE.Access}}(\mathsf{DS}_t, \mathsf{op}_t, \ell_t, v_t))$, $\mathsf{OMapE.Access}_B(\mathsf{EDS}_t)]$,
B's output EDS_{t+1},
and v_t^* output by $(v_t^*, \mathsf{DS}_{t+1}) \leftarrow \tilde{\mathcal{F}}_{\mathsf{OMap}}(\mathsf{DS}_t, \mathsf{op}_t, \ell_t, v_t)$.
Finally, $\mathcal{A}$ outputs bit b.
The experiment returns the same bit b.

Fig. 3. Experiments for Defining OMapE Adaptive Security.

The Setup algorithm is divided into two parts for ease of presentation. The first nine lines in part 1 involve running the key generation algorithm of OMapE, initializing an empty map, computing the number of eLSH instances needed based on the input. In the loop from Line 10 to Line 29 in part 2, we prepare the eLSH in-

stances by randomly sampling the hash functions and store the eLSH instances as part of the secret key. Line 31 to Line 37 deal with computing the eLSH tags from eLSH instances and messages, then adding them to the map. Finally, Setup ends with running the setup algorithm of OMapE on the map.

We now discuss the details related to parameter selection. Given ϵ, δ correctness bounds, upper bound n on the number of messages stored in the database, and upper bound t on the number of query messages supported by the system, we set the parameters theoretically in the Setup algorithm for provable correctness as follows:

- Line 3 to Line 7: We choose I with ξ to guarantee (ϵ, δ)-Search correctness following the results of Lemma D.6 and Lemma D.5. For each (γ_i, r_i)-NN problem, I number of eLSH instances are used to amplify the probability of capturing the approximate near neighbors.
- Line 11 to Line 13 compute the parameters so that the union of (γ_i, r_i)-NN for all $i \in [size_r]$ output by an algorithm called eLSH-cRNN in Search — finding (γ_i, r_i)-NN by checking the data points sharing overlapped eLSH tags with the query point, contains the approximate k-NN with high probability.
- Line 28: The numAccess is used to enforce each search query to yield a fixed number of Φ.Access and is stored as part of the secret key.
- Line 36, 37: We compute and initialize OMapE with the upper bound on the number of items stored in the map based on the maximal number of data points associated with each eLSH tag.

As we mentioned, in practice, the parameters for eLSH instances including $size_r$, s_i, l_i, and I can be set empirically to improve efficiency. Also, locality-sensitive hashing family $\mathcal{H}_i$ with sensitivity $(r_i, \gamma r_i, p_{i,1}, p_{i,2})$ are replaced by specific LSH construction instantiated with appropriate parameters in the implementation (e.g., LSH construction based on stable distribution [7]), and we refer to more details in Section 7.2. Our experiments and prior work [11] show that selecting one single threshold $size_r$ is usually sufficient. Parameters s_i, l_i are part of the eLSH instances. In our experiments, both s_i, l_i are small constants, less or equal to 10.

Algorithm $\mathsf{Setup}(1^\lambda, \epsilon, \delta, Z, \mathbf{M})$ (part 2)

10: **for** $i = 1, \ldots, size_r$ **do**
11: $r_i \leftarrow \gamma^i \cdot r_{\min}$
12: $\mathcal{H}_i \leftarrow (r_i, \gamma r_i, p_{i,1}, p_{i,2})$-sensitive hash family
13: $\rho_i \leftarrow \frac{\ln 1/p_{i,1}}{\ln 1/p_{i,2}}$; $s_i \leftarrow \lceil \log_{1/p_{i,2}} n \rceil$; $l_i \leftarrow \lceil n^{\rho_i}/p_{i,1} \rceil$
14: $K[2i+1] \leftarrow l_i$
15: numAccess $\leftarrow$ numAccess $+ l_i \cdot I$
16: **for** $j = 1, \ldots, I$ **do**
17: **for** $k = 1, \ldots, l_i$ **do**
18: **for** $u = 1, \ldots, s_i$ **do**
19: $h_{i,j,k,u}(\cdot) \xleftarrow{\$} \mathcal{H}_i$
20: **end for**
21: $\hat{g}_{i,j,k}(\cdot) \leftarrow (h_{i,j,k,1}(\cdot), \ldots, h_{i,j,k,s_i}(\cdot))$
22: **end for**
23: $\mathbf{g}_{i,j} \leftarrow (\hat{g}_{i,j,1}(\cdot), \hat{g}_{i,j,2}(\cdot), \ldots, \hat{g}_{i,j,l_i}(\cdot))$
24: **end for**
25: $\mathbf{G}_i \leftarrow (\mathbf{g}_{i,1}, \mathbf{g}_{i,2}, \ldots, \mathbf{g}_{i,I})$
26: $K[2i+2] \leftarrow \mathbf{G}_i$
27: **end for**
28: $K[2 \cdot size_r + 3] \leftarrow$ numAccess
29: **for** each M in $\mathbf{M}$ **do**
30: Parse M as $(M_{\mathcal{D}}, M_{\mathcal{D}_\mathrm{A}})$
31: **Tags** $\leftarrow \mathcal{T}(K, M_{\mathcal{D}})$
32: **for** each T in **Tags** **do**
33: map $\leftarrow$ map $\cup \{(T, M)\}$
34: **end for**
35: **end for**
36: $m \leftarrow n \cdot I \cdot \sum_{i \in [size_r]} l_i$
37: $\mathsf{EDS} \xleftarrow{\$} \Phi.\mathsf{Setup}(1^\lambda, m, K[1], \mathsf{map})$
38: **return** $(\mathsf{k}_{\max}, K, \mathsf{EDS})$

Algorithm $\mathcal{T}(K, M_{\mathcal{D}})$

1: Parse $K[2]$ as $n \| t \| r_{\min} \| r_{\max} \| size_r \| I$
2: **for** $i = 1, \ldots, size_r$ **do** do
3: $l_i \leftarrow K[2i+1]$; $\mathbf{G}_i \leftarrow K[2i+2]$
4: **for** $j = 1, \ldots, I$ **do** do
5: $\mathbf{g}_{i,j} \leftarrow \mathbf{G}_i[j]$; $\mathbf{S}_{i,j} \leftarrow \emptyset$
6: **for** $k = 1, \ldots, l_i$ **do** do
7: $\hat{g}_{i,j,k}(\cdot) \leftarrow \mathbf{g}_{i,j}[k]$
8: $T \leftarrow \langle i \rangle \| \langle j \rangle \| \langle k \rangle \| \hat{g}_{i,j,k}(M_{\mathcal{D}})$
9: $\mathbf{S}_{i,j} \leftarrow \mathbf{S}_{i,j} \cup \{T\}$
10: **end for**
11: **end for**
12: $\mathbf{T}_i \leftarrow (\mathbf{S}_{i,1}, \mathbf{S}_{i,2}, \ldots, \mathbf{S}_{i,I})$
13: **end for**
14: **Tags** $\leftarrow (\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_{size_r})$
15: **return Tags**

Algorithm $[\mathsf{Search}_C(K, Q_{\mathcal{D}}, k), \mathsf{Search}_S(\mathsf{EDS})]$

1: Server S:
2: $\quad \mathsf{EDS}_1 \leftarrow \mathsf{EDS}$
3: Client C:
4: $\quad u \leftarrow 1$
5: $\quad$ Parse $K[2]$ as $n \| t \| r_{\min} \| r_{\max} \| size_r \| I$
6: $\quad \mathtt{numAccess} \leftarrow K[2 \cdot size_r + 3]$
7: $\quad \mathbf{Y} \leftarrow \emptyset$; $\mathtt{ctrI} \leftarrow 0$; $\mathbf{Tags}^* \leftarrow \mathcal{T}(K, Q_{\mathcal{D}})$
8: $\quad$ **for** $i = 1, \ldots, size_r$ and $\mathtt{ctrI} < I$ **do**
9: $\quad\quad l_i \leftarrow K[2i+1]$; $\mathtt{ctrI} \leftarrow 0$
10: $\quad\quad \mathbf{T}_i^* \leftarrow \mathbf{Tags}^*[i]$
11: $\quad\quad$ **for** $j = 1, \ldots, I$ **do**
12: $\quad\quad\quad \mathtt{temp}_{i,j} \leftarrow \emptyset$
13: $\quad\quad\quad$ **loop** For each T in $\mathbf{T}_i^*[j]$
14: $\quad\quad\quad\quad$ Run $(\mathsf{data}^*, \mathsf{EDS}_{u+1}) \leftarrow$
15: $\quad\quad\quad\quad [\Phi.\mathsf{Access}_C(K[1], \mathsf{Find}, T, \perp),$
16: $\quad\quad\quad\quad \Phi.\mathsf{Access}_S(\mathsf{EDS}_u)]$
17: $\quad\quad\quad\quad$ interactively with server S;
18: $\quad\quad\quad\quad$ Parse data^* string into a set $\mathbf{S}$
19: $\quad\quad\quad\quad u \leftarrow u + 1$
20: $\quad\quad\quad\quad$ **if** $\mathbf{S} \neq \emptyset$ **then**
21: $\quad\quad\quad\quad\quad \mathtt{temp}_{i,j} \leftarrow \mathtt{temp}_{i,j} \cup \mathbf{S}$
22: $\quad\quad\quad\quad$ **end if**
23: $\quad\quad\quad\quad$ **if** $|\mathtt{temp}_{i,j}| = 3l_i + k - 1$ **then**
24: $\quad\quad\quad\quad\quad \mathtt{ctrI} \leftarrow \mathtt{ctrI} + 1$
25: $\quad\quad\quad\quad\quad$ **exit loop**
26: $\quad\quad\quad\quad$ **end if**
27: $\quad\quad\quad$ **end loop**
28: $\quad\quad\quad \mathbf{Y} \leftarrow \mathbf{Y} \cup \mathtt{temp}_{i,j}$
29: $\quad\quad$ **end for**
30: $\quad\quad$ **for** $u < \mathtt{numAccess} + 1$ **do**
31: $\quad\quad\quad$ Run $(\mathsf{data}^*, \mathsf{EDS}_{u+1}) \leftarrow$
32: $\quad\quad\quad [\Phi.\mathsf{Access}_C(K[1], \mathsf{Find}, \perp, \perp),$
33: $\quad\quad\quad \Phi.\mathsf{Access}_S(\mathsf{EDS}_u)]$
34: $\quad\quad\quad$ interactively with server S
35: $\quad\quad\quad u \leftarrow u + 1$
36: $\quad\quad$ **end for**
37: $\quad$ **end for**
38: Client C:
39: $\quad$ **if** $|\mathbf{Y}| \geq k$ **then**
40: $\quad\quad$ **return** $Q_{\mathcal{D}}$'s k nearest neighbors in $\mathbf{Y}$
41: $\quad$ **else**
42: $\quad\quad$ **for** $i = 1, \ldots, k - |\mathbf{Y}|$ **do**
43: $\quad\quad\quad M_i \xleftarrow{\$} \mathcal{MS}$; $\mathbf{Y} \leftarrow \mathbf{Y} \cup \{M_i\}$
44: $\quad\quad$ **end for**
45: $\quad$ **end if**
46: $\quad$ **return** $\mathbf{Y}$

The tagging algorithm $\mathcal{T}$ is used as a subroutine in Setup, Search, and Update. It takes as input a message in the metric space, a secret key encapsulating the eLSH instances, then computes and outputs the eLSH tags of the message. Note that at Line 7, $\mathbf{g}_{i,j}$ is a vector, where the k-th position denoted by $\mathbf{g}_{i,j}[k]$ is a placeholder for $\hat{g}_{i,j,k}(\cdot)$; $\hat{g}_{i,j,k}(\cdot)$ is an eLSH instance (Definition 2.3) where $(h_{i,j,k,1}(\cdot), h_{i,j,k,2}(\cdot), \ldots, h_{i,j,k,s_i}(\cdot))$ denotes a concatenation of hash functions.

In interactive algorithm Search, from Line 8 to Line 37, we find the AkNN of the query message by retrieving the messages associated with the query message's eLSH tags from the oblivious map. More specifically, from Line 8 to Line 29, we solve a sequence of (γ_i, r_i)-NN problems using eLSH-cRNN algorithm with an eLSH map. In each iteration, we perform oblivious look-ups on the corresponding eLSH map (cf. the correctness summary in Section 6.4 for more discussion). Note that at Line 25, the loop stops early when enough messages with matched eLSH tags are found. Lemma D.3 shows that the search algorithm with the early exist satisfies the correctness requirement. If enough messages are found, the algorithm stops the loop early. Line 30 to Line 36 enforce a constant number of accesses across the query messages. In particular, at Line 32 and 33, we use $[\Phi.\mathsf{Access}_C(K[1], \mathsf{Find}, \perp, \perp), \Phi.\mathsf{Access}_S(\mathsf{EDS}_u)]$ to denote the dummy accesses for padding needed for security, so that each search query will yield a fixed number of Φ.Access requests. Finally, Line 39 to Line 46 deal with post-processing and finding the AkNN. If not enough messages are found, random messages are selected to ensure the output set size is fixed at k.

Algorithm $\mathsf{Dec}(K, \mathsf{EDS})$

1: $\mathsf{DS} \leftarrow \Phi.\mathsf{Dec}(K[1], \mathsf{EDS})$
2: $\mathbf{M} \leftarrow \emptyset$
3: **for** each (T, M) in DS **do**
4: $\quad\quad \mathbf{M} \leftarrow \mathbf{M} \cup \{M\}$
5: **end for**
6: **return** $\mathbf{M}$

The decryption algorithm Dec of PP-AkNN follows from the decryption algorithm of the underlying OMapE.

Algorithm $[\mathsf{Update}_C(K, U, M^*), \mathsf{Update}_S(\mathsf{EDS})]$

Server S:
  $\mathsf{EDS}_1 \leftarrow \mathsf{EDS}$
Client C:
  Parse M^* as $(M^*_{\mathcal{D}}, M^*_{\mathcal{D}_\mathrm{A}})$; $u \leftarrow 1$
  Tags $\leftarrow \mathcal{T}(K, M^*_{\mathcal{D}})$
  if $U =$ add **then** op $\leftarrow$ Insert
  else if $U =$ remove **then** op $\leftarrow$ Delete
  end if
  for each T in **Tags do**
    Run $(\perp, \mathsf{EDS}_{u+1}) \leftarrow [\Phi.\mathsf{Access}_C(K[1],$
    $\mathsf{op}, T, M^*), \Phi.\mathsf{Access}_S(\mathsf{EDS}_u)]$
    interactively with server S
    $u \leftarrow u + 1$
  end for
  return $\perp$

The Update algorithm utilizes the update algorithm of the underlying OMapE straightforwardly.

6.4 Correctness & Efficiency

Parameter generation. To be able to evaluate correctness of our PP-AkNN protocol, we capture the approaches adopted by theoreticians by defining genAuxInfo algorithm that outputs parameters on a given distribution.

Over some metric space $(\mathcal{D}, d)$, we define a randomized algorithm $Z \xleftarrow{\$} \mathsf{genAuxInfo}(\alpha, \mathbf{M}, \mathbf{Q}, \mathsf{k_{max}})$ that takes as input some failure probability $\alpha \in [0, 1]$, (sufficiently large) $\mathbf{M}$ and $\mathbf{Q} \subseteq \mathcal{D}$, $n = |\mathbf{M}|$, $t = |\mathbf{Q}|$, $\mathsf{k_{max}} \ll n$, and outputs auxiliary information Z in the form of $\mathsf{k_{max}} \| n \| t \| \alpha \| r_{\min} \| r_{\max}$, where $r_{\min}, r_{\max} \in \mathbb{R}^+$ are distance thresholds such that, for each $Q \in \mathbf{Q}$, with at least $1 - \alpha$ probability, there exists a subset $\mathbf{S} \subset \mathbf{M}$ of size $\mathsf{k_{max}}$ such that $Q \in \bigcap_{M \in \mathbf{S}} \mathcal{B}(M, r_{\max})$, and Q does not fall into the ball of radius $r_{\min}$ centered on any message of $\mathbf{M}$, namely, $Q \notin \bigcup_{M \in \mathbf{M}} \mathcal{B}(M, r_{\min})$.

Parameters n, t are set as the upper-bound on the size of the message set, the query set respectively supported by the system.

The proof of the following correctness theorem is in Appendix D.

Theorem 6.3 (Main Correctness Theorem)**.** *For data distribution where* genAuxInfo *exists,* PP-AkNN *constructions instantiated with* OMapE (OAvlTreeE *or* OBSkiplistE) *satisfies* (ϵ, δ)-Search *correctness.*

Remark 6. Note that the above correctness theorem stands for distributions where genAuxInfo exists, and we illustrate its existence by construction in Proposition D.7. It is common for AkNN related works in the theory community to assume that parameters output by genAuxInfo such as closeness thresholds $r_{\min}, r_{\max}$ are given. In practice, we do not need to construct genAuxInfo in order to compute the eLSH parameters, instead we rely on empirically selecting eLSH parameters on data samples. We discuss this aspect in Sections 6.3 and 7.

Remark 7. Since Update straightforwardly calls Φ.Access on the update operation and every eLSH tag of the update messages, its correctness directly follows from that of the underlying OMapE scheme Φ.

Efficiency. The bandwidth cost of the AkNN protocol relies on the underlying oblivious map. If built on an oblivious map that costs $O(\log^2 N)$ bandwith (oblivious AVL tree and Bskiplist), assuming the underlying non-recursive ORAM operation incurs bandwidth cost $O(\log N)$ (e.g., PathORAM), the overall bandwith cost for Search or Update in PP-AkNN is $O(\mathsf{k_{max}} \cdot \log^2 N)$, $\mathsf{k_{max}} \ll N$. The computational complexity on the server side is small since each search or update operation only incurs $O(\log^2 N)$ number of accesses on the memory blocks. The client requires a small temporary storage ($O(\log^2 N)$), and small computational capacity since the client only needs to perform basic AES-based encryption, decryption and linear scan with size bounded in $\log N$ as part of ORAM operations; also for each Search query, filtering out the top AkNN requires one linear scan in $\mathsf{k_{max}}$ ($O(\mathsf{k_{max}})$) is very efficient since $\mathsf{k_{max}} \ll N$.

Although the above analysis is sufficient from the asymptotic complexity perspective, the constants hidden may heavily influence the efficiency in terms of communicational overheads, and thus we implemented the protocol using two instantiations of OMapE and provided performance evaluation on multiple benchmark data sets in Section 7.

6.5 Security Analysis

We state security of our generic PP-AkNN Π.

Theorem 6.4 (Main Theorem)**.** *The* PP-AkNN *protocol* Π *with building block* OMapE Φ *is adaptively* $\mathcal{L}_\Pi$*-secure (cf. Definition 4.1), if* Φ *is adaptively* $\mathcal{L}_\Phi$*-secure (cf. Definition 6.2), where* $\mathcal{L}_\Pi = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}})$

with

$$\mathcal{L}_{\mathsf{Setup}} = \mathcal{L}_{\Phi.\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}} = (\mathcal{L}_{\Phi.\mathsf{Access}}, \mathtt{numAccess_s}),$$
$$\mathcal{L}_{\mathsf{Update}} = (\mathcal{L}_{\Phi.\mathsf{Access}}, \mathtt{numAccess_u}),$$

where $\mathtt{numAccess_s}, \mathtt{numAccess_u}$ *denote the total number of executions of* Φ.Access *on single operation within each search query, each update operation respectively.*

Remark 8. Due to lack of space, we provide the proof sketch of Theorem 6.4 in Appendix A and the full proof in the full version [4]. We discuss the implications of the leakage profile when we present the protocol instantiations.

6.6 Instantiations and their Security

Oblivious Maps Instantiations and their Security. We refer to the same adaptive obliviousness notion as defined in [1] (also provided in Appendix Definition B.1). It is known and can be proved that the two candidates OAvlTree [28] and OBSkiplist [24] built on non-recursive position-based ORAM satisfies the adaptive obliviousness. However, the above two oblivious map candidates do not address the privacy aspect, captured by our OMapE adaptive security (Definition 6.2). Therefore, we will discuss two OMapE instantiations built on these two oblivious map candidates. Let OAvlTreeE denote the OMapE built on OAvlTree with standard symmetric encryption scheme $\mathcal{SE}$ (e.g., AES-based mode of operation). Let OBSkiplistE denote OMapE built on OBSkiplist with $\mathcal{SE}$. We provide their constructions in detail in Appendix B.3. The efficiency statements in Section 6.4 also apply to these two instantiations.

We state the security of the two OMapE candidates — OAvlTreeE and OBSkiplistE. A proof sketch on their adaptive security is provided in Appendix B.4.

Theorem 6.5. *The* OMapE *protocol instantiated with* OAvlTreeE *is adaptively* $\tilde{\mathcal{L}}_{\mathsf{OAvlTreeE}}$*-secure if* OAvlTree *is adaptively oblivious and* $\mathcal{SE}$ *is IND-CPA, where* $\tilde{\mathcal{L}}_{\mathsf{OAvlTreeE}} = (\tilde{\mathcal{L}}_{\mathsf{Setup}}, \tilde{\mathcal{L}}_{\mathsf{Access}})$*,* $\tilde{\mathcal{L}}_{\mathsf{Setup}} = (N, \mathtt{nodeSize}, \mathtt{bucketSize}, \mathtt{height})$*, and* $\tilde{\mathcal{L}}_{\mathsf{Access}} = \perp$*;* N *is total number of nodes or the upper-bound on the number of items can be stored in the* OAvlTreeE*;* nodeSize *is the size of every node of the tree in bits;* bucketSize *is the size of every bucket in bits, used by its underlying ORAM.*

Theorem 6.6. *The* OMapE *protocol instantiated with* OBSkiplistE *is adaptively* $\tilde{\mathcal{L}}_{\mathsf{OBSkiplistE}}$*-secure if* OBSkiplist *is adaptively oblivious and* $\mathcal{SE}$ *is IND-CPA, where* $\tilde{\mathcal{L}}_{\mathsf{OBSkiplistE}} = (\tilde{\mathcal{L}}_{\mathsf{Setup}}, \tilde{\mathcal{L}}_{\mathsf{Access}})$*,* $\tilde{\mathcal{L}}_{\mathsf{Setup}} = (N, \mathtt{nodeSize}, \mathtt{bucketSize}, \beta, \mathtt{height})$ *and* $\tilde{\mathcal{L}}_{\mathsf{Access}} = \perp$*, defined the same in Theorem 6.5, with extra branching factor* β*.*

Protocol Security Analysis. We now combine the above results and the generic PP-AkNN security result (Theorem 6.4) to get the following two security statements for PP-AkNN instantiated with OAvlTreeE and OBSkiplistE, respectively.

Theorem 6.7. *The* PP-AkNN *protocol* Π *instantiated with* OAvlTreeE *is adaptively-*$\mathcal{L}_{\mathsf{OAvlTreeE}}$ *secure, if* OAvlTree *is adaptively oblivious and* $\mathcal{SE}$ *is IND-CPA, where* $\mathcal{L}_{\mathsf{OAvlTreeE}} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}})$*,*

$$\mathcal{L}_{\mathsf{Setup}} = (N, \mathtt{nodeSize}, \mathtt{bucketSize}, \mathtt{height}),$$
$$\mathcal{L}_{\mathsf{Search}} = \mathtt{numAccess_s}, \quad \mathcal{L}_{\mathsf{Update}} = \mathtt{numAccess_u},$$

defined the same as in Theorem 6.4 and Theorem 6.5.

Theorem 6.8. *The* PP-AkNN *protocol* Π *instantiated with* OBSkiplistE *is adaptively-*$\mathcal{L}_{\mathsf{OBSkiplistE}}$ *secure, if* OBSkiplist *is adaptively oblivious and* $\mathcal{SE}$ *is IND-CPA, where* $\mathcal{L}_{\mathsf{OBSkiplistE}} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}}, \mathcal{L}_{\mathsf{Update}})$*,*

$$\mathcal{L}_{\mathsf{Setup}} = (N, \mathtt{nodeSize}, \mathtt{bucketSize}, \beta, \mathtt{height}),$$
$$\mathcal{L}_{\mathsf{Search}} = \mathtt{numAccess_s}, \quad \mathcal{L}_{\mathsf{Update}} = \mathtt{numAccess_u},$$

defined the same as in Theorem 6.4 and Theorem 6.6.

Security Implications. As the above security statements show, the leakage profile of our construction only comprises public parameters from Setup and the number of OMapE accesses incurred from Search and Update, respectively. Since the above public parameters from Setup apply to all messages stored in the system and the queries, and because the number of OMapE accesses is fixed for every search query and update query respectively, we conclude that our construction hides the *query pattern* (query-identity and query-equality pattern defined in [17]) and the *access pattern* (response identity and response-equality). Moreover, our construction hides the *volume pattern* since the transcript size and the number of interactions are fixed for every query request.

7 Implementations

7.1 Overview

We completed our PP-AkNN protocol incorporating the existing eLSH library LSH-kit [8] together with two OMapE implementations — OAvlTreeE and OBSkiplistE. We evaluated the performance of the two on standard AkNN benchmark data sets provided in [2] on a commercial laptop with 16 GB RAM running Ubuntu 20.04. Through our experiments, we found the efficiency bottleneck of our protocol is at the stage of transferring data between the client and the server. This is a direct product following from the communicational overhead of the underlying OMapE protocol. In comparison, extracting eLSH tags and post-processing in Search are both significantly efficient. In this section, we first provide a guideline on selecting parameters for our PP-AkNN constructions built on OAvlTreeE and OBSkiplistE, then evaluate the two on benchmark data sets and compare their performance. Furthermore, we provide two optimization methods — one involves batched search queries which reduces the bandwidth cost by a multiplicative factor (up to $\sum_{i=1}^{size_r} I \cdot l_i$), and the other incorporates parallel accesses that reduces the total number of roundtrips incurred in each Search query by a multiplicative factor in the number of OMapE instances available on the server.

7.2 Parameter Selection

We used the following benchmark data sets in our experiments: `MNIST`, `F-MNIST` (i.e., `Fashion-MNIST`), `SIFT`, and `GIST` [2]. They are all high-dimensional data points in the Euclidean space. In both `F-MNIST` and `MNIST`, each data point is a 28×28 grayscale image that we treat as a single feature vector and is associated with one label from 10 classes. `SIFT` and `GIST` are data sets containing feature vectors only with no labels attached. To select eLSH parameters, we run the parameter selection algorithm provided by LSH-kit for 1000 number of queries on the training set of size $60,000$, finding approximate 10-nearest neighbors (i.e., $\mathsf{k_{max}} = 10$). Looping a range of possible values for l, ϵ, the algorithm will output different s, width w (projection width), *cost*, and *recall* respectively. We selected eLSH parameters with *low cost* (the fraction of data points shared the same eLSH tags as the query and the lower the better) and *high recall* — the fraction that the correct AkNN captured in the response set. We found that setting $I = 1$ and $size_r = 1$ are sufficient to reach 95% recall for approximate k-nearest neighbors for every query point across the benchmark data sets with $\epsilon \geq 0.1$. We present the parameters for our constructions in the following Table 1 after fixing $l = 10$, $\mathsf{k_{max}} = 10$, and $\epsilon = 0.1$:

	Dataset	`MNIST`	`F-MNIST`	`SIFT`	`GIST`
	Dimension	784	784	128	960
	Train size	60 K	60 K	1 M	1 M
	Test size	10 K	10 K	10 K	1 K
	Total storage	217 MB	217 MB	501 MB	3.6 GB
	Feature size	3140 B	3140 B	516 B	3844 B
eLSH	s	7	9	8	8
	w	4275	4563	888	4.65
A	Node size	120 KB	120 KB	20 KB	147 KB
	Bucket size	480 KB	480 KB	80 KB	588 KB
B	BNode size	120 KB	120 KB	20 KB	147 KB
	Bucket size	720 KB	720 KB	120 KB	882 KB

Table 1. Parameters for PP-AkNN scheme A and B instantiated with OAvlTreeE and OBSkiplistE respectively

As shown in the above table, eLSH parameter s does not vary much. In contrast, projection width parameter w used as part of the eLSH construction in the Euclidean space is significantly smaller in `GIST` than the rest since the values' magnitude is small. Nevertheless, each eLSH tag of all the data points with the above parameters can fit in 8 bytes (stored as `uint64_t`). After fixing l and $\mathsf{k_{max}}$, the feature size (i.e., dimensions) plays a critical role determining the node size. We can bound the total number of data points associated with each eLSH tag using $3 \cdot l + \mathsf{k_{max}} - 1$. Namely, we only need to store and check at most 39 points to capture approximate 10-nearest neighbors with high probability. Therefore, we have greatly improved the performance compared with retrieving all the responses — even suitable eLSH parameters with a low cost of 0.006 still imply the response set for every query containing $0.006 \times 60,000 = 360$ points in expectation.

To clarify how to initialize the parameters for the PP-AkNN protocol, we guide through the procedure using OAvlTreeE on `SIFT` benchmark as an example. Given $l = 10$, $\mathsf{k_{max}} = 10$, the number of data points stored in every node with one associated eLSH tag is at most 39. Hence, for SIFT features, each node occupies at least $39 \times 516 = 19.5$ KB. As other attributes in the

node barely occupy the storage and standard blockcipher based IND-CPA encryption scheme incurs almost-negligible storage blow-up, it suffices to set the node size to 20 KB. Also, we set the bucket size 4 times of the block size for OAvlTreeE as suggested in [28]. For OBSkiplistE-based construction, we set bucket size Z, 6 times the expected block size B (20 KB), i.e., 120 KB, and branching factor $\beta = 12$ as in [24].

7.3 Efficiency Comparison

We evaluated the two instantiations on all benchmark data sets listed in Table 1, fixing the query size to $1,000$ and initialize the PP-AkNN protocol with parameters presented in Table 1.

Time Costs. We identified the efficiency bottleneck for our constructions is at the transmission stage due to the large bandwidth costs incurred from OMapE.Access. (1) Setup: Extracting the eLSH tags for every data point across the data sets takes less than 0.2 ms on average. The total setup time on the largest database (GIST) took around 2 hours for OBSkiplistE-based construction and 3 hours for OAvlTreeE-based one. Note that the setup algorithm only needs to be run once. (2) Search: The processing time for data retrieving and post-processing are both efficient, incurring negligible time costs — a sharp contrast to the time costs incurred in data transmission based on our simulation results. We simulated the transmission time under the network conditions: *roundtrip latency* = 30 ms and *bandwidth capacity* = 150 mbps. (3) We found that the time costs for Update and Search are similar for the setting where $I = 1$, $size_r = 1$, and $l = 10$, as both protocols run an equal number of OMapE.Access. In the following table, we present the average statistics for running OMapE.Access on single eLSH tag — the most time-consuming procedure of the whole protocol and discuss its implications.

	Dataset	MNIST	F-MNIST	SIFT	GIST
A	# roundtrips	84	84	102	102
	Bandwidth (MB)	826.88	826.88	199.22	1464.26
	Time (s)	45.16	45.16	10.88	79.97
B	# roundtrips	13	13	15	15
	Bandwidth (MB)	191.95	191.95	43.95	323.00
	Time (s)	10.48	10.48	2.4	17.64

Table 2. Efficiency evaluation for PP-AkNN scheme A and B instantiated with OAvlTreeE and OBSkiplistE respectively

The path length for each node access is $\lceil \log_2(N) \rceil + 1 = 25$. The number of roundtrips incurred is $\lceil 2 \cdot (1 + \log_\beta N) \rceil \approx 15$. The overall (downloading) bandwidth cost for each eLSH tag search is $15 \times 120 \times 25 = 43.95$ MB. We provide an example using OBSkiplistE-based PP-AkNN on SIFT to interpret the results in Table 2. Given $l = 10$, the encrypted data set with eLSH tags occupy approximately 595 MB in storage, this implementation can facilitate at least 1 AkNN query satisfying the strong security guarantee. The temporary requirement on the client storage is around 3 MB. The baseline approach yields more than 198 roundtrips, compared with 150 roundtrips with OBSkiplistE in the same setting. Note that such a comparison is not fair concerning downloading the whole database involves linear scan, which may not be feasible for a large database. Instead, if compared with the same structured data — a search tree based on eLSH tags with duplicated data points, the encrypted storage is at least 5187.9 MB, then our construction can facilitate at least 11 AkNN queries reaching the bandwidth cost of downloading an encrypted database. Note that for a much larger database, or a smaller $\mathsf{k}_{\max}$, such an offset compared with the baseline increases since the overall bandwidth overhead is $O(\mathsf{k}_{\max} \cdot \log^2(N))$, where N the is number of items stored in OMapE.

7.4 Optimizations

Batched queries. We can further optimize the efficiency of the search by batching the queries, specified in Appendix C. The idea is comparing the distances between the returned points and the queries points, if enough points for AkNN are found, abort the search early and continue the next query. To hide the volume leakage, we only need to ensure that the number of OMapE accesses is a multiple of that incurred from one AkNN search. This method will result in efficiency enhancement $\sum_{i=1}^{size_r} l_i \cdot I$ times of the non-optimized scheme in terms of bandwidth cost.

Parallel Access. We can also optimize the protocol using *parallel access*, reducing the number of roundtrips incurred in each Search query. Similar approaches also mentioned in multi-user ORAM literature — creating multiple OMapE instances, so that eLSH tag look-ups can be run in parallel. The number of roundtrips for one AkNN search will equal the number of one single eLSH tag look-up, while the overall bandwidth cost does not change.

Acknowledgments

We thank Arthur Lazzaretti's help with the experiments and the reviewers for very useful and insightful comments. The authors were supported in part by Facebook Faculty Research Award and Cisco Research Award.

References

[1] G. Asharov, I. Komargodski, W.-K. Lin, K. Nayak, E. Peserico, and E. Shi. OptORAMa: Optimal Oblivious RAM. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 403–432. Springer International Publishing, 2020.

[2] E. Bernhardsson. Benchmarking Nearest Neighbors. https://github.com/erikbern/ann-benchmarks.

[3] A. Boldyreva and N. Chenette. Efficient Fuzzy Search on Encrypted Data. In C. Cid and C. Rechberger, editors, *Fast Software Encryption*, Lecture Notes in Computer Science, pages 613–633. Springer, 2015.

[4] A. Boldyreva and T. Tang. Privacy-Preserving Approximate k-Nearest-Neighbors Search that Hides Access, Query and Volume Patterns. *Full version of this paper. IACR Cryptology ePrint Archive*, 2021.

[5] X. Cheng, S. Su, Y. Teng, and K. Xiao. Enabling Secure and Efficient kNN Query Processing over Encrypted Spatial Data in the Cloud. *Security and Communication Networks*, 8, 03 2015.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 79–88. Association for Computing Machinery, 2006.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, June 2004. Association for Computing Machinery.

[8] W. Dong. LSHKIT: A C++ Locality Sensitive Hashing Library. http://lshkit.sourceforge.net.

[9] Y. Elmehdwi, B. K. Samanthula, and W. Jiang. Secure k-Nearest Neighbor Query over Encrypted Data in Outsourced Environments. In *2014 IEEE 30th International Conference on Data Engineering*, pages 664–675, March 2014.

[10] C. Gentry, K. A. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs. Optimizing ORAM and Using It Efficiently for Secure Computation. In E. De Cristofaro and M. Wright, editors, *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 1–18. Springer, 2013.

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 518–529. Morgan Kaufmann, 1999.

[12] P. Grubbs, M.-S. Lacharite, B. Minaud, and K. G. Paterson. Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 315–331. Association for Computing Machinery, 2018.

[13] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson. Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1067–1083, 2019.

[14] S. Har-Peled, P. Indyk, and R. Motwani. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of computing*, 8(1):321–350, 2012.

[15] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613. Association for Computing Machinery, 1998.

[16] S. Kamara and T. Moataz. Computationally Volume-Hiding Structured Encryption. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, Lecture Notes in Computer Science, pages 183–213. Springer International Publishing, 2019.

[17] S. Kamara, T. Moataz, and O. Ohrimenko. Structured Encryption and Leakage Suppression. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, Lecture Notes in Computer Science, pages 339–370. Springer International Publishing, 2018.

[18] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. Data Recovery on Encrypted Databases with k-Nearest Neighbor Query Leakage. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1033–1050, 2019.

[19] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1223–1240, 2020.

[20] D. Micciancio. Oblivious Data Structures: Applications to Cryptography. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 456–464. Association for Computing Machinery, 1997.

[21] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa. Oblix: An Efficient Oblivious Search Index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296, San Francisco, CA, May 2018. IEEE.

[22] M. Naveed. The Fallacy of Composition of Oblivious RAM and Searchable Encryption. *IACR Cryptology ePrint Archive*, 2015:668, 2015.

[23] Y. Peng, H. Li, J. Cui, J. Ma, and Y. Liu. Towards Secure Approximate k-Nearest Neighbor Query over Encrypted High-dimensional Data. *IEEE Access*, PP:1–1, 04 2018.

[24] D. S. Roche, A. Aviv, and S. G. Choi. A Practical Oblivious Map Data Structure with Secure Deletion and History Independence. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 178–197, May 2016.

[25] B. K. Samanthula, Y. Elmehdwi, and W. Jiang. k-Nearest Neighbor Classification over Semantically Secure Encrypted Relational Data. *IEEE transactions on Knowledge and data*

engineering, 27(5):1261–1273, 2015.

[26] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT press, 2006.

[27] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 299–310. Association for Computing Machinery, 2013.

[28] X. S. Wang, K. Nayak, C. Liu, T.-H. H. Chan, E. Shi, E. Stefanov, and Y. Huang. Oblivious Data Structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 215–226. Association for Computing Machinery, 2014.

[29] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis. Secure kNN Computation on Encrypted Databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 139–152. ACM, 2009.

[30] B. Yao, F. Li, and X. Xiao. Secure Nearest Neighbor Revisited. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 733–744. IEEE, 2013.

A Proof Sketch for Security of Generic PP-AkNN Construction

We provide a proof sketch for the main security theorem Theorem 6.4) for generic PP-AkNN construction.

Proof Sketch. As discussed in Section 4, our security definition captures that the query outputs are revealed to the adversary for randomized functionality AkNN. To prove our generic PP-AkNN construction Π satisfies the security definition, we need to show that the joint distributions of the query outputs and the transcripts either output by the simulator or incurred from the honest execution of the protocol are computationally indistinguishable.

Since parameter $\mathsf{k}_{\max}$ is output by Setup in both experiments, and the query outputs are yielded from running the honest execution of protocol Search, the adversary cannot distinguish the two worlds from those outputs. It suffices to show that the transcripts in both worlds are computationally indistinguishable if the adversary is given the query outputs and $\mathsf{k}_{\max}$.

Let $\mathcal{S}_1$ be the simulator for OMapE scheme Φ operating the same as in Appendix B.4, and recall the leakage profile for generic PP-AkNN Π,

$$\mathcal{L}_{\mathsf{Setup}} = \mathcal{L}_{\Phi.\mathsf{Setup}}, \mathcal{L}_{\mathsf{Search}} = (\mathcal{L}_{\Phi.\mathsf{Access}}, \mathtt{numAccess}_{\mathsf{s}}),$$
$$\mathcal{L}_{\mathsf{Update}} = (\mathcal{L}_{\Phi.\mathsf{Access}}, \mathtt{numAccess}_{\mathsf{u}}).$$

We construct simulator $\mathcal{S}_2$ for PP-AkNN scheme Π with $\mathcal{S}_1$ as a subroutine. Simulator $\mathcal{S}_2$ operates as follows: at the setup stage, $\mathcal{S}_2$ outputs the same encrypted data structure as simulator $\mathcal{S}_1$ on given leakage pattern $\mathcal{L}_{\Phi.\mathsf{Setup}}$, and sends to the adversary. On Search queries: $\mathcal{S}_2$ runs $\mathcal{S}_1$ simulating Access with $\mathcal{L}_{\Phi.\mathsf{Access}}$ as input $\mathtt{numAccess}_{\mathsf{s}}$ times. $\mathcal{S}_2$ outputs the same as $\mathcal{S}_1$. Similarly, for Update queries: $\mathcal{S}_2$ runs $\mathcal{S}_1$ simulating Access with $\mathcal{L}_{\Phi.\mathsf{Access}}$ as input $\mathtt{numAccess}_{\mathsf{u}}$ times, and outputs the same as $\mathcal{S}_1$. Since the transcripts are computationally indistinguishable for each Φ.Access for underlying Φ, the transcripts for Π in both worlds are computationally indistinguishable. Although the query outputs are revealed to the adversary together with prior information on the message set, the adversary still can not distinguish the transcripts produced by the simulator $\mathcal{S}_2$ and the one incurred in the honest executions of the real protocols because of the IND-CPA security of the transcripts ensured by underlying OMapE scheme Φ. □

B Oblivious Map with Encryption

B.1 Overview

At a high level, OAvlTree and OBSkiplist both are trees of bounded degree, stored in a non-recursive position-based ORAM. Running Access protocol on each map operation will incur multiple Access executions on Read or Write operations to the underlying ORAM. "Position-based" ORAMs are a class of ORAMs relying on that the client stores a (secret) position map — mapping between logical addresses (block ids) and sets of random physical addresses; for each Read or Write operation on some logical address addr, the client first conducts a position map look-up for addr's associated physical addresses, and then requests the server to access those memory blocks. After each access, the client will write back the accessed blocks to new random positions and updates its local position map. Therefore the overall access protocol hides the logical address from observing the access pattern in a computational setting. The "non-recursive" property guarantees that a single ORAM access only incurs one roundtrip communication. In practice, if the position map is large and cannot fit the client's local storage, it is then stored re-

cursively in a recursive ORAM on the server, yielding multiple roundtrips for a single ORAM block access. To solve this issue, achieving a more efficient look-up in the map structure stored in the ORAM, both OAvlTree and oblivious OBSkiplist exploit their underlying tree structures and smartly store the position maps. More specifically, other than regular data, each tree node also contains its children nodes' position maps, and is then stored as an ORAM block in a non-recursive position-based ORAM (e.g., PathORAM). For the search look-up, starting from the root node, accessing each tree node through ORAM will automatically retrieve the position maps of its children nodes. Hence, it saves the communication costs for position-map look-up — a $\log N$ multiplicative factor improvement in bandwidth compared with performing a naive search on a recursive position-based ORAM.

B.2 Adaptive Obliviousness

$\underline{\mathbf{Real}^{\mathcal{M}}_{\mathcal{A}}(1^\lambda):}$
$(\mathsf{command}_i, \mathsf{inp}_i) \leftarrow \mathcal{A}(1^\lambda)$,
where $\mathsf{command}_i \in \{\mathsf{Access}, \perp\}$
Loop *while* $\mathsf{command}_i \neq \perp$:
$(\mathsf{out}_i, \mathsf{Addrs}_i) \leftarrow \mathcal{M}(1^\lambda, \mathsf{command}_i, \mathsf{inp}_i)$
$(\mathsf{command}_i, \mathsf{inp}_i) \leftarrow \mathcal{A}(1^\lambda, \mathsf{out}_i, \mathsf{Addrs}_i)$

$\underline{\mathbf{Ideal}^{\mathcal{F}}_{\mathcal{A},\mathcal{S}}(1^\lambda):}$
$(\mathsf{command}_i, \mathsf{inp}_i) \leftarrow \mathcal{A}(1^\lambda)$,
where $\mathsf{command}_i \in \{\mathsf{Access}, \perp\}$
Loop *while* $\mathsf{command}_i \neq \perp$:
$\mathsf{out}_i \leftarrow \mathcal{F}(\mathsf{command}_i, \mathsf{inp}_i)$
$\mathsf{Addrs}_i \leftarrow \mathcal{S}(1^\lambda, \mathsf{command}_i)$
$(\mathsf{command}_i, \mathsf{inp}_i) \leftarrow \mathcal{A}(1^\lambda, \mathsf{out}_i, \mathsf{Addrs}_i)$

Fig. 4. Experiments for Defining Adaptive Obliviousness.

In the following definitions, the functionality $\mathcal{F}$ implicitly specifies the command space and input space, and are omitted. For ORAM and OMap, it is sufficient to let the command space be $\{\mathsf{Access}, \perp\}$ as both use Access protocol but with different input space specified in the functionality.

Definition B.1 (Adaptive Obliviousness)**.** *Let $\mathcal{M}$ be a reactive machine that implements functionality $\mathcal{F}$. Let probabilistic experiments be defined in Figure 4 (Appendix). We say that reactive machine $\mathcal{M}$ is an oblivious implementation of $\mathcal{F}$ if there exists a PPT simulator $\mathcal{S}$ such that for all non-uniform stateful PPT adversaries $\mathcal{A}$, the views of $\mathcal{A}$ in the real experiment and ideal experiment are computationally indistinguishable (negligible in λ) [1].*

Algorithm $\mathsf{OMapE.KeyGen}(1^\lambda)$:
$K \xleftarrow{\$} \mathcal{K}(1^\lambda)$
return K

Algorithm $\mathsf{OMapE.Setup}(1^\lambda, Z, K, \mathsf{DS})$:
$(\mathsf{node}_i)_{i \in N'} \leftarrow \mathsf{Helper.Init}(Z, \mathsf{DS})$
$\mathsf{EDS} \leftarrow ()$
for all $i \in [N']$ do
$\mathsf{EDS}[i] \xleftarrow{\$} \mathcal{E}(K, \mathsf{node}_i)$
return EDS

Algorithm $[\mathsf{OMapE.Access}_C(K, \mathsf{op}, \ell, v), \mathsf{OMapE.Access}_S(\mathsf{EDS}_u)]$:
Client C:
$v' \leftarrow \mathcal{E}(K, \ell \| v)$
$\mathsf{OMap.Access}_C(\mathsf{op}, \ell, v')$
Run $(v^*, \mathsf{EDS}_{u+1}) \leftarrow$
$[\mathsf{OMap.Access}_C(\mathsf{op}, \ell, v'),$
$\mathsf{OMap.Access}_S(\mathsf{EDS}_u)]$
interactively with server S
$\hat{v} \leftarrow \mathcal{D}(K, v^*)$
return $\hat{v}$

Algorithm $\mathsf{OMapE.Dec}(K, \mathsf{EDS})$:
$\mathsf{DS} \leftarrow \{\}$
for each $i \in [|\mathsf{EDS}|]$ do
$\mathsf{node}_i \leftarrow \mathcal{D}(K, \mathsf{EDS}[i])$
Parse node_i's data field as (ℓ_i, v_i)
if $\ell_i \neq \perp$ then
$\mathsf{DS} \leftarrow \mathsf{DS} \cup \{(\ell_i, v_i)\}$
return DS

Fig. 5. OMapE Constructions.

B.3 Constructions

We show how we build OMapE schemes with $\mathcal{L}_{\mathsf{OMapE.Access}} = \perp$, and $\mathcal{L}_{\mathsf{OMapE.Setup}}$ comprising public parameters. The two instantiations are OAvlTreeE and OBSkiplistE. The type of the atomic data object node in ODS framework for trees of bounded degree by a tuple

of three attributes:

$$\mathsf{node} = (\mathsf{nid}, \mathsf{ndata}, \mathsf{nchildren}),$$

where the value of nid attribute is in $[N]$, and N is the number of nodes of the data structure; the value of ndata attribute is in $\{0,1\}^w$ by some fixed parameter w, nchildren are pointers to every its child node, which we can parse as a list $(\mathsf{nchildren}_1, \ldots, \mathsf{nchildren}_t)$, where each $\mathsf{nchildren}_i \in \{0,1\}^*$, for all $i \in [t]$. Each element of the map is stored in the ndata field. To connect the logical addresses and physical addresses. The ODS framework stores each node as a block in the underlying ORAM. Note in Figure 5, we provide a helper function $\mathsf{Helper.Init}(Z, \mathsf{DS})$, which can be instantiated differently based on the underlying oblivious map. In the context of [24, 28], $\mathsf{Helper.Init}(Z, \mathsf{DS})$ can be interpreted as running $\mathsf{Access}(\mathsf{Insert}, \ell, v)$ locally on the client side for all (ℓ, v) pairs in the map, then output the plain array storing all the node data. This procedure involves random shuffling, and thus the output (node_i), for each node_i, its physical address does not reveal its logical address.

Remark 9. In the constructions of non-recursive position-based ORAMs, *bucketization* is a common technique used (e.g. PathORAM) — each bucket contains multiple blocks and has a fixed size in bits. Bucket-wise accessing operation is used when accessing the blocks in the access protocol. OBSkiplist [24] utilizes this fact — fixed bucket size, relaxes the block size from a constant w to w in expectation, while still achieves the oblivious simulation of the map functionality.

Remark 10. In the construction, Z is typically set by N, the upper-bound on the number of labels stored in the map. λ is the security parameter used to ensure the obliviousness. More discussion on λ and N can be refereed in [1].

B.4 Adaptive Security of OMapE Schemes

Referring to the security theorems Theorem 6.6 and Theorem 6.5. We first build $\tilde{\mathcal{F}}_{\mathsf{OMap}}$ using $\mathcal{F}_{\mathsf{ORAM}}$, then provide a proof sketch on their adaptive security. First let us recall $\mathcal{F}_{\mathsf{ORAM}}$.

Definition B.2 ($\mathcal{F}_{\mathsf{ORAM}}$ [1])**.** *The functionality is reactive, and holds an internal state — N memory blocks, each of size w (Block size). Denote the internal state an array $\mathbf{X}[1, \ldots, N]$. Initially, $\mathbf{X}[\mathsf{addr}] = 0$ for every $\mathsf{addr} \in [N]$.*

- $\mathsf{Access}(\mathsf{op}, \mathsf{addr}, \mathsf{data})$*: where* $\mathsf{op} \in \{\mathsf{Read}, \mathsf{Write}\}$*,* $\mathsf{addr} \in [N]$*, and* $\mathsf{data} \in \{0,1\}^w$*.*
  1. *If* $\mathsf{op} = \mathsf{Read}$*,* $\mathsf{data}^* \leftarrow \mathbf{X}[\mathsf{addr}]$*.*
  2. *If* $\mathsf{op} = \mathsf{Write}$*,* $\mathbf{X}[\mathsf{addr}] \leftarrow \mathsf{data}$ *and* $\mathsf{data}^* \leftarrow \mathsf{data}$*.*
  3. *Output* data^**.*

Remark 11. For Read operation, input data is set by $\perp$.

We first show that OAvlTree and OBSkiplist using $\mathcal{F}_{\mathsf{ORAM}}$ to implement $\tilde{\mathcal{F}}_{\mathsf{OMap}}$. For tree nodes defined the same as in [24, 28]. Let $\mathsf{nid} \in [N]$ be the logical address of the block in ORAM. Let $\mathsf{node} = (\mathsf{nid}, \mathsf{ndata}, \mathsf{nchildren})$. Consider the reactive map functionality in Definition 6.1. The functionality is reactive, and holds an internal state — N memory blocks, each of (expected) size w denoted by $\mathbf{X}[1, \ldots N]$ storing $\mathsf{map} \subseteq \mathbb{L} \times \mathbb{V}$. Let $(\mathsf{node})_i \leftarrow \mathsf{Helper.Init}(\mathsf{map})$. For each node, store $\mathbf{X}[i] \leftarrow \mathsf{ndata} \| \mathsf{nchildren}$. Load root node in the local stash, parse root node as (ℓ_1, v_1), and access using the algorithms specified in [24, 28] implementing $\tilde{\mathcal{F}}_{\mathsf{OMap}}$ with $\mathcal{F}_{\mathsf{ORAM}}$. For oblivious AVL tree it also needs to make dummy access to hide the operation type, and thus each tree operation (rotation, insert, delete) will result in a constant number of ORAM accesses. Similarly, each operation on BSkiplist also incurs a fixed number of ORAM accesses. Since each ORAM access incurs constant bandwidth cost for PathORAM instantiation (fixed bucket size in bits). Both OAvlTree and OBSkiplist hide the volume pattern.

Proof Sketch. We have showed that we can implement the $\tilde{\mathcal{F}}_{\mathsf{OMap}}$ using $\mathcal{F}_{\mathsf{ORAM}}$, and thus it follows that it satisfies adaptive obliviousness definition. Moreover, OAvlTree and OBSkiplist both implement a map construction with perfect correctness. Although the latter uses random coins for underlying data structure, it does not sacrifice the exact correctness of the map functionality. Thus it is left to argue that we can simulate the encrypted data structure and the transcripts. We construct a simulator $\mathcal{S}$ that generates a fake encrypted data structure and argue that no efficient adversary $\mathcal{A}$ can distinguish a real encrypted data structure from the fake one. Also, the simulator generates transcripts with an honest party indistinguishable from real executions of OMapE.Access between two honest parties. It suffices to show that the distribution of the encrypted data structure; the transcripts for OMapE.Access in the real world and simulated one are computationally indistinguishable. Our main technique is that, given the setup leakage including total N number of nodes in the tree, node size `nodeSize`, the simulator $\mathcal{S}$ first encrypts N' (computed based on

N) $\langle 0 \rangle$'s of bit length nodeSize. For simplicity, assume underlying ORAM is PathORAM built on a complete binary tree containing $2N-1$ buckets to store up to N blocks (nodes); let $N' = (2N-1) \times \texttt{bucketSize}$. Then for $i = 1$ to N', $\mathsf{EDS}_1[i] \leftarrow \mathcal{E}_{K_\mathcal{S}}(\langle 0 \rangle)$, where $\langle 0 \rangle$ is of bit-length nodeSize. At the end, $\mathcal{S}$ outputs the encrypted structure EDS_1 and sends it to $\mathcal{A}$. Adversary $\mathcal{A}$ adaptively makes q queries, simulator $\mathcal{S}$ simulates the transcript of the interactions with honest party B. For each OMapE.Access: The simulator $\mathcal{S}$ sends B a transcript which is a fixed-size list of physical addresses. Honest party B will then send all encrypted nodes associated with those physical addresses to $\mathcal{S}$. Simulator $\mathcal{S}$ then writes back a list of updated encrypted nodes and associated addresses to B. Finally, honest B will update and output EDS_{t+1}. The "adaptive obliviousness" of oblivious map implemented using ORAM ensures the physical addresses are computationally indistinguishable in both worlds, and together with IND-CPA property of each encrypted block (node), ensure computational indistinguishability in transcripts in OMapE.Access and OMapE.Access between the real world and the ones in ideal world. □

C Batch Queries

We outline how we optimize the PP-AkNN protocol to support multiple secure AkNN search queries more efficiently.

Correctness. We can directly reuse the arguments as in the correctness analysis for single query, since we only abort the search early if enough data points are found by examining the distances between the query point and the returned points.

Efficiency. It is straightforward that the optimization yields a multiplicative factor improvement in both the number of roundtrips and the bandwidth cost. The multiplicative factor in these two aspects will be up to $\sum_{i=1}^{size_r} I \cdot l_i$, and the exact number depends on the data distribution.

Security. In the optimization, with batched queries, we may abort the search query early and continue the eLSH tag look-up for next query. The total number of incurred OMapE accesses is padded with dummy accesses, ensuring it is a multiple of numAccess — fixed number of OMapE accesses for single AkNN search query. Together with other public parameters, using similar arguments as before, there exists a PPT simulator that can output

Algorithm Helper.Query$(K, u, Q_\mathcal{D}, k)$

Parse $K[2]$ as $n \| t \| r_{\min} \| r_{\max} \| size_r \| I$
numAccess $\leftarrow K[2 \cdot size_r + 3]$
$\mathbf{Y} \leftarrow \emptyset$; $\mathbf{Tags}^* \leftarrow \mathcal{T}(K, Q_\mathcal{D})$
abortFlag $\leftarrow$ false
for $i = 1, \ldots, size_R$ and abortFlag = false **do**
  $L_i \leftarrow K[2i+1]$;
  $\mathbf{T}_i^* \leftarrow \mathbf{Tags}^*[i]$
  for $j = 1, \ldots, I$ and abortFlag = false **do**
    loop each T in $\mathbf{T}_i^*[j]$
      Run $(\mathsf{data}^*, \mathsf{EDS}_{u+1}) \leftarrow$
      $[\mathsf{OMapE.Access}_C(K[1], \mathsf{Find}, T, \perp),$
      $\mathsf{OMapE.Access}_S(\mathsf{EDS}_u)]$
      interactively with server S;
      Parse data^* string into a set $\mathbf{S}$
      $u \leftarrow u + 1$
      if $\mathbf{S} \neq \emptyset$ **then**
        for each P in $\mathbf{S}$ and $|\mathbf{Y}| < k$ **do**
          Parse P as $(P_\mathcal{D}, P_{\mathcal{D}_\mathrm{A}})$
          if $d(P_\mathcal{D}, Q_\mathcal{D}) \leq \gamma^i \cdot r_{\min}$ **then**
            $\mathbf{Y} \leftarrow \mathbf{Y} \cup \{P\}$
          end if
        end for
      end if
      if $|\mathbf{Y}| = k$ **then**
        abortFlag $\leftarrow$ true
        exit loop
      end if
    end loop
  end for
end for
if $|\mathbf{Y}| < k$ **then**
  for $i = 1, \ldots, k - |\mathbf{Y}|$ **do**
    $M_i \xleftarrow{\$} \mathcal{MS}$; $\mathbf{Y} \leftarrow \mathbf{Y} \cup \{M_i\}$
  end for
end if
return $(u, \mathbf{Y})$

Algorithm $[\mathsf{Search}_C(K, \mathbf{Q}_\mathcal{D}, \mathbf{k}), \mathsf{Search}_S(\mathsf{EDS})]$

Server S:
  $\mathsf{EDS}_1 \leftarrow \mathsf{EDS}$
Client C:
  Parse $K[2]$ as $n\|t\|r_{\min}\|r_{\max}\|size_r\|I$
  $\mathtt{numQuery} \leftarrow |\mathbf{Q}_\mathcal{D}|$
  $\mathtt{numAccess} \leftarrow K[2 \cdot size_r + 3]$
  $u \leftarrow 1$
  for $i = 1, \ldots, \mathtt{numQuery}$ **do**
    $(u, \mathbf{Y}_i) \leftarrow \mathsf{Helper.Query}(K, u, \mathbf{Q}_\mathcal{D}[i], \mathbf{k}[i])$
  end for
  for $((u-1) \mod \mathtt{numAccess}) \neq 0$ **do**
    Run $(\mathsf{data}^*, \mathsf{EDS}_{u+1}) \leftarrow$
    $[\mathsf{OMap.Access}_C(K[1], \mathsf{Find}, \perp, \perp)$
    $\mathsf{OMap.Access}_S(\mathsf{EDS}_u)]$
    interactively with server S
  end for
  $\mathbf{Y}' \leftarrow (\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_{\mathtt{numQuery}})$
  return $\mathbf{Y}'$
Server S:
  return EDS_u

transcripts computationally-indistinguishable from that yielded by the real executions of the protocol as long as the underlying map holds the "obliviousness" property, and the symmetric encryption used to encrypt all data is IND-CPA.

D Proof of Correctness

In the following discussion, for all $\mathbf{M}, \mathbf{Q} \subseteq \mathcal{D}$, $|\mathbf{M}| = n$ and $|\mathbf{Q}| = t$, where $\mathsf{genAuxInfo}$ exists. For $\mathsf{k_{max}} \in \mathbb{N}$, $Z \xleftarrow{\$} \mathsf{genAuxInfo}(\alpha, \mathbf{M}, \mathbf{Q}, \mathsf{k_{max}})$, parse Z as $n\|t\|\mathsf{k_{max}}\|\alpha\|r_{\min}\|r_{\max}$. We refer $(\epsilon, \delta, \beta)$-$\mathsf{Search}$ to a shorthand notation of (ϵ, δ)-Search correctness with a size restriction parameter β on the total number of messages needed to be examined to determine the AkNN. We also discuss how the failure probability δ (depends on α in $\mathsf{genAuxInfo}$) affects parameter selection and compute the size restriction parameter β.

Over arbitrary metric space $(\mathcal{D}, d)$, given $\alpha \in [0, 1]$, all $\mathbf{M}, \mathbf{Q}, \subseteq \mathcal{D}$, $|\mathbf{M}| = n, |\mathbf{Q}| = t$ where $\mathsf{genAuxInfo}$ exists, all $\mathsf{k_{max}} \ll |\mathbf{M}|$ (e.g., $\mathsf{k_{max}} = \sqrt{|\mathbf{M}|}$), all $Q \in \mathbf{Q}$, all $k \in [\mathsf{k_{max}}]$, we parse Z as $n\|t\|\mathsf{k_{max}}\|\alpha\|r_{\min}\|r_{\max}$. Let $size_r = \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil$, where $\gamma = \sqrt{1+\epsilon}$. We first set parameter I by Lemma D.6 with ξ chosen by Lemma D.5, and select parameter s_i, l_i for all $i \in [size_r]$ by Proposition E.2. We may use "α-query-valid" source to denote for $\mathbf{M}, \mathbf{Q} \subseteq \mathcal{D}$, where $\mathsf{genAuxInfo}$ exits, and Z is output by $\mathsf{genAuxInfo}(\alpha, \mathbf{M}, \mathbf{Q}, \mathsf{k_{max}})$. We introduce the following main theorem for our eLSH-based construction.

Proof of Main Correctness Theorem 6.3.
It suffices to show that the eLSH-based construction satisfies $(\epsilon, \delta, \beta)$-$\mathsf{Search}$ correctness on plain LSH tags. For all $\mathbf{M}, \mathbf{Q} \subseteq \mathcal{D}$, $|\mathbf{M}| = n$ and $|\mathbf{Q}| = t$, where $\mathsf{genAuxInfo}$ exists. For $\alpha \in [0, 1], \mathsf{k_{max}} \in \mathbb{N}$, $Z \xleftarrow{\$} \mathsf{genAuxInfo}(\alpha, \mathbf{M}, \mathbf{Q}, \mathsf{k_{max}})$, parse Z as $n\|t\|\mathsf{k_{max}}\|\alpha\|r_{\min}\|r_{\max}$, i.e., For every $Q \in \mathbf{Q}$, with at least $1 - \alpha$ probability, both the following events hold,

A_1: $Q \notin \bigcup_{M \in \mathbf{M}} \mathcal{B}(M, r_{\min})$.
A_2: There exists a subset $\mathbf{S} \subset \mathbf{M}$ of size $\mathsf{k_{max}}$ such that $Q \in \bigcap_{M \in \mathbf{S}} \mathcal{B}(M, r_{\max})$.

We will show setting I based on Lemma D.6 with ξ set by Lemma D.5 guarantees that the eLSH-based scheme satisfies $(\epsilon, \delta, \beta)$-$\mathsf{Search}$ correctness.

Since each $\mathsf{genAuxInfo}$ is a randomized algorithm, we need to establish a relation between failure probability α and δ. First consider a simpler case when $\alpha = 0$, which we prove separately as Lemma D.1.

Now consider the general case with $\alpha \in [0, 1]$. By definition, $\Pr(A_1 \cap A_2) \geq 1 - \alpha$. To connect α and δ, we first define event A_3. Given any eLSH-based construction Π with some source, let A_3 denote the event that Π satisfies the exact correctness, i.e., $(\epsilon, 0, \hat{\beta})$-$\mathsf{Search}$ correctness for some constant $\hat{\beta}$. Then suppose there is an eLSH-based construction Π' with "0-query-valid" source satisfying $(\epsilon, \delta', \beta')$-$\mathsf{Search}$ correctness, where $0 \leq \delta' \leq \delta - \alpha$ and β' is some constant, in other words, $\Pr(A_3) \geq 1 - \delta'$. We show that the scheme Π' with arbitrary α-query-valid source $\mathcal{M}$ satisfies $(\epsilon, \delta, \beta)$-$\mathsf{Search}$ correctness by bounding the probability that all events A_1, A_2, A_3 hold from below.

$$\begin{aligned}\Pr(A_1 \cap A_2 \cap A_3) &= 1 - \Pr(\overline{A_1 \cap A_2} \cup \overline{A_3}) \\ &\geq 1 - \Pr(\overline{A_1 \cap A_2}) - \Pr(\overline{A_3}) \\ &\geq 1 - \alpha - \delta' \\ &\geq 1 - \delta,\end{aligned}$$

where the last inequality is due to $0 \leq \delta' \leq \delta - \alpha$. To complete the proof, we show that scheme Π' does exist with Lemma D.1, and the new size restriction β can be computed according to Lemma D.1. □

Lemma D.1. *Setting I (Lemma D.6) with ξ (Lemma D.5) guarantees with $\mathsf{genAuxInfo}$, by setting $\alpha = 0$, the construction satisfies $(\epsilon, \delta, \beta)$-$\mathsf{Search}$ correctness with $\beta = \sum_{i=1}^{size_r} I \cdot (3l_i + \mathsf{k_{max}} - 1)$.*

Proof. Since we are interested in the setting where $\mathsf{k_{max}} \ll n$ with n denoting the size of $\mathbf{M}$, it suffices to focus on the loop in the $(\epsilon, \delta, \beta)$-Search algorithm for correctness analysis. In particular, we break it down into Lemma D.2 and Lemma D.3 targeting at the termination of the loop. It is easy to see that the size restriction parameter β can be computed as follows,

$$\begin{aligned}\beta &\leq \max_{k\in[\mathsf{k_{max}}]} \sum_{i=1}^{size_r} I \cdot (3l_i + k - 1) \\ &\leq \sum_{i=1}^{size_r} I \cdot (3l_i + \mathsf{k_{max}} - 1).\end{aligned}$$

□

Lemma D.2. *If the loop stops after the Xth iteration where $X = size_r$, the output $\mathbf{Y}$ will include Q's k approximate nearest neighbors in $\mathbf{M}$ with at least $1 - \delta$ probability.*

Proof. Given approximation factor $c = 1 + \epsilon$, $\gamma = \sqrt{c}$ and $size_r = \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil$.

Proof by introduction on $1 \leq k \leq \mathsf{k_{max}}$:

Base case: $k = 1$. Suppose M_1 is $Q's$ nearest neighbor in $\mathbf{M}$ and $d(Q, M_1) = r_1$, then by definition of $r_{\min}$ and $r_{\max}$, $r_1 \in [\gamma^i \cdot r_{\min}, \gamma^{i+1} \cdot r_{\min}]$ for some integer $1 \leq i \leq size_r$, and eLSH instances will output a message $M_1' \in \mathbf{M}$ with

$$d(Q, M_1') \leq \gamma^{i+2} \cdot r_{\min} = c \cdot \gamma^i \cdot r_{\min} \leq c \cdot r_1,$$

which satisfies the correctness requirement of Q's approximate nearest neighbor in $\mathbf{M}$.

Inductive step: assume $k \leq \mathsf{k_{max}} - 1$ and the algorithm has already output Q's k approximate nearest neighbors in $\mathbf{M}$, then consider $k = k + 1$ case.

Suppose M_k is Q's kth nearest neighbor in $\mathbf{M}$ and $d(Q, M_k) = r_k$.

- Case $r_k = r_{k-1}$: the eLSH instances for (γ, r_k)-NN have already output kth approximate nearest neighbor, referring to Lemma D.4.
- Case $r_k > r_{k-1}$: we use the similar argument as in the base case, $r_k \in [\gamma^i \cdot r_{\min}, \gamma^{i+1} \cdot r_{\min}]$ for some integer $1 \leq i \leq \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil$, and the eLSH instances will output a message $M_k' \in \mathbf{M}$ with

  $$d(Q, M_k') \leq \gamma^{i+2} \cdot r_{\min} = c \cdot \gamma^i \cdot r_{\min} < c \cdot r_k.$$

  Thus, we have found Q's k approximate nearest neighbors in $\mathbf{M}$.

Since the algorithm involves multiple randomized procedures, it is left to compute the accumulated error. There are $size_r$ number of iterations in total, and we ensure that the properties in induction hold for all iterations.

For each iteration i with $i \in [size_r]$, we define events A_1 and A_2 as follows,

A_1: $r_k = r_{k-1}$ with the above property holds.
A_2: $r_k > r_{k-1}$ and base case with the above property holds.

Note that event A_1 and A_2 are mutually exclusive. $\Pr(A_1) \geq 1 - \tau_1$ ($\tau_1 = \mathsf{k_{max}} \cdot \xi$ in Lemma D.4) and $\Pr(A_2) \geq 1 - \xi$ (same as in Lemma D.6). Let $\tau = \max\{\tau_1, \xi\}$. We bound the accumulated error for X iterations. Let A_1^i, A_2^i denote event A_1, A_2 in the ith iteration respectively.

$$\begin{aligned}\Pr(\bigcap_{i=1}^{X}(A_1^i \cap A_2^i)) &= 1 - \Pr(\bigcup_{i=1}^{X} \overline{A_1^i \cap A_2^i}) \\ \text{(Union bound)} &\geq 1 - \sum_{i=1}^{X}[\Pr(\overline{A_1^i}) + \Pr(\overline{A_1^i})] \\ (A_1, A_2 \text{ mutually exclusive}) &\geq 1 - X \cdot \max\{\tau_1, \xi\} \\ &= 1 - \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil \cdot \tau.\end{aligned}$$

$\tau = \max\{\tau_1, \xi\}, X = size_r = \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil$.

We set parameters satisfying the following inequality in Lemma D.5,

$$\lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil \cdot \tau \leq \delta, \tag{1}$$

and thus completes the proof. □

Lemma D.3. *If the loop stops after the Xth iteration where $1 \leq X < size_r$, and the counter $\mathtt{ctrI} = I$, the output $\mathbf{Y}$ will include Q's k approximate nearest neighbors in $\mathbf{M}$ with at least $1 - \delta$ probability.*

Proof. We analyze the case when $\mathtt{ctrI} = I$, which occurs if and only if every i-th eLSH instance, $i \in [I]$ for (γ, r_X)-NN problem (Definition 5.1) outputs $3l_i + k - 1$ different messages in some X-th iteration, where $1 \leq X \leq size_r$. Due to Lemma D.2, we only need to focus on the situation where $1 \leq X < size_r$.

We prove the lemma using the property of the eLSH instances. Lemma D.6 guarantees that in at least one of I eLSH instances both events E_1 and E_2 (Definition E.1) hold with at least $1-\xi$ probability for ξ set by Lemma D.5. Moreover, condition $\mathtt{ctrI} = I$ implies that $|\mathbf{M} \cap \mathcal{B}(Q, \gamma \cdot r_X)| \geq k$, and those k points are included in the results with high probability.

We prove the lemma in a similar way as in Lemma D.2 with some variation due to condition $\texttt{ctrI} = I$.

Proof by introduction on $1 \le k \le \mathsf{k_{max}}$:

Base case: $k = 1$. Suppose M_1 is $Q's$ nearest neighbor in $\mathbf{M}$ and $d(Q, M_1) = r_1$, then by definition of $r_{\min}$ and $r_{\max}$, $r_1 \in [\gamma^i \cdot r_{\min}, \gamma^{i+1} \cdot r_{\min}]$ for some integer $1 \le i \le size_r$,

- Case $1 \le i \le X$: refer to Lemma D.2.
- Case $X < i \le size_r$: implies $M_k \notin \mathbf{M} \cap \mathcal{B}(Q, \gamma^X \cdot r_{\min})$, which contradicts $\texttt{ctrI} = I$.

Inductive step: assume $k \le \mathsf{k_{max}} - 1$ and the algorithm has already output Q's k approximate nearest neighbors in $\mathbf{M}$, then consider $k = k + 1$ case.

Suppose M_k is Q's kth nearest neighbor in $\mathbf{M}$ and $d(Q, M_k) = r_k$.

- Case $r_k = r_{k-1}$: refer to Lemma D.2.
- Case $r_k > r_{k-1}$: suppose $r_k \in [\gamma^i \cdot r_{\min}, \gamma^{i+1} \cdot r_{\min}]$ for some integer $1 \le i \le size_r$,
  - Case $1 \le i \le X$: refer to Lemma D.2.
  - Case $X < i \le size_r$: implies $M_k \notin \mathbf{M} \cap \mathcal{B}(Q, \gamma^X \cdot r_{\min})$, which contradicts $\texttt{ctrI} = I$.

Now we compute the accumulated error. Since we set parameters to ensure events A_1 and A_2 defined in Lemma D.2 hold for all iterations, which cover the case $1 \le X < size_r$, it is only left to bound the probability on the extra property we require in the induction holds when $\texttt{ctrI} = I$. Let set $\{M_1, \ldots, M_k\} \subset \mathbf{M}$ denote the Q's k messages that are within $\mathbf{M} \cap \mathcal{B}(Q, \gamma \cdot r_X)$. We need to ensure that they all have hash overlaps with Q with high probability in at least one of I eLSH instances, so they are included in the results.

$$\begin{aligned}
\Pr(\bigcap_{i=1}^{k} E_1(Q, M_i)) &= 1 - \Pr(\bigcup_{i=1}^{k} \overline{E_1(Q, M_i)}) \\
\text{(Union bound)} &\ge 1 - \sum_{i=1}^{k} \Pr(\overline{E_1(Q, M_i)}) \\
\text{(Lemma D.6)} &\ge 1 - k \cdot \xi \\
&\ge 1 - \mathsf{k_{max}} \cdot \xi \\
\text{(Lemma D.5)} &\ge 1 - \delta.
\end{aligned}$$

Hence completes the proof. □

Lemma D.4. *After the Xth iteration for $1 \le X \le size_r$, if there are $v_X \ge 1$ data points in $\mathbf{M} \cap \mathcal{B}(Q, r_X)$ with $r_X = \gamma^X \cdot r_{min}$, the set $\mathbf{Y}$ includes at least $\min\{v_X, k\}$ points in $\mathbf{M} \cap \mathcal{B}(Q, \gamma r_X)$ with constant probability.*

Proof. For all $i \in [size_r]$, we choose s_i, l_i parameters to ensure the same probability bound for solving (γ, r_i)-NN problem in all eLSH instances (Proposition E.2). In addition, we set parameter I to guarantee that, for each (γ, r_i)-NN problem, $i \in [size_r]$, in at least one of its I eLSH instances, event E_1 and event E_2 (Definition E.1) both hold with high probability (Lemma D.6). Without loss of generality, we assume at the Xth iteration, the jth instance satisfies this property where $j \in [I]$. Now consider the following discussion.

Let event B_1 denote $\sum_{u=1}^{L_X} |g_{X,j,u}^{-1}(g_{X,j,u}(Q)) \cap \mathbf{M}| \ge 3L_X + k - 1$; otherwise denoted by event B_2. Event B_1 and B_2 are mutually exclusive.

If $v_X \ge k$, then

- B_1: event E_2 guarantees that there are fewer than $3L_X$ eLSH hash overlaps with Q from the set $\mathbf{M} \setminus \mathcal{B}(Q, \gamma r_X)$, and event E_1 guarantees that those points in $\mathbf{M} \cap \mathcal{B}(Q, r_X)$ have high probability of hash collisions with Q, thus at least k messages in $\mathbf{M} \cap \mathcal{B}(Q, \gamma r_X)$ are included in $\mathbf{Y}$ with high probability for selected parameters.
- B_2: $\mathbf{Y}$ includes all data points which have eLSH hash collisions with Q.

If $v_X < k$, then

- B_1: with high probability this event will not occur, as it contradicts the assumption that both E_1 and E_2 hold.
- B_2: $\mathbf{Y}$ includes all v_X data points with high probability.

It is left to compute the error. Let $w = \min\{v_X, k\}$. We need to ensure that all w points are included. We require that $k \le \mathsf{k_{max}} \ll n$, and $v_X \le n$, where $n = |\mathbf{M}|$. We write $\mathbf{M} \cap \mathcal{B}(Q, r_X)$ as $\{M_i\}_{i \in [w]}$, then we obtain,

$$\begin{aligned}
\Pr(\bigcap_{i=1}^{w} E_1(Q, M_i)) &= 1 - \Pr(\bigcup_{i=1}^{w} \overline{E_1(Q, M_i)}) \\
\text{(Union bound)} &\ge 1 - \sum_{i=1}^{w} \Pr(\overline{E_1(Q, M_i)}) \\
\text{(since } w \le \mathsf{k_{max}}) &\ge 1 - \mathsf{k_{max}} \cdot \xi \\
&= 1 - o(1).
\end{aligned}$$

The last equality is due to $\mathsf{k_{max}} = o(n), \xi = o(n^{-1})$ and Lemma D.5. Thus completes the proof. □

Lemma D.5. *For fixed* $\mathsf{k}_{\max}$*, setting* I *in Lemma D.6 with* ξ *satisfying the following inequality ensures the* $(\epsilon, \delta, \beta)$-Search *correctness for some* β*,*

$$0 < \xi \leq \frac{\delta}{2 \cdot \mathsf{k}_{\max} \cdot \lceil \log_{1+\epsilon} \frac{r_{max}}{r_{min}} \rceil}.$$

Proof. In Lemma D.4, for all $X \in [size_r]$ iteration, we let $w = \min\{v_X, k\}$. Since we can only upperbound v_X by n, we instead limit $\mathsf{k}_{\max}$ to ensure $w = \min\{v_X, k\} \leq \mathsf{k}_{\max}$. Recall that $\tau = \max\{\tau_1, \xi\}$, and we have $\tau_1 = \mathsf{k}_{\max} \cdot \xi$ (Lemma D.4), which implies $\tau = \mathsf{k}_{\max} \cdot \xi$. Consider Inequality 1 in Lemma D.2,

$$\begin{aligned} \delta / \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil &\geq \tau \\ &= \mathsf{k}_{\max} \cdot \xi \end{aligned}$$

$$\xi \leq \frac{\delta}{\mathsf{k}_{\max} \cdot \lceil \log_\gamma \frac{r_{\max}}{r_{\min}} \rceil} = \frac{\delta}{2 \cdot \mathsf{k}_{\max} \cdot \lceil \log_{1+\epsilon} \frac{r_{\max}}{r_{\min}} \rceil}.$$

□

Lemma D.6. *Given set* $\mathbf{M}, \mathbf{Q}$*,* $\xi \in (0,1)$*, let* $n = |\mathbf{M}|, t = |\mathbf{Q}|$*, if there are* $I = \lceil \log_{1/e+1/3} \frac{\xi}{nt} \rceil$ *eLSH instances with same parameters (fixed by* (c, R)*-NN problem), then for all* $M \in \mathbf{M}$*, all* $Q \in \mathbf{Q}$*, there exists* $j \in [I]$ *such that*

$$\Pr[\textit{Both } E_1^j(Q, M) \textit{ and } E_2^j(Q) \textit{ hold }] \geq 1 - \xi.$$

Proof. For eLSH construction solving (c, R)-NN problem (Definition E.1), we choose parameter s, L to ensure both event E_1 and E_2 hold with probability at least $\frac{2}{3} - \frac{1}{e} \geq 0.299$ (Proposition E.2). We amplify the probability by having I copies of eLSH instances constructed with independent random coins.

We bound the error from above using union bound,

$$nt(1/e + 1/3)^I \leq \xi.$$

For construction, we can simply choose $I = \lceil \log_{1/e+1/3} \frac{\xi}{nt} \rceil$. □

To illustrate the existence of genAuxInfo, we provide a proposition setting distance threshold $r_{\min}$ and $r_{\max}$ based on uniform distribution for some constant $\alpha \in (0,1)$.

Proposition D.7. *In Hamming metric space where* $\mathcal{D} = \{0,1\}^d$*, if all the messages output by the source are sampled independently and uniformly at random from* $\{0,1\}^d$*, then for all (sufficiently large)* $\mathbf{M}, \mathbf{Q}$ *output by* $\mathcal{M}(1^\lambda)$ *with size restriction* $t \cdot 2^{-n} < 1$ *and* $t \cdot n \cdot d \cdot 2^{-d} < 1$*, where* $n = |\mathbf{M}|$ *and* $t = |\mathbf{Q}|$*, all positive integer* $\mathsf{k}_{\max} \ll |\mathbf{M}|$*, setting* $r_{min} = 1, r_{max} = d/2$ *guarantees that there exists some constant* $\alpha \in (0,1)$ *such that for each* $Q \in \mathbf{Q}$*, with at least* $1 - \alpha$ *probability (1) there exists a set* $\mathbf{S} \subset \mathbf{M}$ *with* $|\mathbf{S}| = \mathsf{k}_{\max}$ *such that* $Q \in \bigcap_{M \in \mathbf{S}} \mathcal{B}(M, r_{max})$*; (2)* $Q \notin \bigcup_{M \in \mathbf{M}} \mathcal{B}(M, r_{min})$*.*

Proof. In Hamming metric space where $\mathcal{D} = \{0,1\}^d$, without loss of generality, we assume d is even, and suppose $\mathbf{Q} = \{Q_1, \ldots, Q_t\}$ and $\mathbf{M} = \{M_1, \ldots, M_n\}$, where each element of both sets is sampled independently and uniformly from $\{0,1\}^d$.

Assume $1 \leq r_{\min} < r_{\max} \leq d/2$. For any $i \in [t]$, any $j \in [n]$, let event $A_{i,j,r_{\min}}$ denote $d(Q_i, M_j) > r_{\min}$, we obtain

$$\Pr(\overline{A_{i,j,r_{\min}}}) = \frac{\sum_{1 \leq j \leq r_{\min}} \binom{d}{j}}{2^d}.$$

Substitute $r_{\min}$ with 1, we obtain

$$\Pr(\overline{A_{i,j,r_{\min}}}) = d \cdot 2^{-d}.$$

Let random variable $X := \sum_{i=1}^t \sum_{j=1}^n \mathbb{1}_{\{\overline{A_{i,j,r_{\min}}}\}}$, then

$$\begin{aligned} \mathbb{E}X = \sum_{i=1}^t \sum_{j=1}^n \mathbb{E}\mathbb{1}_{\{\overline{A_{i,j,r_{\min}}}\}} &= \sum_{i=1}^t \sum_{j=1}^n \Pr(\overline{A_{i,j,r_{\min}}}) \\ &\leq t \cdot n \cdot d \cdot 2^{-d}. \end{aligned}$$

Let $\alpha \in (0,1)$, we bound the expectation from above,

$$\begin{aligned} \mathbb{E}X &\leq \alpha \\ t \cdot n \cdot d \cdot 2^{-d} &\leq \alpha. \end{aligned}$$

Then by Markov inequality, $\Pr(X \geq 1) \leq \mathbb{E}(X) \leq \alpha$. For any $i \in [t]$, any $j \in [n]$, let event $\overline{A_{i,j,r_{\max}}}$ denote that $d(Q_i, M_j) \leq r_{\max}$. Let event $B_{i,r_{\max}}$ denote that there exists a subset $\mathbf{S} \subset \mathbf{M}, |\mathbf{S}| = \mathsf{k}_{\max}$ s.t. $Q_i \in \bigcap_{M \in \mathbf{S}} \mathcal{B}(M, r_{\max})$.

$$\begin{aligned} \Pr(\overline{A_{i,j,r_{\max}}}) &= \frac{\sum_{1 \leq j \leq r_{\max}} \binom{d}{j}}{2^d} \\ &= 1/2. \end{aligned}$$

Substitute $\Pr(\overline{A_{i,j,r_{\max}}})$ with $1/2$ in the following equation,

$$\begin{aligned} \Pr(\overline{B_{i,r_{\max}}}) &= \Pr(\overline{A_{i,j,r_{\max}}})^{\mathsf{k}_{\max}-1} \Pr(A_{i,j,r_{\max}})^{n-\mathsf{k}_{\max}+1} \\ &= 2^{-n}. \end{aligned}$$

Similarly, let random variable $Y := \sum_{i=1}^t \mathbb{1}_{\{\overline{B_{i,r_{\max}}}\}}$, we have

$$\begin{aligned} \mathbb{E}Y &\leq \alpha \\ t \cdot 2^{-n} &\leq \alpha. \end{aligned}$$

By Markov inequality, $\Pr(Y \geq 1) \leq \mathbb{E}(Y) \leq \alpha$. □

E Extra Definitions & Propositions

Definition E.1. *Given parameter c, R, we choose s, l to ensure that with constant probability the following two events hold. We define the two events for any $q, p^* \in \mathcal{D}$,*

- *$E_1(q, p^*)$ occurs iff either $p^* \notin \mathcal{B}(q, R)$ or $p^* \in \mathcal{B}(q, R)$ and $g_j(p^*) = g_j(q)$ for some $j = 1, \ldots, l$.*
- *$E_2(q)$ occurs iff the total number of collisions of q with points from $P \setminus \mathcal{B}(q, cR)$ is less than $3l$, i.e.,*

$$\sum_{j=1}^{l} |(P \setminus \mathcal{B}(q, cR)) \cap g_j^{-1}(g_j(q))| < 3l.$$

Proposition E.2. *Setting $s = \lceil \log_{1/p_2} n \rceil$ and $l = \lceil n^\rho / p_1 \rceil$ guarantees E_1 and E_2 both hold with probability at least $\frac{2}{3} - \frac{1}{e} \geq 0.299$.*

Proof. Let P_1^*, P_2^* denote the probability that the event E_1, E_2 defined above holds respectively. We define $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}, s = \lceil \log_{1/p_2} n \rceil, l = n^\rho / p_1$, then it is easy to see that $P_1^* \geq p_1^s$.

We have

$$p_1^s \geq p_1^{\log_{1/p_2} n + 1} = p_1 \cdot n^{\log_n p_1^{\log_{1/p_2} n}}.$$

Together with

$$\log_n p_1^{\log_{1/p_2} n} = \log_{1/p_2} n \cdot \log_n p_1 = -\frac{\ln \frac{1}{p_1}}{\ln \frac{1}{p_2}} = -\rho.$$

Therefore,

$$P_1^* \geq p_1^s = p_1 \cdot n^{-\rho}.$$

Also, let q be some query point from the domain $\mathcal{D}$, p' be a point in $P \setminus \mathcal{B}(q, cR)$, and p^* be a point in $\mathcal{B}(q, cR)$.

Then the probability for $g_j(p') = g_j(q)$ for any $j \in [l]$ is at most p_2^s,

$$P_2 = \Pr[g_j(p') = g_j(q)] \leq p_2^s = p_2^{\log_{1/p_2} n} = \frac{1}{n}.$$

For all $j \in [l]$, let random variable X_j denote the total number of hash overlaps of all points that are in $P \setminus \mathcal{B}(q, cR)$ with query point q under g_j, i.e.,

$$X_j := |(P \setminus \mathcal{B}(q, cR)) \cap g_j^{-1}(g_j(q))|.$$

Then we have,

$$\mathbb{E}(X_j) = \sum_{v \in P \setminus \mathcal{B}(q, cR)} P_2' \leq n \cdot \frac{1}{n} = 1.$$

Let random variable Y denote the sum of all hash overlaps for points that are in $P \setminus \mathcal{B}(q, cR)$, i.e.,

$$Y := \sum_{j=1}^{l} |(P \setminus \mathcal{B}(q, cR)) \cap g_j^{-1}(g_j(q))|.$$

By linearity of expectation,

$$\mathbb{E}(Y) = \mathbb{E} \sum_{1}^{l} X_i = l.$$

By Markov inequality,

$$\Pr[Y \geq 3l] \leq \frac{\mathbb{E}(Y)}{3l} = \frac{l}{3l} \leq \frac{1}{3}.$$
$$P_2^* = \Pr[Y < 3l] \geq 1 - \frac{1}{3} = \frac{2}{3}.$$

Then setting $l = \lceil n^\rho / p_1 \rceil$, we bound P_1^* from below,

$$P_1^* = 1 - (1 - n^{-\rho} \cdot p_1)^l \geq 1 - e^{-n^{-\rho} \cdot p_1 \cdot n^\rho / p_1} = 1 - e^{-1}.$$

The lower-bound on the probability that both properties hold is, by union bound $1 - [(1 - P_1^*) + (1 - P_2^*)] = P_1^* + P_2^* - 1 \geq \frac{2}{3} - \frac{1}{e}$. □