Sub-Linear Privacy-Preserving Near-Neighbor Search with Untrusted Server on Large-Scale Datasets

M. Sadegh Riazi*
UCSD
9500 Gilman Dr
San Diego, CA
mriazi@ucsd.edu

Beidi Chen*
Rice University
6100 Main St
Houston, TX
beidi.chen@rice.edu

Anshumali Shrivastava
Rice University
6100 Main St
Houston, TX
anshumali@rice.edu

Dan Wallach
Rice University
6100 Main St
Houston, TX
dwallach@cs.rice.edu

Farinaz Koushanfar
UCSD
9500 Gilman Dr
San Diego, CA
farinaz@ucsd.edu

ABSTRACT

In Near-Neighbor Search (NNS), a new client queries a database (held by a server) for the most similar data (near-neighbors) given a certain similarity metric. The Privacy-Preserving variant (PP-NNS) requires that neither server nor the client shall learn information about the other party's data except what can be inferred from the outcome of NNS. The overwhelming growth in the size of current datasets and the lack of a truly secure server in the online world render the existing solutions impractical; either due to their high computational requirements or non-realistic assumptions which potentially compromise privacy. PP-NNS having query time *sub-linear* in the size of the database has been suggested as an open research direction by Li et al. (CCSW'15). In this paper, we provide the first such algorithm, called Secure Locality Sensitive Indexing (SLSI) which has a sub-linear query time and the ability to handle honest-but-curious parties. At the heart of our proposal lies a secure binary embedding scheme generated from a novel probabilistic transformation over locality sensitive hashing family. We provide information theoretic bound for the privacy guarantees and support our theoretical claims using substantial empirical evidence on real-world datasets.

1. INTRODUCTION

Near-Neighbor Search (NNS) is one of the most fundamental and frequent tasks in large-scale data processing systems. In NNS problem, a server holds a collection of users' data; a new user's objective is to find all similar data to her query given a certain similarity metric. NNS is used in personal recommendations (of friends, events, movies, etc.) especially based on neighborhood models [61], online classification based on K-NN search, face recognition [50], secure biometric authentication [3, 5], privacy-preserving speech recognition [42], etc. The demand for privacy in big-data systems has led to an increasing interest in the problem of Privacy-Preserving Near-Neighbor Search (PP-NNS). In PP-NNS, all of the clients' data must remain private to their respective owners. This implies that not only server(s), but also a new client who queries the database, should not learn information about other clients' data except the NNS result.

The above setting is natural and ubiquitous in the online world where matching and recommendations are common [43]. For example, on dating websites, a client is interested in finding similar profiles (near neighbors) without revealing her attributes to anyone. Note that, it is problematic to assume any trusted server in real settings. For instance, privacy breaches where the data servers have been compromised for a significant blocks of data are common in recent times. A well-publicized recent example is Yahoo's massive leak, which compromised 500 million user accounts including private information such as phone number, date of birth, or even answers to security questions [1]. It is, therefore, desirable that the protocol does not rely on the complete security of participating servers and even if data from the server is compromised, the user's information must remain secure.

Keeping in mind both big-data and modern security challenges, four main requirements have to be satisfied: (i) one shall not assume any trusted server, (ii) data owners (clients) are not trusted, (iii) modern datasets are very high dimensional, and (iv) the query time must be sub-linear (near constant) in the number of clients (or database size) in order to handle web-scale datasets. Finding sub-linear privacy-preserving solution without any trusted party is currently considered to be a critical, yet open, research direction as stated in a recent article [36].

Due to the importance of the PP-NNS problem, there have been many attempts to create a practical solution. In theory, any function (e.g., NNS) with inputs from different

*Indicates equal contribution of the first two authors.

parties can be evaluated securely without revealing the input of each party to another using Secure Function Evaluation (SFE) protocols such as Garbled Circuit (GC) protocol. While the SFE protocols have been continually improving in efficiency, they still suffer from huge execution times and massive communication between executive servers. In addition, realizing NNS with any of the SFE protocols faces the scalability issue. These protocols scale (at best) linearly with respect to the size of the database [47], undermining requirement four. As we describe later, we only utilize the GC protocol for a small part of the computation.

Supporting NNS on encrypted data is an active area of recent research [65, 28, 67, 19]. Unfortunately, available crypto-based solutions fail to support high dimensional data and they usually require multiple rounds of communication between user and the server. Mylar [45] is a system for web applications that works on top of encrypted data which is proved to be insecure by Grubbs et al. [25]. One of the most adopted solutions is Asymmetric Scalar-Product-preserving Encryption (ASPE) [65]. However, not only this scheme has linear query complexity in terms of the size of the database, it has been proven to be insecure against chosen plaintext attack by Yao et al. [67]. More generally, they have proved that secure NNS is at least as hard as Order Preserving Encryption (OPE). Since it has been proven that it is impossible to have secure OPE under standard security models [6, 7], *it is not feasible to have a Secure NNS under standard security models such as Ciphertext Indistinguishability under chosen Plaintext Attack (IND-CPA).* In this paper, we define ϵ-security and show that our solution limits the information leakage (for any arbitrary upper bound) while having a practical sub-linear PP-NNS.

We propose Secure Locality Sensitive Indexing (SLSI) as a practical solution for the sub-linear PP-NNS on high dimensional datasets. Performing NNS on a very high dimensional database is a non-trivial task even when data privacy is not considered a constraint. For example, NNS algorithms based on k-d trees are marginally better than exhaustive search [24]. Our solution has two main components: (i) a novel probabilistic transformation over locality sensitive hashing family (Section 5) and a (ii) secure black-box hash computation method based on the GC protocol (Section 6).

Locality Sensitive Hashing (LSH) is the only line of work which guarantees sub-linear query time approximate near-neighbor search for high-dimensional datasets [29]. One fundamental property of LSH-based binary embedding is that it preserves all pairwise distances with little distortion [31], eliminating the need for sharing original attributes. However, the bits of the binary embeddings have enough information to estimate any pairwise distance (or similarity) between any two users [4], which makes them unsuitable in settings with no trusted party. *We argue that the ability to estimate all pairwise distances is sufficient but not necessary for the task of near-neighbor search.* In fact, we show for the first time, that the ability to estimate distances compromises the security of LSH-based embedding; rendering them susceptible to "triangulation" attack (see Section 5.1). In this work, we eliminate the vulnerability of LSH with minimal modification while not affecting the sub-linear property.

The hash computation process of SLSI takes as input random seeds. These seeds should be consistent among different users' profiles. However, if they are made public, it can be used by an attacker to reconstruct the users' data from the hashes. Therefore, we utilize the GC protocol to mask the hash computation process from both servers and the users as we detail in Section 6. Please note that the GC protocol execution is a one-time process for each store and/or search operation and is performed independently of all other profiles in the database. Therefore, it requires a negligible query time.

Contributions. Our main contributions are as follows:

- We propose the first algorithm for PP-NNS with query time sub-linear in the number of clients. No trusted party or server is needed for handling sensitive data.
- We introduce the first generic transformation which makes any given LSH scheme secure for public release in honest-but-curious adversary setting. Our transformation prevents the estimation of all pairwise distances which is unnecessary for the task of near-neighbor search. This advantage comes at no additional cost, and we retain all the properties of LSH required for the sub-linear search.
- We give information theoretic guarantees on the security of the proposed approach. Our proposed transformation, analysis, and the information theoretic bounds are of independent theoretical interest.
- We provide a practical implementation of *triangulation attack* for compromising the security of LSH signatures in high dimensions. Our attack is based on alternating projections. The proposed attack reveals the vulnerability and unnecessary information leakage by the LSH embeddings. In general, we experimentally verify that the ability to estimate all pairwise distances is sufficient for recovering original attributes.
- We support our theoretical claims using substantial empirical evidence on real-world datasets. We further provide the first thorough evaluation of accuracy-privacy trade-off and its comparison with noise-based privacy. Our scheme can process queries against a database of size 3 Billion entries in *real time* on a typical PC. Performing the same task with the state-of-the-art Yao's Garbled Circuit SFE protocol requires an estimated time of 1.5×10^8 seconds and 1.2×10^7 GBytes of communication (see Section 8.3).

2. PRELIMINARIES AND BACKGROUND

In this section, we briefly review our notation. Then, we discuss our threat model followed by a background on LSH. Finally, we explain how LSH is currently used for large-scale near-neighbor search when the server is trusted. Please refer to [29, 30] for more specific details.

2.1 Key Notations and Terms

A server holds a giant collection $\mathcal{C}$ of clients (or data owners), each represented by some D dimensional attribute vectors, i.e., $\mathcal{C} \subset \mathbb{R}^D$. We are interested in finding the answers to queries. The objective is

$$\arg\max_{x \in \mathcal{C}} Sim(x, q),$$

where $Sim(.,.)$ is a desired similarity measure. However, the process should prevent any given (possibly dishonest) client from inferring the attributes of other clients, except for the information that can be inferred from the answer of the NNS queries.

We interchangeably use the terms clients, users, data owners, vectors, and attributes. They all refer to the vectors in the collection $\mathcal{C}$. Unless otherwise stated, the hash functions h will produce a 1-bit output, i.e., $h(x) \in \{0,\ 1\}$. All the hash functions are probabilistic, and in particular, there is an underlying family (class) of hash functions $\mathcal{H}$ and h is drawn uniformly from this family. The draw can be conveniently fixed using random seeds. Our protocol will require some l-bits embedding and each of these l-bits will be formed by concatenating l independent draws h_i $i \in \{1, 2, ..., l\}$ from some family of hash functions. Similarity search and the near-neighbor search will mean the same thing. Similarity and distances can be converted into each other using the formula distance = 1 - similarity. For any hash function h, the event $h(x) = h(y)$, for given pair x and y, will be referred to as the collision of hashes.

2.2 Threat Model

There are two types of parties involved in our model: servers and clients (data owners). The models in previous works, for example [36], usually consider trusted servers. In many practical scenarios, such as medical records, this is a problematic assumption since the clients often do not desire to share any personal information due to the potential threats common in the online environment. In this paper, we assume Honest-but-Curious (HbC) adversary model for both data owners and servers. In this threat model, each part is assumed to follow the protocol but is curious to extract as much information as possible about other party's secret data. While we do not trust any server, we assume that the servers do not collude with each other. Please note that this is the exact security model of the prior art [19]. We want to emphasize that the assumption of two non-colluding HbC servers is feasible since two servers can represent two different companies, e.g. Amazon and Microsoft. Due to the business reasons and the fact that any collusion will significantly damage their reputation, it would be very unlikely that two companies will collude since it would be against their interests.

The solutions based on the GC protocol, Fully Homomorphic Encryption (FHE), and Oblivious RAM (ORAM) do not leak *any* information about the database and the query [39] other than what can be inferred from the answer of NNS. All other solutions leak some information either in the setup phase (creating the database) or the query phase. Unfortunately, GC, FHE, and ORAM solutions are computationally too expensive to be employed in real-world [65]. In this paper we compare the performance of our proposed solution (SLSI) with GC. In addition, we formalize ϵ-security and prove that the information leakage in our scheme can be made as small as required by tuning a privacy parameter in the protocol. We also compare our solution to the noise addition-based techniques and illustrate, both experimentally and theoretically, that our solution has significantly higher precision/recall for the same security limits.

Note that the answer to NNS may reveal some information about the query and/or the database, regardless of implementation details and security guarantees of any protocol. For example, if client i and j are very close w.r.t similarity measure (near identical), then the near-neighbor query of client i should return j as the correct answer (with a high probability). A correct answer automatically reveals information that j's attributes are likely to be very similar to i's attributes (with a high probability) even without having knowledge of the other client's attributes. This kind of information leak cannot be avoided by any algorithm answering the near-neighbor query with a reasonable accuracy. In addition, the privacy guarantees in the near-neighbor setting all rely on the inherent assumption on bounded computations. Given unbounded computations, the adversary can enumerate the whole space of every possible vector and use near-neighbor query until the generated vector returns the target client as the neighbor. In high dimensions, this process will require exponential computations due to the curse of dimensionality, which turns out to be a boon for the privacy of NNS.

2.3 Locality Sensitive Hashing

A popular technique for approximate near-neighbor search uses the underlying theory of *Locality Sensitive Hashing* [29]. LSH is a family of functions with the property that similar input objects in the domain of these functions have a higher probability of colliding in the range space than non-similar ones. In formal terms, consider $\mathcal{H}$ a family of hash functions mapping $\mathbb{R}^D$ to some set $\mathcal{S}$.

DEFINITION 2.1 (**LSH Family**). *A family $\mathcal{H}$ is called (S_0, cS_0, p_1, p_2)-sensitive if for any two points $x, y \in \mathbb{R}^D$ and h chosen uniformly from $\mathcal{H}$ satisfies the following:*

- *if $Sim(x, y) \geq S_0$ then $Pr(h(x) = h(y)) \geq p_1$*
- *if $Sim(x, y) \leq cS_0$ then $Pr(h(x) = h(y)) \leq p_2$*

For approximate nearest neighbor search typically, $p_1 > p_2$ and $c < 1$ is needed. An LSH allows us to construct data structures that give provably efficient query time algorithms for the approximate near-neighbor problem with the associated similarity measure.

One sufficient condition for a hash family $\mathcal{H}$ to be a LSH family is that the **collision probability** $Pr_{\mathcal{H}}(h(x) = h(y))$ is monotonically increasing function of the similarity, i.e.

$$Pr_{\mathcal{H}}(h(x) = h(y)) = f(Sim(x, y)), \tag{1}$$

where f is a monotonically increasing function. In fact most of the popular known LSH families, such as MinHash (Section 2.4) and SimHash (Section 2.5), actually satisfy this stronger property. It can be noted that Equation 1 automatically guarantees the two required conditions in the Definition 2.1 for any S_0 and $c < 1$.

It was shown [29] that having an LSH family for a given similarity measure is sufficient for efficiently solving near-neighbor search in sub-linear time:

DEFINITION 2.2. *Given a family of (S_0, cS_0, p_1, p_2)-sensitive hash functions, one can construct a data structure for c-NN with $O(n^\rho \log n)$ query time and space $O(n^{1+\rho})$, where $\rho = \frac{\log p_1}{\log p_2} < 1$.*

2.4 Popular LSH 1: Minwise Hashing (MinHash) and Resemblance Similarity

One of the most popular measures of similarity between web documents is resemblance (or Jaccard similarity) $\mathcal{R}$ [10]. This similarity measure is only defined over sets which can be equivalently thought of as binary vectors over the universe, with non-zeros indicating the elements belonging to the given set.

The resemblance similarity between two given sets x, $y \subseteq \Omega = \{1, 2, ..., |\Omega|\}$ is defined as

$$\mathcal{R} = \frac{|x \cap y|}{|x \cup y|} = \frac{a}{f_1 + f_2 - a}, \tag{2}$$

where $f_1 = |x|$, $f_2 = |y|$, and $a = |x \cap y|$.

Minwise hashing [11] is the LSH for resemblance similarity. The minwise hashing family applies a random permutation $\pi : \Omega \to \Omega$, on the given set x, and stores only the minimum value after the permutation mapping. Formally MinHash and its collision probability is given by

$$h_\pi^{min}(x) = \min(\pi(x)); \quad Pr(h_\pi^{min}(x) = h_\pi^{min}(y)) = \mathcal{R}. \tag{3}$$

2.5 Popular LSH 2: Signed Random Projections (SimHash) and Cosine Similarity

SimHash is another popular LSH for the cosine similarity measure, which originates from the concept of Signed Random Projections (SRP) [14, 46, 27]. Given a vector x, SRP utilizes a random w vector with each component generated from i.i.d. normal distribution, i.e., $w_i \sim N(0,1)$, and only stores the sign of the projection. Formally SimHash is given by

$$h_w^{sign}(x) = sign(w^T x). \tag{4}$$

It was shown in the seminal work [23] that collision under SRP satisfies the following equation:

$$Pr(h_w^{sign}(x) = h_w^{sign}(y)) = 1 - \frac{\theta}{\pi}, \tag{5}$$

where $\theta = cos^{-1}\left(\frac{x^T y}{||x||_2 \cdot ||y||_2}\right)$. The term $\frac{x^T y}{||x||_2 \cdot ||y||_2}$, is the cosine similarity. There is a variant of SimHash where, instead of $w_i \sim N(0,1)$, we choose each w_i independently as either +1 or -1 with probability $\frac{1}{2}$. It is known that this variant performs similar to the one with $w \sim N(0,1)$ [46]. Since $1 - \frac{\theta}{\pi}$ is monotonic with respect to cosine similarity $\mathcal{S}$, SimHash is a valid LSH.

2.6 Mapping LSH to 1-bit

LSH, such as MinHash, in general, generates an integer value, which is expensive from the storage perspective. It would gain a lot of benefits from having a single bit hashing schemes, or binary locality sensitive bits. It is also not difficult to obtain 1-bit LSH. The idea is to apply a random universal hash function to the LSH and map it to 1-bit.

A commonly used universal scheme is given by

$$h_{1bit}(x) = a \times x \mod 2, \tag{6}$$

where a is an odd random number, see [12] for more details. With this 1-bit mapping, any hashing output $h(x)$ can be converted to 1-bit by applying universal 1-bit hash function $h_{1bit}(.)$. Collision probability of this new transformed 1-bit hashing scheme is given by

$$Pr(h_{1bit}(h(x)) = h_{1bit}(h(y))) = \frac{Pr(h(x) = h(y)) + 1}{2}. \tag{7}$$

It is not difficult to show that $h_{1bit}(h(x))$ is also a valid LSH familiy for the same similarity measure associated with $h(.)$ [14, 51]. Another convenient (and efficient) 1-bit rehashing is to use the parity, or the most significant bit, of $h_\pi^{min}(x)$ as 1-bit hash [51].

2.7 PP-NNS in Sub-linear Time with a Trusted Server

In the trusted server settings, LSH-based protocols are well-known for privacy-preserving near-neighbor search [30]. The protocol with sub-linear query time search involves the following two major steps.

1. **Hash Function Generation and Computation (Pre-processing):** The trusted server fixes random seeds for hash functions. Every client $x \in \mathcal{C}$ sends its attributes to the server. The server computes the l-bit binary embedding $E(x)$, using appropriate (pre-chosen) LSH schemes $h_i(x)$s. Computing l bits involves generating multiple 1-bit hashes using independent randomization and concatenating them $E(x) = [h_1(x); h_2(x); ...; h_l(x)]$, where $h_i(x)$ is an independent hashing scheme. The server also generates hash tables, as a part of preprocessing for sub-linear time search. New clients can be dynamically inserted into the tables.

2. **Sub-linear Searching with Hamming Distance (Querying):** To find near-neighbors of any given query point q, the trusted server computes the l-bit embedding of q, $E(q)$, using the pre-decided function $E(.)$. Due to the LSH property of E, it suffices to find points $y \in \mathcal{C}$ such that $E(q)$ and $E(y)$ are close in Hamming distance. Searching for close Hamming distance can be done very efficiently in sub-linear time by only probing few buckets in the pre-constructed hash tables [30].

The above protocol requires a trusted server which handles all the data. The security relies on the fact that no client is allowed to see any part of the computation process. The sub-linearity of the search is due to the classical sub-linear LSH algorithm for Hamming distance search [46].

3. CHALLENGES WITH UNTRUSTED SERVER AND TYPES OF ATTACKS

For obtaining sub-linear solutions, we do not have many choices. LSH-based techniques are the only known methods that guarantee efficient sub-linear query time algorithms even in high dimensions [22]. Thus, one cannot hope to deviate from the philosophy of generating a binary embedding for data vectors x, given by $E(x)$, which preserves original near-neighbors in the obtained Hamming space.

Conceptually, there can be three types of attacks when the server is untrusted: (i) querying the database and brute-forcing the space of inputs (ii) extracting the original attribute vector from the hashes using compressive sensing theory, and (iii) analyzing the combination of hashes and measuring their mutual correlation to estimate the original attribute of a user.

Brute-force/Probing Attacks. An attacker can ask for the hash embedding of a random attribute vector and check whether it is equal to another user's hash (if the database is compromised). However, exploring the entire input space is computationally infeasible. If each element of the attribute vector is represented as a 32-bit number, we have $(2^{32})^D$ possible unique inputs. National Institute of Standards and Technology (NIST) states that any attack that requires 2^{128} operations is computationally infeasible [62]. For example, for the two datasets that are

Phases	User(s)		Servers
1. Generating Random Seeds			- server 1 generates random $seeds_1$ (off-line) - server 2 generates random $seeds_2$ (off-line)
2. Constructing the Database (Hash Tables)	for all x_is in $\mathcal{C}$, each data owner computes random pad v_i and $Enc_{vi}(x_i)$	v_i to server 1 $\rightarrow$ $Enc_{vi}(x_i)$ to server 2	- two servers jointly perform black-box computation of $S(x_i)$: server 1 inputs: $seeds_1$ and v_i, server 2 inputs: $seeds_2$ and $Enc_{vi}(x_i)$ $\rightarrow$ output: $S(x_i)$ - server 1 adds the new $S(x_i)$ to the hash tables
3. Searching (Querying)	new user with query q computes random pad v_q and $Enc_{vq}(q)$	v_q to server 1 $\rightarrow$ $Enc_{vq}(q)$ to server 2 $\leftarrow$ NNS(q)	- two servers jointly perform black-box computation of $S(q)$: server 1 inputs: $seeds_1$ and v_q, server 2 inputs: $seeds_2$ and $Enc_{vq}(q)$ $\rightarrow$ output: $S(q)$ - server 1 searches the hash tables using $S(q)$ and finds NNS(q)

Figure 1: The SLSI scheme consists of three phases: (i) generating random seeds, (ii) constructing the database, and (iii) searching phase. The internal mechanism of $S(.)$ is explained in Seciton 5. Black-box hash computation of $S(x)$ is described in Section 6 which is based on the GC protocol and takes as input $Enc_v(x)$ (encryption of x) and the pad v.

considered in this paper, $D \geqslant 186$. Thus, there are 2^{5952} possible inputs which are far beyond the security standards. Note that each element of attribute vector might not have uniform distribution, e.g., the value for *age* is typically a number between 0 and 100. Therefore, each number may not have maximum randomness (entropy). However, even if each number has minimum randomness (1-bit entropy), for any input vector with $D \geqslant 128$, the attack is not possible.

Compressive Sensing/Reconstruction Attacks. The theory of compressed sensing makes it possible to approximately recover x from the hash embedding $E(x)$ given the random seeds used in $E(.)$. Thus, we need to ensure that neither users nor the server have any information about the random seeds used in the computation of $E(.)$. Every embedding $E(x)|x \in \mathcal{C}$, however, should be created using identical random seeds. We show that using secure function evaluation protocols, it is possible to create secure binary embeddings using the same set of random seeds while noone knows the seeds used in the hashing function. We describe the solution in Section 6. Since the generation of the hash embedding is a one-time operation, it is allowed to be costlier as it is independent of other query processes.

Multilateration/Correlation/Triangulation Attacks. Although, recovering x from $E(x)$ is not possible without knowing the random seeds inside h_i $\forall i$, it is still possible to recover x from $E(x)$ by combining a "few" calls to the function $E(.)$ over few known inputs $y_i's$ (similar to chosen-plaintext attack). The LSH property allows the estimation of any pairwise distance. Such estimations open room for "triangulation" attack which is hard to prevent. We explain the problem and the solution in Section 5. This information leakage with LSH is one of the major reasons why sub-linear search with semi-honest clients and absence of trusted party is an open research direction.

We use a novel probabilistic transformation to show that converting the bits generated from LSH family into secure bits is suitable for public release in the semi-honest model since it is secure against triangulation attack. Our secure bits preserve only the near-neighbors in Hamming space, unlike LSH, do not allow estimation of all possible distances. Our final l-bit embedding functions will be denoted by $S(x)$ instead of $E(.)$ to signify the secure nature of $S(.)$. Our solutions for making LSH secure is the main contributions of this paper, which makes sub-linear time PP-NNS possible in the semi-honest setting with no trusted party. In the process, we fundamentally leverage the theory of LSH from the privacy perspective.

Before we describe the technical details of our solution in Sections 5 and 6 respectively, we briefly give an overview of our final protocol.

4. PROPOSED SLSI PROTOCOL FOR SUB-LINEAR QUERY TIME PP-NNS

The security of the final protocol is based on the proposed secure LSH (described in Section 5). Utilizing Secure LSH, we can generate l-bit embeddings, for some l, $S(.)$, such $S(x)$ is safe for public release. Assuming that we know such embedding $S(.)$, our final protocol for sub-linear query time PP-NNS works in three phases. The first phase is the one-time random seeds generation (off-line). Next phase accounts for one-time pre-processing stage and making the database (hash tables). The third phase is the online query phase. More precisely:

1. Generating Random Seeds of $S(.)$: This process needs to be performed only once and does not require any communication between servers (off-line). Two servers generate random seeds that are required in the black-box hash computation of $S(.)$ (in phase two and three). The final internal random seeds of $S(.)$ is not known to anyone and is secure. The mathematical detail of secure LSH embedding, $S(.)$, are described in Section 5 while the details on its black-box computation are described in Section 6.

2. Constructing the Database (Hash Tables): Every client x computes his/her l-bit secure binary embedding $S(x)$ using black-box hash computation by communicating to the servers. This l-bit signature $S(x)$ serves as the secure public identifier for client x. Server #1 which possesses all $S(x)$s, pre-processes the collection of l-bit binary strings $\{S(x) : x \in \mathcal{C}\}$ to create hash tables using the classical algorithm for sub-linear time search with Hamming distance.

3. Searching in Sub-Linear Time (Query Phase): To find near-neighbors of point x, it suffices to find points y such that the corresponding secure embeddings, $S(x)$ and $S(y)$, are near-neighbors in Hamming distance. Searching for close Hamming distance can be done very efficiently in sub-linear time using the well-known algorithms [46].

It should be noted that other than the set $S_{\mathcal{C}} = \{S(x) : x \in \mathcal{C}\}$, no information is transfered between clients. Hence if $S_{\mathcal{C}}$ is not sufficient to recover any of the client's information, the protocol is secure. For better readability we summarize the end-to-end protocol in Figure 1.

5. THE KEY INGREDIENT: LSH TRANSFORMATION

We explain why traditional LSH (or any scheme) which allows for estimation of *any* pairwise distance is not secure in the HbC adversary model. We describe the attack followed by its solution. We later formalize the privacy budget.

5.1 "Triangulation" Attack

To give more insight into the situation, we describe *triangulation attack* which leads to an accurate estimation of any target client's attribute q. For illustration, we focus on two dimensions, but the arguments naturally extend in higher dimensions. Assume that we are given the LSH embedding $E(q)$ of the target point q (instead of secure embedding $S(.)$). An attacker, who wants to know the attributes of q, can create three random data (points) in the space A, B, and C. Creating few random points is not hard, e.g., fake online profiles with random attributes. The protocol allows computation of their LSH embeddings $E(A)$, $E(B)$, and $E(C)$ via the publicly available function $E(.)$.

Given the random points A, B, C, and their corresponding hashes $E(A)$, $E(B)$, and $E(C)$, the attacker can compute the number of matches between the hash values of $E(A)$, $E(B)$, and $E(C)$ with the target hash, $E(q)$. Using these number of matches, the distances of q with A, B, and C, denoted by d_A, d_B and d_C, can be accurately estimated from their corresponding binary embeddings [4].

Estimation of Distances from LSH Embeddings: Let us focus on estimating d_A from l-bit binary LSH embedding $E(A)$ and $E(q)$. For illustrations let l be equal to 5 and $E(A) = 11010$ and $E(q) = 10110$. Let m be the measured number of bit matches between $E(A)$ and $E(q)$ out of l. For our case, we have $m = 3$, because bit numbered 1, 4 and 5 of $E(A)$ and $E(q)$ are equal. Since every bit comes from an independent 1-bit LSH scheme, we have $\mathbb{E}[n_{match}] = l \times Pr(h_i(q) = h_i(A)) = m$, where n_{match} is the number of bit matches between two LSH embeddings and $\mathbb{E}[.]$ denotes the expected value of a random variable.

Thus we can estimate, in an unbiased way, the collision probability $Pr(h(A) = h(q))$ by the expression $\frac{m}{l}$, the mean number of bit matches. As we discussed in Section 2, the collision probability is usually a monotonic function of the distance (or similarity) $Pr(h(A) = h(q)) = f(dist(A, q))$ where f is the monotonic function. Every monotonic function has an inverse, thus

$$dist(A, q) = f^{-1}\left(\frac{m}{l}\right),$$

is an accurate estimator of the distance or similarity [51, 37, 38]. See Section 9 for details where we describe the implementation of triangulation attack.

After estimating the distances d_A, d_B and d_C, the attributes of q can be inferred using triangulation. Figure 2 shows a two-dimensional illustration of our setup.

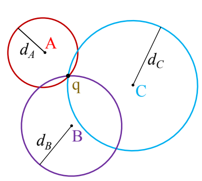

Figure 2: The user q can be located using random points A, B, and C along with the distances d_A, d_B, and d_C which are estimated from the available hashes.

It should be noted that even if the distance estimation is not very accurate, generating many distance estimates from different random points would be sufficient to achieve a very good accuracy in locating any target point.

The above illustration only shows two dimensions. For higher dimensions, we show an efficient iterative process, using the idea of alternating projections [9], to infer the attributes even for high-dimensional vectors. In Section 9, we describe the process in details. Our inference process shows the power of simple iterative machine learning in breaking the security, which itself can be of independent interest. The ease of triangulation-based inference of attributes further emphasizes the need for more secure hashing schemes which we propose in the next section.

5.2 Probabilistic Transformations for Generating Secure LSH

Our proposal is a generic framework for making any given LSH privacy-preserving. In particular, we prevent LSH from leaking the distance information without compromising on the accuracy of the near-neighbor search.

We illustrate the main idea using 1-bit MinHash and later we formally introduce the methodology. The collision probability, for any two given data points x and y, under 1-bit MinHash is given by $\frac{\mathcal{R}(x,y)+1}{2}$ (Equation 7). This quantity varies linearly, between 1 to 0.5 as $\mathcal{R}(x, y)$ varies from 1 to 0, with a constant gradient of $\frac{1}{2}$. Thus, even when $\mathcal{R}(x, y)$ is small, the variation of the collision probability with distance keeps changing and gets reflected in the Hamming distance between the public l-bit hash strings. This property allows us to estimate the distances accurately by counting the number of bit matches out of the l-bits which are public. For example if 65% of bits matches, then a good estimate of similarity between x and y is $0.65 \times 2 - 1 = 0.3$ (Equation 7).

To make LSH privacy-preserving without losing the accuracy in near-neighbor search tasks, it is necessary to have the flat collision probability with no gradient if the similarity between the pair x and y is below the satisfactory level. Thus, for any pair of random points x and y, the Hamming distance between the publicly available l-bit hash codes is around $l/2$ (due to the 0.5 probability of agreement), which prohibits the estimation of distances between x and y.

Until now, we have realized that we need to transform the collision probability. The primary challenge is to find the precise expression for the curve which has the desired behavior and at the same time represents the collision probability of some 1-bit hashing scheme. It should be noted that not every curve is a collision probability curve [14], therefore, it is not even known if such a mathematical expression exists.

We show that the expression given by $\frac{\mathcal{R}(x,y)^k+1}{2}$, for some large enough k, has the required "sweet" property. In particular, we construct a new 1-bit secure MinHash with collision probability $\frac{\mathcal{R}(x,y)^k+1}{2}$ for any positive integer k, instead of $\frac{\mathcal{R}(x,y)+1}{2}$. The observation is that since $\mathcal{R} \leq 1$, $\mathcal{R}^k$ for reasonably large k quickly falls to zero as $\mathcal{R}(x, y)$ goes away from 1. Therefore, the quantity $\frac{\mathcal{R}(x,y)^k+1}{2}$ will be very close to $\frac{1}{2}$ for even moderately high similarity. Furthermore, we can control the decay of the probability curve by choosing k appropriately. The function $\frac{\mathcal{R}(x,y)^k+1}{2}$ follows the desired trend of collision probability and is secure from information theoretic perspective.

The key mathematical observation is that we can generate 1-bit hash functions with collision probability $\frac{\mathcal{R}(x,y)^k+1}{2}$

by combining k independent MinHashes. Note that, given x and y, the probability of agreement of an independent MinHash value is $\mathcal{R}(x,y)$. Therefore, the probability of agreement of all k independent MinHashes will be $\mathcal{R}(x,y)^k$, see [52] for details. Also, to generate a 1-bit hash value from k integers, we need a universal hash function that takes a vector of k MinHashes and maps it uniformly to 1-bit. The final collision probability of this new 1-bit scheme is precisely $\frac{\mathcal{R}(x,y)^k+1}{2}$, as required. The overall idea is quite general and applicable to any LSH. We formalize it in the next section.

5.3 Formalization

As we argued in the previous section, we need a universal hashing scheme, $h_{univ}: \mathbb{N}^k \mapsto \{0,1\}$, which maps a vector of k integers uniformly to 0 or 1. There are many ways to achieve this and a common candidate is

$$h_{univ}(x_1,\ x_2, ...,\ x_k) = (r_{k+1} + \sum_{i=1}^{k} r_i x_i)\ mod\ p,\ mod\ 2,$$

where r_i are fixed randomly generated integers.

Given a hash function h, uniformly sampled from any given locality sensitive family $\mathcal{H}$, let us denote the probability of agreement (collision) of hash values of x and y by P_c,

$$P_{collision} \equiv P_c \equiv Pr_{\mathcal{H}}(h(x) = h(y)). \tag{8}$$

DEFINITION 5.1 (**Secure LSH**). *Our proposed secure 1-bit LSH, h_{sec}, parameterized by k, for any point x is given by*

$$h_{sec}(x) = h_{univ}(h_1(x), h_2(x), ..., h_k(x)), \tag{9}$$

where h_is, $i \in \{1,\ 2, ...,\ k\}$ are k independent hash functions sampled uniformly from the LSH family of interest $\mathcal{H}$.

It is not difficult to show the following:

THEOREM 5.1. *For any vectors x and y, under the randomization of h_{sec} and r_i we have*

$$P_c^{sec} = Pr_{\mathcal{H},r}\left(h_{sec}(x) = h_{sec}(y)\right) = \frac{P_c^k + 1}{2} \tag{10}$$

PROOF. It should be noted that $h_{sec}(x) = h_{sec}(y)$ can happen due to the random bit collision with probability $\frac{1}{2}$. Otherwise the two are equal if and only if

$$(h_1(x), h_2(x), ..., h_k(x)) = (h_1(y), h_2(y), ..., h_k(y)),$$

which happens with probability P_c^k, because each h_i is independent and $Pr(h_i(x) = h_i(y)) = P_c$. Therefore, the total probability is $\frac{1}{2} + \frac{1}{2}P_c^k$ leading to the desired expression. □

We illustrate the usefulness of the framework proposed above in deriving secure 1-bit hash for two most popular similarity measures: 1) Secure MinHash for Jaccard similarity and 2) Secure SimHash for Cosine similarity. The idea is applicable to any general LSH including ALSH for Maximum Inner Product Search (MIPS) [53, 56, 57].

5.3.1 *Making Minwise Hashing Secure (Secure MinHash)*

As an immediate consequence of Theorem 5.1, we can obtain secure 1-bit MinHash in order to search based on the Resemblance similarity,

$$h_{sec}^{min}(x) = h_{univ}(h_{\pi_1}^{min}(x),\ h_{\pi_2}^{min}(x), ...,\ h_{\pi_k}^{min}(x)), \tag{11}$$

with the following Corollary:

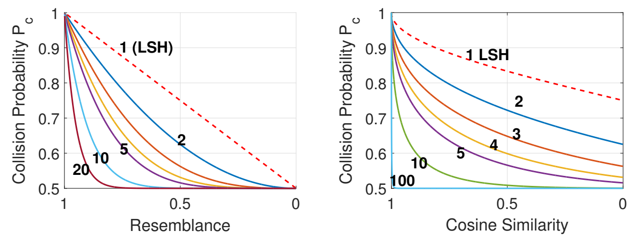

Figure 3: **Left:** The probability of collision of Secure MinHash as a function of R (resemblance) for different values of k. **Right:** The probability of collision of Secure SimHash as a function of θ (Cosine Similarity) for different values of k. Increasing k makes the collision probability drop to the constant 0.5 rapidly.

COROLLARY 1. *For MinHash we have:*

$$P_c^{sec} = Pr\left(h_{sec}^{min}(x) = h_{sec}^{min}(y)\right) = \frac{\mathcal{R}^k + 1}{2} \tag{12}$$

Figure 3 shows that the nature of new collision probability follows the desired trend. The parameter k gives us the knob to tune the probability curve. In section 5.4, we discuss how to tune this knob.

To generate our final l-bit binary embedding $S(x)$, we simply generate l independent h_{sec}^{min}, by using independent permutations for MinHashes and independent random numbers for the universal hashing. Therefore, $S(x)$ is the concatenation of l different h_{sec}^{min}.

In Section 5.3.3, we formally show that our transformed bits are more secure than LSH. In particular, we prove that the mutual information between the two secure 1-bit MinHashes, $h_{sec}^{min}(x)$ and $h_{sec}^{min}(y)$ decays sharply (*exponentially* with k) to zero as the similarity between x and y (i.e. $\mathcal{R}$) decreases. Thus, there is negligible mutual information about x in the embedding of some random (non-neighbor) y.

5.3.2 *Making Signed Random Projections Secure (Secure SimHash)*

Analogous to MinHash, we can make SimHash secure with the same properties.

$$h_{sec}^{sign}(x) = h_{univ}(h_{w_1}^{sign}(x),\ h_{w_2}^{sign}(x), ...,\ h_{w_k}^{sign}(x)), \tag{13}$$

where w_is for all i are independently chosen. Figure 3 (right) summarizes the collision probability as a function of similarity for different values of k.

COROLLARY 2. *For Secure SimHash we have:*

$$P_c^{sec} = Pr\left(h_{sec}^{sign}(x) = h_{sec}^{sign}(y)\right) = \frac{(1 - \frac{\theta}{\pi})^k + 1}{2} \tag{14}$$

5.3.3 *Info. Theoretic Bound as a Function of k*

We provide the theoretical property of our transformation by quantifying the mutual information between two l-bit secure embeddings. The similarity of x and y ($Sim(x,y)$) is denoted as $S_{x,y}$.

THEOREM 5.2. *For any two data points x and y, with $S_{x,y}$ being the similarity between them, the mutual information between $h_{sec}(x)$ and $h_{sec}(y)$ is bounded by*

$$I(h_{sec}(x); h_{sec}(y)|S_{x,y}) < l \cdot (2P_c^{sec} - 1)log(\frac{P_c^{sec}}{1 - P_c^{sec}}) \tag{15}$$

PROOF. For simplicity let us call the i^{th} bit of $h_{sec}(x)$, u_i and i^{th} bit of $h_{sec}(y)$, u'_i. and derive the mutual information between these two bits conditioned on $S_{x,y}$ as follows:

$$
\begin{aligned}
I(u_i;u'_i|S_{x,y}) \equiv& \\
&\sum_{u_i,u'_i\in\{0,1\}} P(u_i,u'_i|S_{x,y}) log \frac{P(u_i,u'_i|S_{x,y})}{P(u_i|S_{x,y})P(u'_i|S_{x,y})} \\
&= P_c^{sec} log(2P_c^{sec}) + (1-P_c^{sec}) log(2(1-P_c^{sec})) \\
&< (2P_c^{sec}-1) log(\frac{P_c^{sec}}{1-P_c^{sec}})
\end{aligned}
$$

Since every bits of the binary embeddings are generated independently, the mutual information between l-bit embeddings is multiplied by l. □

Substituting P_c^{sec} from Equation 12 and Equation 14, the mutual information can be expressed as a function of Resemblance and Cosine similarities, respectively.

COROLLARY 3. *For secure MinHash we have:*

$$I(h_{sec}^{min}(x); h_{sec}^{min}(y)|\mathcal{R}) < \mathcal{R}^k log(\frac{1+\mathcal{R}^k}{1-\mathcal{R}^k}) \tag{16}$$

and for Secure SimHash:

$$I(h_{sec}^{sign}(x); h_{sec}^{sign}(y)|\theta) < (1-\frac{\theta}{\pi})^k log(\frac{1+(1-\frac{\theta}{\pi})^k}{1-(1-\frac{\theta}{\pi})^k}) \tag{17}$$

As can be seen from Equations 16 and 17, the mutual information drops rapidly (exponentially with k) to zero for x and y that have small similarity. Thus, for any two non-neighbor points (small $Sim(x,y)$) the generated bits behave like random bits revealing no information about each other. Obviously, $k=1$, which is the traditional choice for LSH, is not secure, as the bits contain significant mutual information. The choice of k controls the decay of the mutual information and hence is the privacy knob (see Section 5.4 for details on how to tune this knob).

5.4 Formalism of Privacy Budget

Suppose, the application at hand considers any pair of points x and y with $Sim(x,y) < s_0$ as non-neighbors, for some problem-dependent choice of s_0. The application also specifies an ϵ such that the collision probability of any two non-neighbors should not exceed $\frac{1}{2}+\epsilon$ (be very close to half (random)). Forcing this condition ensures that whenever $Sim(x,y) < s_0$, the released bits cannot distinguish x and y with any randomly chosen pair. Formally,

DEFINITION 5.2 (**ϵ-Secure Hash at Threshold s_0**). *For any x and y with $Sim(x,y) \le s_0$, we call a 1-bit hashing scheme h_{sec} secure at threshold s_0 if the probability of bit-matches satisfies:*

$$\frac{1}{2} \le Pr(h_{sec}(x) = h_{sec}(y)) \le \frac{1}{2} + \epsilon.$$

Note that the expression of ϵ-secure hash is not symmetric since the probability of collision is always greater than or equal to $\frac{1}{2}$ (see Equation 10).

We show that for any ϵ-secure hash function, the mutual information in the bits of non-neighbor pairs is bounded.

THEOREM 5.3 (**Information Bound**). *For any 1-bit ϵ-secure hash function at threshold s_0, the mutual information between $h(x)$ and $h(y)$, for any pair with $Sim(x,y) \le s_0$, is bounded as*

$$I(h(x);h(y)) \le 2\epsilon \log \frac{1+2\epsilon}{1-2\epsilon} \tag{18}$$

PROOF. Follows from Theorem 5.2 and Definition 5.2. □

In triangulation attack, we have access to m attributes y_is: $Y = y_1, y_2, ..., y_m$, and their corresponding hashes $h(y_i)$s. Assuming y_is are independent, we can bound the mutual information about any target x conditional on knowing y_i's and h_is as follows:

THEOREM 5.4. *For any 1-bit ϵ-secure hash function at threshold s_0, the mutual information between $h(x)$ and $\{h(y_1), h(y_2)...h(y_m)\}$, for any pair with $Sim(x,y_i) \le s_0$ and any pair of y_i, y_j are independent, is bounded as*

$$I(h(x); h(y_1)h(y_2)...h(y_m)) \le 2m\epsilon \log \frac{1+2\epsilon}{1-2\epsilon} \tag{19}$$

PROOF. Define subsets $\mathcal{T} \subseteq \mathcal{V}$, where $\mathcal{V} = n$.

$$
\begin{aligned}
I(h(x); h(y_1)h(y_2)...h(y_m)) =& I(h(y_1)h(y_2)...h(y_m); h(x)) \\
=& \sum_{T\subseteq 2,...,n} (-1)^{|T|} I(T; h(x)) \\
\le& 2m\epsilon \log \frac{1+2\epsilon}{1-2\epsilon}
\end{aligned} \tag{20}
$$

Note that the mutual information of any y_i, y_j pair is 0 because they are independent. □

Thus, for small enough ϵ, it is impossible to get enough information about any non-neighbor x via triangulation. We verify this observation empirically in the experiments. Next, we show that Secure LSH can always be made ϵ-secure hash function for any ϵ using an appropriate choice of k.

THEOREM 5.5. *Any Secure LSH, h_{sec}, is also an ϵ-secure hash function at any given threshold s_0, for all*

$$k \ge \left\lceil \frac{\log 2\epsilon}{\log\left(Pr(h(x) = h(y)|Sim(x,y) = s_0)\right)} \right\rceil, \tag{21}$$

where $\lceil . \rceil$ is the ceiling operation. Here, $h(x)$ is the original hash function from which the h_{sec} is derived.

PROOF. Follows from the definition of ϵ-secure hashing added with fact that $h(x)$ satisfies Definition 2.1. □

In order to obtain ϵ-secure MinHash, we need $k = \left\lceil \frac{\log 2\epsilon}{\log s_0} \right\rceil$.

For secure SimHash, we need to choose $k = \left\lceil \frac{\log 2\epsilon}{\log\left(1-\frac{cos^{-1}(s_0)}{\pi}\right)} \right\rceil$.

To get a sense of quantification, if we consider $s_0 = 0.75$ (high similarity) and $\epsilon = 0.05$, then we have $k = 8$ (MinHash) and $k = 12$ (SimHash).

5.5 Utility-Privacy Trade-off of Secure LSH

As mentioned in Section 2.3, the querying time and space for approximate Near-Neighbor search are directly quantified by $\rho = \frac{\log p_1}{\log p_2} < 1$. The space complexity grows as $n^{1+\rho}$, while the query time grows as n^{ρ}, where n is the size of the dataset. Thus, smaller ρ indicates better theoretical performance. The collision probability of our secured LSH is $P_c^{sec} = \frac{P_c^k+1}{2}$. The new ρ' for Secure LSH would be $\frac{\log \frac{P_1^k+1}{2}}{\log \frac{P_2^k+1}{2}}$.

THEOREM 5.6. *ρ' is monotonically increasing with k.*

$$\frac{d\rho'}{dk} = \frac{p_1^k \ln(p_1)}{\ln\left(\frac{p_2^k+1}{2}\right)(p_1^k+1)} - \frac{\ln\left(\frac{p_1^k+1}{2}\right) p_2^k \ln(p_2)}{\ln^2\left(\frac{p_2^k+1}{2}\right)(p_2^k+1)} > 0 \quad (22)$$

Therefore, when we increase k, we get the privacy at the cost of reduced space and query time. The quantification of this tradeoff is given as $\frac{\log \frac{P_1^k+1}{2}}{\log \frac{P_2^k+1}{2}}$

6. HIDING THE MECHANISM OF S(.)

We are now ready to describe the final piece of our protocol. We describe in detail how we can reasonably hide the random seeds inside of $S(.)$ from the users in addition to *both servers*. To compute the hash of the client's input, we need random seeds (e.g., for MinHash, random seeds are the random permutations and for SimHash, they are random vectors used in the projection step). Since these random seeds should be equal for all clients, we cannot let each client generate her seeds independently. Seeds should be chosen in a consistent fashion. However, seeds should not be revealed to any server, otherwise, they might be used to reconstruction the secret attributes. As a result, we need to design a mechanism such that no party knows the seeds, which is an important and yet difficult task. To compute $S(x)$ securely without revealing the actual random seeds to any party, at least two different (non-colluding) servers need to be deployed. While we do not trust either server, we require that two servers do not collude.

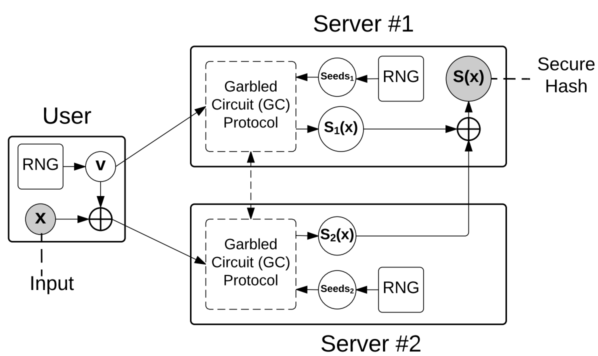

Figure 4: Global flow of black-box hash computation. RNG stands for Random Number Generator.

In the initial phase, each server generates its own version of the seeds randomly. Whenever a client wants to compute a secure hash, $S(x)$, she generates a random D dimensional vector v (same dimensionality as her input value) and then XORs this vector with x (resulting in $x \oplus v$). She then sends v to server #1 and $x \oplus v$ to server #2. The term $x \oplus v$ is One-Time Pad (OTP) encryption of x using v as the pad and we denote it as $Enc_v(x)$. Given this information and the two initial random seeds, then both servers engage in a two-party secure computation. Here, we utilize Garbled Circuit (GC) protocol in order to compute $S(x)$. The GC protocol is one of the generic secure function evaluation protocols that allows two parties to jointly compute a function on their inputs while keeping each input private to their respective owners. In this protocol, the function that is evaluated securely has to be described as a Boolean circuit. The computation and communication complexity of this algorithm is proportional to the number of non-XOR gates in the circuit.

The global flow of our approach is illustrated in Figure 4. Server #1 inputs v and server #2 inputs $Enc_v(x)$ $(x \oplus v)$ to the GC protocol. In addition, each server inputs her random seeds to the GC protocol. *Actual seeds* used for generating the hash of x are based on the two random seeds from two servers and are generated using the Boolean circuit that is used inside the GC protocol. In our case, the Boolean circuit is the secure hash computation suggested in Section 5. For this reason, we have designed the corresponding Boolean circuits for securely computing secure MinHash and SimHash. After two servers run the GC protocol, they both acquire secret shared values of the hash ($S_1(x)$ and $S_2(x)$) and server #1 needs to XOR the two values to get the real hash ($S(x)$). RNG stands for Random Number Generator. We utilize the recent advances which make hashing algorithmically faster [54, 55].

The above procedure is called *XOR-sharing* technique and is secure in HbC attack model because: (i) server #1 receives nothing but a true random number which contains no information about x and (ii) server #2 receives the encryption of the message x using v as the encryption pad and is perfectly secure [40]. Since both servers are assumed to not collude, they cannot infer any information about the user's input x. The theory behind the GC protocol guarantees that neither of the parties that execute the protocol can infer any information about the intermediate values [66]. Since the actual random seeds used to compute $S(x)$ is created by the GC protocol as an intermediate value, none of the servers nor the users know the value of true seeds and hence our protocol is secure.

7. NOISE ADDITION METHODS AND THEIR POOR UTILITY-PRIVACY TRADE-OFF

Obfuscating information by adding noise is one of the most popular techniques for achieving privacy. By adding sufficient noise to the hashes, one can construct ϵ-secure scheme satisfying Definition 5.2. However, any protocol based on adding a noise will obfuscate the information uniformly in every bit, which will significantly affect the utility of near-neighbor search. We elaborate this poor utility-privacy trade-off. This is not the first time when such poor utility-privacy trade-off is being observed by adding a noise [21].

Following popular noise addition mechanism [34], in order to achieve the requirement in Definition 5.2, we can choose to corrupt 1-bit LSH $h(x)$ with a random bit, with probability f. Formally, the generated hash function is

$$h_{corr}(x) = \begin{cases} random\_bit, & \text{with probability f} \\ h(x), & \text{with probability 1-f} \end{cases}$$

THEOREM 7.1. *The new collision probability after this corruption, for any x and y, is given by:*

$$P(h_{corr}(x) = h_{corr}(y)) = (1-f)(Pr(h_{1bit}(x) = h_{1bit}(y)) + \frac{f}{2}. \quad (23)$$

Let us define $P(s) = Pr(h_{1bit}(x) = h_{1bit}(y) | Sim(x,y) = s)$. Using this quantity, it is not difficult to show:

THEOREM 7.2. *h_{corr} is ϵ-secure at threshold s_0, iff*

$$(1-f)P(s_0) + 0.5f \leq 0.5 + \epsilon; \quad f \geq 1 - \frac{\epsilon}{\left(P(s_0) - \frac{1}{2}\right)}.$$

For 1-bit MinHash with corruption, the collision probability boils down to $\frac{R(1-f)+1}{2}$. Thus, f only changes the slope of collision probability curve. To ensure ϵ-secure hash at similarity s_0 threshold, we must have

$$f \geq 1 - \frac{2\epsilon}{s_0}. \tag{24}$$

To understand its implication, consider, an example with $s_0 = 0.75$ (high similarity) and $\epsilon = 0.05$. This combination requires $f \geq 0.86$. Such high f implies that most bits (86%) are randomly chosen, and hence they are uninformative. Even for very similar (almost identical) x and y, the collision probability is close to random. This degrades the usefulness of LSH scheme significantly. In contrast, for the same threshold $s_0 = 0.75$ and same epsilon $\epsilon = 0.05$, secure LSH needs $k = 8$ which leads to the collision probability expression $\frac{\mathcal{R}^8+1}{2}$. For $x = y$, i.e. $\mathcal{R} = 1$, this expression is *always* 1. For x and y with similarity 0.95, the collision probability is greater than 0.83, significantly higher than 0.56 obtained using noise addition (very close to the random probability 0.5).

8. EVALUATIONS

8.1 Utility-Privacy Tradeoff

In this section, we provide thorough evaluations of the accuracy and privacy trade-off in our framework. Our aim is two-fold: (i) We want to evaluate the benefits of our proposal compared to traditional LSH in preventing triangulation attack and simultaneously evaluate the effect of our proposal on the utility of near-neighbor search. (ii) We also want to understand the utility-privacy of noise addition techniques in practice and further quantify it with the trade-offs of our approach. It is important to have such evaluations, as pure noise addition may be a good heuristic on real datasets that prevents the triangulation attack without hurting accuracy.

Datasets: We use the IWPC [15] and Speed Dating datasets [20]. They belong to different domains but both contain private and sensitive attributes of the concerned individuals. The IWPC is a medical dataset which consists of 186 demographic, phenotypic, and genotypic features like race, medicines taken, and Cyp2C9 genotypes of 5700 patients. We split the records to 80% for creating hash tables and 20% for querying. The dataset is publicly available for research purposes. The type of data contained in the IWPC dataset is equivalent to that of other private medical datasets that have not been released publicly [21]. Speed-Dating dataset has 8378 text survey samples, each has 190 features representing geometric features or answers to designed questions for the volunteered subjects.

We focus on the cosine similarity search, therefore, our underlying LSH scheme is SimHash (or Signed Random Projections). The gold standard neighbors for every query were chosen to be the points with cosine similarity greater than or equal to 0.95. Please note that LSH is threshold-based [29]. Hence, we chose a reasonable high similarity threshold.

Baselines: We chose the following three baselines for our comparisons. **1. LSH:** This is the standard SimHash-based embedding. **2. Secure LSH:** As described in Section 5, we use our proposed transformation to make LSH secure. To study the utility-privacy trade-off, a range for privacy parameter $k = 2, 4, 6, 8, 12$ is chosen. Note, $k = 1$ is vanilla SimHash. **3. Noise-based LSH:** [34] shows a way to release user information in a privacy-preserving way for near-neighbor search. The paper showed that adding Gaussian noise $N(0, \sigma^2)$ after the random projection preserves differential privacy. To compute the private variant of SimHash, we used the sign of the differentially random private vector (generated by perturbed random projections) as suggested in [34]. To understand the trade-off the noise levels are varied over a fine grid $\sigma = 0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0$.

We generated 32-bit hashes for IWPC and 64-bit hashes for Speed-Dating using each of the competing candidate hashing schemes. For each query data, we ranked points in training data based on the Hamming distance of the competing hash codes. We then computed the precision and recall of the Hamming-based ranking on the gold standard neighbors. We summarized the complete precision-recall curves for both the datasets and all the competing scheme in Figure 5. This is a standard evaluation for hashing algorithms in the literature [64]. Higher precision-recall under a given ranking indicates a better correlation of binary Hamming distance with the actual similarity measure. A better correlation directly translates into a faster algorithm for sublinear near neighbor search [46] with Hamming distance.

In Figure 5, the first and third plots from the left-hand side show the retrieval precision and recall curve using various σ in noise-based LSH. The Vanilla LSH line, which is the performance of LSH $k = 1$ or $\sigma = 0$ serves as the reference in the plots. By increasing σ, the accuracy of noise-based hashing drops dramatically. Adding noise as argued in Section 7 leads to poor collision probability for similar neighbors which in turn leads to a significant drop in accuracy compared to LSH as evident from the plots. As the privacy budget is increased, by adding more noise, the performance drops significantly. In contrast, the second and fourth plots from the left-hand side show the precision and recall curve using different ks with Secure SimHash.

By increasing privacy budget k, the accuracy does not drop and even gets better than vanilla LSH. This improvement is not surprising and can be attributed to the enhanced gap between the collision probability of near-neighbor and any random pair (Figure 3). It is known that with hashing-based techniques, such enhanced gap leads to a better accuracy [29]. The plots show a consistent trend across the datasets and clearly signify the superiority of our proposed transformation over both LSH and noise addition based methods in terms of retrieving near-neighbors. The result clearly establishes the importance of studying problem-specific privacy before resorting to obfuscation based on noise.

8.2 Effectiveness Against Triangulation Attack

We showed that irrespective of the privacy budget, our proposal is significantly more accurate than LSH and Noise-based LSH. Our theoretical results suggest that the proposal is also secure against triangulation attack, whereas, vanilla LSH is not. We validate this claim in this section. Furthermore, we also study the effectiveness of noise addition in preventing the attack.

To evaluate the privacy, we implemented the "triangulation attack" and inferred its accuracy on real datasets,

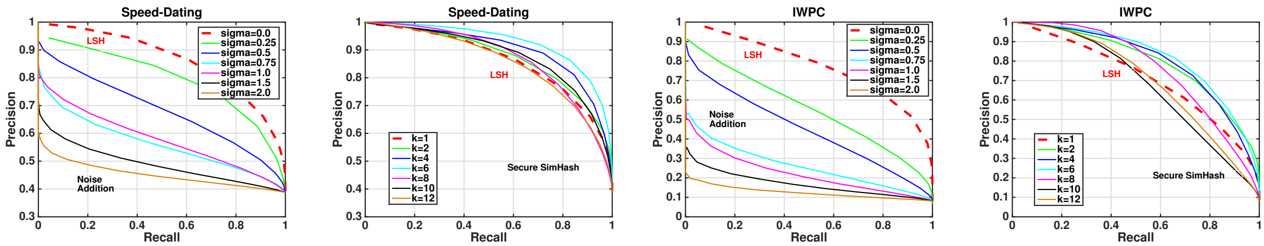

Figure 5: **Utility-Privacy Tradeoff:** The plots represent the precession recall curves (higher is better) based on noise addition (first and third from the left) and secure cosine similarity (second and fourth from the left) for both datasets. The dotted red line is the vanilla LSH. We can clearly see that adding noise loses utility while the proposed approach is significantly better.

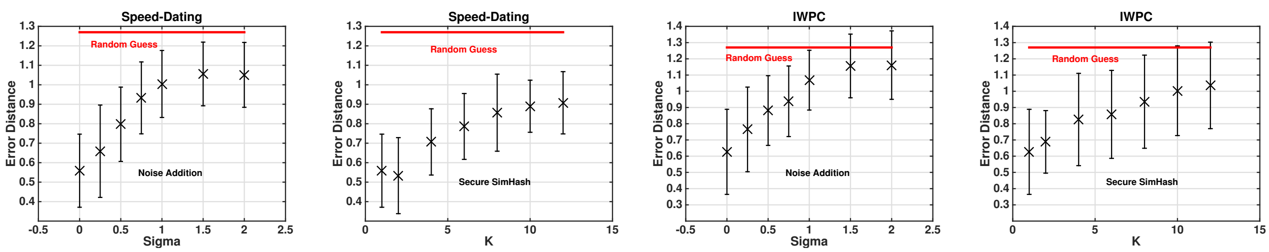

Figure 6: **Effectiveness Against Triangulation Attack:** Plots show the error (mean and error bars) in the triangulation-attack-based inference of attributes (higher is more secure, random is holy grail). We can see that both adding noise (first and third form the left) and increasing k with secure hashing (second and fourth form the left) lead to increased security. Contrasting this with Figure 5 clearly shows the superiority of our proposal in retaining utility for the same level of privacy.

IWPC, and Speed-Dating. The task was to infer sensitive attributes of a given target query vector by triangulating it with respect to randomly chosen points as explained in Section 5.1. For IWPC, we selected some sensitive attributes for inference like cancer or not, set of medicines taken, or Cyp2C9 genotypes to form the attack data points. For Speed-Dating, we randomly chose the attributes for inferring. To scale-up the implementation for higher dimensions, we use a novel iterative projection algorithm which successively approaches the target. The procedure is described separately in Section 9, which could be of separate interest.

We used the same privacy budget, i.e., $k = 1, 2, 4, 6, 8, 12$ for Secure SimHash and $\sigma = 0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0$, for noise-based SimHash. Again, $k = 1$ and $\sigma = 0$ corresponded to the vanilla LSH method which will serve as our reference point. We computed the error of the estimated target using triangulation attack with the actual target. We then calculated the mean and standard deviations of the errors over 100 independent triangulation attacks. The errors for varying k for our proposed secure LSH and varying σ for noise-based LSH were summarized in Figure 6. We also plotted the accuracy of random guess which will serve as our holy grail for privacy. The attack accuracy for $k = 1$ ($\sigma = 0$) is substantially better than the random guess which clearly indicates the vanilla LSH is not secure. The decrease in attack accuracy with an increase in k clearly shows the high security level of our solution.

As indicated by our theoretical results, the accuracy of the triangulation attack decreases and slowly approaches the random level (holy grail for privacy) as the privacy budget increases. We can conclude that both noise addition and our proposal effectively prevent triangulation attack. Increasing noise, as expected, preserves privacy but at a significant loss in utility. However, the retrieval experiments show that our proposal provides privacy *without* loss in accuracy. For all σ, there always exists some k which could achieve significantly better performance for the same level of security.

8.3 Computational Cost Comparison with SFE Protocols

In this section, we compare the performance of our protocol with the GC protocol, one of the most promising and efficient Secure Function Evaluation (SFE) protocols. In our scheme, we have integrated the GC protocol only for our black-box hash computation step that is computed independently and only once for each client. We will compare the performance of our protocol with the *pure execution of NNS* in GC to show the shortcomings of this approach. While GC protocol can compute NNS without any computational error (compare to Figure 5), it has rather limited practical usage and scalability. The recent work of [59] has implemented the K-Near-Neighbor (KNN) search using TinyGarble [60] framework, one of the most efficient GC frameworks. Based on their performance results, they report execution time of 6.7s for $N = 128,000$ when processing 31-bit data. According to their cost functions (which scales linearly with N and input bit length), for $N = 3$ Billion and input size of 1280-bit (same as ours), the execution time exceeds *74 days*. In contrast, our protocol requires 0.415 second for black-box hash computation and 0.887 second to search the hash-tables, resulting in an overall 1.3 s execution time on the same machine. We have also modified their solution and synthesized the circuit for NNS based on the Cosine similarity. For the exact same problem and parameters as ours, their solution requires an estimated processing time of 1.5×10^8 seconds and communication of 1.2×10^7 GBytes. This clearly illustrates the superiority of our novel scheme over GC.

9. ALTERNATING PROJECTIONS FOR TRIANGULATION ATTACK

We provide the details of our implementation for the triangulation attack over SimHash with cosine similarity (angles) as the measure. We start with all normalized vectors. Given the target point q, we generate $D+1$ random points X_is in the space.

$$q \in R^D, X_i \in R^D, \|X_i\|_2 = \|q\|_2 = 1, \quad \forall i \in \{1, 2, ..., (D+1)\}. \tag{25}$$

The distance between every X_i and q,

$$d_i = \|X_i - q\|_2, \quad \forall i \in \{1, 2, ..., (D+1)\} \tag{26}$$

is estimated as described in Section 5.1, first we estimate the angle θ using hash matches between $H(X_i)$ and $H(q)$: Then, we can get the distance d_i, from θ easily as the data is normalized.

After finding all of the distances, we use Alternating Projection Method [26] to find the possible intersection of $D+1$ D-dimensional spheres, $S_1, ..., S_{D+1}$, each with central point X_i and radius $\|X_i - q\|_2$. Any point in the intersection will likely be very close to the target point. The procedure for computing the point in the intersection is summarized in Algorithm 1. t_0 is initialized to a random vector (representing the estimated location for the target point q) and is iteratively updated. $\mathcal{P}_{S_i}(t_k)$ denotes the projection of point t_k on sphere S_i. We generate the sequence of projections:

$$t_{k+1} = \mathcal{P}_{S_N}\left(\mathcal{P}_{S_{N-1}}(...\mathcal{P}_{S_1}(t_k))\right),$$

Algorithm 1: POCS Algorithm

Initialize the maximum number of iteration $I_{\max}$
$t_0 = \text{rand}(1, \text{D})$ //D-dimensional random vector
$\textit{counter} = 0$
repeat
  for $j = 1$ *to* $D+1$ **do**
    $t_j = P_{S_j}(t_{j-1})$ //P is projection
      //of t_{j-1} on S_j
  end for
  $\textit{counter}++$
until Convergence == true or $\textit{counter} == I_{\max}$

10. PRIOR ART

PP-NNS is a heavily studied problem. However, existing solutions are limited with respect to at least one of the three requirements outlined in Section 1. In addition to PP-NNS approaches discussed in Section 1, we briefly discuss most relevant prior works. Several PP-NNS solutions are built upon the principals of cryptographically secure computation with the ability to compute on encrypted data [19]. The security of this approach, like cryptographic tools, is based on the hardness of certain problems in number theory (e.g. factorization of large numbers). Since every single bit in the computation is encrypted, distance calculations are computationally demanding and slow.

Another popular approach is to use information-theoretic secure multi-party computations, which guarantees that even with unlimited computational power no adversary can compromise the data. This method is based on secret-shared information to perform the secure computation and requires three or more servers. Securely computing pairwise distances needs "comparison" which cannot be carried out using secret-sharing alone and needs additional cryptographic blocks which limit the overall scalability [35, 16]. These algorithms work by first computing all possible distances securely, before they find the near-neighbors based on minimum distance values. Irrespective of the underlying technique, calculating all distance pairs incurs $O(N)$ complexity. Thus, the sub-linear time requirement cannot be satisfied by this class of techniques, rendering it unscalable and impractical to modern massive datasets.

There has been successful advances in the area of Differential Privacy (DP) [18]. However, their security model and use cases of DP is different than ours. DP usually assumes a trusted server and aims to bound the information leakage when answering each query. In a very high level, a certain noise is added to the data stored on the database such that the statistical information of the database is preserved but an attacker cannot infer significant information about single entry in the database.

Order Preserving Encryption (OPE) [6, 44] allows to carry out the comparison on encrypted version of data instead of the raw version. Wang et al. [63] have proposed a solution based on OPE and R-tree for faster than linear PP-NNS. However, Naveed et al. [39] introduced several attacks that can recover original users' data from an encrypted database that are based on OPE or Deterministic Encryption (DTE). They have illustrated that the encrypted databases based on OPE or DTE are insecure. Another line of research is based on the searchable encryption [58, 17, 33, 32, 41, 13, 49] which allows a user to store the encrypted data on the cloud server while being able to perform secure search. However, these solutions are limited to *exact* keyword search and are not compatible with NNS algorithms.

LSH is the algorithm of choice for sub-linear near-neighbor search in high dimensions [29]. LSH techniques rely on randomized binary embeddings (or representations) [48, 30, 2, 8]. These embeddings act as a probabilistic encryption which does not reveal direct information about the original attributes [30, 8]. Due to the celebrated Jonson-Lindenstrauss [31] or LSH property, it is possible to compare the generated embedding for a potential match.

11. CONCLUSION

This paper addresses the important problem of privacy-preserving near-neighbor search for multiple data owners while the query time is sub-linear in the number of clients. We show that the generic method of Locally Sensitive Hashing (LSH) for sub-linear query search is vulnerable to the triangulation attack. To secure LSH, a novel transformation is suggested based on the secure probabilistic embedding over LSH family. We theoretically demonstrate that our transformation preserves the near-neighbor embedding of LSH while it makes distance estimation mathematically impossible for non-neighbor points. By combining our transformation with Yao's Garbled Circuit protocol, we devise the first practical privacy-preserving near-neighbor algorithm, called Secure Locality Sensitive Indexing (SLSI) that is scalable to the massive datasets without relying on trusted servers. The paper provides substantial empirical evidence on real data from medical records of patients to online dating profiles to support its theoretical claims.

12. REFERENCES

[1] Yahoo security notice. `https://help.yahoo.com/kb/account/SLN27925.html`.

[2] A. Aghasaryan, M. Bouzid, D. Kostadinov, M. Kothari, and A. Nandi. On the use of LSH for privacy preserving personalization. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2013.

[3] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, et al. Privacy-preserving fingercode authentication. In *Proceedings of the 12th ACM workshop on Multimedia and security*, 2010.

[4] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.

[5] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*. 2011.

[6] A. Boldyreva, N. Chenette, Y. Lee, and A. Oneill. Order-preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2009.

[7] A. Boldyreva, N. Chenette, and A. ONeill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*, 2011.

[8] P. Boufounos and S. Rane. Secure binary embeddings for privacy preserving nearest neighbors. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2011.

[9] S. Boyd and J. Dattorro. Alternating projections. 2003.

[10] A. Z. Broder. On the resemblance and containment of documents. In *the Compression and Complexity of Sequences*, 1997.

[11] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC*, 1998.

[12] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *STOC*, 1977.

[13] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*, 2014.

[14] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.

[15] I. W. P. Consortium et al. Estimation of the warfarin dose with clinical and pharmacogenetic data. *N Engl J Med*, 2009.

[16] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation, an introduction. *Contemporary cryptology*, 2009.

[17] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 2011.

[18] C. Dwork. Differential privacy. In *Automata, languages and programming*. 2006.

[19] Y. Elmehdwi, B. K. Samanthula, and W. Jiang. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *International Conference on Data Engineering (ICDE)*, 2014.

[20] R. Fisman, S. S. Iyengar, E. Kamenica, and I. Simonson. Gender differences in mate selection: Evidence from a speed dating experiment. *The Quarterly Journal of Economics*, 2006.

[21] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *USENIX Security*, 2014.

[22] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, 1999.

[23] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 1995.

[24] J. E. Goodman, J. O'Rourke, and P. Indyk. *Nearest neighbours in high-dimensional spaces*. Handbook of Discrete and Computational Geometry (2nd edition), CRC Press, 2004.

[25] P. Grubbs, R. McPherson, M. Naveed, T. Ristenpart, and V. Shmatikov. Breaking web applications built on top of encrypted data. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[26] L. Gubin, B. Polyak, and E. Raik. The method of projections for finding the common point of convex sets. *USSR Computational Mathematics and Mathematical Physics*, 7(6):1–24, 1967.

[27] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[28] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *IEEE International Conference on Data Engineering (ICDE)*, 2011.

[29] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[30] P. Indyk and D. Woodruff. Polylogarithmic private approximations and efficient matching. In *Theory of Cryptography*. 2006.

[31] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 1984.

[32] S. Kamara and C. Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, 2013.

[33] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the ACM conference on Computer and communications security*, 2012.

[34] K. Kenthapadi, A. Korolova, I. Mironov, and N. Mishra. Privacy via the Johnson-Lindenstrauss transform. *Journal of Privacy and Confidentiality*, 2013.

[35] J. Kilian. Founding crytpography on oblivious transfer. In *Proceedings of annual ACM symposium on*

Theory of computing, 1988.

[36] F. Li, R. Shin, and V. Paxson. Exploring privacy preservation in outsourced k-nearest neighbors with multiple data owners. In *Proceedings of the ACM Workshop on Cloud Computing Security Workshop*, 2015.

[37] P. Li, M. Mitzenmacher, and A. Shrivastava. Coding for random projections. In *ICML*, 2014.

[38] Y. Moon, S. Noh, D. Park, C. Luo, A. Shrivastava, S. Hong, and K. Palem. CaPSuLe: Camera based positioning system using learning. In *Proceedings of international IEEE System-on-Chip Conference*, 2016.

[39] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[40] C. Paar and J. Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[41] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *IEEE Symposium on Security and Privacy (S&P)*, 2014.

[42] M. A. Pathak and B. Raj. Privacy-preserving speaker verification as password matching. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[43] H. Polat and W. Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *IEEE International Conference on Data Mining (ICDM)*, 2003.

[44] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[45] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, and H. Balakrishnan. Building web applications on top of encrypted data using Mylar. In *NSDI*, 2014.

[46] A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[47] S. Rane and P. T. Boufounos. Privacy-preserving nearest neighbor methods: Comparing signals without revealing them. *IEEE Signal Processing Magazine*, 2013.

[48] S. Rane, W. Sun, and A. Vetro. Privacy-preserving approximation of L1 distance for multimedia applications. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2010.

[49] M. S. Riazi, E. M. Songhori, and F. Koushanfar. PriSearch: Efficient search on private data. In *Design Automation Conference*, 2017.

[50] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *Information, Security and Cryptology*. 2009.

[51] A. Shrivastava and P. Li. Fast near neighbor search in high-dimensional binary data. In *ECML*, 2012.

[52] A. Shrivastava and P. Li. Beyond pairwise: Provably fast algorithms for approximate k-way similarity search. In *NIPS*, 2013.

[53] A. Shrivastava and P. Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS*, 2014.

[54] A. Shrivastava and P. Li. Densifying one permutation hashing via rotation for fast near neighbor search. In *ICML*, 2014.

[55] A. Shrivastava and P. Li. Improved densification of one permutation hashing. In *UAI*, 2014.

[56] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW*, 2015.

[57] A. Shrivastava and P. Li. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). In *UAI*, 2015.

[58] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy (S&P)*, 2000.

[59] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, and F. Koushanfar. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In *Design Automation Conference*, 2015.

[60] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[61] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

[62] `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf`. National institute of standards and technology. 2017.

[63] B. Wang, Y. Hou, and M. Li. Practical and secure nearest neighbor search on encrypted large-scale data. In *International Conference on Computer Communications*, 2016.

[64] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in neural information processing systems*, 2009.

[65] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the ACM SIGMOD International Conference on Management of data*, 2009.

[66] A. C.-C. Yao. How to generate and exchange secrets. In *Annual Symposium on Foundations of Computer Science*, 1986.

[67] B. Yao, F. Li, and X. Xiao. Secure nearest neighbor revisited. In *IEEE International Conference on Data Engineering (ICDE)*, 2013.