Private Outlier Detection and Content based Encrypted Search

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science (by Research)
in
Computer Science

by

Nisarg Raval
200907012
nisarg.raval@research.iiit.ac.in

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2012

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled “Private Outlier Detection and Content based Encrypted Search” by Nisarg Raval, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. C. V. Jawahar
Date	Adviser: Dr. Kannan Srinathan

To Mom and Dad

Acknowledgments

I am extremely grateful to Prof C. V. Jawahar and Dr. Kannan Srinathan for guiding me in my work, motivating me to aim bigger and giving me the opportunity and freedom to explore new domains and problems. I thank all my co-authors Piyush Bansal, Madhuchand Rushi Pillutla and Rashmi Tonge for their cooperation and valuable inputs. A special thanks to my friends Nitin, Manisha, Rajvi, Laxit, Kushal, Siddhartha, Anand and Praveen for making this journey memorable. I also thank all my colleagues in CVIT and CSTAR lab, and batch mates at IIIT, for their constant support and feedback. And lastly, I am grateful to Mom, Dad and my sister Shivangi, for constant encouragement and support at every step of my life!

Abstract

Today, our daily lives are constantly driven and influenced by services like social networking, personalized advertisements, cloud infrastructure etc. This makes any form of personal or professional data vulnerable to both security and privacy threats. User looses control over the data as soon as its published on the Internet. Recently, popular photo sharing service *Instagram* tried to change its privacy policy, giving itself the right to sell users' information including their photographs to advertisers without any notification or compensation. With incidents like AOL and Netflix debacle, organizations are refraining from releasing customer data even after anonymization. This hinders future research and developments in many ways due to unavailability of customer data, behavior and patterns. Techniques like k-anonymity and l-diversity help to preserve user privacy to some extent but fails to provide any solid guarantees. Recently proposed, differential privacy provides promising results in this area with theoretical bounds on information leak and preservation of privacy even against the auxiliary information. However, differential privacy considers individual's record independent of other data and can not be applied to linked data like social graph, time series etc. Furthermore, all these techniques compromise utility of the data while maintaining privacy.

At present, two approaches are widely used to ensure security of the data. First one is based on Secure Multiparty Computations (SMC), where users communicate with each other using SMC protocol to securely evaluate any function on their data. SMC gives provable security without compromising utility of the data. However, the solutions based on the general protocol of SMC requires enormous computational and communication overhead, thus limiting the practical deployment of the secure algorithms. The second approach is to encrypt the data at the user end and perform any operations in the encrypted domain to preserve the confidentiality of the data. This leads to difficulty in performing various operations like search, retrieval and computations. Homomorphic encryption allows secure computations over encrypted data but its impractical due to high computational cost.

In this thesis, we focus on developing privacy preserving, communication and computationally efficient algorithms in the domain of Data Mining and Information Retrieval. Specifically, we design efficient privacy preserving solutions for two widely used tasks - outlier detection and content based similarity search over encrypted data. Private computation of outliers among data collected from different sources has many applications in the area of data mining. First, we propose a novel algorithm of distance based outlier detection using Locality Sensitive Hashing (LSH) in a distributed setting where data is horizontally partitioned among data owners. We show that the proposed algorithm is scalable

in case of large datasets with high dimensionality. Then, we extend our distributed outlier algorithm to propose an algorithm for private outlier detection that broke the previous known bounds of quadratic cost. We compare our algorithm with the state of the art technique and show that our algorithm is better both in terms of computational as well as communication complexity.

Next, we explore the properties of hierarchical index structures and propose an algorithm for content based similarity search over encrypted data that do not reveal user query to the server. To the best of our knowledge, we are the first one to propose a definition and a scheme for the problem of Content Similarity Searchable Symmetric Encryption (CS-SSE). Finally, we extend our CS-SSE scheme for Private Content Based Image Retrieval, which is essential for many applications like searching health records using CT scans and patent search for logos. We also show that our algorithm achieves optimal computational cost with similar accuracy and privacy when compared to the existing techniques.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Today, our daily lives are constantly driven and influenced by services like social networking, personalized advertisements, cloud infrastructure etc. This makes any form of personal or professional data vulnerable to both security and privacy threats. These services extensively use Data Mining and Information Retrieval algorithms. We need to build efficient and secure solutions for these algorithms in order to protect the user from potential privacy breach. In this thesis, we propose efficient privacy preserving solutions for two widely used tasks - outlier detection and content based similarity search over encrypted data.

1.1 Need of Privacy

Perhaps the greatest societal change of the last two decades has been of telecommunication: the ubiquity of *smart* mobile devices has led to a world where human beings are constantly connected to one another. Everyday decisions are influenced by social networks, review portals, and targeted advertising to such an extent that a business' online presence is often its best advertisement. Underpinning this change are large, centralized providers who allow users to freely use their infrastructure to share such information in return for free access to the users data. Such seemingly innocuous data may result in very real disclosures when correlated; for instance Target[1] has been known to identify pregnancy in customers even before their family is aware of it. Privacy concerns about provider data are far from just theoretical and with disclosures from both the AOL[2] and Netflix [52] public data sets, organizations are refraining from releasing even anonymized customer data. This unavailability of large-scale data hinders future research and developments. Techniques like k-anonymity [67, 42, 59] and l-diversity [50, 79] help preserve user privacy to some degree in the face of de-anonymization techniques, but fail to provide any solid guarantees [1, 44]. Recently, differential privacy [19] has shown promises in providing theoretical bounds on information leakage, but only with the simplifying assumption that all sources of an individual's data are independent of one another. Thus, it can not be applied to many large scale linked

[1] http://www.nytimes.com/2012/02/19/magazine/shopping-habits.html
[2] http://www.nytimes.com/2006/08/09/technology/09aol.html

data sets like social graph, time series, etc., which are very important to certain tasks like behavioral analysis and information diffusion.

All these techniques reduce utility of the data because of the privacy vs. utility trade off. Problems which require both high utility and privacy (e.g. fraud detection, targeted advertisements), need to be addressed by techniques like Secure Multiparty Computation (SMC) [82]. However, SMC is usually computationally intensive and difficult to apply in many practical scenarios [25]. In this thesis, we propose novel techniques of privacy preserving computations which have very low computational and communication cost and hence can be applied to many real world problems. Specifically, we propose privacy preserving solutions for two different problems in the area of Data Mining and Information Retrieval.

1.2 Privacy in Data Mining

Data Mining, a process of extracting patterns from large datasets, has become an important research area due to the exponential growth of digital data and storage capabilities. However, in case of large datasets collected from various input sources, oftentimes the data is distributed across the network, raising privacy and security concerns while performing distributed data mining. In many situations sharing information leads to mutual gain, but due to data confidentiality issues it is not always feasible. For example, large collection of patient medical records across hospitals significantly aids medical research but since it is very sensitive information, it is not shared by hospitals due to trust issues. To overcome this problem, Privacy Preserving Data Mining (PPDM) methods have been proposed, which enable data owners to jointly compute specific operations without disclosing sensitive information.

PPDM methods can be broadly classified into two types - Randomization techniques [3] and Cryptographic techniques [61]. In this thesis, we will focus on cryptographic techniques. PPDM methods based on cryptographic techniques were introduced by Lindell *et al.* [45], who proposed an algorithm for Privacy Preserving ID3 Classification. Subsequently, privacy preserving algorithms have been proposed for various data mining tasks such as association rule mining [64], classification [76, 83, 81], clustering [30, 33, 32] and outlier detection [75]. For more details on various privacy preserving data mining algorithms please refer to [2].

1.2.1 Privacy Preserving Outlier Detection (PPOD)

Outlier detection is a fundamental task in data mining to uncover abnormal patterns. Outlier detection has been extensively researched in the recent past due to its applications in various fields such as fault detection, medical diagnosis, measuring ecosystem disturbances etc. Outlier detection is important for mainly two reasons: (1) In many data analysis tasks, outliers need to be detected and removed before processing the data, to enhance system performance. (2) In some applications, outlier detection in itself is crucial since the outliers provide useful or even critical information. These applications

include credit card fraud detection, network intrusion detection etc. A few definitions exist for outliers which are considered general enough to cover various types of data. Hawkins [29] defines an outlier as *an observation that deviates so much from the other observations as to arouse suspicion that it was generated by a different mechanism*.

Problem Statement: Consider a scenario where a bank wants to detect fraudulent customers by mining their transaction patterns. One possible solution is to find customers whose transaction patterns are abnormal (outlier) among all the customers in that bank. The results can be improved by enriching customer data gathered from various banks. For example, a specific customer pattern might look abnormal with respect to a certain bank but it can be perfectly normal when compared to other bank's transaction patterns. However, to maintain confidentiality of their customers, often banks do not share their data with each other. This leads to an important question of how do banks detect fraudulent customers using data from other banks without revealing any information to each other, apart from that of the fraudulent customers? The answer lies in privacy preserving outlier detection. In this thesis, we address the problem of privately computing outliers in horizontally partitioned data distributed among data owners, such that no other information about the data, apart from the outliers is revealed. Here, by horizontal distribution we mean that each player has all the attributes (dimensions) for some subset of total objects, such that the union of these subsets is equal to the total dataset.

Our Approach: As opposed to the current PPOD algorithms which provide privacy for already existing outlier detection algorithms, we develop a new outlier detection scheme based on Locality Sensitive Hashing (LSH) [31] in order to achieve efficient algorithm with runtime sub-quadratic in the database size. We show that the proposed solution is significantly better than the state of the art quadratic result. The idea is to use an efficient pruning technique to prune most of the non-outliers and exhaustively search for outliers among the remaining data. For outlier detection we use the distance based definition proposed by Knorr *et al.* [37] which uses near neighbors of the objects to detect outliers. Approximate near neighbor queries can be answered very efficiently (in sublinear time) using LSH. Given a query object, LSH returns each neighbour (object within specified radius) with high probability. Due to the probabilistic nature of LSH, it may not return all the neighbours. However, instead of all the neighbours, a small subset of neighbours is sufficient to prune most of the non outliers. Therefore, using LSH we can quickly detect and prune most of the non-outliers (objects with sufficient neighbours) and then process only the remaining objects to detect the outliers. First, we propose a distributed algorithm to compute outliers among horizontally partitioned data without considering privacy. Then, we extend this algorithm to privately compute outliers among horizontally distributed data. To compute the outliers in a privacy preserving manner, all the players engaged into secure evaluation of global LSH bin structure using *Secure Union* and *Secure Sum* protocol.

The computational complexity of the proposed algorithm is $\mathcal{O}(ndL)$ for d-dimensional dataset with n objects. The parameter L is defined as $n^{1/1+\epsilon}$, where $\epsilon > 0$ is an approximation factor. The communication complexity is $\mathcal{O}(N_b L \log n)$; where N_b is the average number of bins created during each of the L iterations of LSH. Due to the locality sensitive property, many objects which are near by fall into

a single bin. Hence, average number of bins created will be much less than the total number of objects in a dataset ($N_b << n$). We also give empirical evidence for the same in experimental section of Chapter 3. Thus in both computational as well as communication cost, we show a significant improvement over the previous known result of $\mathcal{O}(n^2d)$ [75]. Further, the communication cost of our algorithm is independent of dimensionality of the data and thus works very efficiently even for datasets of very large dimensionality as opposed to the existing algorithms. We achieve the above mentioned improvements at the cost of an approximate solution to outlier detection. However, we can use techniques proposed in [60, 63] to achieve very low approximation errors.

1.3 Privacy in Information Retrieval

Privacy in Information Retrieval can be broadly classified into two different but related problems - Searching on Encrypted Data and Private Information Retrieval (PIR). The problem of searching on encrypted data usually deals with private database, where user data is stored in an encrypted form and user should be able to search on the encrypted data. On the other hand the problem of PIR generally deals with public database, where data is stored in plain text and any user can retrieve specific data item without revealing it to the server. Next, we discuss both the problems in the context of content based similarity search.

1.3.1 Private Search over Encrypted Data

In light of the explosion in the scale of data being collected by various organizations, cloud storage is being increasingly used by these organizations to store their data. However, in order to protect the integrity of the data, it has to be stored in an encrypted form on the outsourced servers. This leads to the problem of searching in the encrypted domain. The problem of exact search in encrypted domain, termed as Searchable Symmetric Encryption (SSE) has received considerable attention in the research community [10, 15, 34, 71]. In these schemes a user can securely search an encrypted database for those items which contain exact query feature (or keyword). Later, the schemes proposed in [43, 51, 59] address the problem of Similarity SSE, the aim of which is to securely search for data items containing similar features to that of the query feature. All these methods solve the problem of "Finding a set of data items that contain a specific feature or similar features". But, what if one wants to find those items which are similar to a specific item?

Problem Statement: Let us say user is interested in searching for a scenery in her own collection of encrypted photographs stored on a remote server. In case of techniques based on keyword search, user has to describe the photo of interest in terms of keywords, for example mountain, river, forest etc. But description of a photograph is usually ambiguous and differs from person to person. It would be convenient if the user is able to give some examples of nature photographs to the server which in turn returns similar photographs. Since, the photographs on the server is encrypted we need a secure

matching mechanism over encrypted data to retrieve similar photographs. We call such a scheme as Content Similarity SSE (CS-SSE). One trivial solution is that user retrieves all the photographs from the server, decrypts it and matches them with the query image on client side. However, this solution is not practical as it requires entire database to be transfered to the client for every query.

We consider the popular setting of a cloud framework, where a data owner stores the data on a third party cloud server after encrypting it on the client side. Later on, the data owner or any client (with the permission of the data owner) can search for similar data items based on the notion of content similarity. Server should not learn anything about the database or query. Note, that the query item may or may not belongs to the database stored on the server.

Our Approach: We formalize the notion of CS-SSE and propose an efficient algorithm for the same using hierarchical index structures. Thus, given a query item, our method can securely search an encrypted database for similar items without revealing the query to the server. The idea is to construct a tree-like secure index structure at user end and store it on the server with encrypted data. Thus, server has no information about the stored data as well as the index structure. This will enable client to perform oblivious tree traversal to search and retrieve results based on the query, without server knowing anything about query or the results.

We also capture the information leak of our algorithm and give a security analysis based on adaptive semantic security definition. By exploiting the properties of tree-like index structures, we are able to achieve optimal computational cost of searching. The computation and communication costs of the proposed scheme are both $\mathcal{O}(m \log_k n)$ where m is the size of each tree node, k is the branching factor and n is the number of leaf nodes. The computation cost of the protocol is optimal since any similarity search protocol without considering privacy would at least need $\mathcal{O}(m \log_k n)$ computations. Next, we employ our scheme to search images and evaluate its efficiency on an encrypted image database indexed using popular index structure - vocabulary tree [53].

Note, that the proposed solution is inspired by the approach given in [69] but it is very different in terms of both problem formulation as well as results. In [69], the server holds public database in a plain-text form, while in our case the database is private and encrypted by the database owner. For private retrieval, Oblivious Transfer (OT) protocol is used in [69], which results into computational complexity of linear in the database size. For single server PIR, OT has to access entire database to keep the query private. However, we can avoid this step because in our setting both the database and data structure are encrypted and permuted respectively. Instead of using OT, we leverage the properties of encryption and tree structure perturbation to perform oblivious tree traversal, which results into computational complexity of logarithmic in the database size.

1.3.2 Private Content Based Image Retrieval (PCBIR)

Traditional image retrieval systems are based on text retrieval approaches by annotating each image with its description and querying description of a desired image. Availability of inexpensive smart cameras and increasing usage of applications like *Flikr, Picasa and Instagram*, lead to massive and diverse

collection of multimedia data - specifically images. Due to this, annotation became both ambiguous and laborious. On the other hand, state of the art image descriptors and efficient indexing mechanism gave rise to Content Based Image Retrieval (CBIR) system, where query is an image and the aim is to retrieve similar images. Query by image aids the user in expressing her query more accurately. This decreases the semantic gap between the user and the system, thereby circumventing the cardinal limitation of the conventional annotation based systems.

Problem Statement: Although, CBIR has better functionality, it is not widely used in practice. One reason is user's privacy. In CBIR, user needs to send an example query image and it might be the case that this image contains personal information. For example, using mobile app like *Google Goggles*, user can take a photograph anywhere and instantly get information about the photograph from the web or search the web for similar photographs. This leads to major privacy concerns as the photographs taken by the user may be very personal and can reveal personal information like location, habits, etc. Using these information adversary can easily profile the user. Since, users can not control their own collection of photographs it raises the potential risk of misuse. Thus, to preserve user's privacy the system should retrieve similar images or information about the image without revealing query image to the server.

Our Approach: We extend the proposed CS-SSE scheme to address the problem of PCBIR in a non-colluding two-server setting. In case of single server Private Information Retrieval (PIR), since the database at the server is in unencrypted form, a computation cost (at the server) is linear in the database size. This is unavoidable as failure to access any element would lead to a privacy disclosure. To overcome this limitation, Chor *et al.* [13] gave the idea of replicating the database at multiple, non-colluding servers. Similarly, we use the two-server setting to reduce the server side computational cost. With the help of an honest but curious third party server, database server will securely construct index structure in such a manner that it will have no information about the data stored in it. Third party server will also be used to authenticate any client who wants to query the database server. However, this needs to be done only once. Clients will interact with database server using CS-SSE protocol to privately retrieve similar images to the query image. We also perform extensive experiments and evaluate our algorithm with various performance measures and show that our algorithm outperforms the existing techniques.

1.4 Contributions of the Thesis

- **Distributed Outlier Detection using Locality Sensitive Hashing**
  We propose a novel algorithm of distance based outlier detection using locality sensitive hashing technique in a distributed setting where data is horizontally partitioned among data owners. We show that the proposed algorithm is scalable in case of large datasets with high dimensionality.

- **Privacy Preserving Outlier Detection using Locality Sensitive Hashing**
  Private computation of outliers among data collected from different sources has many applications in the area of data mining. We extend our distributed outlier algorithm to propose an algorithm for

private outlier detection that broke the previous known bounds of quadratic cost. We compare our algorithm with the state of the art technique and show that ours is better in terms of computational as well as communication complexity.

- **Private Content Based Search on Encrypted Data using Hierarchical Index Structures**
  To ensure confidentiality, data is often encrypted on the client and then stored on third party storage like cloud. This opaqueness hinders operations like search and retrieval, and computations on the data. We explore the properties of hierarchical index structures and propose an algorithm for content based similarity search over encrypted data that do not reveal user query to the server. To the best of our knowledge, we are the first one to propose a definition and a scheme for the problem of Content Similarity Searchable Symmetric Encryption (CS-SSE).

- **Private Content Based Image Retrieval**
  We extend our CS-SSE scheme for Private Content Based Image Retrieval, which is essential for many applications like searching health records using CT scans and patent search for logos. We show that our algorithm achieves optimal computational cost with similar accuracy and privacy when compared to the existing techniques.

1.5 Organization of the Thesis

Remaining part of the thesis is organized as follows:

Chapter 2 gives brief overview of the required background. It describes outlier detection methodologies, Locality Sensitive Hashing and Vocabulary Tree. It also gives overview of cryptographic primitives used through out the thesis.

Chapter 3 presents our work on privacy preserving outlier detection. We start by describing the problem and related work in this field. We proceed to present our algorithm in a distributed setting with horizontal partitioning of the data. After that, we extend our algorithm for privacy preserving outlier detection in horizontal partitioning. Finally, we give experiments on various datasets and conclude with the summary of the proposed method.

Chapter 4 presents our work on private content based search on encrypted data. First, we formalize the problem and give definition of content similarity searchable symmetric encryption (CS-SSE). Then, we present our algorithm for CS-SSE and its complexity analysis. We also give a formal security analysis of the proposed method. After that, we extend our CS-SSE scheme for private content based image retrieval. Finally, we show experiments on different image datasets and compare the proposed scheme with existing techniques.

Chapter 5 is the last chapter of this thesis and summarizes our work and establishes the key lessons learned from our work. We touch upon a number of topics for further research in this field, and conclude this thesis.

Chapter 2

Background and Preliminaries

In this chapter, we provide a brief introduction to the background details which are used in the subsequent chapters. First, we describe outlier detection methodologies including distance based outliers which will be used later in Chapter 3. Then, we describe Locality Sensitive Hashing (LSH), which is a fundamental building block of our privacy preserving outlier detection (PPOD) algorithm. After that, we presents vocabulary tree which is used to build image retrieval system in Chapter 4. Finally, we describe Secure Sum and Secure Union protocols which will be used in the proposed PPOD algorithm.

2.1 Outlier Detection Methodologies

In this section we outline various outlier detection approaches and give details about distance based outlier detection method used in our algorithm.

2.1.1 Statistical Outlier Detection

Statistical approaches were the earliest algorithms used for outlier detection. Some of the earliest are applicable only for single dimensional data sets. The intuition of these approaches is that normal data objects follow a generating mechanism and abnormal objects deviate from this generating mechanism. Given a certain kind of statistical distribution (e.g., Gaussian), algorithms compute the parameters assuming all data points have been generated by such a distribution (e.g., mean and standard deviation). Outliers are points that have a low probability to be generated by the overall distribution (e.g., deviate more than 3 times the standard deviation from the mean). These methods have the limitation that they assume the data distribution is known before hand. Another limitation of the statistical methods is that they do not scale well to large datasets or datasets of high dimensions.

2.1.2 Distance Based Outlier Detection

While normal data points have several neighbours at small distances, outliers are far apart from their neighbours i.e. have less dense neighbourhood. If one calculates the distance between all points and

group them by proximity, points that have none or few neighbours could be considered as outliers. This is the underlying assumption of distance based outlier approaches. These approaches do not make any assumptions about the data distribution. They calculate the nearest neighbours of an object using suitable distance calculation measure such as Euclidean or Mahalanobis distance and use K-NN or similar clustering algorithms to group them. However, the computation grows exponentially with data size as they are based on calculation of distances between all objects. The computational complexity is directly proportional to both the dimensionality of the data and the number of objects.

Knorr *et al.* [37] proposed the DB(p_t, d_t) Outlier detection scheme, wherein an object o is considered to be an outlier if at least fraction p_t of the total objects have greater than d_t distance to o. They defined several ways to find such objects. For instance the index based approach computes distance range using spatial index structure and excludes an object if its d_t-neighbourhood contains more than $1 - p_t$ fraction of total objects. They proposed nested loop algorithm to avoid the cost of building an index. They also proposed building a grid such that any two objects from the same grid cell have a distance of at most d_t to each other. This way objects need to be compared to those from neighboring cells to check if they are outliers.

Ramaswamy *et al.* [62] proposed an outlier scoring scheme based on k-NN distances to produce a ranked list of potential outliers. k-NN distance of an object is its outlier score. Since the entire distance matrix must be calculated for all objects, this approach is susceptible to high computational costs. So they included techniques for speeding the k-NN algorithm such as partitioning the data into cells. If any cell and its directly adjacent neighbours contain more than k objects, then the objects in the cell are deemed to lie in a dense area of the distribution so the objects contained are unlikely to be outliers. If the number of objects lying in cells more than a specified distance apart is less than k then all objects in the cell are labeled as outliers.

2.1.3 Density Based Outlier Detection

The general idea of density based outlier detection is to compare the density around a point with the density around its local neighbors. The relative density of a point compared to its neighbors is computed as an outlier score. This class of approaches differ in how to estimate density of a given population.

Since distance based outlier detection models have problems with different densities, relative density is used to compare the neighborhood of points from areas of different densities. One determines the difference of an object from its nearest neighbors rather than from the entire set of data considered as a whole. To reveal outliers, a special metric, the outlier factor (OF), is introduced. Then, a threshold value is set, and all objects whose outlier factors exceed this value are considered to be outliers. Two metrics - local outlier factor (LOF) and connectivity based outlier factor (COF), introduced in [9, 8] and [73] respectively are most often used. LOF is characterized by a high rate of the outlier detection, but it also has a quadratic complexity.

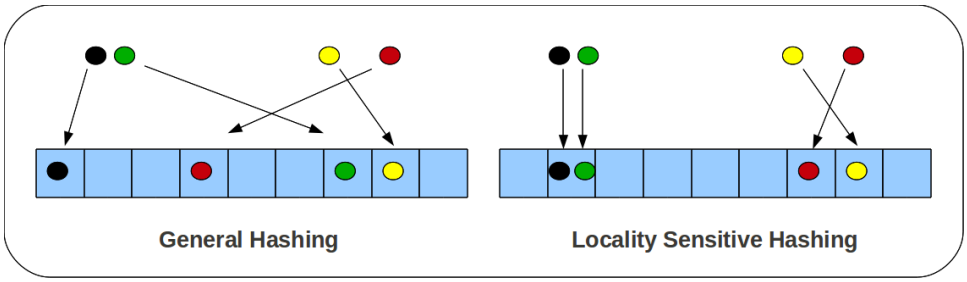


Figure 2.1: Conceptual difference between general hashing and LSH.

2.2 Locality Sensitive Hashing

We briefly give the overview of Locality Sensitive Hashing (LSH), which will later be used in our privacy preserving outlier detection scheme. The idea of Locality Sensitive Hashing was first introduced by Indyk *et al.* [31]. The basic concept of LSH is to hash all the objects such that similar objects are hashed to the same bin with high probability. Figure 2.1 shows the conceptual difference between regular hash function and LSH. Mathematically, this idea is formalized as follows:

Definition 1 *A family $H = h : S \rightarrow U$ is called (r_1, r_2, p_1, p_2) sensitive if for any two objects p, q in S, with $d(p, q)$ is the distance between objects p and q, following properties hold:*

$$if\ d(p,q) \leq r_1 : Pr[h(p) = h(q)] \geq p_1 \tag{2.1}$$

$$if\ d(p,q) \geq r_2 : Pr[h(p) = h(q)] \leq p_2 \tag{2.2}$$

For the hashing scheme to be locality sensitive, two conditions to be satisfied are $r_2 > r_1$ and $p_2 < p_1$. In order to amplify the gap between the probabilities p_1 and p_2, standard practice is to concatenate several hash functions to obtain a function family $G = \{g : S \rightarrow U^k\}$ such that $g(p) = (h_1(p), h_2(p), \ldots h_k(p))$; where k is the width of each hash function and $h_i \in H$. For a hash function family G, the probabilities in Equation 2.1 and 2.2 are modified as:

$$if\ d(p,q) \leq r_1 : Pr[g(p) = g(q)] \geq p_1^k \tag{2.3}$$

$$if\ d(p,q) \geq r_2 : Pr[g(p) = g(q)] \leq p_2^k \tag{2.4}$$

When $0 < p_1, p_2 < 1$ and k is large positive then both the probabilities p_1^k and p_2^k will be very low. However, p_2^k will be much smaller than p_1^k because $p_2 < p_1$. This will amplify the gap between good collision (between objects within r_1) and bad collision (between objects more than r_2 distance apart). Although this will amplify the gap between probabilities, overall probabilities will be reduced. To boost

the probability of collision for near neighbors, each object is hashed using L hash functions randomly drawn from G and stored in the corresponding hash tables. i.e. each object $o \in D$ is stored in the bins $g_j(o)$ for $j = 1, 2, \ldots L$; where each g is drawn independently and uniformly at random from G. At query time, the query object is hashed using the same L hash functions and all the database objects that are hashed into the corresponding L bins, are retrieved for near neighbor search.

Though the proposed privacy preserving algorithms are independent of the underlying hash function used, we briefly present an LSH scheme based on the p-stable distributions introduced in [17]. Intuitively, this hashing scheme projects all the objects onto a random line and then divide this random line into multiple equal length segments to generate bins. Thus, objects which are closer will fall into a single bin with high probability. Formally, each hash function $h_{a,b}(v) : R^d \rightarrow N$ maps a d dimensional vector v onto the set of integers. For a fixed a and b the hash function $h_{a,b}$ is given by:

$$h_{a,b}(v) = \lfloor \frac{a.v + b}{w} \rfloor \tag{2.5}$$

where a is a d dimensional random vector whose entry is drawn from a p-stable distribution, b is a real number randomly chosen from the range $[0, w]$ and w is the bin width. Let $f_p(t)$ denotes the probability density function of the absolute value of the p-stable distribution. For two vectors v_1 and v_2, $a.v_1 - a.v_2$ is distributed as cZ, where $c = ||v_1 - v_2||_p$ [†] and Z is a random variable drawn from a p-stable distribution ($Z \sim D_p$). Now, the probability of collision between two vectors v_1 and v_2 is given by,

$$p(c) = Pr[h_{a,b}(v_1) = h_{a,b}(v_2)] = Pr[\lfloor \tfrac{a.v_1+b}{w} \rfloor = \lfloor \tfrac{a.v_2+b}{w} \rfloor]$$

The collision will occur when $|a.v_1 - a.v_2| < w$ and a divider does not fall between the projections $a.v_1$ and $a.v_2$. The first condition can be rewritten as $|(v_1 - v_2).a| < w$. According to the definition of p-stable distribution, this condition is equivalent to:

$$|||v_1 - v_2||Z| \equiv |cZ| < w, where Z \sim D_p$$

The probability that the divider falls between $a.v_1$ and $a.v_2$ is $\frac{|(v_1-v_2).a|}{w}$. Thus the probability of collision is:

$$p(c) = \int_{t=0}^{w/c} f_p(t)(1 - \frac{tc}{w})dt \tag{2.6}$$

For a fixed bin width w, the collision probability $p(c)$ depends only on the distance c and is monotonically decreasing in c. In other words, for smaller distance (closer objects) the collision probability is higher. Thus, as per Definition 1 the family of hash functions above is (r_1, r_2, p_1, p_2) sensitive for $p_1 = p(r_1)$ and $p_2 = p(r_2)$ for $r_2 = r_1(1 + \epsilon)$, where $\epsilon > 0$ is an approximation factor.

The LSH scheme can be used to solve the approximate nearest neighbor problem which is proposed in [24]:

[†] $||X||_p$ denotes the L_p-norm of X.

Definition 2 *(ϵ-Nearest Neighbor): Given a set D of objects in a d-dimensional Euclidean space, preprocess D so as to efficiently return an object $o \in D$ for any given query object q, such that $d(o,q) \leq (1+\epsilon)d(q,D)$ where $d(q,D)$ is the distance of q to its closest object in D.*

Definition 3 *((R,c)-NN problem): Given a set D of objects in a d-dimensional Euclidean space and parameters $R > 0$, $\delta > 0$, construct a data structure which, given any query object q, does the following with probability $1-\delta$: For any query q, if there exists $o \in D$ such that $d(q,o) \leq R$, find an object $o' \in D$, such that $d(q,o') \leq cR$.*

The ϵ-Nearest Neighbor (ϵ-NN) problem can be reduced to the (R,c)-NN problem, which is a decision version of the nearest neighbor problem [17], where $c = (1+\epsilon)$. The (R,c)-NN problem is solved by using the LSH scheme on the dataset D by setting the parameter $r_1 = R$ and $r_2 = cR$. To process the query q, all L bins (every object is hashed using L hash functions, so q will go to L hash bins, one for every hash table.) to which q is hashed, are searched and for each object o in those bins, if $d(o,q) \leq r_2$ YES is returned, else NO is returned.

Many improvements in terms of space and time complexity of LSH has been proposed recently. Multi-probe LSH [49] reduces the number of hash tables by an order of magnitude by intelligently probing multiple buckets that are likely to contain query results in a hash table. Random projections [31, 11] used in LSH are dataset independent and make no prior assumption about the data. However, they require large code length for good retrieval accuracy. Norouzi *et al.* [54] proposed a method for learning similarity preserving hash functions that map high dimensional data onto binary codes. The formulation is based on structure prediction with latent variables and a hinge-like loss function. Recently, Kulis *et al.* [40] proposed a locality-sensitive hashing scheme to accommodate arbitrary kernel functions, making it possible to preserve the algorithm's sub-linear time similarity search guarantees for a wide class of useful similarity functions. In this thesis, we have used the original version of LSH without any of these improvements. However, in future our method can be extended to incorporate these improvements.

2.3 Vocabulary Tree

Sivic *et al.* [70] proposed a retrieval method for images using a popular text retrieval approach called Bag of Words (BoW). The idea is to quantize extracted local features of an image into visual words defined by centers of k-means clustering performed on a large set of training image descriptors. Each image is then represented by a weighted histogram of these visual words and Term Frequency Inverse Document Frequency (TF-IDF) scoring with inverted file structure is applied to get the relevant images. The retrieval quality of this method is directly proportional to the quantization error and to reduce the quantization error we need large number of visual words which will increase the time of quantization. To solve this problem Nister *et al.* [53] proposed an efficient scheme of hierarchical

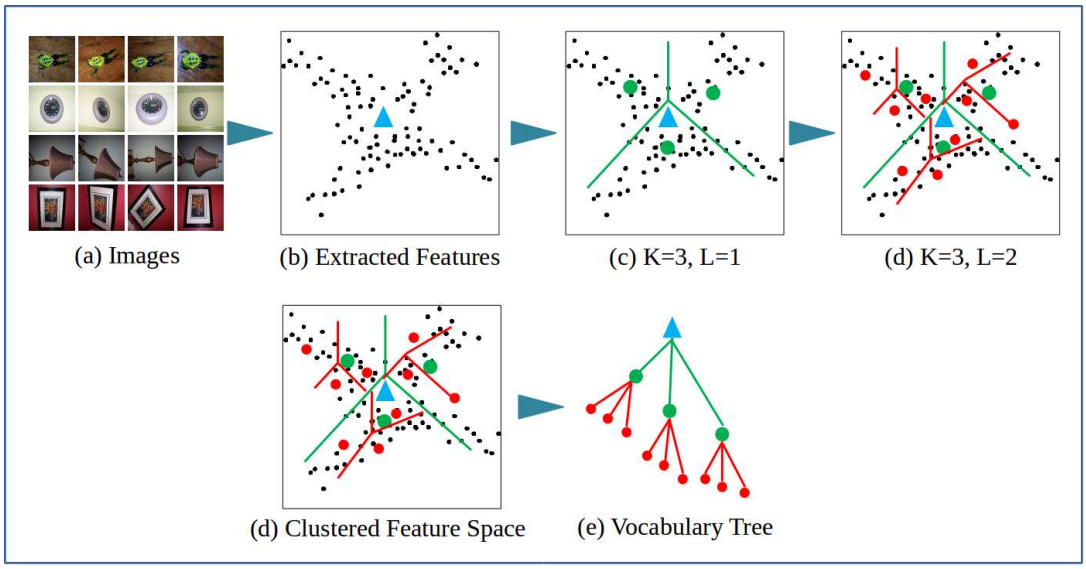


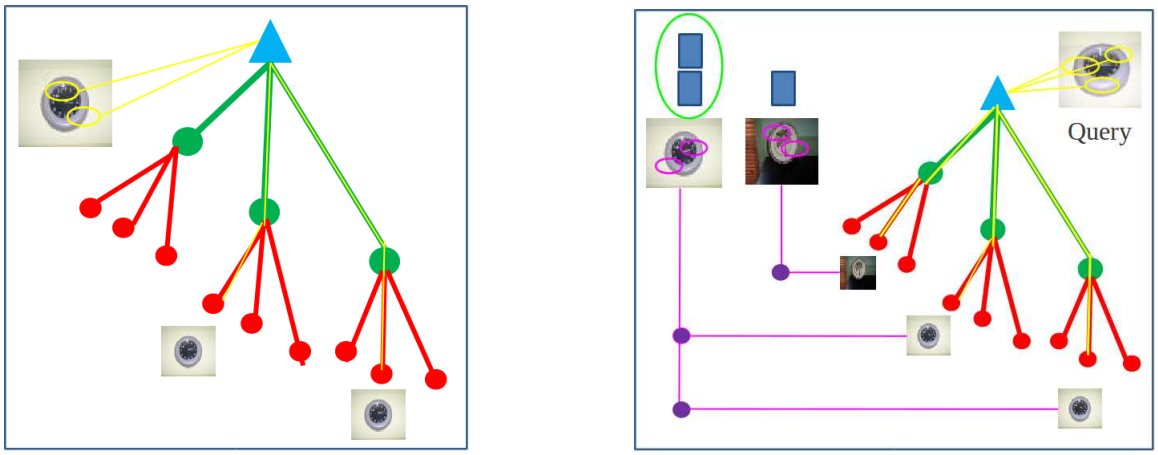
Figure 2.2: Building Vocabulary Tree: First, the features are extracted from training images. Then, hierarchical k-means is performed to cluster those features. Finally, those clusters form visual words.

quantization and scoring using vocabulary tree. Here we give a brief overview of their method. More details are available in [53]. The process of building, quantizing, and searching is explained through an example in Figure 2.2, 2.3(a) and 2.3(b) respectively.

Visual Words: In text retrieval system based on BoW approach, all documents are represented by a vector of word frequencies. A text is retrieved by computing its vector of word frequencies and returning the documents with the closest vectors. To use this approach in the context of images, a visual analogy of a *word* called *visual word* is required. Visual words are defined by vector quantizing the feature vectors of an image. A set of features are extracted from a large collection of training images and clusters are formed by clustering these features. These clusters are called visual words and any new feature is quantized into the closest cluster. This way, all the features of an image are mapped to their respective visual words. An image is represented by a vector of visual word frequencies same as a document is represented by a vector of word frequencies.

Building Vocabulary Tree: A set of local features are extracted from a large collection of training images and hierarchical k-means clustering is applied on those features to build the tree. Here k is the branching factor, that is at each node k-means clustering is applied on the data of that node to generate k-child nodes and corresponding k-cluster centers are stored in that node. At root node all the data is considered and partitioned into k clusters and the process is repeated recursively to some predefined depth h, which results into vocabulary tree with $n = k^h$ leaf nodes that acts as visual words.

Quantization: To represent an image as BoW, a set of local features are extracted from an image. Each feature vector is compared with the k-cluster centers of the node and a child node having the closest cluster center is selected. Starting from the root node this process is repeated until the leaf node is reached giving the path down the tree for every feature vector which will be used for scoring. This way



(a) Quantization (b) Search

Figure 2.3: Quantization: Image is quantized by traversing vocabulary tree based on its extracted features. Search: Query image is quantized using vocabulary tree and all the database images that share traversal path with the query image are scored.

all the features of an image are mapped to one of the visual words (represented by leaf nodes). Finally, an image is represented by a histogram of visual words (of length equal to number of leaf nodes). For every visual word, the histogram denotes the number of image features quantized into the corresponding leaf node. All the images in a database are quantized and their respective traversal path are stored as a preprocessing step.

Search: Intuitively we want to retrieve the images which share maximum number of similar features with the query image. This can be done by quantizing query image as described above and a score is computed with every database image based on the similarity of their traversal path. Rather than computing scores with all the database images we can apply inverted index scheme to compute the score with only those database images whose traversal path share at least one node with that of the query image. We are omitting the database images who do not have any visual word common with the query image. In other words those are the images who do not share any features with the query image. Hence, their relevance score will be zero and we can safely omit those images during search.

Scoring: Both query image and database image are represented by a histogram of visual words (leaf nodes) and a score is given to the database image based on the normalized difference between these two vectors. If Q and D are query image and database image respectively then i^{th} value of their respective vectors is given by,

$$Q_i = n_i w_i$$

$$D_i = m_i w_i$$

Here, n_i and m_i are the number of features of a query image and database image respectively, whose traversal path pass through node i. In an analogy with text retrieval these values corresponds to Term Frequency (TF) of the i^{th} term (word). w_i is the weight of the node and is given by,

$$w_i = ln\frac{N}{N_i}$$

where, N is the total number of images in database and N_i is the number of images having at least one feature whose traversal path pass through node i, which can be thought of as Inverse Document Frequency (IDF). The relevance score of database image D, given query image Q is defined as,

$$s(q,d) = ||\frac{Q}{||Q||_p} - \frac{D}{||D||_p}||_p$$

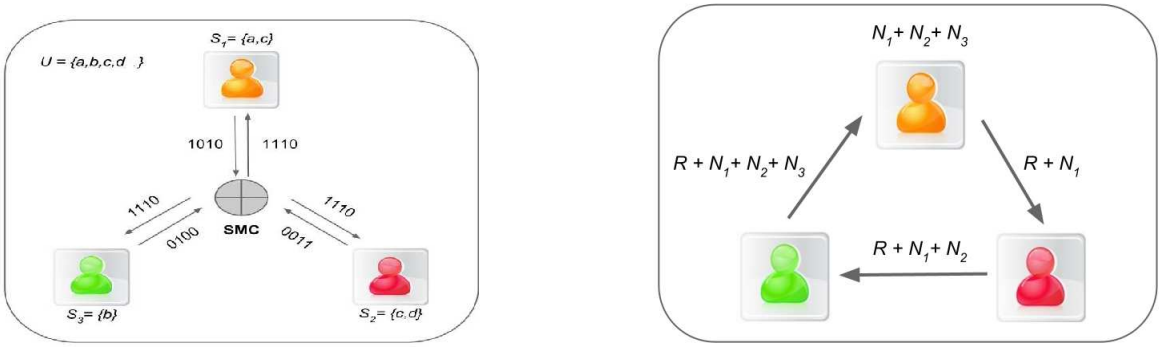
Here, $||X||_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$ denotes the L_p-norm for X. Usually, L_1-norm or L_2-norm is used for scoring. The database images having high relevance score are returned as a result. The computation of scores can be made faster by storing inverted files at every node in a vocabulary tree. The inverted file of a node will store the list of database image IDs which has a traversal path through this node along with their frequency. The frequency of an image for a specific node will not change because the number of features of an image that has a traversal path along that node remains the same. In other words, frequency of an image for a node (visual word) is the number of times that visual word occurring in that image, which is constant. Thus given a query image its features are propagated down the tree and at each node its inverted file is accessed to compute the scores with corresponding database images.

2.4 Cryptographic Primitives

In this section we give brief overview of two cryptographic primitives - Secure Union and Secure Sum, which are used later in the thesis to build our privacy preserving protocols. Although, many algorithms exists for these primitives with varying complexity and security, we give only one specific instance of each for the ease of explanation. Note that, the proposed methods use these primitives as a black box and are independent of the underlying algorithm.

2.4.1 Secure Union

We use Secure Union, in our privacy preserving outlier detection method, to securely compute label information of the global LSH bin structure. Figure 2.4(a) shows the conceptual view of secure union protocol. If the domain of the set elements is small, then secure union of sets can be computed using SMC protocols. First, each player constructs a string of length n filled with 0 at every position, where n is the cardinality of the domain of the set elements. Then, each player sets value 1 to the position whose corresponding element exists in her set. Finally, all players securely perform OR operation on all the



(a) Secure Union

(b) Secure Sum

Figure 2.4: Conceptual View of Cryptographic Primitives

strings using SMC to find the union of all the sets. However, due to large domain size we can not use this approach in our protocol.

Another approach for Secure Union is based on commutative encryption [14]. The idea is that each player encrypts their set of elements as well as the encrypted elements from other players and later on decrypt the entire set to get the union. For encryption to be commutative, applying encryption on the same plain text with different keys in any order will generate the same cipher text. Thus, duplicate elements can be detected as they will have same cipher text and can be removed. However, this will reveal number of common elements across the players. Note that only duplicate elements can be detected as the probability of generating same cipher text on two different plain text using any number of keys and order is negligible. Due to commutative property decryption can also be performed in any order. So, every player applies permutation before decrypting the elements to prevent players form tracking the source of an element. The protocol assumes that players are honest but curious and non-colluding. Efficient collusion resistant protocol for Secure Union is given in [36, 22].

2.4.2 Secure Sum

We use Secure Sum protocol, in our privacy preserving outlier detection method, to securely compute global LSH bin count. We briefly describe the protocol for Secure Sum given in [14]. Figure 2.4(b) shows the conceptual view of the secure sum protocol.

Assume that there are m players $P_1, P_2, \ldots P_m$, with each having corresponding values $v_1, v_2, \ldots v_m$. They want to securely compute the summation $v = \sum_{i=1}^{m} v_i$ where $v < N$. The master player say P_1 generates a random number R uniformly distributed in $Z_n = \{0, 1, \ldots N-1\}$. After that, P_1 computes $R + v_1$ mod N and sends the result to P_2. Because of the addition of uniformly generated random number R, the value received by P_2 is independent of v_1. Hence, P_2 learns nothing about v_1. Now, Player P_2 adds its own value v_2 to the received number to get $R + v_1 + v_2$ mod N and sends it to P_3

and so on. Finally, after receiving $R + \sum_{i=1}^{m-1} v_i \bmod N$, P_m computes $R + \sum_{i=1}^{m} v_i \bmod N$ by adding its own value v_m and sends the result back to P_1. Since, P_1 knows the value of R it subtracts it from the received value and computes the desired result v. Note, that P_1 can learn $\sum_{i=2}^{m} v_i$ by subtracting v_1 from the computed result v. However, this can be learned from the global result irrespective of how it has been computed.

The proposed method works well under the assumption of three or more players communicating over secure channel with no collusion. For example, player P_{i-1} and P_{i+1} can collaborate to compute v_i. One approach to handle collusion in case of honest majority, is to divide each v_i into multiple shares and then sum of each share is computed individually. However, the path used is permuted for each share, such that no player has the same neighbor twice. To compute v_i, the neighbors of i from each iteration would have to collude. Varying the number of shares varies the number of colluding players required to violate security. Efficient collusion resistant protocol for Secure Sum is given in [80].

Chapter 3

Privacy Preserving Outlier Detection using Locality Sensitive Hashing

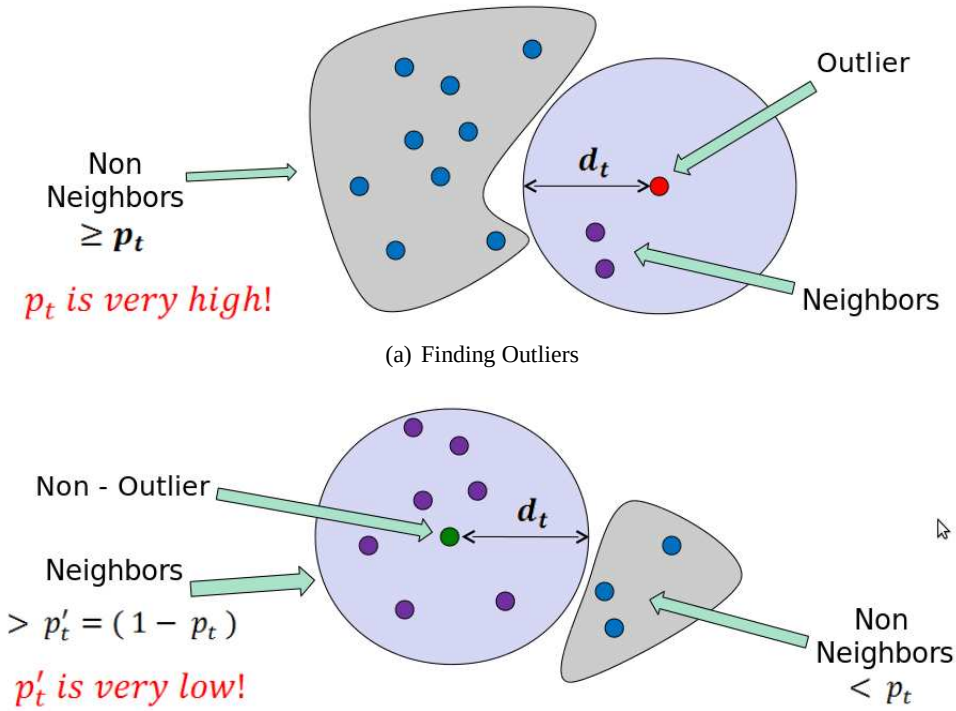
In this chapter, we give an approximate algorithm for distance based outlier detection using Locality Sensitive Hashing (LSH) technique. First, we propose a constant round protocol for outlier detection with low communication cost, in a distributed setting with horizontal partitioning, where each player has a subset of the total number of objects. Our central idea is to use LSH to efficiently prune all the objects which can not be outliers and process only the remaining objects to find the actual outliers. Our distributed algorithm is efficient and scalable in terms of communication cost as the amount of actual data communicated between players is extremely low compared to the entire database. Next, we extend this algorithm for privacy preserving distance based outlier detection using secure LSH evaluation. Each player will communicate to securely evaluate LSH using cryptographic primitives - Secure Union and Secure Sum. The communication complexity of the proposed algorithm is independent of the dimensionality of the data, hence it is efficiently scalable for high dimensional data. We also compare our algorithm with the existing state of the art technique and show that it is better in terms of both communication as well as computational cost.

3.1 Outlier Detection using LSH

Our solution is based on the approach of [63], where authors presented an outlier detection scheme (*CentralizedOD*) using LSH in a centralized setting. We extend their approach in a distributed setting which also preserves data privacy. *CentralizedOD* uses distance based outlier definition introduced by Knorr *et al.* [37], given as below:

Definition 4 *$DB(p_t, d_t)outlier$: An object o in a dataset D is a $DB(p_t, d_t)$ outlier if at least fraction p_t of the objects in D lie at a distance greater than d_t from o.*

where $DB(p_t, d_t)$ is a shorthand notation for a distance based outlier detection using parameters p_t and d_t. The above definition says that an object o is an outlier if a very large fraction p_t of the total objects in the dataset lie outside the radius d_t from o.



(a) Finding Outliers

(b) Finding Non-Outliers

Figure 3.1: *CentralizedOD* Approach: Pruning non-outliers is efficient because its easy to find small number of neighbours.

The idea is to use converse of this definition and consider an object to be a non-outlier if it has sufficient neighbours (more than p'_t) within distance d_t, where $p'_t = (1 - p_t) \times |D|$. Since the fraction p_t is very high (usually set to 0.9988 [38]), the modified point threshold p'_t will be very less compared to the number of objects in D. This allows for easy detection of most of the non-outliers by finding more than p'_t objects within distance d_t.

LSH is used to efficiently find the near neighbours of an object. Given a set of objects, LSH scheme hashes the database objects in such a way that all the objects within a specified distance are hashed to the same value with high probability. This way, all those non-outliers in the data which have many near neighbours can be identified easily, without calculating the distances to every other object in the dataset. Moreover, using LSH properties, whenever we identify a non-outlier we will be able to say most of its neighbours as non-outliers without even considering them separately. This leads to a very efficient pruning technique, where most of the non-outliers in the dataset can be easily pruned. The remaining objects after pruning are the set of probable outliers which may contain very few non-outliers. To further remove these non-outliers, the probabilistic nature of LSH is used. The idea is to take the intersection

of the sets of probable outliers over multiple runs of an algorithm, to output the final set of approximate outliers. The approach works because each set of probable outliers will contain the actual outliers with high probability. Since, the proposed approach is based on LSH which is probabilistic in nature, the output of outlier detection algorithm will have false positives and false negatives. In the context of outlier detection, a false positive is to label a non-outlier as an outlier and false negative is to label an outlier to be a non-outlier. The bounds on both false positives and false negatives are given in [63].

3.1.1 Centralized Outlier Detection

We now describe LSH based centralized algorithm for outlier detection - *CentralizedOD*. Using LSH, instead of finding the nearest object in the dataset, *CentralizedOD* finds the number of objects lying with in distance d_t from an object. Given a dataset D and the parameters d_t and p_t, LSH scheme is executed on D by setting the parameter $r_1 = d_t/(1+\epsilon)$, where ϵ is an approximation factor. For an object o from the dataset D, all L bins to which it is hashed are considered. A set S of all the objects stored in those L bins (without removing duplicates) is constructed. The objects in this set are the probable neighbours of o. More precisely, from Equation 2.2, we know that each object in S is within the distance $r_2 = (1+\epsilon) \times r_1 = d_t$ from o, with a probability at least $(1-p_2^k)$. To boost this probability, we consider only those objects which are repeated more than $b_t (bt \leq L)$ times in the L bins and store only those objects in a new set S'. In other words, we are reducing the error probability of considering an actual non-neighbour as a neighbour of the object o. Here, b_t is a *Bin Threshold* which can be computed based on the desired false negative probability. A method to compute the optimal value of b_t is given in [63].

If the cardinality of S' is greater than the modified point threshold p'_t then with a very high probability o cannot be an outlier since it has sufficient neighbours within distance d_t. Moreover, this holds true for all the objects in S' i.e., every object other than o also has more than p'_t neighbours with in the distance d_t, so they all can not be outliers (with a very high probability). Hence, all the objects in S' are marked as non outliers. If, on the other hand the cardinality of S' is less than or equal to the point threshold p'_t, the object o is marked as a probable outlier. This procedure is repeated till all the objects are either marked as non-outliers or probable outliers. The resulting set of probable (approximate) outliers can have a few false positives. These false positives can be removed by again considering each of the probable outliers o' and calculating its distance to every object in D. If the number of objects in D lying at distance greater than d_t from o' is at least fraction p_t of the size of D then o' is marked as an actual outlier. After processing each probable outlier, we have the set of the actual outliers in the dataset D.

We use above centralized algorithm in our privacy preserving protocol for horizontally distributed data. Each player locally computes its own set of probable outliers using the centralized algorithm with global distance and point threshold parameters. In the next phase, all the players securely communicate to obtain their subset of the actual outliers in the total database. We consider Honest-But-Curious (HBC) adversary model [25], where an adversary must follow the protocol correctly but can infer any additional information from the data it has.

3.2 Prior Work

The foremost algorithms introduced for outlier detection are statistics based [65, 77]. These methods have the limitation that they assume the data distribution is known before hand. Another limitation of the statistical methods is that they do not scale well to large datasets or datasets of high dimensions. To overcome these limitations several approaches have been proposed to efficiently mine the outliers. These include density based approaches [9, 57], distance based [37, 38, 62, 4], depth based [35] and clustering based methods [21, 28, 84].

The notion of distance based outliers was introduced by Knorr *et al.* in [37]. They introduced a definition for distance based outlier detection and gave two algorithms [37, 38] for outlier detection. The first one is a nested loop algorithm which is inefficient while dealing with large datasets due to its quadratic complexity in the database size. The second algorithm is cell-based and is linear in terms of the database size. However, it is exponential in terms of the data dimensionality and is inefficient for datasets of high dimensions ($d > 4$). Ramaswamy *et al.* [62] propose a slightly modified definition for distance based outliers based on k-nearest neighbour and propose a partition-based algorithm for mining outliers. Subsequent approaches [4, 5, 74] use efficient data structures and pruning techniques to further reduce the complexity of outlier detection to near linear in the dataset size. Recently, in [78], an efficient method for outlier detection has been proposed using the LSH technique. Their method is based on the definition of an outlier provided in [62], whereas our approach uses the definition provided in [37]. Also, most of these techniques are designed for centralized setting and do not scale well to the distributed setting.

In the case of distributed setting, an outlier detection scheme for wireless sensor networks was presented in [7]. Distributed outlier detection schemes handling mixed attribute datasets are proposed in [56, 39]. In [18] a distributed top-k outlier detection scheme is presented using Distributed Exploration of Massive Astronomy Catalogs. They use PCA based technique to give distributed outlier detection scheme for vertically partitioned data. An outlier detection scheme for distributed data streams is presented by Su *et al.* [72].

Privacy preserving outlier detection (PPOD) was introduced by Vaidya et al. in [75]. They use the definition for distance based outliers provided in [37], and give PPOD algorithms for both horizontal and vertical partitioning of data. Subsequently, a PPOD algorithm using the k-nearest neighbour based definition [62] was given in [85], considering only vertical partitioning, where data is distributed across the dimensions. However, all of the above mentioned algorithms have quadratic communication and computational complexities in the database size, making them infeasible while dealing with large datasets. To the best of our knowledge, no other work in the field of PPDM based on cryptographic techniques has addressed distance based outlier detection. Privacy preserving density based outlier detection algorithms have been proposed in [16, 68].

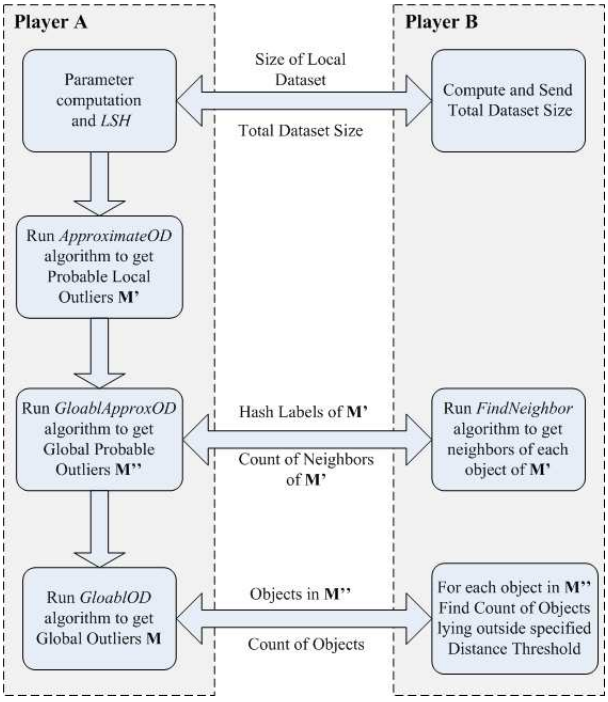


Figure 3.2: Distributed Outlier Detection using LSH

3.3 Distributed Outlier Detection

First, we give outlier detection algorithm for horizontally partitioned data without considering privacy. Consider a distributed setting with p players, each player having a subset of objects in the whole database. In this setting, each player first computes its set of local outliers by using the centralized algorithm on its local dataset. After the local outliers are generated, all the players communicate to compute the global outliers from the sets of local outliers. At the end of the algorithm each player will have its subset of the actual global outliers.

We consider the horizontal distribution where each player has a subset of the total number of objects. The distributed algorithm *DistributedOD* (Algorithm 1) is divided broadly into three phases. In the first phase, all players communicate to compute the global parameters. Then each player locally computes its set of local probable outliers M'. In the second phase *GlobalApproxOD* (Algorithm 4), the players engage in communication to compute their subsets of global probable outliers. Finally, in the third phase *GlobalOD* (Algorithm 5), the players again engage in communication to compute their subsets of the actual global outliers. Figure 3.2 gives an overview of the process in the distributed setting from the perspective of one player in a two player setting. It is clear from the figure that the round complexity of our algorithm is 3, which also holds true for multi player setting.

Before giving a detailed description of the algorithm, we define global and local outliers as follows:

Algorithm 1 DistributedOD: Outlier Detection Algorithm for Horizontal Distribution

Require: Players P^A and P^B, P^A's Dataset D^A, P^B's Dataset D^B, Distance Threshold d_t, Point Threshold p_t, Approximation Factor ϵ
Ensure: P^A's Outliers M^A
1: At P^A :
2: P^A sends $|D^A|$ to P^B
3: At P^B :
4: $n = |D^A| + |D^B|$
5: P^B sends n to P^A
6: At P^A :
7: $p'_t = (1 - p_t) \times n$
8: $R = d_t/(1 + \epsilon)$
9: $T^A_{L\times H} = LSH(D^A, R)$
10: compute b_t
11: $M'^A = ApproximateOD(D^A, T^A_{L\times H}, p'_t, b_t)$
12: $M''^A = GlobalApproxOD(M'^A)$
13: $M^A = GlobalOD(M''^A)$

Definition 5 global outlier*: Given a distance threshold d_t and a point threshold p_t, an object o in a dataset D is a global outlier if at least fraction p_t of the objects in D lie at a distance greater than D from o.*

Definition 6 local outlier*: Given a distance threshold d_t and a point threshold p_t, an object o with player P_i is a local outlier if the number of objects in the local dataset D_i lying at a distance greater than D is at least a fraction p_t of the total dataset N.*

3.3.1 Algorithm

We give the distributed algorithm in a two player setting, which can be easily extended to a p player setting. Consider two players denoted by P^A and P^B with local datasets D^A and D^B. We present the algorithm such that one player, say P^A will be able to compute its subset of the global outliers at the end of the algorithm. Similarly the algorithm can be used to enable P^B to compute its subset of the global outliers by simply interchanging the roles of P^A and P^B in the algorithm.

In the first phase, player P^A sends the size of her dataset to P^B and gets the size of the entire dataset (i.e. $|D^A$+$D^B|$). Player P^A locally computes its local probable outliers M'^A by running the centralized algorithm on her dataset D^A. In the second phase, for each object o in the set M'^A, P^A forms the set l^A_o which is the set of L labels of the bins (generated by LSH) to which o is hashed to (these labels are stored while performing LSH). P^A sends the set l^A to P^B. Player P^B considers each element l in l^A and runs the FindNeighbors algorithm on its local dataset. P^B then counts the cardinality of the sets

Algorithm 2 ApproximateOD: Calculate Local Probable Outliers

Require: Dataset D, Hash Table $T_{L\times H}$, Point Threshold p'_t, Bin Threshold b_t,
Ensure: Outliers M'

$M' = \{\}$
$\forall o \in D, flag[o] = 0$ // mark all objects as outliers
for each object o in D **do**
  if $flag[o] = 0$ **then**
    $l_o = \{g_i(o)|i = 1toL\}$
    $S' = FindNeighbors(D, T_{L\times H}, o, l_o, b_t)$
    if $|S'| > p'_t$ **then**
      $\forall o' \in S', flag[o'] = 1$ // mark objects in S' as non-outliers
    else
      $M' = M' \cup \{o\}$ // add current object into set of probable outliers M'
    end if
  end if
end for

Algorithm 3 FindNeighbors: Calculate Neighbours using LSH

Require: Dataset D, Hash Table $T_{L\times H}$, object o, hash labels l_o, Bin Threshold b_t
Ensure: Set S' of objects satisfying Bin Threshold criteria

$S = S' = \{\}$
for $i = 1$ to $|l_o|$ **do**
  $S = S||T[i, l_o[i]]$
end for
$\forall o' \in D, count[o'] = 0$
for each object $o' \neq o$ in S **do**
  $count[o'] = count[o'] + 1$
  if $count[o'] > b_t$ **then**
    $S' = S' \cup \{o'\}$
  end if
end for

returned by FindNeighbors and sends the set of counts c^B to P^A. Player P^A considers each object o in M'^A and runs the FindNeighbors algorithm to obtain its count of the number of objects repeating more than b_t times (this step is actually redundant if P^A stores this count in first phase). P^A then computes the sum of this count and the corresponding count in c^B, and if this sum is less than or equal to point threshold p'_t, P^A stores o in the set of global probable outliers M''^A.

The set M''^A contains some false positives because of the LSH approximation, which can be removed in third phase with another round of communication as follows: P^A sends the set M''^A to P^B. Player P^B considers each object o in M''^A and computes the distance to each object in her dataset D^B and counts the number of objects which lie at a distance greater than the distance threshold d_t from o. P^B sends all the counts back to P^A. P^A then considers each object o in M''^A and computes the distance to each object in its dataset D^A and counts the number of objects lying at distance greater than d_t. P^A then

Algorithm 4 GlobalApproxOD: Calculate Global Probable Outliers

Require: Players P^A and P^B, P^A's Dataset D^A, P^B's Dataset D^B, Point Threshold p'_t, Bin Threshold b_t, P^A's Hash Table $T^A_{L\times H}$, P^B's Hash Table $T^B_{L\times H}$, P^A's Local Approximate Outliers M'^A
Ensure: P^A's Global Approximate Outliers M''^A

At P^A :
for each object o in M'^A **do**
  $l^A_o = \{g_i(o)|i = 1toL\}$
end for
send l^A to P^B
At P^B :
$\forall l \in l^A, c^B[l] = 0$
for each label l in l^A **do**
  $S' = FindNeighbors(D^B, T^B_{L\times H}, null, l, b_t)$
  $c^B[l] = |S'|$
end for
send c^B to P^A
At P^A :
$M''^A = \{\}$
for each object o' in M'^A **do**
  $l_o = \{g_i(o)|i = 1toL\}$
  $S' = FindNeighbors(D^A, T^A_{L\times H}, o', l_o, b_t)$
  $count = |S'| + c^B[o]$
  if $count \leq p'_t$ **then**
    $M''^A = M''^A \cup \{o'\}$
  end if
end for

computes the sum of this count and the corresponding count received from P^B and if the total count is greater than or equal to fraction p_t of n, P^A marks the object o as an outlier.

3.3.2 Example

For the ease of understanding we explain the work flow of the proposed algorithms using a simple example. First, we show how to compute outliers in a distributed setting without considering privacy (Algorithm 1). Later on in Section 3.4 we show how to compute outliers in a privacy preserving manner using Algorithm 7. Let's assume that player P^A has dataset D^A consists of 6 objects $a_1 = (1, 1), a_2 = (1, 3), a_3 = (2, 1), a_4 = (2, 3), a_5 = (5, 1)$ and $a_6 = (4, 5)$. Player P^B has dataset D^B consists of 4 objects $b_1 = (3, 1), b_2 = (4, 2), b_3 = (5, 2)$ and $b_4 = (4, 1)$. We fix point threshold $p_t = 0.8$, distance threshold $d_t = 2$ and approximation factor $\epsilon = 0$. The total dataset size is $n = 10$ and the modified point threshold is $p'_t = 2$. Initially, both the players will reveal the size of their dataset to each other. Any one player will compute the LSH parameters and publish them to the other. Using these parameters, both the players run LSH with radius $R = d_t/(1 + \epsilon) = 2$, on their respective datasets to compute bin

Algorithm 5 GlobalOD: Calculate Global Outliers

Require: Players P^A and P^B, P^A's Dataset D^A, P^B's Dataset D^B, Distance Threshold d_t, Point Threshold p_t, P^A's Global Approximate Outliers M''^A
Ensure: P^A's Global Outliers M^A

send M''^A to P^B
At P^B :
$\forall o'' \in M''^A, c^B[o''] = 0$
for each object o'' in M''^A **do**
  for each object o in D^B **do**
    if $Dist(o'', o) > d_t$ **then**
      $c^B[o''] = c^B[o''] + 1$
    end if
  end for
end for
send c^B to P^A
At P^A :
for each object o'' in M''^A **do**
  $count = 0$
  for each object o in D^A **do**
    if $Dist(o'', o) > d_t$ **then**
      $count = count + 1$
    end if
  end for
  $count = count + c^B[o'']$
  if $count \geq (p_t \times n)$ **then**
    $M^A = M^A \cup \{o''\}$
  end if
end for

structures T^A and T^B as shown in Table 3.1 and Table 3.2 respectively. In each bin structure, we have $L = 3$ different hash functions creating 3 hash tables. A bin structure is considered as a matrix and each entry of this matrix is accessed by indexing a two dimensional array. Thus, $T^A[3, 2]$ denotes the objects that are hashed to second bin of the third hash table, i.e. $T^A[3, 2] = \{a_4, a_5\}$. For simplicity, we set bin threshold $b_t = 1$, i.e. an object is considered as neighbour if its hashed to the same bin as the query object at least once.

ApproximateOD: Both the players run *ApproximateOD* to get the local probable outliers using their respective LSH bin structure. All the objects who are not already marked as non-outliers are processed one by one. For every object, all L bins to which it is hashed, are considered. A union of all the objects which occur more than b_t times in those L bins forms the neighbour set. If the size of this neighbour set is greater than point threshold p'_t, then all the objects in this neighbour set and the current object are marked as non-outliers, otherwise the current object is marked as local probable outlier. For example, player P^A, considers first object a_1, which is hashed into 3 bins $\{1, 2, 1\}$ using 3 hash functions. The

Table 3.1: Player P^A's LSH Bin Structure T^A

$\{a_1, a_2, a_3, a_4\}$	$\{a_6\}$	$\{a_5\}$
$\{a_6\}$	$\{a_1, a_2, a_3, a_4\}$	$\{a_5\}$
$\{a_1, a_2, a_3\}$	$\{a_4, a_5\}$	$\{a_6\}$

Table 3.2: Player P^B's LSH Bin Structure T^B

$\{b_1, b_4\}$	$\{\}$	$\{b_2, b_3\}$
$\{\}$	$\{b_4\}$	$\{b_1, b_2, b_3\}$
$\{b_1, b_4\}$	$\{b_2, b_3\}$	$\{\}$

neighbours of a_1 are the objects which are hashed into those 3 bins and repeat more than b_t times. Thus, the neighbour set corresponding to a_1 is $\{a_2, a_3, a_4\}$, the size of which (C^A) is greater than p'_t. Hence, a_1, a_2, a_3 and a_4 are marked as non-outliers. Next, we consider a_5 for which the neighbour set is $\{a_4\}$. Since, a_5 does not have sufficient neighbours (more than p'_t), we add it to the set of local probable outliers M'^A. Similarly a_6 is also added to M'^A. In case of P^B, b_1 has neighbour set $\{b_4, b_2, b_3\}$, the size of which (C^B) is greater than p'_t. Thus, all the objects of D^B are marked as non-outliers.

GlobalApproxOD: The local probable outlier set may contains some non-outliers for which enough neighbours are not available locally, but are available in other player's database. To consider these neighbours and further prune the non-outliers, both players engage into communication to calculate global probable outliers. For every local probable outlier, player P^A sends $L = 3$ bin labels to which it is hashed to. Player P^B sends back the total number of objects C^B which repeat more than b_t times in the corresponding bins of T^B. After receiving the count of neighbours (C^B) from P^B, player P^A calculates the total number of neighbours $C = C^A + C^B$. If C is greater than p'_t the object is marked as non-outlier otherwise it is marked as global probable outlier. In our example scenario, P^A sends bin labels corresponding to a_5 and a_6 which are $\{3, 3, 2\}$ and $\{2, 1, 3\}$ respectively. P^B will calculate the neighbour set count C^B for each bin label set. In case of label set $\{3, 3, 2\}$, the count of objects repeating more than b_t times (C^B) in T^B is 3 because $b_t = 1$ and three objects b_1, b_2 and b_3 are hashed into those bins. Similarly, in case of bin label set $\{2, 1, 3\}$, the neighbour set count C^B is 0, as no objects of D^B are hashed into those bins. Player P^A receives 3 and 0 as neighbour counts corresponding to a_5 and a_6 from P^B and calculates global neighbour count C by adding own neighbour counts 1 and 0. Thus, global neighbour count for a_5 and a_6 is 4 and 0 respectively. Since, $4 > p'_t$ and $0 < p'_t$, a_5 is marked as non-outlier (enough neighbours are available globally) and a_6 is added into the global probable outlier set M''^A.

GlobalOD: Finally, both the players engage into communication to compute global outliers. For every global probable outlier, number of objects in both the datasets (D^A and D^B) which has greater than d_t distance is calculated. If this count is greater than point threshold p_t then the object is an outlier otherwise it is a non-outlier. In our example, a_6 is the global probable outlier. P^A sends a_6 to P^B. Both P^A and P^B compute the distance between a_6 and all other objects in the database D^A and D^B respectively. Both the players count the number of objects which have distance greater than d_t. The respective counts are $C^A = 5$ and $C^B = 4$ and the total number of objects which are more than d_t distance apart from a_6 is $C^A + C^B = 9$ which is greater than fraction p_t of the dataset size (i.e. $p_t \times n = 0.8 \times 10 = 8$). Thus, according to the Definition 4 a_6 is an outlier.

3.3.3 Analysis

Computational Complexity: We give the total computational complexity of a single player P^A, required to enable both the players to compute their subsets of the Global outliers. In DistributedOD, Fixing the parameters in steps 7 and 8 is of constant complexity. The LSH binning in Step 9 has a computational complexity of $O(n^A Ld)$. ApproximateOD on player P^A's input has a computational cost of $O(n^A Lm'^A)$, where $m'^A = |M'^A|$. In GlobalApproxOD, calculating the labels of each object in M'^A (Steps 2-4) has a complexity $O(m'^A L)$. While sending this set of labels (Step 5) to P^B, player P^A receives a corresponding set of labels for objects in M'^B from P^B. For each label in this set, P^A, needs to run the FindNeighbors protocol and get the count of the set returned by FindNeighbors. This has a computational complexity of $O(m'^B n^A L)$. Computing the set of global probable outliers from the local probable outliers (Steps $15-22$), has a complexity $O(n^A m'^A L)$. In GlobalOD, P^A sends the set M''^A to P^A (no computation). Similarly, P^A receives the corresponding set M''^B from P^B. For each object in this set P^A has to compute distances to all the objects in D^A. This would have a computational complexity of $O(m''^B n^A d)$. Finally, to compute its subset of the global outliers (Steps $13-24$), the computational complexity is $O(n^A m''^A d)$. Using the inequalities $m'^A, m'^B << n^A$; $m''^A, m''^B << n^A$ [63] and considering only the dominating terms, the total computational complexity of player P^A would be $O(n^A dL)$.

Communication Complexity: Similar to the computational analysis, We give the total communication complexity of a single player P^A, required to enable both the players to compute their subsets of the Global outliers. In DistributedOD, Step 2 has a communication of $O(\log(n^A))$. In GlobalApproxOD, step 5 has a communication of $O(m'^A L)$. In Step 12, P^A receives a count for each label in the set l^A which P^A has sent. P^A has to send to P^B, a similar count for each label in the set l^B received from P^B. This would have a communication complexity of $O(m'^B L \log(n^A))$. In GlobalOD, Step 1 has a communication of $O(m''^A d)$. In Step 11, P^A receives a count for each object in the set M''^A which P^A has sent. P^A has to send to P^B, a similar count for each object in the set M''^B received from P^B. This would have a communication complexity of $O(m''^B \log(n^A))$. For large datasets of high dimensionality, Considering only the dominating terms, the total communication complexity of player P^A would be $O(m'^A L + m''^A d)$. In other words, the major communication is sending the labels of local probable outliers and sending the entire global probable outliers set.

3.4 Private Outlier Detection

We now describe an algorithm for privately computing outlier detection over horizontally partitioned data. The algorithm for privacy preserving outlier detection over horizontally distributed data is executed in two phases. In the first phase, each player locally computes its set of local probable outliers by running centralized algorithm on its local dataset. These local outliers contain the global outliers as well as few non-outliers for which enough neighbours do not exist in the respective local datasets but exist in the entire dataset considering all players. To prune these non-outliers, all the players engage in

Algorithm 6 Pruning: Pruning Non-Outliers using LSH

Require: Dataset D, Hash Table T, Point Threshold p'_t, Bin Threshold b_t,
Ensure: M

```
M = {}
∀o ∈ D, pruned[o] = false
for each object o in D do
    if pruned[o] = false then
        Neighbors = ⋃_{i=1}^{L} T[g_i(o)] // g is the hash function
        RepeatedNeighbors = {o' | o' ∈ Neighbors and occurrence(o') ≥ b_t}
        if |RepeatedNeighbors| > p'_t then
            ∀o' ∈ RepeatedNeighbors, pruned[o'] = true
        else
            M = M ∪ {o}
        end if
    end if
end for
```

communication in the second phase in order to generate the global neighbour information and compute their subsets of the global probable outliers. The pruning technique used here is slightly different from the one mentioned previously. Hence, for the sake of clarity, we will incorporate the modified centralized algorithm in our privacy preserving algorithm.

3.4.1 Algorithm

We consider t players, each with dataset D_i for $i = 1$ to t and $n_i = |D_i|$ such that the size of the entire dataset D is $n = \sum_{i=1}^{t} n_i$. We assume that n is known beforehand to all the players, otherwise it could be computed using a *Secure Sum* protocol. Apart from this, the algorithm takes as input distance and point thresholds. At the end of the algorithm each player has its subset of the outliers in the dataset D. Our algorithm is based on LSH scheme using p-stable distributions [17], where each hash function for a d-dimensional object v is computed as:

$$h_{a,b}(v) = \lfloor \frac{a.v + b}{w} \rfloor \tag{3.1}$$

While performing LSH on the local datasets, in each iteration of LSH, all the players need to use the same randomness so that, in any one iteration, all the objects are hashed using same hashing function. Hence, any one player, let us say P_1 computes the LSH parameters k (width of a hash function) and L (number of hash tables) and generates the random matrices A and B. Here, each element a of the $k \times L$ matrix A is a d-dimensional vector whose each entry is independently drawn from a p-stable distribution and each element b of the $k \times L$ matrix B is a random number in $[0, w]$. P_1 then publishes these values to all the other players. Each player then computes the modified point threshold $p'_t = (1 - p_t) \times n$ and LSH distance parameter $R = d_t/(1 + \epsilon)$. Note that the parameter p'_t is computed based on the size of the entire dataset instead of the size of the local dataset. The LSH scheme is then run on the local dataset

Algorithm 7 PrivateOD: Privacy Preserving Outlier Detection in Horizontal Distribution

Require: t Players, Datasets D_i for $i = 1$ to t, Total Dataset size n, Distance Threshold d_t, Point Threshold p_t, Approximation Factor ϵ

Ensure: Outliers M_i; $i = 1$ to t

At P_1:
  Compute LSH parameters k, L and w
  Generate A and B // A and B are $k \times L$ matrices
  Publish k, L, w, A and B
for each player $i = 1$ to t **do**
  $p'_t = (1 - p_t) \times n$
  $R = d_t/(1 + \epsilon)$
  $T_i = LSH(D_i, R, A, B)$
  Compute b_t
  $M'_i = Pruning(D_i, T_i, p'_t, b_t)$
end for
for each player $i = 1$ to t **do**
  $BinLabels_i = \{$label of each bin in $T_i\}$
end for
$BinLabels = SecureUnion(BinLabels_i)$; i = 1 to t
for each player $i = 1$ to t **do**
  for $l = 1$ to $|BinLabels|$ **do**
    if $BinLabels(l) \in BinLabels_i$ **then**
      $C_i(l) = |T_i(l)|$
    else
      $C_i(l) = 0$
    end if
  end for
end for
$C = SecureSum(C_1, C_2, \ldots C_t)$
for each player $i = 1$ to t **do**
  $\widehat{C}_i = C - C_i$
  for each object o in M'_i **do**
    $Neighbors_i = \bigcup_{j=1}^{L} T_i[g_j(o)]$
    $RepeatedNeighbors_i = \{q \in Neighbors_i \mid occurrence(q) \geq b_t\}$
    $req\_nn_i = p'_t - |RepeatedNeighbors_i|$
    $ValidBins_i = \{b_j \mid b_j = g_j(o)\ and\ \widehat{C}_i[g_j(o)] > req\_nn_i, j = 1, 2, \ldots L\}$
    if $|ValidBins_i| < b_t$ **then**
      $M_i = M_i \cup \{o\}$
    end if
  end for
  Return M_i
end for

using the above computed parameters which outputs the binning structure T_i (Here, the LSH scheme is a bit different from that of the centralized setting in that the random vectors A and B are given as input to the LSH scheme whereas in the actual LSH scheme these values are generated with in the LSH protocol). The protocol *Pruning* is then invoked, which returns the set of local probable outliers M_i'.

In the second phase, to form the set of global outliers, each player requires the total number of objects with all the other players that are hashed to each bin. Since the binning at each player is performed with the same global LSH parameters, we can find the correspondence between LSH bin structure across the players. However, the bin labels with each player might be different (considering only the non-empty bins). Hence the players communicate to form the union of all the bin labels (*BinLabels*) using the *Secure Union* protocol [14] (steps $12 - 15$). While forming *BinLabels*, each bin label needs to be indexed with the iteration number during LSH (1 to L) in order to differentiate between bins of same labels created during different iterations of LSH. Each player then counts the number of objects it has in each of the bins in *BinLabels* to form the local bin count structure C_i (steps $17 - 23$). Next, the *Secure Sum* protocol [14] is used to compute the global bin count structure C from the corresponding local bin count structures C_i of all the players (step 25). Each player i then locally computes the sum of the other $t - 1$ player's counts for all the bins i.e., $\widehat{C}_i = C - C_i$ (step 27).

At every player, each object o in the set of local probable outliers M_i' is then considered to determine whether it is a global outlier. To get the number of neighbours of o in the local dataset, the cardinality of the set $RepeatedNeighbors_i$ for o is computed. This step is actually redundant if the value is stored in the first phase (during *Pruning*). This value would be less than p_t' since the object was considered a local probable outlier in the first phase. The number of neighbours required to make it a non-outlier is computed as: $req\_nn_i = p_t' - |RepeatedNeighbors_i|$. To get the count of neighbours of o which are available with other players, the corresponding bins in $\widehat{C}_i$ (i.e. those L entries in $\widehat{C}_i$ indicated by $g_j(o)$ where $j = 1$ to L) are considered. If any of these L bins have count greater than req_nn_i, we can consider the object o to be a non-outlier. However, as explained in the centralized scheme, to reduce the false negative probability, we require b_t such bins to make the object a non-outlier. To achieve this, only those bins which have object count greater than req_nn_i are said to be $ValidBins_i$. If the cardinality of $ValidBins_i$ is less than b_t, the object o is considered as a global outlier and added to the set of global probable outliers M_i.

As in the case of distributed setting, the whole algorithm is run over multiple iterations and the intersection of all the global probable outlier sets is returned as the final set of outliers at each player. Since the procedure to find global outliers is a bit different from that of finding outliers in the distributed scheme, the false positive rate increases slightly in comparison to that of the distributed algorithm.

3.4.2 Example

We continue with the example scenario described in Section 3.3.2 and show how to compute outliers in a privacy preserving manner using Algorithm 7. Initially, both the players will reveal the size of their dataset to each other. Any one player will compute the LSH parameters and publish them to the

other. Using these parameters, both the players will run LSH on their own dataset and compute local outliers as described in Section 3.3.2. Thus, player P^A computes the local outlier set $M'^A = \{a_5, a_6\}$ while player P^B does not have any local outliers. In case of non-private algorithm, players can directly share their bin structure to calculate global outliers. However, in case of private algorithm, players use *SecureUnion* and *SecureSum* protocols to share binning information. Any *SecureUnion* and *SecureSum* protocol can be used as the proposed algorithm is independent of the underlying protocols. One simple method for both the protocols is given in Section 2.4. Here, we use them as a black box.

Both the players construct a set of bin labels of their bin structure and then execute secure union protocol to compute global bin label set. In our example player P^A has bin labels $\{[1,1],[1,2],[1,3],[2,1],[2,2],[2,3],[3,1],[3,2],[3,3]\}$ and player P^B has bin labels $\{[1,1],[1,3],[2,2],[2,3],[3,1],[3,2]\}$. Thus global bin label set is union of these two sets which is $\{[1,1],[1,2],[1,3],[2,1],[2,2],[2,3],[3,1],[3,2],[3,3]\}$. Each player then constructs a counter corresponding to each bin label in the global bin label set. The counter is set to 0 if the corresponding bin doesn't exist in the local bin structure, otherwise its the number of objects hashed in to that bin. For player P^A the counter array corresponding to global bin structure is $C^A = \{4,1,1,1,4,1,3,2,1\}$ and for player P^B its $C^B = \{2,0,2,0,1,3,2,2,0\}$. Both the players use secure sum protocol to compute the global counter array which gives the total number of objects hashed into each bin of a global bin structure. In our example, the global counter $C = \{6,1,3,1,5,4,5,4,1\}$.

As explained earlier, few of the local outliers are actually non-outliers because they have enough neighbours but not all of them are available locally. Thus, for every local outlier, a player counts the number of neighbours required (req_{nn}) by subtracting number of local neighbours from point threshold p'_t. Then, a player checks if this local outlier has req_{nn} neighbours available in other player's database by checking corresponding bin counts in global counter C. Every local outlier who has enough neighbours is pruned and marked as non-outliers. Consider a local outlier a_5. The number of local neighbours of a_5 is 1, hence 2 more neighbours are required in D^B (database with P^B) to mark it as non-outlier. The bin labels corresponding to a_5 is $\{[1,3],[2,3],[3,2]\}$ and the counts at P^B corresponding to these labels are $\{2,3,2\}$. To enforce bin threshold criteria, we consider only those bins which has at least req_{nn} objects. If total number of such bins is less than bin threshold b_t then the object is marked as an outlier otherwise it is marked as non-outlier. In case of a_5, the number of bins satisfy such criteria is 3 which is greater than bin threshold. Hence, a_5 is marked as non-outlier. Next, consider another local outlier a_6. The bin labels corresponding to a_6 is $\{[1,2],[2,1],[3,3]\}$ and the counts at P^B corresponding to these labels are $\{0,0,0\}$. Since, a_6 doesn't have any more neighbours available in D^B it is marked as an outlier.

3.4.3 Analysis

Computational Complexity: We give the computational complexity from the perspective of player P_1. The computational complexity of step 3 is $O(kL)$. Steps 8 and 10 have complexity of $O(n_1dkL)$ and $O(n_1N_{pr_1}L)$ respectively, where N_{pr_1} is the number of processed objects of player P_1. The complexity of step 13 and steps $17-23$ depends on the average number of non-empty bins N_b created

Table 3.3: Dataset Description

Dataset	Objects	Attributes	Source
Corel	68040	32	kdd repository
Landsat	275465	60	vision lab, ucsb
Darpa	458301	23	Lincoln Laboratory, MIT
Household	1000000	3	US Census Bureau

during each iteration of LSH. In the worst case where each object in the dataset is hashed to a different bin, the number of bins would be equal to the dataset size. However, the number of bins would be much less than the total data size in the average case. Thus the average case complexity for step 13 and steps $17-23$ is $O(N_bL)$; where $N_b << n$. Similarly, the complexity of step 27 is $O(N_bL)$. The complexity for steps $28-36$ is $O(m'_1L)$; where $m' = |M'_1|$. Considering the dominating terms, the computational complexity for player P_1 is $O(n_1dL)$. The overall computational complexity of Algorithm 7 is $O(ndL)$.

Communication Complexity: In Algorithm 7, communication among the players happens in steps $4, 15$ and 25. Step 4 has a communication complexity of $O(kL)$. The average case communication complexity for step 15 is $O(N_bL)$. The corresponding average case communication complexity for step 25 is $O(N_bL \log n)$. Thus the overall communication complexity for Algorithm 7 is $O(N_bL \log n)$.

Security Analysis: In Algorithm 7, the values communicated in step 4 are public values and do not reveal any private information. After executing steps 15 and 25, each player has the count of objects in the entire database that are hashed to each bin. From this, each player can infer about the distribution of the objects of all the other players but cannot infer about the distribution of any single player when more than two players are involved (since *Secure Union* and *Secure Sum* are used). However, in most cases where the objects with each player come from a same distribution, this information is usually known before hand and thus there is no extra information revealed.

3.5 Experiments

We perform experiments on datasets listed in Table 3.3. For implementation of LSH, we have used E2LSH package[1], which is based on the p-stable distribution [17]. For all our experiments the approximation factor ϵ is set to 2 (empirically determined). All experiments are executed on Intel(R) Core $i7$ CPU $3.33GHz$ machine. To simulate horizontal partitioning, we randomly divide the dataset among all the players such that each player will have n/t distinct objects with all the attributes, where n is the size of the dataset and t is the number of players. For all the experiments, we consider $t = 2$, unless specified otherwise.

[1] http://www.mit.edu/∼andoni/LSH/

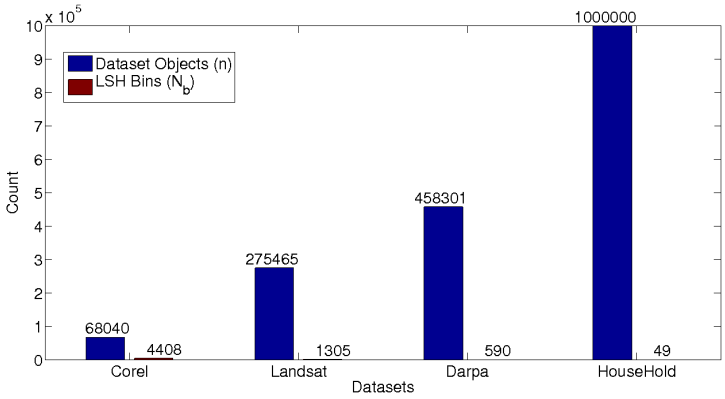


Figure 3.3: Number of LSH bins created for various datasets. Number of LSH bins are much less compared to the size of the database.

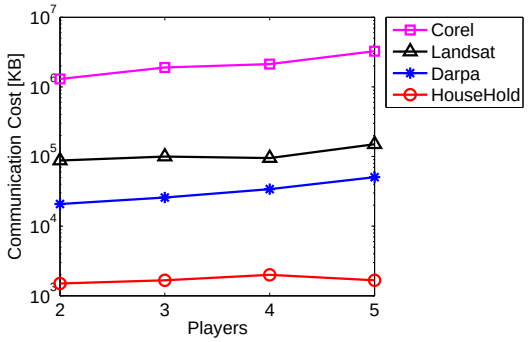


Figure 3.4: Communication Cost of PrivateOD over multiple players.

LSH Bin Structure: The communication cost of *PrivateOD* algorithm is $O(N_b L \log n)$, where N_b is the average number of nonempty bins created during each iteration of LSH, L is the number of hash tables and n is the number of database objects. We compare N_b and n by running LSH on entire dataset without partitioning. The results are shown in Figure 3.3. The numbers on the bar shows the respective counts. It is evident from the figure that indeed N_b is extremely less than n. Further, this is also true for large datasets. Due to the locality sensitive property, many objects which are near by fall into a single bin. Hence, average number of bins created will be much less than the total number of objects in a dataset ($N_b << n$). Even in case of sparse data this holds true as the LSH radius R will be large because of high distance threshold d_t.

3.5.1 Accuracy

We compute outliers by running both the algorithms – *DistributedOD* (Algorithm 1) and *PrivateOD* (Algorithm 7), in horizontally distributed setting with two players. We computed the optimal bin threshold b_t as described in [63] and run both the algorithms using the same bin threshold. The accuracy of these algorithms is measured by comparing the computed outliers with the ground truth. The ground truth is a set of actual outliers computed using exhaustive distance computation [37]. The results are summarized in Table 3.4. The detection rate for each dataset at the optimal b_t is 100% (i.e., no false negatives). The false positive rate is shown as a percentage of the total dataset. In both distributed and private setting, the false positive rate is quite less, which shows that the proposed algorithms have high accuracy. The false positives of *PrivateOD* is on an average 0.02% higher than that of *DistributedOD*. The small increase in false positives is due to lack of *GlobalOD* step, where all the players exchange their global probable outliers for further pruning. In case of *PrivateOD*, the players can not exchange their global probable outliers as that will reveal those objects to the other players. One solution is to use

Table 3.4: Accuracy of DistributedOD and PrivateOD

Dataset	BinThreshold (b_t)	False Positives [%]	
		DistributedOD	PrivateOD
Corel	45	0.011	0.041
Landsat	30	0.014	0.025
Darpa	30	0.031	0.059
Household	25	0.009	0.02

secure multiparty computation (SMC) [75] to privately execute *GlobalOD*. However, SMC protocols are very expensive in terms of computational as well as communication cost.

3.5.2 Comparison

We compare *PrivateOD* with the algorithm given in [75], which has a cost of $O(n^2d)$. The comparison in terms of communication cost for different datasets by varying the dataset size is shown in Figure 3.5. We calculate the communication cost in terms of amount of data (in KB) transfered between players during the entire protocol execution. Not only our method has less communication cost in comparison, but also the rate of increase in the cost with respect to dataset size is very less compared to the $O(n^2d)$ method. Also, for large datasets, this increase in communication cost is very less compared to smaller datasets.

We repeat the experiment by varying the number of players from 2 to 5. The accuracy of the algorithm remains the same while communication cost increases with the number of players. The results are given in Figure 3.4. For large datasets, the increase in communication cost is very less compared to smaller datasets. Because in case of large datasets, most of the non-outliers are pruned locally as enough neighbours are available locally. The communication cost will be even less in many practical scenarios, where the dataset of each player is generated locally instead of a random distribution of a single dataset (as in the case of our simulated horizontal partitioning). This is because by randomly distributing a single dataset there might be a case where many neighbours of an object of one player, will be with other players. This will increase the number of local outliers which in turn will increase the communication cost.

3.6 Summary

We proposed an approximate algorithm for distance based outlier detection. Our approach uses the Locality Sensitive Hashing (LSH) technique to achieve high level of efficiency and low communication as well as computational cost. First, we gave algorithm for distributed outlier detection among horizontally partitioned data. Later, we extended our method to incorporate privacy preserving outlier detection. We have given a thorough complexity analysis for both algorithms and have provided the empirical results on various datasets to support our claims. We also compared our algorithm with the previous

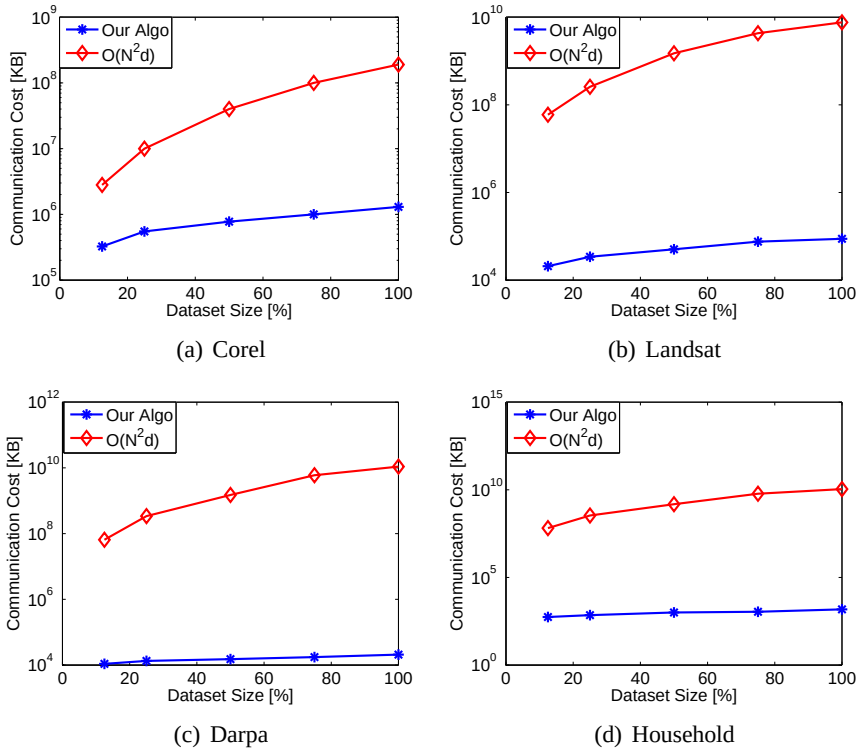


Figure 3.5: Comparison of the proposed PrivateOD algorithm. Our algorithm performs far better than previous known result of quadratic cost.

known results and showed that ours perform better in terms of communication as well as computational cost.

Chapter 4

Private Content Based Search on Encrypted Data using Hierarchical Index Structures

4.1 Introduction

With the advent of third party data storage services like *DropBox* and *SkyDrive*, it has become increasingly important to have an efficient search mechanism on the encrypted data. In this chapter, we formally define Content Based Similarity Search on encrypted data and propose an efficient protocol for the same. We further show that our method outperforms present state of the art techniques for similarity search on encrypted data, not only in accuracy but also in retrieval time and communication. To achieve this we propose an algorithm which make use of hierarchical index structures. The computation as well as communication cost of the protocol is $\mathcal{O}(m \log_k n)$, where m is the size of each tree node, k is the branching factor and n is the number of leaf nodes. Empirically we show that on image datasets of the order $5K$ size, we achieve 25 times lower communication cost. Also, we achieve an average improvement of 30% in terms of retrieval quality. We also extend our algorithm to solve the problem of private content based image retrieval in public database. In this setting, when compared with the existing techniques, our technique is 10^5 times faster on an index tree with 1 Million leaf nodes. We begin by formalizing the problem of Content Similarity Searchable Symmetric Encryption (CS-SSE).

Definition 7 *(Content Similarity SSE): A content owner with database D should be able to store D on an outsourced server S, so that users can privately query for similar data from S. The database is indexed using any hierarchical index structure and stored on S in encrypted form so that the server cannot know anything about the data. For a query item Q, the client should be able to retrieve the data items which have maximum similarity in terms of content with respect to Q.*

4.2 Prior Work

The problem of private exact keyword search is to test for the existence of a particular keyword on a database in a privacy preserving fashion. Oblivious RAMs [26, 55] were proposed to solve this problem which offer optimal security but are computationally expensive and use poly-logarithmic rounds of communication. To overcome the problems of the Oblivious RAMs, Song *et al.* [71] proposed a model considering a weaker notion of privacy. While their scheme is proven to satisfy the requirements of a Secure Encryption scheme, it fails to satisfy those of a Secure Searchable Encryption scheme. Later, Goh *et al.* [20] gave a formal definition for a Secure Searchable Encryption scheme and proposed a Secure Index based scheme using Bloom filters [6] and pseudo random functions. In [15], Curtmola *et al.* proposed an adaptive semantic security definition for Secure Searchable Encryption scheme and provided a scheme which satisfies the definition. In [58], the authors address the practicability of the Searchable encryptions schemes by relaxing the adversary model. In the case of Similarity Searchable Symmetric Encryption, the authors of [59] proposed a scheme for similarity matching on encrypted data for the case of approximate string matching under hamming distance. Another scheme for fuzzy keyword search over encrypted data was proposed in [43]. Recently, Kuzu *et al.* [51] proposed an efficient protocol for Similarity Search on encrypted data, which is based on Hashing based index structures.

CS-SSE was addressed in [48] for the specific case of multimedia data. In [48], the authors gave a system to privately search for similar images on an encrypted image database, using Order Preserving Encryptions (OPE) which are not secure against Chosen Plaintext Attack (CPA). For the case of searching on a public database, a *Private Content Based Image Retrieval* protocol has been presented in [69]. A slightly different setting, in which the server's privacy is also considered was studied in [66], where the authors proposed an *Oblivious Image Retrieval* protocol.

4.3 Content Similarity SSE

The problem of CS-SSE can be solved using any *Similarity SSE* protocol as a subroutine. Given a query item Q with a set of features, a *Similarity SSE* protocol is executed for each feature in parallel and the results are merged to obtain data items having maximum content similarity with respect to Q. More specifically, we show how to use the recently proposed *Similarity SSE* [51] protocol to construct a protocol for CS-SSE. Next we show the limitations of this approach and then propose an efficient algorithm for CS-SSE which overcomes these limitations.

4.3.1 Content Similarity SSE using LSH

Similarity SSE scheme proposed in [51] uses Locality Sensitive Hashing (LSH) for secure indexing. The content owner extracts the features of all the database elements and hashes them using LSH, which generates L hash tables. Due to the locality sensitive property of LSH, similar features are hashed to the same bucket with high probability. To obtain the index, a bucket vector of length N bits is created

for each bucket, where i^{th} bit corresponds to i^{th} database element. The i^{th} bit of a bucket vector is set to 1 if at least one of the features of the corresponding database element is hashed to that bucket, else it is set to 0. Intuitively, a bucket vector represents the set of data items whose features are hashed to that bucket. In order to secure the index, all the bucket vectors are encrypted and all the bucket labels are randomly permuted. At the time of query, user similarly applies LSH on the query feature f to generate L bucket labels and permutes them. After that user obtains the corresponding L bucket vectors from S and scores the database elements using the cumulative sum of all the retrieved bucket vectors. Based on theses scores user then retrieves the relevant database elements from server S. We now show how to construct a protocol for CS-SSE by directly using the *Similarity SSE* scheme as a subroutine. We name the resulting protocol as *Content Similarity via LSH (CSL)* for further reference.

Since the difference is only in terms of querying mechanism, index construction is same as explained above. At the time of query, user first extracts all the features of a query data item. After that, all the features are queried in parallel to retrieve the corresponding L bucket vectors as explained above. User then scores database elements using all L bucket vectors of all the features of a query. Based on the score, the user retrieves similar database elements from S. Since there are L bucket vectors, each of length N, for each feature, $N \times L$ bits needs to be transfered. The communication of this protocol is $O(NL)$ bits per query feature, which grows linearly with the size of the database. Hence, for large databases this approach is not practical as it is as good as transferring the entire database. Moreover, the limitation of this approach is in terms of the used scoring mechanism. Since the bucket content is only one bit per database element, it only captures the information of whether any of the features of a data item are hashed or not. It does not capture the frequency of a particular feature which is a very important attribute in many efficient scoring techniques like Term Frequency Inverse Document Frequency (TF-IDF).

4.3.2 Content Similarity SSE using Hierarchical Structure

In order to overcome limitations of LSH based approach, we suitably modify the scheme of [51] for the case of hierarchical tree based indexes and provide a protocol for CS-SSE. This also provides the additional benefit of tree indexes being exact as opposed to the probabilistic nature of LSH which, together with the efficient scoring mechanisms improves the accuracy of our system. Furthermore, we reduce the required communication from $O(NL)$ to $O(m \log_k n)$, where m is the size of a node, k is the branching factor and n is the number of leaf nodes in the tree. We also give experimental evidence to show these improvements of our algorithm over existing techniques. However, the number of rounds required for our protocol is $O(\log_k n)$, where CSL is a constant round protocol. Next we give a brief overview of our algorithm which has two phases - Offline Processing by content owner and Online Querying by user.

In offline processing, the content owner with dataset D, extracts the features of database elements and builds an index structure using any hierarchical indexing scheme like Hierarchical k-means, kd-Tree etc. Once the indexing is done, all the nodes of an index tree are encrypted using any symmetric

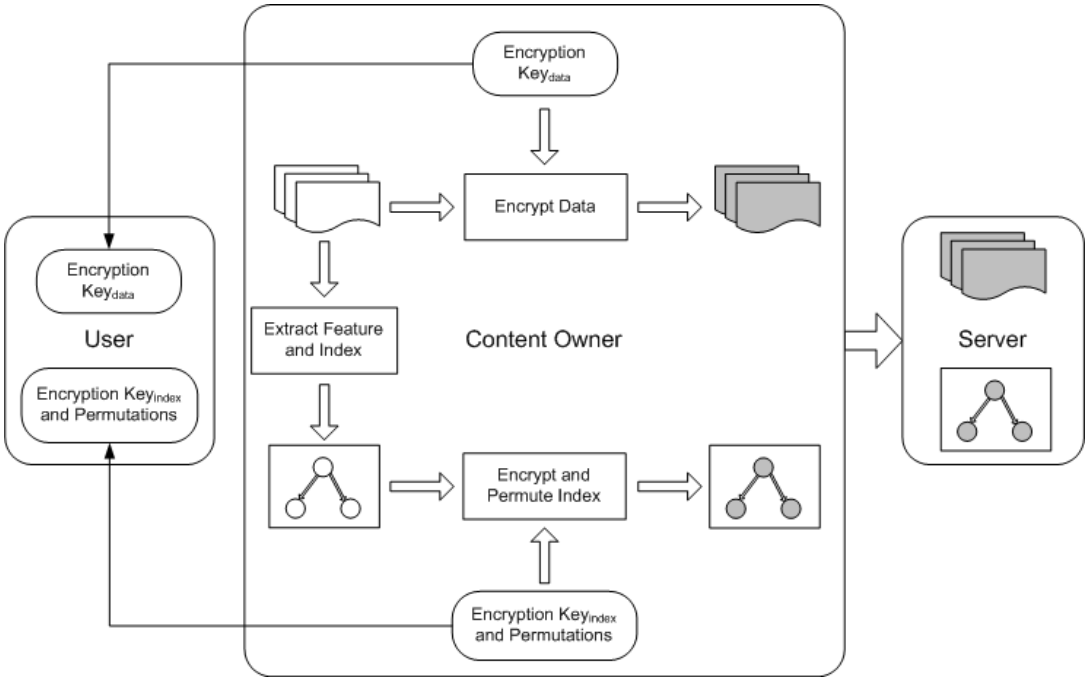


Figure 4.1: Offline Processing: Content Owner generates secure index structure and stores it on a database server as a preprocessing step.

encryption scheme and the labels of their child nodes are randomly permuted using any pseudo random permutations. The encryption is used to secure the content of the index and permutation is used to secure the traversal path of a query. The content owner then generates unique ids for all the data elements and encrypts entire database. At the end, content owner sends the encrypted database with corresponding ids and encrypted index structure to S. To enable the users to privately search on S, content owner shares the encryption keys and random permutations with the legitimate users. The entire process is illustrated in Figure 4.1.

In online querying, user first extracts the set of features of a given query data item Q. User then obtains the root node information from the server and decrypts it using the secret key shared by content owner. For each feature, user takes a decision as to which node needs to be accessed on the next level by comparing it with retrieved node information and applies the random permutation on the label of the node obtained. The user then obtains the information of the node with this permuted label on the next level and this process is repeated till a leaf node is retrieved. Using the leaf node information obtained for each feature, the user scores each element in the database. Based on the computed scores, relevant data items are retrieved from S. Note that all query features can be executed in parallel and then scores can be computed to achieve speedup. The query mechanism is illustrated in Figure 4.2.

Although, the proposed algorithm can be applied in general for any type of data, in this thesis we give a specific example of a Content Based Image Retrieval (CBIR) using the popular index structure Vocabulary Tree [53].

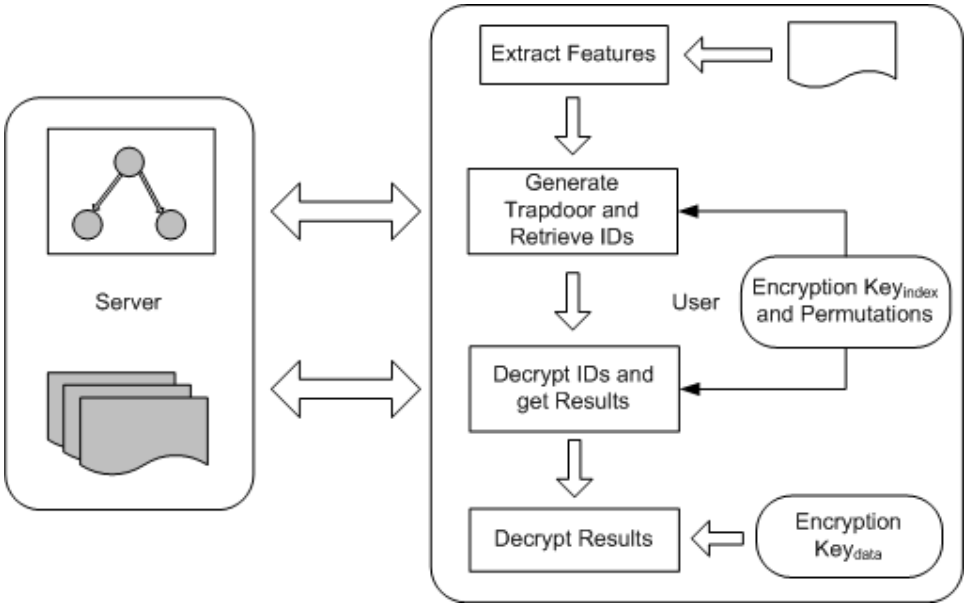


Figure 4.2: Online Querying: User runs CS-SSE protocol to privately query over encrypted data.

4.3.3 Algorithm

We now give the details of our algorithm for CS-SSE. The definition of a Searchable Symmetric Encryption scheme is given as below [15]:

Definition 8 *(Searchable Symmetric Encryption): An Index based SSE scheme over a Database collection D is a collection of five polynomial time algorithms $SSE = (Gen,\ Enc, Trpdr, Search, Dec)$, where:*

- *$Gen(1^{\lambda})$ is a probabilistic key generation algorithm run by the content owner. It takes as input a security parameter λ and returns key K.*
- *$Enc(K, D)$ is a probabilistic algorithm run by the content owner to encrypt the database. It takes as input the Key K and the database collection $D = (D_1, D_2, \ldots D_N)$ and returns a secure index I and the encrypted database $C = (C_1, C_2, \ldots C_N)$.*
- *$Trpdr(K, Q)$ is a deterministic algorithm run by the user to generate a trapdoor for a given query Q. It takes as input the Key K and query Q and returns a trapdoor T.*
- *$Search(I, T)$ is a deterministic algorithm run by the server to search for similar items in database D for the query Q. It takes as input the encrypted index I built on the database D and the trapdoor T and outputs a set of identifiers.*

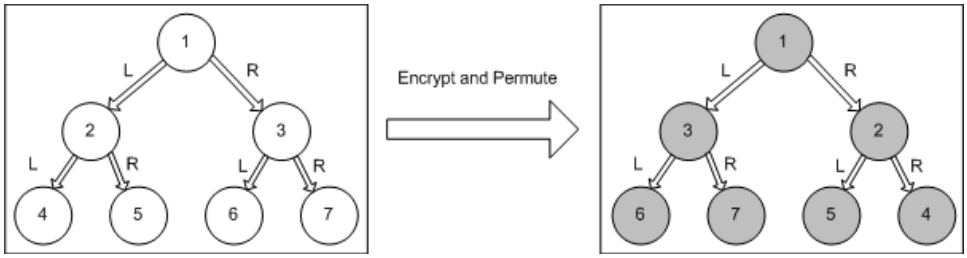


Figure 4.3: Conceptual View of Secure Index representing encryption and permutation operations on tree-like data structures.

- $Dec(K, id(C_j))$ *is a deterministic algorithm run by the user to recover similar items in* D*. It takes as input the key* K *and an identifier* $id(C_j)$ *and returns the corresponding database item* D_j*.*

The definition of SSE scheme will hold even in the case of CS-SSE, with the query Q now being a data item. We now explain each of these algorithms for our scheme. Throughout this chapter, by $R \xleftarrow{U} \{0,1\}^\lambda$ we mean assigning a uniformly sampled value of length λ to R.

1. Key Generation $Gen(1^\lambda)$: Sample Keys $K_1, K_2 \xleftarrow{U} \{0,1\}^\lambda$ where λ is the security parameter.

2. Index Construction $Enc(K_1, K_2, D)$: The content owner with the database D first generates the features for each element in D and builds the index structure on the feature set using any Hierarchical index scheme. We represent by n and k the number of leaf nodes formed in the hierarchical tree and the branching factor at each internal node of the tree respectively. Consequently, the height of the tree will be $h = \log_k n$. The content owner then encrypts the indexed data structure using key K_1. Note that during encryption, each node of the tree is encrypted independently. A pseudo random permutation π is then applied on the encrypted index to obtain the secure index I. During pseudo random permutation, at every internal node in the data structure, the content owner randomly permutes the labels of its k child nodes. For a binary data structure, this means randomly rotating the child nodes as illustrated in Figure 4.3. Also, each item in the database D is encrypted using key K_2 to get the encrypted database C. Each item in C is then indexed with an identifier id. The content owner then stores this encrypted database along with the secure index I on an outsourced server S. For the symmetric encryptions schemes E_{K_1} and E_{K_2}, we use a PCPA (Pseudorandomness against chosen plaintext attacks) secure scheme. To enable the users to securely search on the encrypted database, the content owner shares (K_1, K_2, π) with the users.

3. Trapdoor Generation $Trpdr(K_1, \pi, Q)$: The user with a query item Q generates the feature set F for Q where $F = \bigcup_{i=1}^{r} F_i$ for $r \geq 1$. The user then generates the trapdoor information $T = (T_1, T_2, \ldots T_r)$ using the key K_1 and permutation π which have been shared by the content owner. The trapdoor T_i for each feature F_i is again a vector $T_i = (T_{i_0}, T_{i_1}, \ldots T_{i_h})$ where h is the height of the tree. Each element T_{i_l} is the permuted label of the node to be accessed at level l of the index tree I stored on server S. However, the label of the node to be accessed on a particular level is dependent

Algorithm 8 Offline Processing: Content Owner Constructs Secure Index using CS-SSE

1: $K_1, K_2 \leftarrow \{0,1\}^\lambda$
2: Generate Feature set F for dataset D
3: Build the index structure I' on F
4: Encrypt indexed database to obtain $I^e = E_{K_1}(I')$
5: Apply random permutations on the encrypted index structure to obtain $I = \pi(I^e)$
6: Encrypt Database elements to obtain $C = E_{K_2}(D)$
7: Store I and C on outsourced server S
8: Share (K_1, K_2, π) with users

on the information obtained at the previous level. Hence in our scheme, the trapdoor information for a particular feature is generated by user in communication with the server S.

4. Search $Search(I, T)$: Search is done at the server in communication with the user at each level of the tree. For each feature F_i of a query Q, the user first requests S for the encrypted root node information and decrypts it using K_1 to obtain the actual root information. Based on the comparison of feature F_i with the root node information, the label δ of the child node which is to be accessed is decided. The type of information stored on the root node and all subsequent internal nodes and the metric used for comparison depend on the actual indexing scheme used. For example, if hierarchical k-means is used, the information in each internal node will be the cluster centers of the child nodes and the child node having the least distance from the query needs to be accessed. Once the label δ of the child node is determined, its permuted label δ^π is then computed by applying the permutation π on δ. The user then goes on to request the information I_{l,δ^π} of the child node with label δ^π on level l. This procedure is continued till a leaf node is reached and the data on the corresponding leaf node (in encrypted form) is retrieved and decrypted using K_1. The label δ at each level form the tree traversal path for the feature F_i and the permuted labels δ^π at each level form the trapdoor information T_i for F_i i.e., $T_{i_l} = \delta_l^\pi$ for $l = 0$ to h. At S, traversal corresponding to F_i is done using T_i and hence the actual traversal path is hidden from S. We note that for the first level ($l = 0$), $\delta_0^\pi = \delta_0$ since there is only one node (root) on this level. Once the leaf node information R_i for each feature is obtained, scoring is performed using this information to generate the ids of the similar data in D. The scoring procedure $Score$ is dependent on the type of information stored in the leaf nodes which in turn depends on the kind of indexing scheme used.

5. Decryption $Dec(K_2, id(C_j))$: The user obtains the data items C_j corresponding to the ids obtained in the Search phase and decrypts using key K_2 to recover the similar data items D_j.

We categorize the above described algorithms into two phases:

- Offline processing, done by the Content Owner, comprising the Key Generation and Index Construction algorithms.
- Online Querying, done by the users in communication with the server S, comprising the Trapdoor Generation, Search and Decryption algorithms.

The two phases are summarized in Algorithms 8 and 9. We now give a simple example to explain the working of these algorithms followed by detailed analysis of CS-SSE.

Algorithm 9 Online Querying: Users Privately Query using CS-SSE

Generate feature set for query Q, $F = (F_1, F_2, \ldots F_r)$
for each feature F_i in F **do**
  $\delta^\pi = 1$ // pointing the root node
  for each level l in the tree **do**
    From S obtain I_{l,δ^π} the information of node δ^π on level l
    Decrypt to obtain the actual information $i_{l,\delta^\pi} = D_k(I_{l,\delta^\pi})$
    if leaf node is reached **then**
      $R_i \leftarrow i_{l,\delta^\pi}$ //store the leaf node information
    else
      Compare i_{l,δ^π} with F_i and decide label δ of the child node to be accessed
      Apply random permutation π on δ to obtain the permuted label $\delta^\pi = \pi(\delta)$
    end if
  end for
end for
Compute $id(C_j)$ using $Score(R_1, R_2, \ldots R_r)$
Obtain C_j from S
Decrypt C_j using K_2 to obtain D_j

4.3.4 Example

The CS-SSE protocol has two phases - Offline Processing and Online Querying. During offline processing, the content owner encrypts database, constructs secure index structure using secret keys and random permutations and stores them on the server. In the second phase, users privately query the secure index structure to search and retrieve similar items from the encrypted database using the secret keys and random permutations provided by the content owner. Through a simple example, we explain the core steps of the algorithms – how to secure the index and how to privately retrieve similar items using secure index. Given a query, all the features of the query is extracted and queried to retrieve information from the corresponding leaf nodes of the secure index structure. The scoring function is applied on the aggregated information to rank the database items. Finally, top ranked items are retrieved from the encrypted database. Once all the leaf node information is available to the client, it can be easily decrypted using the secret key given by the content owner and any scoring function can be applied to rank the items. The server database is a pair of item id and corresponding encypted item. Based on the item id of the top ranked items, client retrieves similar items from the server in an encrypted form and decrypts it. Thus, given the information of the leaf nodes corresponding to the query features, retrieving similar items is a straight forward task. Hence, we focus on the crucial task of retrieving leaf node information corresponding to the query features.

Secure Index Construction: We assume the features are available and index structure is constructed using those features. Although, the method is applicable to any hierarchical index structures, we consider a specific case of vocabulary tree. The details of building vocabulary tree from the database and searching vocabulary tree for image retrieval is given in Section 2.3. Vocabulary tree is constructed using hierarchical k-means, where each non-leaf node contains k cluster centers (one for every child node) and each leaf node contains the inverted list of images whose features belong to the visual word (cluster) represented by that leaf node. Let us consider the vocabulary tree shown in Figure 4.4(a), where $k = 2$ and the height of the tree is 2. The nodes of the tree are labeled as $l_1, l_2, \ldots l_7$, starting from the root node and scanning left to right at each level. We consider 2 dimensional data (features). The cluster centers are shown as a set of k two dimensional points inside every node except leaf nodes. For example, root node contains 2 cluster centers $(2, 2)$ and $(5, 4)$ for left and right child respectively. First, we encrypt every node of the tree using any symmetric encryption scheme. Then, we randomly permute the child node labels for every node. We start with the root node which has two child nodes left (l_2) and right (l_3). After applying permutation $\pi_1 = [3, 2]$, l_2 points to right child and l_3 points to left child. That is left and right nodes are swapped together with their subtree. Similarly we apply random permutation $\pi_2 = [7, 6, 4, 5]$ to the nodes at next level. The resulting encrypted and permuted tree is shown in Figure 4.4(b) (The content of the nodes are not encrypted in the figure for the ease of explanation). This secure index structure together with encrypted database is sent to the server. Note, that server does not know the permutations or the secret keys used to generate secure index and encrypted database. Hence, the server does not have any information about the index structure as well as data. The entire permutation set $\pi = \{\pi_1, \pi_2\}$ and secret keys are given to the legitimate clients to enable private searching over this secure index.

Searching: For retrieval, query is compared with k cluster centers of a root node and the child node corresponding to the closest cluster center is visited. This process is repeated until a leaf node is reached. Upon reaching the leaf node, the information on the leaf node is transferred to the client as a result of the query. In order to keep query private, client obliviously traverse the tree. Initially, client retrieves root node information, decrypts it using the secret key provided by the content owner and compare it with the query to decide which child node should be accessed next. The resulting child node label is permuted using the respective permutation π and the node with permuted label is retrieved from the server. Since, the server doesn't know the permutations and all the nodes are encrypted, it can not know which nodes are being accessed and what data are being retrieved. Let's consider the query feature $(1, 3)$. Client retrieves the root node (l_1) information and decrypts it to get two cluster centers $[(2, 2), (5, 4)]$. By finding distances between query feature $(1, 3)$ and both the cluster centers, client knows that the query lies in the first cluster $(2, 2)$. Hence, the left child (l_2) of the root node needs to be accessed next. Using the permutation π_1 client finds the permuted label of l_2 which is l_3. Now, client retrieves information from node (l_3) and decrypts it to get the cluster centers $[(2, 1), (2, 3)]$. By repeating the same process, client finds the next query cluster as $(2, 3)$ and next child as (l_5). Client finds its permuted label l_7 after applying permutation π_2 and retrieves information from that node. Since, l_7 is the leaf node, it contains

(a) Vocabulary Tree (b) Secure Index

Figure 4.4: Example Secure Index Construction

list of images containing features similar to the query feature. Similarly, all the features of a query is processed and the retrieved information is used to score the database images as explained earlier.

4.3.5 Analysis

Complexity Analysis: Since the information of one node per level is obtained from server S (step 5 in Algorithm 9) and then processed at the client, both communication and computational cost of querying mechanism is $O(m \log_k n)$; where m is the size of each node and $(\log_k n)$ represents the depth of a tree with a branching factor k and n leaf nodes. The number of rounds for the protocol is $\log_k n$ since the operations for each level of the tree cannot be carried in parallel. However, even for a tree with 1 Million leaf nodes and a branching factor of 10, the depth and correspondingly the number of rounds required will be 6 which is practical.

During Index construction, even though encrypted data is stored on the server S, applying pseudo random permutations is still necessary since otherwise the server can learn information about a query just by looking at the traversal path of a query. Also, the size of all leaf nodes should be made same, which will be the maximum leaf size M, by suitably using padding for some leaf nodes so that the server cannot learn anything by looking at the size of the leaf node accessed for a particular query. Note that the index scheme should generate a perfect k-ary tree, otherwise it can be made so by adding dummy nodes at the time of index construction. Next, we give a formal proof of security for the proposed scheme considering an honest-but-curious adversary model.

Security Analysis: We use the adaptive semantic security definition for *SSE* scheme given in [15]. First we give the necessary auxiliary definitions before proceeding to the security definition:

Definition 9 *(History): A q-query history over a Database collection D is a tuple $H = (D, Q')$ that includes the database D and a vector of q query items $Q' = (Q_1, Q_2, \ldots Q_q)$.*

Definition 10 *(Access Pattern): The Access Pattern induced by a q-query history $H = (D, Q')$ is a tuple $\alpha(H) = (D(Q_1), D(Q_2), \ldots D(Q_q))$, where $D(Q_i)$ is the collection of identifiers of those database elements in D which are most similar in terms of content to query Q_i.*

Definition 11 *(Search Pattern): The Search Pattern induced by a q-query history $H = (D, Q')$ is a symmetric binary matrix $\sigma(H)$ s.t for $1 \leq i, j \leq q$ the element in the i^{th} row and j^{th} column is 1 if $Q_i = Q_j$ and 0 otherwise.*

Thus, *Search Pattern* is the information whether two searches were for the same query or not. In our scheme, each query Q_i is a vector of r features i.e., $Q_i = (F_1^i, F_2^i, \ldots F_r^i)$. Hence even if two queries are different, they can have some features in common. Moreover, each feature F_j^i of a particular query Q_i generates a traversal path $F_j^i = (F_{j_0}^i, F_{j_1}^i, \ldots F_{j_h}^i)$, where $F_{j_l}^i = \delta_l$ is a label of the node to be accessed at level l. So, even if two features are different, there might be some overlap in their traversal path. For example, in case of images, two different images may have some features in common and based on the number of common features, an adversary might infer some information about the similarity of the images. Also, for two different features, the adversary might gain some information about their similarity based on the overlap in their traversal paths on the hierarchical tree since the trapdoor generation is deterministic. We capture this information in the definition *Content Similarity Pattern* given below and include this in the *Trace*, which is the maximum amount of information the content owner is willing to leak. The notion of *Content Similarity Pattern* is an extension to the *Similarity Pattern* definition given in [51].

Definition 12 *(Content Similarity Pattern): The Content Similarity Pattern induced by a q-query history $H = (D, Q)$ is defined by $\zeta(H)$ s.t $\zeta([i, j, m], [u, v, w]) = 1$ if $F_{j_m}^i = F_{v_w}^u$ and 0 otherwise, where $1 \leq i, u \leq q$ and $1 \leq j, v \leq r$ and $0 \leq m, w \leq h$.*

Definition 13 *(Trace): The Trace induced by a q-query history $H = (D, Q)$ is a sequence $\tau(H) = (|C_1|, |C_1|, \ldots |C_N|, \alpha(H), \zeta(H))$ consisting of the lengths of documents in C and the Access and Content Similarity Patterns.*

Definition 14 *(Adaptive semantic Security): A Content Similarity SSE is said to be adaptively semantic secure if for all polynomial-size adversaries, there exists a simulator Sim such that for all polynomial-size distinguisher $Dist$:*

$$Pr[Dist(v(H)) = 1] - Pr[Dist(Sim(\tau(H))) = 1] < neg(\lambda)$$

where, $neg(\lambda)$ represents a negligible function in λ. Here, $v(H)$ is the view of the adversary, generated from an adaptively chosen history H, which is all the information available to the adversary and $sim(\tau(H))$ represents the view generated by the simulator given the trace $\tau(H)$. Intuitively, the definition says that a Content Similarity SSE scheme is secure if all the information available to the adversary can be simulated by the trace $\tau(H)$ which is exactly the maximum information we are leaking, with a probability negligibly close to 1. In other words, any polynomial-size distinguisher can not distinguish the view generated by the simulator from that of the actual view of the adversary with a greater than negligible probability.

Since the adversary has the knowledge of the secure index I, the encrypted database $C = (C_1, C_2, \ldots C_N)$ and the trapdoor information T, the view of the adversary over any adaptively chosen history is $v(H) = (I, (C_1, C_2, \ldots C_N), T)$. To prove that our scheme is adaptively semantic secure, we show how a simulator Sim can generate a view $v^*(\tau(H)) = (I^*, (C_1^*, C_2^*, \ldots C_N^*), T^*)$ that is indistinguishable from the adversary's view $v(H)$ using the trace $\tau(H)$.

- The index structure I is a tree of height h and branching factor k where all internal nodes are of the same size L_{int} and all the leaf nodes are of the size L_{leaf}. Here, L_{int} and L_{leaf} are defined by the type of the indexing scheme used. To build I^*, Sim builds a tree by adding k^l internal nodes at each level $l = 0$ to $h - 1$. Each of these internal nodes are assigned a random value $R_{l,j} \xleftarrow{U} \{0,1\}^{L_{int}}$ for $l = 0$ to $h - 1$, $j = 1$ to k^l. Next, Sim adds k^h leaf nodes and assigns a random value $R_j \xleftarrow{U} \{0,1\}^{L_{leaf}}$ to each of them, where $j = 1$ to k^h. Finally, for each node on level l, a label is assigned at random $Label \xleftarrow{U} \{0,1\}^{|\delta_l|}$, for $l = 0$ to h. During index construction, since each node is encrypted using a PCPA secure scheme and the labels are permuted using a pseudorandom permutation, I^* is indistinguishable from the actual index structure I.

- From trace $\tau(H)$, Sim has the knowledge of the lengths of the documents in C. The simulator choses N random values $(C_1^*, C_2^*, \ldots C_N^*)$ such that $|C_i^*| = |C_i|$. Since a PCPA secure scheme is used for generating C, each C_i^* is computationally indistinguishable from C_i.

- From the trace $\tau(H)$, Sim has the knowledge of Content Similarity pattern $\zeta(H)$. The trapdoor information T is generated as follows. For $1 \leq i$, $u \leq q$ and $1 \leq j$, $v \leq r$ and $0 \leq m$, $w \leq h$, If $\zeta([i, j, m], [u, v, w]) = 1$, set $T_{j_m}^{*i} = T_{v_w}^{*u} \xleftarrow{U} \{0,1\}^{|\delta|}$ else set $T_{j_m}^{*i} \xleftarrow{U} \{0,1\}^{|\delta|}$ where $|\delta|$ is the length of a node label. Since a pseudorandom permutation is used to generate each label $T_{j_m}^{*i}$, the simulated trapdoor $T_{j_m}^{i}$ is indistinguishable from the real one.

4.4 Private Content Based Image Retrieval

A closely related problem to *SSE* is that of securely searching on a public database, in which case the data is in unencrypted form at the server. In this section, we consider the problem of secure content

Algorithm 10 PCBIR: Secure Index Construction using CS-SSE

At $S1$:
Build the index structure I on D
Encrypt indexed database to obtain $I_{K_1} = E_{K_1}(I)$
Encrypt database elements to obtain $C = E_{K_2}(D)$
Send I_{k_1} and C to server $S2$
At $S2$:
Encrypt I_{K_1} to obtain $I_{K_3} = E_{K_3}(I_{K_1})$
Apply random permutations on the encrypted index structure to obtain $I^{\pi}_{k3} = \pi(I_{k_3})$
Encrypt C to obtain $C_{K_4} = E_{K_4}(C)$
Send $I^{\pi}_{k_3}$ and C_{K_4} to server $S1$
At S1:
$I_f = D_{K_1}(I^{\pi}_{k_3})$ //remove local encryption
$C_f = D_{K_2}(C_{K_4})$

similarity search on a public image database. Shashank *et. al.* considered this problem in [69] and gave a private content based image retrieval protocol using hierarchical indexing. Their idea is to perform oblivious walk on the tree structure at the server end and retrieve the relevant results. For this purpose they used *Oblivious Transfer (OT)* protocol to privately retrieve the node information at each level from the server. The retrieved information is compared with the query and decision of which child node to retrieve next is taken at user end. Due to the use of OT, the computation on the server at each level is $O(n_l)$, where n_l is the number of nodes at level l. This linear computation is unavoidable since the data is in unencrypted form in the server and failure to access each element will lead to a privacy disclosure. Because of the large data size, most of the hierarchical index structures have huge amount of nodes to improve efficiency. For example, in case of vocabulary tree the number of leaf nodes are usually in the order of Million. Thus, it is impractical to have computation which is linear in the number of nodes in a tree for real time applications like online search.

In order to overcome this limitation, we reduce this problem to that of searching on an encrypted database using two-server model with CS-SSE protocol. Consider a server $S1$ with a database D on which the users would like to securely search for similar data. The idea is to use a third party server $S2$ which builds the secure index on the database D that will eventually be stored on $S1$. During index construction, $S2$ uses secret keys for encryption which are unknown to server $S1$ (we assume non colluding servers). In other words, server $S2$ does the work of content owner in the CS-SSE protocol. However, $S1$ might not be willing to release its database D to a third party in an unencrypted form. Hence, D is first encrypted at $S1$ before sending to $S2$, where index is constructed on this encrypted database. Finally, $S1$ removes its local encryption on the secure index obtained from $S2$. Here, the encryption schemes used at both servers need to be *Commutative* so that the order of decryption does not matter. Once this off line processing is over, users can securely search on the secure index stored on $S1$ in communication with server $S2$. We now explain the offline processing stage in detail.

During Key Generation phase, $S1$ generates keys K_1, K_2 and $S2$ generates keys K_3 and K_4. During Index construction, the server $S1$ with the database D builds the index structure I using vocabulary tree after extracting features for each image in D. The obtained index structure is then encrypted using key K_1 to obtain $I_{K_1} = E_{K_1}(I)$ which is sent to a third party server $S2$ along with the encrypted database $C = E_{K_2}(D)$. At $S2$, the received index structure and the encrypted database are encrypted using K_3 and K_4 to obtain $I_{K_3} = E_{K_3}(I_{K_1}) = E_{K_3}(E_{K_1}(I))$ and $C_{K_4} = E_{K_4}(C) = E_{K_4}(E_{K_2}(D))$. A random permutation π is then applied at each node of I_{K_3} to obtain $I^{\pi}_{k_3} = \pi(I_{k_3})$. The permutation process is similar to the one used in the CS-SSE protocol. The obtained structures $I^{\pi}_{k_3}$ and C_{K_4} are then sent to server $S1$ where the local encryption is removed to obtain the final Index structure I_f and the database C_f on which the users can search for similar images securely. Here, the encryption schemes used at both the servers need to be commutative so that the order of decryption does not matter. We have: $I_f = D_{K_1}(I^{\pi}_{k_3}) = D_{K_1}(\pi(E_{K_3}(E_{K_1}(I))) = \pi(E_{K_3}(I))$ and $C_f = D_{K_2}(C_{K_4}) = D_{K_2}(E_{K_4}(E_{K_2}(D))) = E_{K_4}(D)$. The Index Construction is summarized in Algorithm 10.

The online querying is same as that of the CS-SSE protocol. The only difference is that user, before querying on $S1$, needs to communicate with $S2$ to obtain the keys and random permutations used while index construction. However, this communication is one time even for multiple queries.

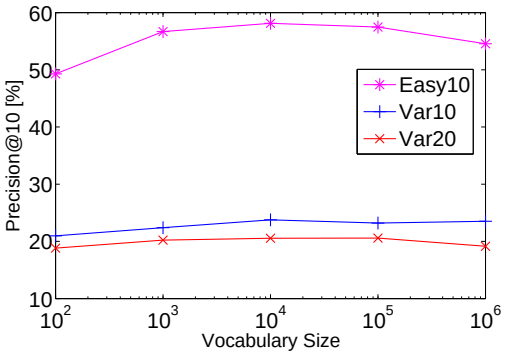
4.5 Experiments

We implemented the proposed CS-SSE system to privately search similar images from the image database and measured the performance in terms of retrieval quality, query time and communication. We performed experiments on various image datasets using vocabulary tree as an index structure and compared it with existing techniques. For feature extraction we use state of the art Scale Invariant Feature Transform (SIFT) [47]. SIFT describes local features in an image which is invariant to scale, rotation, viewpoint and illumination. As a result, images taken under different conditions can be easily matched and thus it is widely used in recognition and image matching problems. Symmetric encryption scheme used in all the experiments is AES in CTR mode with the key size of 128 bit which is known to be PCPA (Pseudorandomness against chosen plaintext attacks) secure. All the experiments are executed on Intel(R) Core $i7$ CPU $3.33GHz$ machine. Now, we describe the performance measures used to evaluate our algorithm.

Precision@topk: For each query image, $topk$ similar images are retrieved from the server. Among these $topk$ images, percentage of images that share the same category with the query image is computed which is then averaged over all query images to calculate $precision@topk$.

Query Time: Search time is the time between sending a query image and retrieving $topk$ similar images from the server. We only consider the time till retrieval of $topk$ ids of the result images and not the actual images since it is same for all algorithms and also depends on the size of database. In CS-SSE schemes a query is an image which consists of multiple features, which can be queried in parallel and the results are accumulated to calculate the score.

(a) Search results on Ukbench dataset: The bounding box displays the query image followed by the retrieved results in order from left to right.



(b) Retrieval Quality of CS-SSE scheme in terms of Precision@10 for various datasets.

Figure 4.5: Performance of CS-SSE Scheme

Communication: The communication is the amount of data transfered between user and server during one search transaction. For the same reason mentioned above, here also we consider the communication till the retrieval of $topk$ ids of the result images.

4.5.1 Retrieval Quality

In order to test the functionality of the proposed CS-SSE system, we perform private image search on Ukbench dataset [53] using vocabulary tree. The dataset consists of 2550 groups of 4 images each (total 10200), taken under different positions and varying illumination conditions. We used a subset of this dataset to train the vocabulary with branching factor $k = 10$ and number of leaf nodes $n = 100K$. To measure the performance we count how many of the 4 images come into top-4 results when querying an image from that set of four images. Averaging these scores over all the queries, we achieved final score of 3.01. Few query images and their retrieved results (after removing identical result image) are shown in Figure 4.5(a).

Next we consider the popular object detection dataset Caltech256 [27]. We perform experiments on three different subsets of Caltech256 - Easy10, Var10 and Var20 as described in [12]. The set Easy10 consists of 10 object categories that are easiest to classify, Var10 and Var20 consist of 10 and 20 object categories respectively that span the full range of classification difficulty. For all the sets we randomly chose 40 images of each category as database and 25 images of each category as a query set. We report the precision@10 by varying vocabulary size from 100 to $1M$. As shown in Figure 4.5(b) the precision increases by increasing the vocabulary size. For all the datasets we achieve highest precision between vocabulary size of $10K$ to $100K$, which shows that large index structures are needed to improve the

Dataset	Dataset Size	Precision@10		Query Time (ms)		Communication (MB)	
		CS-SSE	*CSL*	*CS-SSE*	*CSL*	*CS-SSE*	*CSL*
Easy10	100	37.6	2.9	0.055	0.013	3.80	0.50
	200	44.5	3.1	0.055	0.049	3.84	2.79
	300	52.07	3.66	0.055	0.058	3.87	4.20
	400	57.48	8.05	0.055	0.090	3.90	5.60
Var10	100	21.4	5.9	0.055	0.213	3.81	1.39
	200	21.9	4.25	0.068	0.346	3.84	4.22
	300	25	11.47	0.068	0.213	3.89	6.33
	400	23.2	3.1	0.055	0.493	3.96	8.45
Var20	200	15.1	3.35	0.055	0.093	3.85	2.79
	400	17.65	3.45	0.055	0.080	3.91	3.33
	600	18.47	2.73	0.056	0.300	4.01	12.68
	800	20.58	3.88	0.056	0.660	4.09	16.91

Table 4.1: Performance Comparison with *CSL* on various datasets.

accuracy of the system. Because of the optimal computation cost our algorithm is best suited for large index structures.

4.5.2 Performance Evaluation

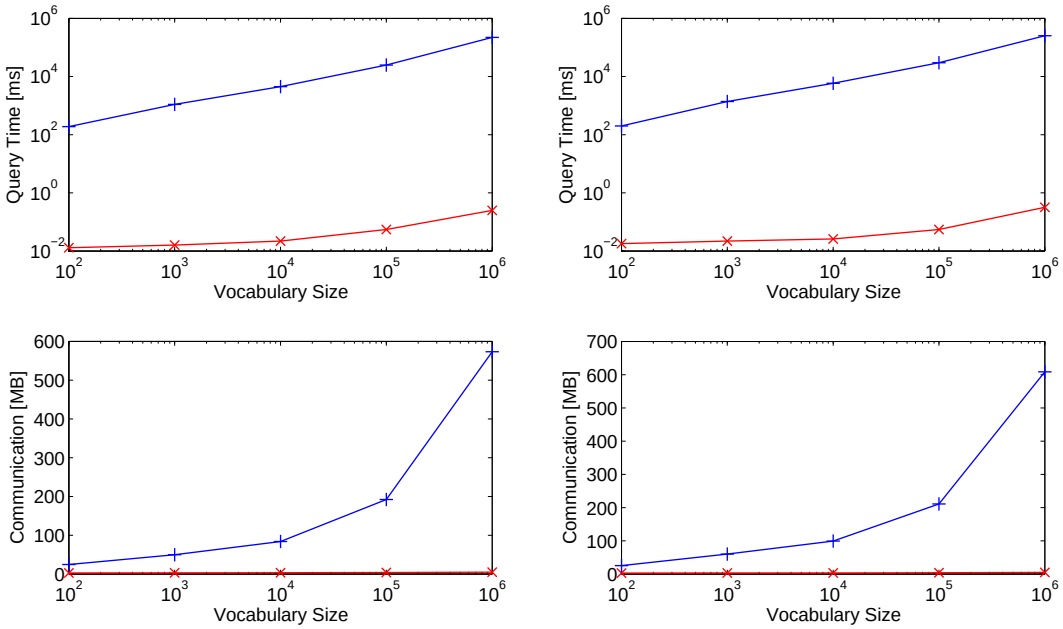
We compare the performance of the proposed CS-SSE scheme with that of CSL. For our scheme we use a vocabulary tree of size fixed to $10K$ with branching factor of 10. For the case of CSL we run the algorithm with multiple LSH parameter combinations and report the query time and communication against the best precision@10. The results are summarized in Table 4.1 where we vary the dataset size and report precision@10, query time and required communication. The results show that our method achieves better precision with very less communication and time compared to CSL. The improvement in precision is due to the sophisticated TF-IDF scoring method used in our algorithm as opposed to just counting the number of common features between query and result image. Using TF, not only we capture the information of number of common features but also capture the information about their frequency in a specific image. Further more, using IDF we compute the normalized score which gives fairness between images with high number of features and low number of features. On an average we achieve 30% improvement in accuracy across various datasets.

For small datasets the required communication and query time of CSL is better than our algorithm, because each LSH bucket contains very few bits. But as we increase the database size, the required communication increases rapidly in case of CSL while in our case it does not increase much. The reason is that in case of CSL, the LSH bucket contains bit vector of length N, where N is the size of the dataset, while in case of our algorithm only inverted indexes of a respective leaf nodes are transfered.

In the vocabulary tree with $10K$ leaf nodes, each leaf node will contain very small amount of data. To show the effect of large data size we also perform the experiments on a slightly bigger dataset of approximately 4500 images Scene15 [41], which consists of fifteen natural scene images. On the full dataset, our algorithm requires only 1 MB of communication as opposed to 25 MB in case of CSL. That is our algorithm has 25 times lower communication cost than that of CSL on the dataset of 4500 images. Even though very few amount of data is transfered per feature during our querying mechanism, overall communication increases because each query image contains large number of features. However, in other cases in which query item contains very small number of features the communication will be comparatively very low. Also, Note that CSL is a constant round protocol whereas our algorithm uses $\log_k n$ rounds. But, this is not a serious limitation as the number of rounds is very small even for large datasets.

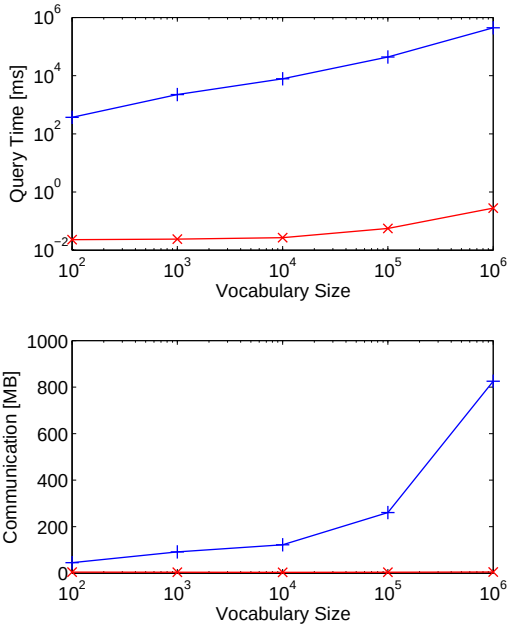
4.5.3 Image Search on Public Database

We run our algorithm with the modified setting to search on public database and compare it with the state of the art algorithm of *PCBIR* [69]. As both algorithms give exact results in terms of precision, we only compare the performance in terms of time and communication. For the datasets mentioned above we vary the vocabulary size from 100 to $100K$ with branching factor of 10 and report the time and communication as shown in Figure 4.6. While implementing the protocol of [69], we use the same OT protocol as the one used in [69], which has a communication complexity of $O(\sqrt{n})$ for a 1-out-of-n bit transfer. Note that better OT protocols [23, 46] in terms of communication exist, using which the communication of [69] would be considerably reduced, although, it will be still more than that of our algorithm. Also, this reduction in communication gap comes at the cost of further increasing the computation cost of [69] due to the costly cryptographic techniques used in the OT protocols of [23]. As can be seen from Figure 4.6, as the vocabulary size increases search time increases rapidly in case of PCBIR while in case of our algorithm it increases very slowly as it only depends on the height of the tree and not on the number of nodes. For a vocabulary size of 1 Million leaf nodes, our algorithm is $O(10^5)$ times faster. Similarly, this is also true for communication. In a specific case of Figure 4.6(d), unlike other results, the communication decreases from vocabulary size of 100 to $10K$, this is because Scene15 dataset is large compared to other datasets. Hence, in case of small vocabulary each leaf node contains relatively large amount of images. The dominating factor will be the leaf node data in case of small vocabulary and as we increase the vocabulary size the number of images in each leaf node decreases which results into low communication as shown in that figure for the case of vocabulary size of $100K$.

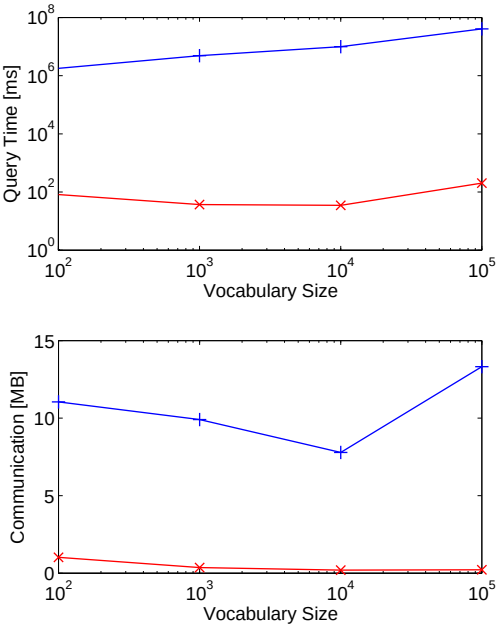


(a) Easy10

(b) Var10



(c) Var20



(d) Scene15

Figure 4.6: Performance Comparison with *PCBIR* [+ : PCBIR × : CS-SSE]

4.6 Summary

We formally defined the notion of Content Similarity Search on Encrypted Data and proposed an efficient CS-SSE scheme for the case of hierarchical index structures. We provided thorough security analysis and proved our construction is adaptively semantic secure against a polynomial time adversary. We demonstrated the applicability of the proposed scheme for the particular case of image datasets, using a state of the art indexing scheme and evaluated the performance in comparison with the current known techniques. Further, we considered the case of similarity search on public image database and extended the CS-SSE scheme for this setting. Through experimental validation, we showed that our scheme outperforms the existing protocols for this setting. Although, we have considered only hierarchical index structures, the proposed framework could be explored using other data structures for better functionality and performance.

Chapter 5

Conclusion and Future Work

In this thesis, we addressed privacy issues in the domain of Data Mining and Information Retrieval. Specifically, we gave privacy preserving algorithms for Outlier Detection and Content Based Similarity Search over Encrypted Data. First, we proposed a novel algorithm of distance based outlier detection using Locality Sensitive Hashing in a distributed setting where data is horizontally partitioned among data owners. We showed that the proposed algorithm is scalable in case of large datasets with high dimensionality. Then, we extended our distributed outlier algorithm to propose an algorithm for private outlier detection that broke the previous known bounds of quadratic cost. We gave thorough complexity analysis for both the algorithms and have provided empirical results on various datasets to support our claims. We also compared our algorithm with the previous known results and showed that ours perform better in terms of communication as well as computational cost.

Next, we formally defined the notion of content similarity search on encrypted data and proposed an efficient CS-SSE scheme for the case of hierarchical index structures. We provided thorough security analysis and proved our construction is adaptively semantic secure against a polynomial size adversary. We demonstrated the applicability of the proposed scheme for the particular case of image datasets, using a state of the art indexing scheme and evaluated the performance in comparison with the current known techniques. We showed that our algorithm achieves optimal computational cost with similar accuracy and privacy when compared to the existing techniques. We also extended our CS-SSE scheme for Private Content Based Image Retrieval and showed that our scheme outperforms the existing protocols for this setting.

5.1 Future Work

We showed the advantage of LSH based approaches in the proposed outlier detection method in terms of low communication as well as computational cost. Similarly, we can explore the use of LSH in other data mining tasks in order to achieve efficient privacy preserving algorithms. Since, LSH can be considered as a clustering mechanism, efficient privacy preserving clustering algorithms can be built using LSH. Another future direction of research is to extend the proposed privacy preserving

outlier detection algorithm for hybrid data partitioning which include both horizontal as well as vertical partitioning of the data. Also, one can use LSH to design an algorithm for private density based outlier detection which performs better than distance based outlier detection in certain scenarios.

Apart from hierarchical index structures, the proposed scheme of CS-SSE can be extended to other data structures like hashing which may further improve performance in terms of server side computations. One may also work towards achieving constant round protocol for the proposed CS-SSE scheme as opposed to the logarithmic round protocol.

Publications

1. Madhuchand Rushi Pillutla, **Nisarg Raval**, Piyush Bansal, Kannan Srinathan and C.V. Jawahar, *LSH based outlier detection and its application in distributed setting,* 20^{th} ACM International Conference on Information and knowledge management, **CIKM** (2011).

2. **Nisarg Raval**, Madhuchand Rushi Pillutla, Piyush Bansal, Kannan Srinathan and C.V. Jawahar, *Privacy Preserving Outlier Detection Using Locality Sensitive Hashing,* 11^{th} IEEE International Conference on Data Mining Workshops, **ICDM** (2011).

3. **Nisarg Raval**, Madhuchand Rushi Pillutla, Piyush Bansal, Kannan Srinathan and C.V. Jawahar, *Efficient Content Similarity Search on Encrypted Data using Hierarchical Index Structures,* Transactions on Data Privacy (*Under Review*).

4. **Nisarg Raval**, Rashmi Tonge and C.V. Jawahar, *Image Retrieval using Eigen Queries,* 11^{th} Asian Conference on Computer Vision, **ACCV** (2012).[1]

[1]This work is independent and not part of this thesis.

Bibliography

[1] C. C. Aggarwal. On k-anonymity and the curse of dimensionality. In *VLDB*, 2005.

[2] C. C. Aggarwal and P. S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*. 2008.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, 2000.

[4] F. Angiulli and F. Fassetti. Very efficient mining of distance-based outliers. In *CIKM*, 2007.

[5] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *SIGKDD*, 2003.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.

[7] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta. In-network outlier detection in wireless sensor networks. In *ICDCS*, 2006.

[8] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Optics-of: Identifying local outliers. In *PKDD*, 1999.

[9] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGOD*, 2000.

[10] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, 2005.

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

[12] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large scale online learning of image similarity through ranking. *JMLR*, 2011.

[13] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 1998.

[14] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD*, 2002.

[15] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, 2006.

[16] Z. Dai, L. Huang, Y. Zhu, and W. Yang. Privacy preserving density-based outlier detection. *CMC*, 2010.

[17] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, 2004.

[18] H. Dutta, C. Gianella, K. Borne, and H. Kargupta. Distributed top-k outlier detection in astronomy catalogs using the DEMAC system. In *ICDM*, 2007.

[19] C. Dwork. Differential privacy. In *ICALP*, 2006.

[20] E. Goh. Secure indexes, cryptology eprint archive. In *Report 2003/216*, 2003.

[21] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 1996.

[22] K. Frikken. Privacy-preserving set union. In *ACNS*, 2007.

[23] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP*, 2005.

[24] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[25] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. 2004.

[26] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM*, 1996.

[27] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007.

[28] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *SIGMOD*, 1998.

[29] D. Hawkins. *Identification of outliers*. Monographs on applied probability and statistics. Chapman and Hall, London [u.a.], 1980.

[30] A. Inan, Y. Saygyn, E. Savas, A. Hintoglu, and A. Levi. Privacy preserving clustering on horizontally partitioned data. In *Data Engineering Workshops*, 2006.

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

[32] G. Jagannathan, K. Pillaipakkamnatt, R. N. Wright, and D. Umano. Communication-efficient privacy-preserving clustering. *Data Privacy*, 2010.

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

[34] E. jin Goh. Secure indexes. In *Cryptology Eprint Archive*, 2004.

[35] T. Johnson, I. Kwok, and R. Ng. Fast computation of 2-dimensional depth contours. In *KDD*, 1998.

[36] L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO*, 2005.

[37] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, 1998.

[38] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 2000.

[39] A. Koufakou and M. Georgiopoulos. A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes. *DMKD*, 2010.

[40] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *TPAMI*, 2012.

[41] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

[42] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *ICDE*, 2006.

[43] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou. Enabling efficient fuzzy keyword search over encrypted data in cloud computing. In *Cryptology ePrint Archive, Report 2009/593*, 2009.

[44] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, 2007.

[45] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *JOURNAL OF CRYPTOLOGY*, 2000.

[46] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In *ISC*, 2005.

[47] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[48] W. Lu, A. Swaminathan, A. L. Varna, and M. Wu. Enabling search over encrypted multimedia databases. In *MFS*, 2009.

[49] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*.

[50] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM TKDD*, 2007.

[51] M. K. Mehmet Kuzu, Mohammad Saiful Islam. Efficient similarity search over encrypted data. In *ICDE*, 2012.

[52] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *SP*, 2008.

[53] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[54] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, 2011.

[55] R. Ostrovsky. Efficient computation on oblivious rams. In *STOC*, 1990.

[56] M. E. Otey, A. Ghoting, and S. Parthasarathy. Fast distributed outlier detection in mixed-attribute data sets. *DMKD*, 2006.

[57] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, 2003.

[58] V. Pappas, M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin. Private search in the real world. In *ACSAC*, 2011.

[59] H.-A. Park, B. H. Kim, D. H. Lee, Y. D. Chung, and J. Zhan. Secure similarity search. In *GRC*, 2007.

[60] M. R. Pillutla, N. Raval, P. Bansal, K. Srinathan, and C. V. Jawahar. Lsh based outlier detection and its application in distributed setting. In *CIKM*, 2011.

[61] B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *SIGKDD*, 2002.

[62] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, 2000.

[63] N. Raval, M. Pillutla, P. Bansal, K. Srinathan, and C. Jawahar. Privacy preserving outlier detection using locality sensitive hashing. In *ICDMW*, 2011.

[64] S. J. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *VLDB*, 2002.

[65] P. Rousseeuw and A.M.Leroy. Robust regression and outlier detection. In *John Wiley and Sons*, 1997.

[66] P. R. Sabbu, U. Ganugula, S. Kannan, and B. Bezawada. An oblivious image retrieval protocol. In *ICAINA*, 2011.

[67] P. Samarati. Protecting respondents' identities in microdata release. *IEEE TKDE*, 2001.

[68] M. Shaneck, Y. Kim, and V. Kumar. Privacy preserving nearest neighbor search. *ICDMW*, 2006.

[69] J. Shashank, P. Kowshik, K. Srinathan, and C. Jawahar. Private content based image retrieval. In *CVPR*, 2008.

[70] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.

[71] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *SP*, 2000.

[72] L. Su, W. Han, S. Yang, P. Zou, and Y. Jia. Continuous adaptive outlier detection on distributed data streams. In *HPCC'07*, 2007.

[73] J. Tang, Z. Chen, A. W. chee Fu, and D. Cheung. A robust outlier detection scheme for large data sets. In *Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, 2001.

[74] Y. Tao and X. Xiao. S.: Mining distance-based outliers from large databases in any metric space. In *KDD*, 2006.

[75] J. Vaidya and C. Clifton. Privacy-preserving outlier detection. In *ICDM*, 2004.

[76] J. Vaidya, M. Kantarciouglu, and C. Clifton. Privacy-preserving naive bayes classification. *The VLDB Journal*, 2008.

[77] V.Barnett and T.Lewis. Outliers in statistical data. In *John Wiley*, 1994.

[78] Y. Wang, S. Parthasarathy, and S. Tatikonda. Locality sensitive outlier detection: A ranking driven approach. In *ICDE*, 2011.

[79] X. Xiao and Y. Tao. Anatomy: simple and effective privacy preservation. In *VLDB*, 2006.

[80] B. Yang, H. Nakagawa, I. Sato, and J. Sakuma. Collusion-resistant privacy-preserving data mining. In *KDD*, 2010.

[81] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *SDM*, 2005.

[82] A. C. Yao. Protocols for secure computations. In *FCS*, 1982.

[83] H. Yu, X. Jiang, and J. Vaidya. Privacy-preserving svm using nonlinear kernels on horizontally partitioned data. In *SAC*, 2006.

[84] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. *SIGMOD*, 1996.

[85] Z. Zhou, L. Huang, Y. Wei, and Y. Yun. Privacy preserving outlier detection over vertically partitioned data. In *EBISS*, 2009.