PRIVACY PRESERVING RANGE QUERY SEARCH OVER ENCRYPTED NUMERIC DATA IN CLOUD

by

Yongmin Yan

B.Eng., Zhejiang University, 2011

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

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

APPROVAL

Name:	Yongmin Yan
Degree:	Master of Science
Title of Thesis:	Privacy Preserving Range Query Search over Encrypted Numeric Data in Cloud

Examining Committee: Dr. Ze-Nian Li, Professor
Chair

Dr. Ke Wang, Senior Supervisor, Professor

Dr. Wo-Shun Luk, Supervisor, Professor

Dr. Andrei A.Bulatov, SFU Examiner, Associate Professor

Date Approved: August 12, 2013

Abstract

In cloud computing, to protect privacy when outsourcing database management systems, confidential data (e.g., bank account balance) has to be encrypted which however renders traditional query processing methods (e.g., the B-tree index) inapplicable. In this thesis, we consider processing range query search (e.g., retrieving all records with a bank account balance being less than some threshold) over a confidential numeric attribute with two main challenges. First, the confidentiality requires hiding the value and the relative order of the attribute in records from the cloud server, but computing a range query requires comparing the values of this attribute. Second, since the encrypted data does not preserve the closeness of plaintext values (for privacy reasons), a straightforward search has to scan the entire encrypted data to retrieve wanted records, which is not scalable for data of the cloud scale. In this work, we present a novel searchable encryption/index scheme to address these challenges.

Keywords: Privacy; Range Query; Encryption; Cloud Computing; Database as a Service; Outsourcing; Numeric Attribute; Index

Acknowledgments

My special thanks goes to my senior supervisor, Dr. Ke Wang, for his invaluable guidance and support. This work would not have been finished without his insightful comments and suggestions. I am inspired by his rigorous attitude of scholarship. I am grateful for benefiting a lot from his extensive knowledge and extraordinary research experience.

I want to express my gratefulness to my supervisor, Dr. Wo-Shun Luk and examiner Dr. Andrei A.Bulatov for their precious time and generous help on commenting my thesis. And I would also like to thank Dr. Ze-Nian Li for taking the time to chair my thesis defense.

To my former and current labmates: Judy Yeh, Chao Han, Peng Wang, Bo Hu, Zhihui Guo, Chengyi Zhang, Weipeng Lin, Hongwei Liang, Yue Wang, thank you all for your generous help, warm encouragement as well as making my graduate studies an enjoyable experience.

Last but very importantly, I would like to thank my parents for their endless love, support and encouragement!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cloud computing is no doubt an effective approach to deal with big data, through providing on-demand high quality services from powerful and configurable computing resources. One application of cloud computing is “database as a service” [20][21], where the data owner outsources complex database management systems into the cloud server. To protect data privacy against attacks from the cloud server, confidential data must be encrypted before being uploaded to the cloud server. This, however, makes it difficult to perform traditional query processing operations.

In this thesis, we consider the problem of processing range query search over a confidential numeric attribute K in an outsourced setting of cloud computing and we assume exact answers must be returned. In our model, the cloud server is the adversary and it knows all the plaintext domain values (i.e., all the distinct actual values of K) and has access to all the encrypted records and queries. In addition, the cloud server possesses background knowledge of the frequency information about the K values and knows partial correspondence between some plaintext values and ciphertext values (i.e., the cloud server knows palintext values for some ciphertext values). Our goal is to protect the complete correspondence between plaintext values and ciphertext values for both records and queries from being disclosed except those obtained by the cloud server from the background knowledge (i.e., the cloud server should not be able to figure out plaintext values for other ciphertext values it does not know). We will discuss this in detail in Chapter 3.

This problem faces two challenges. First, the confidentiality requires hiding the values and the relative order of the attribute in records from the cloud server, but computing a range query requires comparing the values of this attribute. We will discuss the reason why protecting the relative order is important soon in the following context. Second, since the encrypted data does not preserve the closeness of plaintext values (for privacy reasons), a straightforward search has to scan the entire encrypted data to retrieve wanted records, which is not scalable for data of the cloud scale. In this work, we present a novel searchable encryption/index scheme to address these challenges.

Additionally, the exactness of the returned answers could expose the plaintext frequency and order information which could be used to help the cloud server to identify the plaintext values for certain ciphertext values. Here comes the controversy part because the exact answers, however, are often required. This is because, if additional data is returned (i.e., there are false positives), the user will probably get more information than allowed which could cause privacy concerns for the data owner. If less data is returned (i.e., there are false negatives), the user will get incomplete answers which will probably become useless. In this sense, exact answers are often required. In our work here, we do not intend to achieve full frequency and order privacy to guarantee the exactness of the returned answers. Brief reasons on why exact answers could disclose the frequency and order information will be discussed soon and we consider to resolve the issue of exact answers as our future work. Also, we will use the terms “privacy” and “security” interchangeably in the rest of the thesis and either of them means that the plaintext values are hidden rather than exposed. And the attacker here refers to the cloud server.

1.1 Background

In modern society, cloud computing [28] is a promising paradigm which becomes more and more ubiquitous. In cloud computing, people can flexibly configure computing resources (e.g., servers, storage systems and applications) and this brings stupendous scalability.

The great advantage of cloud computing can be explained by the following example. For instance, a startup company wants to do a throwaway computation on big data and this needs ten servers with database management systems included for storage and computation. However, after the computation, the company actually does not need the data and the hardware resources anymore. Before cloud computing, the company has to buy the hardware (i.e., the ten servers including database management systems) and all these resources would be useless to that company after computation and thus would be wasted. In addition, the company also needs to spend time configuring these machines.

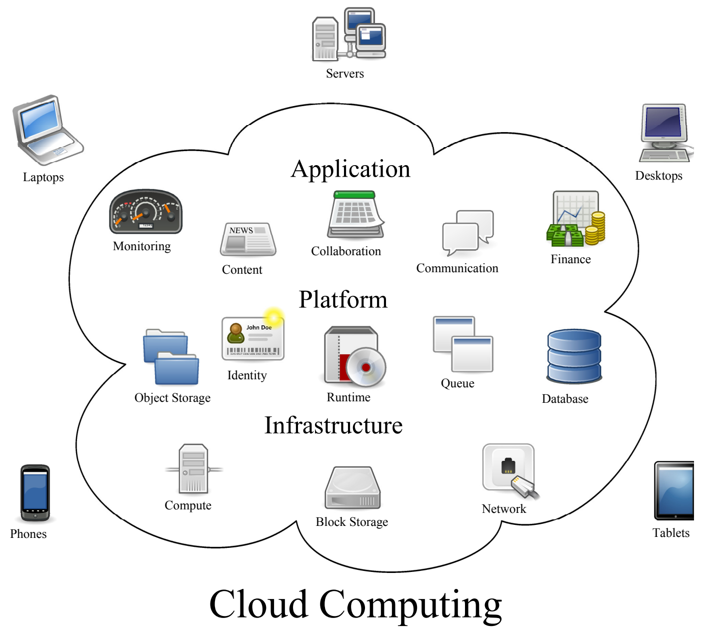

Figure 1.1: Cloud computing logical diagram

But now, in the paradigm of cloud computing, the company can actually rent ten (virtual) machines from Amazon EC2 and use S3 [1] for data storage with much cheaper cost and all the resources are already configured. This totally frees the company from complicated configuration by itself and also enable the company to just focus on the computation logic. More importantly these resources could be collected back and utilized by others after this company finishes the computation. These resources won't be wasted and the company saves cost.

As shown in Figure 1.1: Cloud computing logical diagram [1], cloud computing provides on-demand services through three "X as a Service" models with three different levels of resources [28] as follows.

- Infrastructure as a Service (IaaS): In this model, the service provider offers hardware infrastructure (usually virtual machines) in which the customers can install arbitrary softwares including operating systems as they want. The customers also have control over the hardware resources like network and disk storage. Typical examples are Amazon EC2 [1].

[1]`http://en.wikipedia.org/wiki/File:Cloud_computing.svg`

- Platform as a Service (PaaS): In this model, the customers don't have control over low-level services like operating systems but they can freely deploy applications created by certain programming languages and tools supported by the provider. Typical examples include Google App Engine [2], Windows Azure Cloud Services [3] and so on.

- Software as a Service (SaaS): In this model, the customers don't have to be worried about any deployment issues (since they don't even have control over any software installations) but they can just use the applications supported by the service provider. For example, Gmail, Google App, Microsoft Office 365 are all instances of SaaS.

One application of cloud computing is "database as a service" [20][21], where the data owner outsources complex database management systems into the cloud (e.g., Amazon Relational Database Service: MySQL, Microsoft SQL Azure: MS SQL). The on-demand high quality computation power of the cloud server for query evaluation attracts great attention from the customers and makes such a paradigm itself popular.

As promising as it is, there are also problems and challenges with this "database as a service" model. The concern comes from the fact that the data is outside the trusted domain of the data owner but within the cloud server which could be malicious. For instance, a hospital (the data owner) wants to outsource the medical database into a cloud server for authorized researchers (the data users) to do analysis. The medical data is confidential so the server is not allowed to see any plaintext values in the database. As a result, the medical database needs to be encrypted before it is outsourced. This gives rise to the difficulty to do query evaluation. Searchable encryption which supports query evaluation directly over the encrypted data is a promising approach to achieve both the query computation power from the cloud server and data privacy.

1.2 Motivation

As a big part, range query search is quite common in people's daily life. It is worthwhile to support range query search over encrypted data in our context where the cloud server is the adversary and the cloud server is not allowed to see any plaintext values. So here we want to consider the problem of processing range query search over a confidential numeric attribute K for a relational database outsourced to a cloud server. In addition, range query search is so general that we will later on show query evaluation (i.e., equality test) on a categorical attribute could also be treated by the same way as treating a special kind of range query search (see more discussion in Chapter 3).

More specifically, a range query over a numeric attribute K with domain $[\alpha, \beta]$ will retrieve all records having a K value within a specified query interval $[a, b]$ ($a \leq b, a, b \in [\alpha, \beta]$). We assume that K is confidential (e.g., bank account balance, income, transaction time, and other sensitive financial information) and our goal here is to protect any plaintext K values from being disclosed except those obtained by the cloud server due to prior background knowledge. We aim to achieve the protection of the plaintext values from three confidentiality requirements by protecting the secret key, the frequency information and the relative order of the encrypted K values. Also, the cloud server should be able to compute the exact answers from the outsourced encrypted database. We will discuss the goal, the model and the specific requirements as well as what the cloud server can do and what the cloud server knows in much more detail in Chapter 3.

Note we give a stronger confidential model by requiring the relative order also to be confidential here. The confidentiality of the order information results from the following observation: as we discussed briefly at the beginning of this chapter, the cloud server knows all the plaintext K values and has access to all the encrypted records. Once the order information of the encrypted values is exposed, that is, the cloud server knows the ordered encrypted values $u_1 < \cdots < u_m$. Then with ordered plaintext domain values $v_1 < \cdots < v_m$, it is immediately known that u_i is the encryption of v_i. As a result, it is necessary for us to protect the relative order information as well.

1.3 Challenges

Although there already has been quite some work on secure processing of equality test, range query search, aggregation query computation and keyword based query search or similarity query search (see Chapter 2), little work on secure processing of range query search without preserving the order information has been reported. A basic challenge is that computing a range query requires comparing the order of values, but the confidentiality of the attribute requires that this order be concealed to the cloud server. The usual trick of injecting noises into values does not work here because it could change the outcome of comparison. The order preserving encryption [6][15] does not meet our confidentiality requirement because it preserves the order of plaintext values. The bucketing scheme in [20][24] maps values to coarser buckets at the cost of returning false positives, which can be arbitrarily many. Since the boundary of a bucket corresponds to some range of the numeric attribute, it may be disclosed by correlating the growth of buckets with background knowledge on such growth. The homomorphic encryption scheme [22], which is good for aggregation, however, is not useful for range query search. See more discussions in Chapter 2.

Another challenge is that, since we require the encryption scheme not to preserve the order information, the encrypted data will not preserve the closeness of plaintext values. So all the indexing methods based on the order information such as the B-tree index won't be useful in our setting. To our knowledge, no efficient indexing method exists to help make the query evaluation on encrypted data much faster. A straightforward search has to scan the entire encrypted data to retrieve wanted records. This is not scalable for data of the cloud scale.

In addition, the frequency analysis attack is a strong attack where the attacker tries to identify corresponding plaintext values for certain ciphertext values by linking the same frequency statistics obtained from background knowledge. To resist this attack, a probabilistic encryption (frequency not preserved) has to be adopted instead of a deterministic one (frequency preserved), yet, we still need to return exact answers. In the simplest case of frequency analysis attack, to compromise the plaintext content of a search query when a deterministic encryption (frequency preserved) is adopted, the attacker could collect the frequency information gradually by keeping a counter for each encrypted query. Then every time the attacker sees an encrypted query which previously appeared, the counter for this query is incremented by one. After some time, with high probability, the attacker could map the encrypted query with the largest counter number (most frequent) to the most frequent one in plaintext which the attacker knows from the background knowledge. As a result, the plaintext values of the query could be exposed.

1.4 Contributions

In this thesis, we present a novel probabilistic encryption scheme to address the problem of range query search over a confidential numeric attribute K without preserving the order information in an encrypted setting and we also give an efficient indexing method directly over the encrypted database to give more efficient query evaluation than the linear scan method. As we briefly discussed at the beginning of this chapter, we need to guarantee the exactness of the returned answers which however could disclose the plaintext frequency and order information. The reason is that repeated queries could be identified based on the same returned answers and thus frequency information could be inferred. Also, exact answers guarantee the consecutive property of K among the returned records. For instance, query answers $\{X, Y, W\}$ and $\{F, X, Y\}$ indicate records X, Y are consecutive in the order of K and thus order information could be exposed. As a result, we do not achieve full privacy on frequency and order information in our work here and we will discuss the details of these privacy issues of the exact answers in Chapter 4.6. The contributions of our work can be summarized as follows.

- The first contribution is a new probabilistic encryption scheme for processing range query search over a confidential numeric attribute K, assuming that the cloud server follows the prescribed protocols but is curious about plaintext contents. The scheme protects the confidential K values and does not preserve the relative order of such values in encrypted records.

- Also, we propose an indexing method to efficiently evaluate range query search over encrypted data. Traditional indexing methods on numerical values are usually based on the order information. However, in our encryption, the order won't be preserved to protect privacy and thus making the traditional indexing methods inapplicable here. The proposed indexing method turns out to be much more efficient than linear scan by our experiment results which yet does not need to preserve the order information.

1.5 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 thoroughly describes the related work in this area of secure query processing. Chapter 3 formulates the problem of processing range query search in a systematic way and defines all the assumptions in our setting as well as the confidential requirements. We also give our system model and the symbol table which includes a convenient reference to all the symbols we use throughout the thesis in this chapter. We give the complete detailed encryption scheme in Chapter 4. The indexing method over the encrypted database is presented in Chapter 5. And the experiment results and analysis will be discussed in Chapter 6. We conclude the thesis with a summary of conclusion and future work in Chapter 7.

Chapter 2

Related Work

Encryption is a perfect way to protect data privacy which also gives rise to the difficulty in executing regular query on such encrypted data. To keep the advantage of computing ability on the server side, it is not reasonable to download the entire encrypted data and execute the query on the data after decryption, the cloud server has to have the ability to execute query directly over the encrypted data and return the correct results to the user.

There have been considerable interests in querying encrypted data and various queries are considered. Briefly, the works include equality test (e.g., [20][36]), range query search (e.g., [6][30][32]), aggregation query computation (e.g., [7][22]) as well as keyword based query search or similarity query search (e.g., [10][35][27]). However, little work on secure processing of range query search without preserving the order information has been reported. The basic challenge comes from the following dilemma: a range query requires comparing the values of the attribute, whereas the privacy model requires hiding the order of such values. Next, we give a detailed description of the related work.

2.1 Database Area

In the database field, typical works include the following. [20][24] proposed to partition the domain of K into coarser buckets and replace exact domain values with their corresponding bucket identifiers. This method suffers from false positives and is vulnerable to linking attacks [13]. The exactly correct answers still need to be found by decrypting the data. In addition, if the data growth concentrates on a small number of buckets, the boundary of such buckets may be re-identified based on the background knowledge on the growth of certain values in real life.

The work in [23] further addresses how to control the amount of false positives by adding a second step called "Controlled-Diffusion" to the bucketization algorithm. While they mainly deal with multidimensional range queries, their redistributed buckets are still fixed there. As a result, the false positives are not diverse at all.

The order preserving encryption scheme in [6][30] with property that $E_k(v_i) \leq E_k(v_j)$ holds for every pair $v_i \leq v_j$ where E_k is the encryption algorithm with secret key k directly discloses the relative order of the confidential attribute in records. This gives the attacker a chance to figure out the order preserved encryption $u_1 < \cdots < u_m$, and in a case where the ordered domain values $v_1 < \cdots < v_m$ are known, it is possible to figure out u_i is the encryption of v_i. And [26] gives a specific example about how to compromise the scheme in [6]. In a word, the order preserving encryption schemes do not meet our confidentiality requirement.

The work in [32] presents a scheme to allow an authorized auditor to retrieve encrypted log records that fall into an interval of time. Their motivation and setting are quite different from ours, and they do not consider the selection attribute as confidential because the order and intervals of such attributes are all known to the sever.

The line of research on aggregation queries, e.g., [7], assumes that the aggregation attribute is sensitive but the selection attribute is not, and employs the homomorphic encryption scheme to compute the aggregation operation [22]. This encryption scheme is not applicable to range query search. In addition, as discussed in [38], the homomorphic encryption in [22] is not fully secure.

In [38], secure computation of k-nearest neighbor queries (kNN) were studied in an outsourced setting. Their technique is based on a notion of distance between data points and the query point, and approximation of such distances. The comparison operation required by range query search does not have a natural notion of distance or approximation.

The MONOMI's design in [36] takes an approach of split client/server execution of queries and considers a larger class of SQL queries. The idea is to execute as much of the query as possible over encrypted data on the server, and execute the remaining components by shipping encrypted data to a trusted client. Their approach relies on various encryption methods for the execution on the server, rather than replacing encryption methods.

An alternative approach is to build a B-tree over plaintext, but encrypt every record and the B-tree at the node level using conventional encryption. The B-tree index is maintained by the user to locate the data of interest [13][33][17][37]. In general, this approach does not provide access confidentiality or pattern confidentiality because it could be observed that an access aims at a specific data and two accesses aim at the same data.

The proposals of oblivious RAM [19] and Private Information Retrieval (PIR) [11][12] [34] aim to address this issue. The idea is that each data read or write request will generate a completely random sequence of data accesses from the server's perspective and they proposed several techniques, namely, cover searches, cached searches, and shuffling. But they incur a heavy workload at the client, communication costs, high computational complexity, or require special-purpose hardware, so they are not applicable to large databases of the cloud scale.

A secure B+-tree index is also used in [37] where a salted IDA encoding scheme and column-access-via-proxy query processing has been processed. To hide access patterns, the client side has to keep the B+tree index to direct the search while the sever stores and retrieves the encrypted data.

2.2 Keyword Based Search

In [16], a simple but effective scheme has been proposed to encrypt a look-up directory consisting of (key, value) pairs. The goal is to allow the corresponding value to be retrieved if and only if a valid key is provided. However, their idea is using an associated one-way hash function mapping the original keys in a deterministic way and as a result it could suffer from frequency analysis attacks if the attackers are armed with background knowledge about the data distribution or frequency information.

In [10], multi-keyword similarity searches over encrypted documents are considered. Both documents and queries are represented as vectors over keywords and the similarity between a document and a query is measured by the inner product between their represented vectors. The query answers are the set of documents that have a similarity score (inner product) above a specified threshold. They further introduce noises into the inner product to resist attacks specific to their schemes while sacrificing the accuracy of the results. We will later on show for our encryption scheme, we also introduce noises but in such a way that the exact answers are not affected (see Chapter 4).

In [35], interesting searchable symmetric key encryption schemes are proposed to support keyword searches over an encrypted text repository such as email messages. However, only equality tests of a single keyword are considered in their work and the time complexity is linear in its repository size per query. In [27], an index based on locality sensitive hashing is proposed to support similarity search for high dimensional data like text data. It is not clear how these techniques can be adapted for range query search in relational databases.

2.3 Others

There are also some more works based on trusted hardware. The recent advent of tamper-resistant trusted hardware [8] advocates a hardware solution. The trusted hardware, which is placed at the server side and is tamper-resistant, has limited capacity in storage and processing power compared to general-purpose hardware, therefore, only operations involving sensitive data are performed by the trusted hardware and the remaining operations are performed by general-purpose hardware.

Not only that the computation power of the cloud server is not fully utilized, but also this partitioning implies that intermediate results will be communicated back and forth between trusted hardware and general-purpose hardware, which could open doors for attacks based on access patterns or make subsequent processing more difficult. For example, before sending the intermediate result of the range query $Salary \geq 200K$ for the sensitive attribute $Salary$ back to general-purpose hardware, it may be necessary to encrypt the non-sensitive attribute $Name$, making it difficult to further process $Name$, such as join by $Name$. Futhermore, as pointed out in [36], such approaches require relatively intrusive changes on behalf of the service provider.

In the field of cryptography, there are mainly two areas: symmetric-key cryptography and public-key cryptography [14]. In symmetric-key cryptography (e.g., Data Encryption Standard (DES) [4], Advanced Encryption Standard (AES) [5], etc.), the same key is used for both encryption and decryption while in public-key cryptography (e.g., RSA [25]), anyone can do encryption using the public key, but only the paired private key (different key) can be used to do decryption.

However, it is not clear at all how these existed cryptography methods such as RSA [25] could be adapted to support direct query search over encrypted data. The essential reason is that these cryptography schemes are originally designed only for the purpose of protection of confidential data rather than supporting query search over encrypted data. As a result, such techniques in cryptography might not be that suitable in our context.

Oracle's Transparent Data Encryption (TDE) [29] considers a different setting from ours. TDE assumes the server is trusted and the adversary could be other malicious ones who try to compromise the server. TDE encrypts data on disks (so even when the encrypted database is stolen by malicious people, it is still protected) but the query evaluation is performed on plaintext data after decryption in the server (i.e., the server is trusted) without impacting existing applications. That is, the server could see the plaintext data which, however, is not acceptable in our model where the cloud server is assumed to be malicious and not trusted and thus not allowed to see plaintext data. In theory, fully homomorphic encryption [18] could be used to evaluate any function over encrypted data. In practice, this construction is prohibitive for databases of the cloud scale, requiring slowdowns on the order of 10^9 times [9].

So far, we have summarized all the typical works in database area, keyword based search and other works or issues including hardware based approaches, techniques in cryptography or homomorphic encryption. As we can see, every approach has its own limits and it seems no perfect approaches exist to fully solve the problem of secure query processing over encrypted data in the cloud. We conclude this chapter here.

Chapter 3

Problem Formulation

In this chapter, we describe the data, queries, parties involved. We give detailed discussion on our model, our goal and confidentiality requirements and the motivation behind them. We first formulate the specific problem (data and queries) we want to solve and then we discuss our setting of cloud computing including the system model. Finally, we give out our confidentiality requirements as well as all notations we use throughout the whole thesis. The details of our encryption scheme will be discussed in next chapter.

3.1 Range Queries

As a big part, the problem we considered here is to support processing range query search over a numeric attribute K for a relational database outsourced to a cloud server. More specifically, we consider a relational table of the form $R(rid, K)$, where K is a numeric attribute and rid is the record id. K is confidential and is the searchable attribute.

For the data, we assume that K has the domain $[\alpha, \beta]$ for some $\beta > \alpha > 0$. $[\alpha, \beta]$ here is not a real number interval but a discrete interval which contains all the actual discrete K values of all the records in the database.

On the other hand, for the query, a range query has a condition $a \leq K \leq b$, where a and b are in $[\alpha, \beta]$. This query retrieves all records having a K value within the range $[a, b]$ exactly (i.e., there are no false positives or false negatives in the returned answers). Note a single side range query ($K \geq a$, $K \leq b$) and exact equality query $K = a$ could all be converted into the general form $a \leq K \leq b$ as shown in Table 3.1.

We clarify the domain $[\alpha, \beta]$ a bit more. Note in the query $a \leq K \leq b$, a, b are assumed

original condition	equivalent form
$K = a$	$a \leq K \leq a$
$K \leq b$	$\alpha \leq K \leq b$
$K \geq a$	$a \leq K \leq \beta$

Table 3.1: The translation from other conditions into the general range query $a \leq K \leq b$

to be in $[\alpha, \beta]$. That is, a, b are actual domain values in the database by the definition of $[\alpha, \beta]$. This is fine because any range query $a' \leq K \leq b'$ where a', b' are not domain values could be equivalently transformed into $a \leq K \leq b$ where a and b are in $[\alpha, \beta]$ and both of them retrieve the same set of records.

In summary, we consider such a problem of processing range query search $a \leq K \leq b$ over a numeric attribute K with domain $[\alpha, \beta]$ for a relational database outsourced to a cloud server where $\beta > \alpha > 0$ and $a, b \in [\alpha, \beta]$.

A categorical attribute K can be treated as a special case for a numeric attribute with the domain $[1, |K|]$, where $|K|$ is the number of distinct values of K, and only queries with a condition $a \leq K \leq a$ are allowed, where a is an integer in the range $[1, |K|]$.

3.2 System Model

Figure 3.1 shows the big picture of the outsourcing model considered in this work, which involves four entities Owner, Users, Proxy, and Server. This is a distributed setting where Server is on the remote side and not trusted. More specifically, Owner collects and owns the data R and has all rights to upload, query and encrypt data, and may also grant the query right to authorized users with access control keys. Users is a group of users authorized to post a query and receive the answers. Owner encrypts the data and then uploads it to the Server (or delegates this task to a trusted Proxy as shown in the figure). Proxy serves a bridge between Users and Server.

A query from Users will go though the trusted Proxy, which encrypts the query and submits the query to Server. Server computes and returns the answer to Proxy. Proxy then decrypts the answer, and returns the answer to Users. For example, Owner is a hospital, who outsources patient records to the cloud, and Users are various medical research labs, who post queries to retrieve patient records of interests.

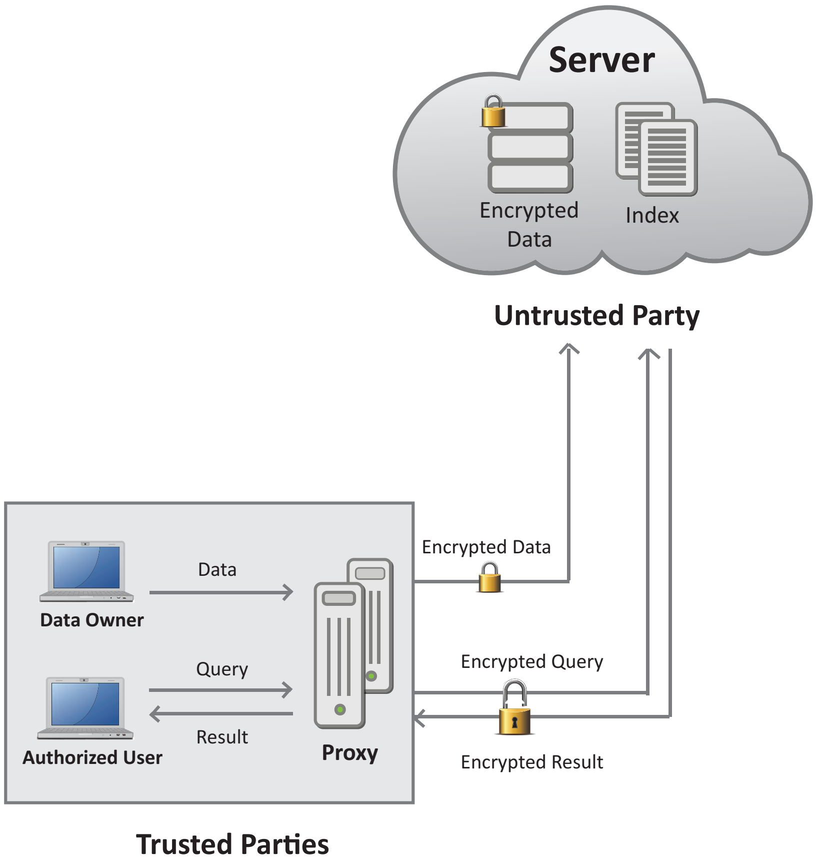

Figure 3.1: System model

We assume an honest but curious cloud server model in our setting. That is, the Server will follow our protocols to do query computation correctly but it is curious about the confidential plaintext values and tries to learn information from everything stored, received, and processed. More specifically, the only assumption is that the Server will perform the query evaluation correctly and return the correct answers. There are no constraints on any other operations the Server could do or could not do. That is, the Server will do the query computation correctly and can do any other operations it wants to do.

In our model, Server is the adversary and it is not allowed to see any plaintext values. Server knows all the distinct plaintext values of K and all the encrypted records and queries. Server can do any operations over things stored, received or processed. In addition, Server has prior background ground knowledge of the frequency information of the K values among the records and queries and knows partial correspondence between some plaintext values and ciptertext values. The only thing Sever does not know and tries to figure out is the complete correspondence between the plaintext values and ciphertext values.

Our system must prevent Server from learning any additional correspondence between plaintext values and ciphertext values except those obtained by prior knowledge. That is, we must protect the plaintext values for any encrypted records or queries from being disclosed to Server. The detailed goals are described as below.

3.3 Our Goal and Confidentiality Requirements

Our goal in this work is to protect the plaintext K values for both records and queries from being disclosed. Specifically, the goal is to protect the complete correspondence between plaintext values and ciphertext values on attribute K from being disclosed to Server except those obtained by Server from its prior background knowledge.

Firstly, Server knows everything stored, received and processed which are directly known. Additionally, as we discussed previously, Server possesses background knowledge which we define as follows: (1) Server knows all the distinct plaintext K values and the frequency of any particular K values among the records and queries; (2) Server knows plaintext values for some ciphertext values (i.e., partial correspondence).

According to how the Server could make use of the background knowledge to compromise plaintext values for any particular encrypted records or queries (i.e., correspondence) we give the following privacy definitions which all aim to contribute to achieve the ultimate goal of protecting the correspondence between plaintext values and ciphertext values. “X privacy” where X could be Secret Key, Frequency (for ciphertext values), or Order (for ciphertext values) in the following is defined as that X is confidential and should not be disclosed. The detailed definitions and discussions are described as follows.

- **Secret Key Privacy**: The secret key is the direct protection on the database and query content. So it is extremely important to protect the secrete key from being compromised. To our knowledge, in our case, the strongest attack to compromise the secret key is to establish a linear system based on the partial correspondence between plaintext values and ciphertext values which Server knows from the background knowledge (known as *known-plaintext attack* in cryptography [14]). So if we achieve the secret key privacy, the other unknown plaintext should still be safe even when partial correspondence is exposed.

- **Frequency Privacy**: This is to protect the original plaintext frequency information among ciphertext values from being disclosed. This is important to help protect the complete correspondence because Server knows the frequency of plaintext values and all the plaintext values. If the original plaintext frequency information among ciphertext values is exposed, Server could identify the plaintext of encrypted values with high probability by identifying those with similar frequency. We call this attack frequency analysis attack. As a result, original plaintext frequency should not be preserved in the ciphertext values, and a probabilistic encryption scheme that encrypts each occurrence of a value probabilistically without preserving the frequency information would provide a stronger protection.

- **Order Privacy**: This is to protect the order information for ciphertext values from being disclosed. Since Server knows all the plaintext K values, it of course knows the order (i.e., $v_1 < \cdots < v_m$). Once the order information for ciphertext values (i.e., ordered encrypted values $u_1 < \cdots < u_m$) is exposed, it is known that u_i is the encryption of v_i. As a result, it is necessary for us to protect the relative order information and an encryption scheme which does not preserve the order information gives stronger protection.

Note the "Frequency Privacy" and "Order Privacy" are important to protect but also difficult to fully achieve if we want to guarantee the exactness of the returned answers. As we mentioned previously in Chapter 1.4, the exactness of the returned answers could disclose the frequency and order information. As long as we guarantee the exact answers, we cannot achieve full frequency and order privacy. More details will be discussed in Chapter 4.

In summary, in our work here, we try to achieve the goal to protect the complete correspondence between plaintext values and ciphertext values. That is, Server could not learn any additional plaintext values for any particular encrypted records or queries except the partial correspondence known by Server from prior background knowledge. To achieve this goal, we define more specific requirements (i.e., Secret Key Privacy, Frequency Privacy and Order Privacy) based on what the Server could do using its background knowledge. We will discuss these again and what we actually achieve in more detail in the next chapter.

3.4 Symbol Table

As shown in Table 3.2, we give a summary of all the frequently used notations or symbols in this thesis.

Symbol	Definition
R	The relational database table $R(rid, K, A)$
K	The searchable numerical attribute in the table R
$[\alpha, \beta]$	The domain for the attribute K
$[a, b]$	The query interval in the range query $a \leq K \leq b$
p	p is a column vector representing the encoded form of a plaintext record in the database table
q	q is a column vector representing the encoded form of a plaintext query $a \leq K \leq b$
$e(p)$	The encrypted form of the plaintext record p
$e(q)$	The encrypted form of the plaintext query q
$\varepsilon_i^1, \cdots, \varepsilon_i^U$	$\varepsilon_i^t, t = 1, \cdots, U$ are randomly chosen from $(0, 1]$ and appended to each record vector as extended dimensions
$f^1, \cdots, f^U$	$f^t, t = 1, \cdots, U$ are randomly chosen and determined in Theorem 4.4.1
C_q	$C_q = r_q(1 + \frac{a}{b})$ is the encrypted form of the given value to the server
U	The number of extended positions in record and query vector
S	$S = (s_1, \cdots, s_{2+U})$ is the split indicator vector
s	The number of non-zero bits in $S = (s_1, \cdots, s_{2+U})$
ℓ	The security parameter defined as the number of dimensions of an encrypted record
M	The $\ell \times \ell$ invertible transformation matrix
SK	$SK = \{S, M\}$ is the secret key consisting of the split indicator vector S and the transformation matrix M
T_q	$T_q = (e(q), C_q)$ which is the trapdoor of query q
R_e	The encrypted form of the plaintext relational table $R(rid, K, A)$
Ans_e	The set of the returned answer records in encrypted form

Table 3.2: The symbols we use in this thesis

Chapter 4

A New Encryption Scheme

In this chapter, we present a new encryption scheme for processing range query search over the confidential numeric attribute K. The idea is to encrypt the attribute K in both data and queries in such a way that the query condition can be tested in the encrypted form of data and queries. There are totally five building blocks as follows in our scheme.

4.1 Building Blocks

- $\mathsf{Setup}(1^{\ell})$: Taking a security parameter ℓ as input, Owner outputs a secret key as SK.
- $\mathsf{Build}(R,\ SK)$: Based on the dataset R, Owner builds a searchable encrypted form of R, R_e, then outsources it to Server.
- $\mathsf{Trapdoor}(q)$: With a query q as input, Proxy runs this algorithm to generate a corresponding encrypted query, T_q, called the trapdoor.
- $\mathsf{Query}(T_q,\ R_e)$: Server runs this algorithm to perform the encrypted query T_q on the encrypted data R_e, and return the encrypted answer to Proxy.
- $\mathsf{Decrypt}(Ans_e, SK)$: Proxy runs this algorithm to decrypt the answer Ans_e received using the key SK and return the plaintext answer to Users.

Before going into the details of these building blocks, we summarize the high level ideas of our scheme. First, we encode a record p and a query q as column vectors such that p satisfies q if and only if $p^T \cdot q \leq 1 + \frac{a}{b}$ holds, where p^T is the transpose of p, $p^T \cdot q$ is the inner product of p and q, and $[a, b]$ is the query interval. This test in plaintext does not provide confidentiality. We then transform these vectors into encrypted vectors $e(p)$ and $e(q)$ such that $p^T \cdot q \leq 1 + \frac{a}{b}$ holds if and only if $e(p)^T \cdot e(q) \leq C_q$ holds, where C_q is an encrypted value representing the query q and we will show soon that C_q is random such that it is different even for the same repeated query q. Next, we first present the vector encoding scheme for p and q, then present the encryption scheme for computing $e(p)$ and $e(q)$.

4.2 Encoding Records and Queries

We adopt the following encoding scheme for records and queries. For each record p with the value v on K, we encode it as a vector $p = (\frac{1}{v}, v)$. For a query q with the condition $a \leq K \leq b$, we encode it as a vector $q = (a, \frac{1}{b})$. All vectors are column vectors. Table 4.1 shows a concrete example for encoding of the attribute K. And for example, a range query $5 \leq$ bank account balance ≤ 8 would be encoded as a column vector $q = (5, \frac{1}{8})$.

K = bank account balance (10^3)	Encoded Form
5	$(1/5, 5)$
10	$(1/10, 10)$
8	$(1/8, 8)$

Table 4.1: Example of encoding: the relational table R with K as bank account balance

Therefore, we have the following equation:

$$p^T \cdot q = \frac{a}{v} + \frac{v}{b} \tag{4.1}$$

Let $y = \frac{a}{v} + \frac{v}{b}$ and Figure 4.1 shows an example of the plot of the function $y = \frac{2}{v} + \frac{v}{8}$. It can be seen that, as v increases, y first decreases, reaches the minimum value 1 at $v = 4$, and then increases.

Generally, we have the same pattern, that is, for $y = \frac{a}{v} + \frac{v}{b}$, as v increases, y first decreases, reaches the minimum value $2\sqrt{a/b}$ at $v = \sqrt{ab}$, and then increases. At $v = a$ and $v = b$, $y = 1 + \frac{a}{b}$. Therefore, for $a \leq v \leq b$, $y \leq 1 + \frac{a}{b}$, and all other values of v, $y > 1 + \frac{a}{b}$.

Such pattern is very important. Firstly, it gives us a sufficient and necessary condition to test if $a \leq v \leq b$ (we will show this more specifically soon). Secondly, the non-monotone pattern (y decrease first and then increase) prevents the inference of the relative order of

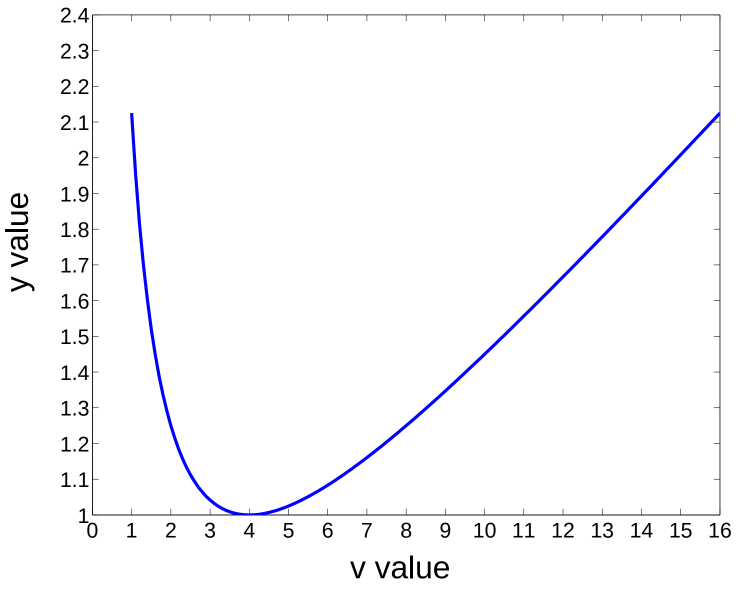

Figure 4.1: Shape of the function $y = \frac{2}{v} + \frac{v}{8}$

the K value of records (we will see more details in Chapter 4.6). Thus, we give a formal proof for this property in Lemma 4.2.1.

Lemma 4.2.1 *For $b \geq a > 0$ and $v > 0$, as v increases, function $y = \frac{a}{v} + \frac{v}{b}$ first decreases, reaches the minimum value $2\sqrt{a/b}$ at $v = \sqrt{ab}$, and then increases. At $v = a$ and $v = b$, $y = 1 + \frac{a}{b}$.*

Proof: First, we take derivative of the function y, we have

$$f'(v) = \frac{dy}{dv} = \frac{-a}{v^2} + \frac{1}{b}$$

We then take the second derivative, we have

$$f''(v) = \frac{df'(x)}{dv} = \frac{2a}{v^3}$$

Since $b \geq a > 0$ and $v > 0$, then $f''(v) = \frac{2a}{v^3} > 0$, as a result, $f'(v)$ is strictly increasing as v increases. We know at $v = \sqrt{ab}$, $f'(v) = 0$. Then we have when $0 < v < \sqrt{ab}$, $f'(v) < 0$, thus, y is decreasing as v increases, when $v > \sqrt{ab}$, $f'(v) > 0$, thus, y is increasing as v increases. So we have proved that as v increases, function $y = \frac{a}{v} + \frac{v}{b}$ first decreases, reaches the minimum value at $v = \sqrt{ab}$, and then increases. The minimum value at $v = \sqrt{ab}$ is

$2\sqrt{a/b}$ and at $v = a$ and $v = b$, $y = 1 + \frac{a}{b}$ as one can easily compute the values. We finish the proof here. $\square$

The following fact and lemma follow from Lemma 4.2.1.

Fact 1: (1) For $b \geq a > 0$ and $v > 0$, $p^T \cdot q \leq 1 + \frac{a}{b}$ if and only if $a \leq v \leq b$. (2) For $a \leq v \leq b$, as v increases, $p^T \cdot q$ first decreases, reaches the minimum $2\sqrt{a/b}$ at $v = \sqrt{ab}$, then increases.

Fact 1(2) implies that $p^T \cdot q$ is non-monotone within the query range $[a, b]$. As we will see in Chapter 4.6, this property prevents the inference of the relative order of the K value of records.

Lemma 4.2.2 *A record p is in the answer to a query q if and only if $p^T \cdot q \leq 1 + \frac{a}{b}$.*

4.3 Encrypting Records and Queries

Lemma 4.2.2 tests whether p satisfies q in plaintext. To preserve privacy, this test must be done in an encrypted form. We present our encryption scheme below. Let p and q be the encoded vectors defined above.

4.3.1 Setup Secret Key

Our secret key consists of a split indicator vector S and a transformation invertible matrix M. Basically, the split indicator vector S is used to help break the correspondence between the plaintext and ciphertext values and the transformation matrix M is used to transform the record vector from plaintext space into the encrypted space. We now describe the detailed way of how we construct the secret key.

$\mathsf{Setup}(1^\ell)$. $\ell > 2$ is the security parameter and is defined as the number of dimensions of an encrypted record. Owner randomly picks an integer $U > 0$ in the range $[\lceil \frac{\ell}{2} \rceil - 2, \ell - 2]$ and let $s = \ell - U - 2$, and randomly generates a split indicator vector

$$S = (s_1, \cdots, s_{2+U})$$

where $s_i \in \{0, 1, -1\}$, such that there are exactly s non-zero s_i's. Here, $s_i = 0$ indicates we do not perform split on the dimension i, $s_i = 1/-1$ means we perform split (in different ways for -1 and 1, details could be found in $\mathsf{Build}(R, SK)$). Finally, Owner generates a $\ell \times \ell$ invertible matrix M. $SK = \{S, M\}$ is the secret key for encryption. Note that $\{U, s\}$

is a random (integer) partition of $\ell - 2$. More detailed discussion on the purpose of S, M could be found in Chapter 4.6.

The reason for $U \in [\lceil \frac{\ell}{2} \rceil - 2, \ell - 2]$ is by the constraint of $s \in [0, 2 + U]$ which results from the definition for s to be the non-zero bits in the vector $S = (s_1, \cdots, s_{2+U})$ of length $2 + U$, so U can't be too small or too large, otherwise, it will lead to invalid secret key and make the encryption impossible. U cannot be too large because $U = \ell - 2 - s \leq \ell - 2$, U cannot be too small because $s = \ell - 2 - U \leq 2 + U$, so $U \geq \frac{\ell}{2} - 2 \geq \lceil \frac{\ell}{2} \rceil - 2$, and we will see soon that U actually is the number of extended positions appended to p, for privacy reason, we require $U > 0$ and we will discuss this shortly in Chapter 4.6.

To make the process more clear, let's consider a concrete example of choosing $\ell = 10$. That is, the dimension of the encrypted record should be 10, and we could pick any $U \in [3, 8]$, but if we choose U out of $[3, 8]$ such as $U = 2$, then $s = \ell - 2 - U = 6$, it is not possible since the maximum value for s is $2 + U = 4$ (the reason is by definition, s is the non-zero bits in the vector $S = (s_1, \cdots, s_{2+U})$).

4.3.2 Data Encryption

As discussed before, a probabilistic encryption scheme has to be adopted. We achieve such a probabilistic encryption by injecting randomness into each record vector p_i such that every record (including those having the same K values) looks different. More specifically, the encryption process is as follows.

$\mathsf{Build}(R, SK)$. For each record $p_i = (\frac{1}{v_i}, v_i)$ (with value v_i on K and note p_i is in the vector form by the encoding scheme presented previously rather than a single original v_i value) in the database R, the ciphertext of p_i, $e(p_i)$, is produced in three steps. Step 1, extend p_i into $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)$ with $\varepsilon_i^t, t = 1, \cdots, U$ randomly chosen from $(0, 1]$. The purpose of $\varepsilon_i^t, t = 1, \cdots, U$ is to inject randomness and make each record different in encrypted form. More importantly, $\varepsilon_i^t, t = 1, \cdots, U$ introduces randomness into the inner product result while such randomness does not affect the correctness of the answer (Theorem 4.4.1). As we will see shortly in **Fact 2**, the split in next step 2 does not influence the inner product computation result, and thus, $\varepsilon_i^t, t = 1, \cdots, U$ is necessary to help prevent Server from identifying same records or repeated queries (see more details in Chapter 4.6).

Step 2, split $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)$ into (p', p''), where p' is a length $2 + U$ vector and p'' is a length s vector: if the jth bit of S is 0, $p'[j] = (p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)[j]$; if the jth bit of S is 1 and is the j'th non-zero in S, $p'[j]$ and $p''[j']$ are a random split of $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)[j]$, i.e.,

$p'[j] + p''[j'] = (p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)[j]$; if the jth bit of S is -1 and is the j'th non-zero in S, $p'[j]$ and $p''[j']$ are equal to $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)[j]$. As discussed in $\mathsf{Setup}(1^\ell)$, this split step is useful for breaking the correspondence between the plaintext and ciphertext values, we give the details of how it helps in Chapter 4.6.

Since step 2 is a bit complicated, let's consider the following concrete example to clarify the process so far. Suppose we select the security parameter as $\ell = 5$ and pick $U = 1, s = 2$, and then we generate $S = (1, 0, -1)$ and M as a $\ell \times \ell$ (i.e., 5×5) invertible matrix. And we take the first record in Table 4.1 as our $p = (\frac{1}{5}, 5) = (0.2, 5)$. In step 1, $p = (\frac{1}{5}, 5) = (0.2, 5)$ would be extended by $U = 1$ more dimension as $p = (0.2, 5, 0.7)$ (note the 0.7 on the last dimension is the random number generated corresponding to $\varepsilon_i^1 \in (0, 1]$).

In step 2, we need to split $p = (0.2, 5, 0.7)$ into (p', p'') where p' is a vector of length $2 + U = 3$ and p'' is a length $s = 2$ vector. Then we determine the dimension values of p', p'' according to $S = (1, 0, -1)$. $S[1] = 1$, it is non-zero and is the 1st non-zero bit in S, then $p'[1], p''[1]$ need to be decided and they are a random split of $p[1] = 0.2$. Let's say $p'[1] = -2.7, p''[1] = 2.9$. By random split, it means as long as we keep $p'[1] + p''[1] = p[1]$ here, then it is fine. Then we have $S[2] = 0$, since it is zero, we leave $p'[2] = p[2] = 5$ and don't need to take care of p''. Lastly, $S[3] = -1$, and it is the 2nd non-zero bit in S, then $p'[3], p''[2]$ need to be decided and they are set to the same value as $p'[3] = p''[2] = p[3] = 0.7$. Then the record $p = (0.2, 5, 0.7)$ will be split into $(p', p'') = (-2.7, 5, 0.7, 2.9, 0.7)$. We now talk about the last step.

Step 3, compute the encrypted record for p_i as $e(p_i) = M^T \cdot (p', p'')$. Note that $e(p_i)$ is a ℓ-dimensional column-vector. Let R_e denote the set of all encrypted records $e(p_i)$ for the records p_i in R. To finish the example, we will multiply $(p', p'') = (-2.7, 5, 0.7, 2.9, 0.7)$ with the 5×5 transformation matrix M. Note after split, now the record becomes of length 5, and it is valid to do multiplication between the 5×5 invertible matrix M and $(p', p'') = (-2.7, 5, 0.7, 2.9, 0.7)$.

After all the three steps of $\mathsf{Build}(R, SK)$, the encryption extends p_i with $U + s$ new dimensions, U dimensions for random values ε_i and s dimensions for random splits. Since $\{U, s\}$ is a random split of $\ell - 2$, Server does not know exactly how many of the added $U + s$ dimensions are due to the random split (i.e., s). We will discuss more details on the purpose of adding these dimensions in Chapter 4.6.

4.3.3 Query Encryption

The query encryption (i.e., $\mathsf{Trapdoor}(q)$) here will generate the trapdoor of a query q which involves two parts. The encrypted vector $e(q)$ of the same length as the encrypted record vector $e(p_i)$ in $\mathsf{Build}(R, SK)$ and $C_q = r_q(1 + \frac{a}{b})$ which is the encrypted form of $1 + \frac{a}{b}$. The detailed process is as follows.

$\mathsf{Trapdoor}(q)$. This function produces the trapdoor of a query q: Randomly choose a positive number $r_q > 0$ which is used to hide the plaintext inner product $1 + \frac{a}{b}$ in Equation (4.3) and $p_i^T \cdot q$ in Lemma 4.3.1. Let $f^1, \cdots, f^U$ be positive values (i.e., $f^t > 0, t = 1, \cdots, U$) determined later (i.e., Theorem 4.4.1). $f^1, \cdots, f^U$ serves two purposes, one is to inject randomness into the encryption form of queries such that even identical queries have different encrypted form. The other is together with $\varepsilon_i^t, t = 1, \cdots, U$, to introduce randomness into the inner product result as in Lemma 4.3.1 and more importantly, we can control $f^1, \cdots, f^U$ such that the randomness does not affect the correctness of the returned answer (Theorem 4.4.1).

Next, we split the vector $(r_q q, -r_q f^1, \cdots, -r_q f^U)$ into (q', q'') exactly as for $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)$ (see the concrete example in $\mathsf{Build}(R, SK)$ for further reference), except for using S', where S' is the 1/-1 complement of S, that is, if the jth bit of S is 1, the jth bit of S' is -1, and if the jth bit of S is -1, the jth bit of S' is 1. All zeros in S are preserved in S'. The split for the query here corresponds to the split in data records but in a complement way by using a complement split indicator vector S' instead of S. Such a way of split in query and records not only injects randomness into the encryption but also could recover $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)^T \cdot (r_q q, -r_q f^1, \cdots, -r_q f^U)$ which will be shown in **Fact 2**.

Finally, we compute the encryption of q as $e(q) = M^{-1} \cdot (q', q'')$, where M^{-1} is the inverse of M. $e(q)$ is a ℓ-dimensional vector. Return the trapdoor

$$T_q = (e(q), C_q) \tag{4.2}$$

where

$$C_q = r_q(1 + \frac{a}{b}) \tag{4.3}$$

Since r_q is randomly chosen for each query, C_q does not reveal $\frac{a}{b}$.

Fact 2: $(p', p'')^T \cdot (q', q'') = (p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)^T \cdot (r_q q, -r_q f^1, \cdots, -r_q f^U) = r_q({p_i}^T \cdot q - \sum_{t=1}^{U} f^t \varepsilon_i^t)$.

The second equality follows trivially. The first equality follows from our construction of (p', p'') and (q', q''). To see this, let x denote $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)$ and let y denote $(r_q q, -r_q f^1, \cdots, -r_q f^U)$. And we want to verify $(p', p'')^T \cdot (q', q'') = x^T \cdot y$ in Fact 2.

If the jth bit of S is 0, $p'[j] = x[j]$ and $q'[j] = y[j]$. So $x[j]y[j] = p'[j]q'[j]$.

If the jth bit of S is 1 and is the j'th non-zero in S, $p'[j] + p''[j'] = x[j]$ and $q'[j] = q''[j'] = y[j]$, so $x[j]y[j] = (p'[j] + p''[j'])y[j] = p'[j]y[j] + p''[j']y[j] = p'[j]q'[j] + p''[j']q''[j']$. For a similar reason, if the jth bit of S is -1 and is the j'th non-zero in S, the equality $x[j]y[j] = p'[j]q'[j] + p''[j']q''[j']$ also holds.

Note x, y, p', q', S are of length $2+U$ while p'', q'' are of length s. By the above discussion, we have $x[j]y[j] = p'[j]q'[j]$ when $S[j] = 0$ and $x[j]y[j] = p'[j]q'[j] + p''[j']q''[j']$ when $S[j] \neq 0$. Thus,

$x^T \cdot y$
$= \sum_{j=1}^{2+U} x[j]y[j]$
$= \sum_{j,S[j]=0} x[j]y[j] + \sum_{j,S[j]\neq 0} x[j]y[j]$
$= \sum_{j,S[j]=0} p'[j]q'[j] + \sum_{j,S[j]\neq 0} p'[j]q'[j] + p''[j']q''[j']$
$= \sum_{j,S[j]=0} p'[j]q'[j] + \sum_{j,S[j]\neq 0} p'[j]q'[j] + \sum_{j,S[j]\neq 0} p''[j']q''[j']$
$= \sum_{j,S[j]=0} p'[j]q'[j] + \sum_{j,S[j]\neq 0} p'[j]q'[j] + \sum_{j'=1}^{s} p''[j']q''[j']$
$= \sum_{j=1}^{2+U} p'[j]q'[j] + \sum_{j'=1}^{s} p''[j']q''[j']$
$= (p', p'')^T \cdot (q', q'')$.

The next Lemma 4.3.1 follows from Fact 2.

Lemma 4.3.1 *For a query q and a record p_i, $e(p_i)^T \cdot e(q) = r_q({p_i}^T \cdot q - \sum_{t=1}^{U} f^t \varepsilon_i^t)$.*

Proof: $e(p_i)^T \cdot e(q)$
$= \left(M^T \cdot (p', p'')\right)^T \cdot (M^{-1} \cdot (q', q''))$
$= (p', p'')^T \cdot M \cdot M^{-1} \cdot (q', q'')$
$= (p', p'')^T \cdot (q', q'')$
$= r_q({p_i}^T \cdot q - \sum_{t=1}^{U} f^t \varepsilon_i^t)$ (Fact 2). $\square$

4.4 Computing the Query

Based on the relationship between $e(p_i)^T \cdot e(q)$ and ${p_i}^T \cdot q$ in Lemma 4.3.1, we can derive the encrypted test condition for the query.

Query(T_q, R_e) returns the encrypted records $e(p_i)$ in R_e such that $e(p_i)^T \cdot e(q) \leq C_q$, where $(e(q), C_q)$ is the trapdoor in Equation (4.2). Lemma 4.3.1 suggests that a sufficiently large $\sum_{t=1}^{U} f^t \varepsilon_i^t$ always leads to $e(p_i)^T \cdot e(q) \leq C_q$, which means that all records $e(p_i)$ will be returned. To ensure that the set of returned records is exactly the answer to the query, we must choose proper values for $f^1, \cdots, f^U$. Let $f = \sum_{t=1}^{U} f^t \varepsilon_i^t$. We first identify the range for f such that $e(p_i)^T \cdot e(q) \leq C_q$ if and only if p_i is the answer to the query q. We then determine the value for each $f^t > 0, t = 1, \cdots, U$ such that f is in this range.

Recall that $[\alpha, \beta]$ is the domain range of K containing all the discrete values for K. Consider any domain value c for K. Let c^+ denote the smallest domain value greater than c, and c^- denotes the largest domain value less than c. If c is the smallest domain value α, let c^- be $\alpha/2$, and if c is the largest domain value β, let c^+ be $\beta + 1$. Note that $\alpha/2$ and $\beta + 1$ are not domain values. We define

$$f^* = min\{\frac{a}{b^+} + \frac{b^+}{b} - 1 - \frac{a}{b}, \frac{a}{a^-} + \frac{a^-}{b} - 1 - \frac{a}{b}\}$$

Note f^* is positive by the above definition. This is because by Lemma 4.2.1 and the definition of a^-, b^+, we have that for $y = \frac{a}{v} + \frac{v}{b}$, when v ranges from a^- to a, y is decreasing, that is, $\frac{a}{a^-} + \frac{a^-}{b} > \frac{a}{a} + \frac{a}{b} = 1 + \frac{a}{b}$, so we have $\frac{a}{a^-} + \frac{a^-}{b} - 1 - \frac{a}{b} > 0$. Similarly, when v ranges from b to b^+, y is increasing (Lemma 4.2.1). So we also have $\frac{a}{b^+} + \frac{b^+}{b} > \frac{a}{b} + \frac{b}{b} = 1 + \frac{a}{b}$, that is, $\frac{a}{b^+} + \frac{b^+}{b} - 1 - \frac{a}{b} > 0$. As a result, $f^* > 0$. And we claim:

Theorem 4.4.1 *For any f such that $0 < f < f^*$, p_i is in the answer to q if and only if $e(p_i)^T \cdot e(q) \leq C_q$; therefore, Query($T_q$, R_e) computes the exact answer for q.*

Proof: According to Lemma 4.3.1, $e(p_i)^T \cdot e(q) \leq C_q \Leftrightarrow r_q({p_i}^T \cdot q - f) \leq r_q(1 + \frac{a}{b}) \Leftrightarrow {p_i}^T \cdot q - f \leq 1 + \frac{a}{b}$. That is, $e(p_i)^T \cdot e(q) \leq C_q$ holds if and only if $p_i^T \cdot q - f \leq 1 + \frac{a}{b}$. With Lemma 4.2.2, p_i is in the answer to q if and only if $p_i^T \cdot q \leq 1 + \frac{a}{b}$. So as long as we show that $p_i^T \cdot q \leq 1 + \frac{a}{b}$ holds if and only if $p_i^T \cdot q - f \leq 1 + \frac{a}{b}$ holds, then we are done.

"Only if" is trivial. Since we have $f = \sum_{t=1}^{U} f^t \varepsilon_i^t$ by definition and f^t, ε_i^t are all defined as positive numbers. So $f > 0$. That is, because $f > 0$, if $p_i^T \cdot q \leq 1 + \frac{a}{b}$, we must have $p_i^T \cdot q - f \leq 1 + \frac{a}{b}$.

For "If", suppose $p_i^T \cdot q - f \leq 1 + \frac{a}{b}$, we need to show $p_i^T \cdot q \leq 1 + \frac{a}{b}$. The condition $0 < f < f^*$ implies

$$\frac{a}{b^+} + \frac{b^+}{b} - f > 1 + \frac{a}{b} \tag{4.4}$$

$$\frac{a}{a^-} + \frac{a^-}{b} - f > 1 + \frac{a}{b} \tag{4.5}$$

Let $y(v) = \frac{a}{v} + \frac{v}{b}$. From Lemma 4.2.1, $y(b^+)$ is the minimum value for any domain value $v > b$ and $y(a^-)$ is the minimum value for any domain value $v < a$. Therefore, if $v > b$, $y(v) \geq y(b^+)$, thus, $y(v) - f \geq y(b^+) - f > 1 + \frac{a}{b}$ (Equation (4.4)). But this contradicts the assumption that $p_i^T \cdot q - f \leq 1 + \frac{a}{b}$. Therefore, $v \leq b$. Similarly, if $v < a$, $y(v) \geq y(a^-)$, thus, $y(v) - f \geq y(a^-) - f > 1 + \frac{a}{b}$ (Equation (4.5)). This, however, contradicts the assumption that $p_i^T \cdot q - f \leq 1 + \frac{a}{b}$ again. So we must have $a \leq v \leq b$ under the assumption that $p_i^T \cdot q - f \leq 1 + \frac{a}{b}$. Then Fact 1 implies $p_i^T \cdot q \leq 1 + \frac{a}{b}$ as required. □

Finally, to determine $f^t > 0, t = 1, \cdots, U$, such that $0 < \sum_{t=1}^{U} f^t \varepsilon_i^t < f^*$, it suffices to have $0 < \sum_{t=1}^{U} f^t < f^*$ because $\varepsilon_i^t \in (0, 1]$. Any f^t such that $0 < f^t < \frac{f^*}{U}$, $t = 1, \cdots, U$, serves this purpose. Alternatively, $f^t > 0, t = 1, \cdots, U$ can be chosen differently for each query, as long as these inequalities hold.

4.5 Decryption

Basically, the decryption is just a reversed process of encryption, the detailed process is described as follows.

Decrypt(Ans_e, SK). A record $e(p_i)$ in Ans_e is decrypted by computing $(M^T)^{-1} \cdot e(p_i)$, where $e(p_i) = M^T \cdot (p', p'')$, and reconstructing p_i from (p', p'') using S. Note this is just for the K in $R(rid, K, A)$, to decrypt part A associated with $e(p_i)$ we just use the conventional decryption approach according to what we use to encrypt A.

4.6 Analysis

Let us analyze the privacy properties of the proposed encryption scheme and the query computation given by Theorem 4.4.1. Server knows $e(p_i)$, the trapdoor $T_q = (e(q), C_q)$, and the security parameter ℓ and all these are directly known. In addition, Server possesses background knowledge and knows all the distinct plaintext K values and the frequency of any particular K value in the database and the queries. Server also knows partial correspondence between the plaintext values and ciphertext values. However, Server does not know the secret key $SK = \{S, M\}$ where M is a ℓ-dimensional invertible matrix. And we try to achieve the goal to protect the complete correspondence between plaintext values and ciphertext values from being disclosed by achieving the following specific privacy. We now discuss what we actually achieve as follows.

4.6.1 Secret Key Privacy

To our knowledge, the known-plaintext attack is the strongest attack to compromise the secret key based on what Server knows in our assumption. So we first focus on analyzing this attack. In this attack, Server tries to derive the secret key M by establishing enough number of equations $M^T \cdot p_i = e(p_i)$. Since M is a $\ell \times \ell$ invertible matrix. As long as Server obtains ℓ such equations which are linearly independent with each other, Server could compromise M. That is, suppose there are ℓ such linear independent $M^T \cdot p_i = e(p_i), i = 1, \cdots, \ell$. Let $B_1 = (p_1, \cdots, p_\ell), B_2 = (e(p_1), \cdots, e(p_\ell))$, then the linear system $M^T B_1 = B_2$ is known, and $M^T = B_2 B_1^{-1}$.

In a simple case (e.g., the encryption is just a matrix transformation by multiplying a plaintext vector p_i with the matrix M^T and there is no extension or split) where p_i is the plaintext encoding vector $(\frac{1}{v}, v)$, Server is able to establish the linear system based on the partial correspondence which Server already knows (i.e., Server knows v for p_i and the corresponding $e(p_i)$).

However, in our case, this is not straightforward. More specifically, in our case, the ciphertext $e(p_i) = M^T \cdot (p', p'')$ of a record p_i is constructed from a random split $\{p', p''\}$ of $(p_i, \varepsilon_i^1, \cdots, \varepsilon_i^U)$, determined by the split indictor vector S. Similarly, $e(q) = M^{-1} \cdot (q', q'')$ for a query q is constructed from a random split $\{q', q''\}$ of $(r_q q, -r_q f^1, \cdots, r_q f^U)$.

Without knowing the exact random splits, i.e., $\{p', p''\}$ and $\{q', q''\}$, Server could not establish the linear system. And for the split indictor vector S, not only Server does not know exactly how many splits (value of s) we perform, but also, Server is not able to know whether it is a random split on record or on query (i.e., non-zero bit in S could be either -1 or 1). Please refer to Theorem 6 in [38] for more discussion which gives a formal proof on such a splitting scheme could resist the known-plaintext attack.

4.6.2 Frequency Privacy

Frequency privacy is protected by the random values $\varepsilon_i^t, t = 1, \cdots, U$ for a record p_i and the random value $r_q, f^t, t = 1, \cdots, U$ for a query q. Note that these values are chosen randomly for each *occurrence* of a record or query, thus, even duplicate records or repeated query requests get encrypted differently. Such probabilistic encryption is highly effective to resist frequency analysis attacks where the attacker tries to identify corresponding plaintext values for certain ciphertext values by linking the same frequency statistics obtained from background knowledge. In addition, the random splits of dimensions $p_i[j]$ or $q[j]$ mentioned above further contribute to the probabilistic nature of ciphertext values.

Lemma 4.3.1 also implies that two identical records p_1 and p_2 do not necessarily yield the same inner products $e(p_1)^T \cdot e(q)$ and $e(p_2)^T \cdot e(q)$ even for the same query q, thanks to the random values $\varepsilon_i^t, t = 1, \cdots, U$ introduced to records. Let's see in a bit more detail the importance to introduce $\varepsilon_i^t, t = 1, \cdots, U$ into records and $f^t, t = 1, \cdots, U$ into queries.

For two duplicated record p_1, p_2 (i.e., they are identical on attribute K), without using $f^t, t = 1, \cdots, U$ (and $\varepsilon_i^t, t = 1, \cdots, U$), from Lemma 4.3.1, $e(p_1)^T \cdot e(q)$ and $e(p_2)^T \cdot e(q)$ are identical for every query q, therefore, Server could infer p_1 and p_2 are identical. With the use of $f^t, t = 1, \cdots, U$ (correspondingly, $\varepsilon_i^t, t = 1, \cdots, U$), even if p_1 and p_2 are identical, the inner products $e(p_1)^T \cdot e(q)$ and $e(p_2)^T \cdot e(q)$ are guaranteed to be different if $\varepsilon_i, t = 1, \cdots, U$ is randomly chosen for each p_i and thus they are different from each other. This is how $f^t, t = 1, \cdots, U$ and $\varepsilon_i^t, t = 1, \cdots, U$ prevent inferring whether two records are same based on their inner products with the same query.

4.6.3 Order Privacy

Firstly, from a direct point of view, the encryption of a record p, $e(p) = M^T \cdot (p', p'')$, goes through the random split (p', p'') and the multiplication by the matrix M, which aggregates over multiple dimensions, including random values ε_i's. As a result, the order of the K value in a record p is randomized in $e(p)$. That is, original order in not preserved under our encryption scheme.

Furthermore, inferring the relative order of two records p_1 and p_2 by comparing $e(p_1)^T \cdot e(q)$ and $e(p_2)^T \cdot e(q)$ for the same query q does not help here. From Lemma 4.3.1, this is equivalent to comparing $r_q({p_1}^T \cdot q - \sum_{t=1}^{U} f^t \varepsilon_1^t)$ and $r_q({p_2}^T \cdot q - \sum_{t=1}^{U} f^t \varepsilon_2^t)$. From Fact 1(2), ${p_i}^T \cdot q$ is non-monotone, which means that a large ${p_i}^T \cdot q$ could be produced by either a large K value of p_i or a small K value of p_i.

4.6.4 Additional Analysis and Summarization

So far, we assume that exact answers must returned. The reason is that if more data than asked is returned, the user will know more than allowed and if less data than asked is returned, the user will not get complete answers which are not useful. There are additional privacy leakages due to the exactness of the returned answers. However, this does not only apply to our particular scheme here but applies to all the schemes which guarantee the exactness of the returned answers. We analyze the privacy concerns regarding the exact answers as follows.

The first concern here is that a solution that always returns the exact answers may open a door to frequency analysis attack mentioned above because Server may identify repeating queries based on the identity of the results. This is not difficult to understand, if the answers are exactly the same (even in encrypted form, Server could always tell whether the answer A_1 to q_1 is the same as another answer A_2 to q_2), then Server could conclude that these two queries q_1 and q_2 are the same query with a high probability.

Secondly, a more tricky issue is that an exact answer for a range query has the property that all records in the answer are consecutive in the order of K. Server could use this property to infer the relative order of K in records. For example, if one query retrieves $\{X, Y, W\}$ and another query retrieves $\{F, X, Y\}$, then X and Y must be consecutive in the order of K. Why? Because if X and Y are not consecutive, then it must be true that F is in the middle of X and Y since $\{F, X, Y\}$ is returned, and in any other answer, if X, Y are returned at the same time, then F must also be returned due to the consecutive property. However, the returned answer $\{X, Y, W\}$ does not include F. So we could conclude that X, Y must be consecutive in the order of K.

After the answers for a large number of queries are observed, Server may learn a long chain of consecutive records by eliminating permutations that are inconsistent among query answers. Once the relative order of two records on this chain is known, so is the relative order of all records on the chain.

In a context where incorrect answer (e.g., false positives, false negatives) are allowed, possible ways to combat the above attacks could be generating false positive and/or false negative records into the answers in a controlled manner. The purpose is to destroy the above "consecutive property" and the exactness. We consider this as our future work.

Lastly, we give a summary of our analysis here. Essentially, the goal we try to achieve is to protect the complete correspondence between plaintext values and ciphertext values from being disclosed. That is, Server cannot identify plaintext values for any particular ciphertext values except those Server obtained from prior background knowledge. Based on what Server knows, we try to analyze all the possible attacks Server could do and check whether our scheme can resist against these attacks or not (i.e., the three privacy aspects we discussed previously).

What we actually achieve is this: our scheme can resist against all the specific attacks as we discussed previously. Other potential attacks may or may not apply but for now, to our knowledge, it is not clear how other attacks would apply. We did not achieve our goal fully because the exactness of the returned answers may cause additional privacy leakages. We are making progress regarding existed works which directly preserve the order information. We conclude this chapter here.

Chapter 5

Indexing Method

In this chapter, we propose an efficient indexing method to support faster query evaluation than the trivial linear scan manner. Until now, the query answer is computed by evaluating the condition $e(p_i)^T \cdot e(q) \leq C_q$ in Theorem 4.4.1 for every encrypted data point $e(p_i)$ in the database. This linear scan is not acceptable for large databases. We adopt the *ball tree* index in [31] to retrieve such points while pruning as many data points as possible. Briefly, a ball tree is a binary tree such that each non-leaf node represents a ball and has two child nodes. A data point belonging to a parent goes to the child ball whose center is closer to the data point. All data points are only stored at the leaf nodes. To build such a ball tree, we could keep separating the data point space into two partitions (left and right child) recursively until the number of data points in some partition is below a predefined threshold and we make this partition as a leaf node. We call this threshold as "max leaf size" and we will test it in our experiment chapter. The detailed algorithm to construct the ball tree (i.e., how to encrypted records into balls) could be found in [31] and we don't bother to bring the exactly same algorithm here.

Algorithm 1 gives the outline of our branch-and-bound pruning search based on such a ball tree index just described. Let's explain how exactly this algorithm works. Essentially, we need to retrieve all data points $e(p_i)$ whose inner product with $e(q)$ is no more than C_q. For a query q, we traverse the ball tree in a depth-first branch-and-bound manner. At each node N, two bounds are estimated using Theorem 5.0.1 below: $UB(e(q), N)$ is defined as the maximum possible inner product between $e(q)$ and any point inside the node N, and $LB(e(q), N)$ is defined as the minimum possible inner product between $e(q)$ and any point inside the node N. Then we have the following:

Algorithm 1 BallTreeSearch(Query $e(q)$, TreeNode N)

```
1: if $C_q < LB(e(q), N)$ then
2:    The subtree rooted at $N$ will be pruned
3: else if $C_q \geq UB(e(q), N)$ then
4:    Return all the data points in the leaf
      nodes of the subtree rooted at $N$
5: else
6:    if isLeaf($N$) then
7:       LinearSearch($e(q), N$)
8:    else
9:       BallTreeSearch($e(q)$, $N$.leftChild);
10:      BallTreeSearch($e(q)$, $N$.rightChild);
11:   end if
12: end if
```

If $C_q < LB(e(q), N)$, the subtree rooted at N is pruned because even the minimum possible inner product $LB(e(q), N)$ is larger than C_q, then none of the data points in the leaf nodes of the subtree rooted at N would have an inner product with $e(q)$ to be no more than C_q.

If $C_q \geq UB(e(q), N)$, all the data points in the subtree rooted at N are returned. This is because even the maximum possible inner product $UB(e(q), N)$ is less than or equal to C_q, then it is sure that all of the data points in the leaf nodes of the subtree rooted at N would have an inner product with $e(q)$ to be no more than C_q. Note this is also a way to achieve the same effect as pruning in a sense that we don't have to linear scan each data point.

Otherwise in the third case, if N is a leaf node, we evaluate $e(p_i)^T \cdot e(q) \leq C_q$ for every data point $e(p_i)$ in N; if N is not a leaf node, we examine the two child nodes of N recursively.

Furthermore, the ball tree index can be implemented on disk storage by allocating the nodes to disk pages in the depth-first order of the tree, where several adjacent nodes in the order can be allocated to the same disk page. This makes sense because the nodes are examined in the depth-first order of the tree nodes. The branch-and-bound operation can be supported by having each node pointing to its sibling node, so the subtree of the node can be skipped by following this pointer in the depth-first traversal.

The only remaining thing is to find the exact values for $LB(e(q), N), UB(e(q), N)$. We give Theorem 5.0.1 as follows. And from now on we don't particularly distinguish a point

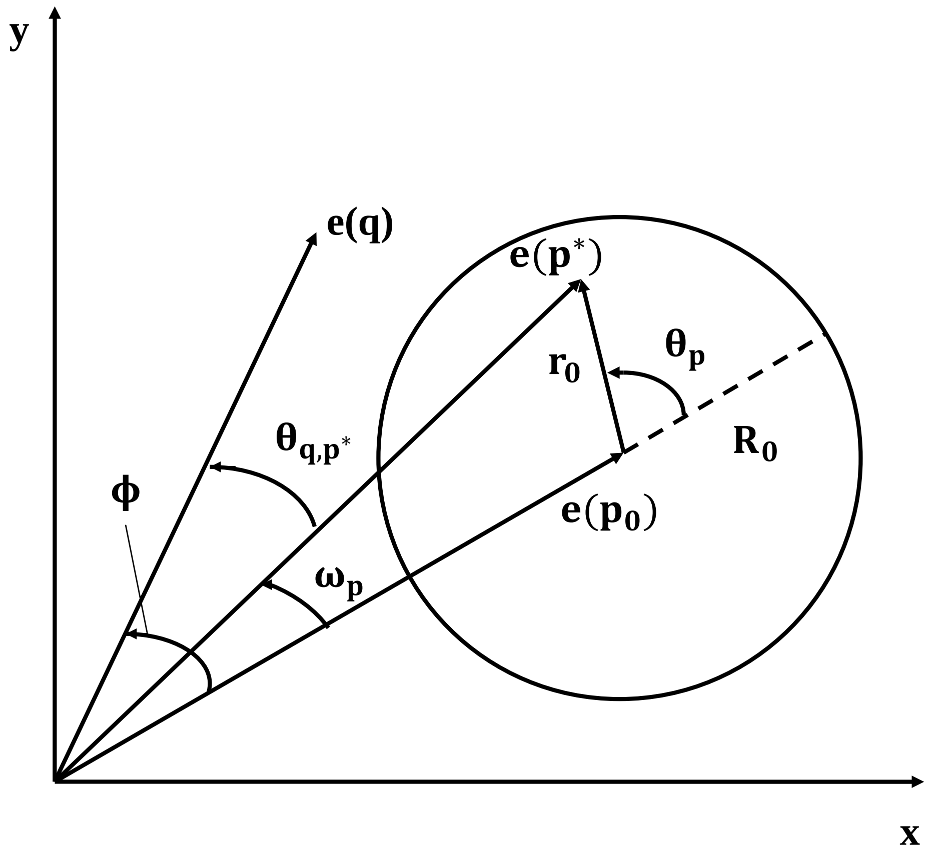

Figure 5.1: Plot example for ball node (a circle) in 2-dimensional space

from a vector which starts from the origin when it does not cause any ambiguity. That is, we will call an encrypted record $e(p)$ either a vector or a point in the encrypted space. We give the proof for $LB(e(q), N)$ in Theorem 5.0.1 while the proof for $UB(e(q), N)$ could be found in [31].

Theorem 5.0.1 *Given a ball tree node N with the center $e(p_0)$ and the radius R_0, and an encrypted query $e(q)$,*

$$UB(e(q), N) = e(p_0)^T \cdot e(q) + R_0|e(q)|$$
$$LB(e(q), N) = e(p_0)^T \cdot e(q) - R_0|e(q)|$$

Proof: As shown in Figure 5.1, suppose that $e(p^*)$ is the vector(point) in the ball node N for the query vector $e(q)$ with minimum inner product and r_0 be the Euclidean distance between the ball center $e(p_0)$ and $e(p^*)$ (by definition, $r_0 \leq R_0$ and note everything is in the encrypted space). Let θ_p be the angle between the vector $e(p_0)$ and the vector $e(p^*) - e(p_0)$, the one starting from point $e(p_0)$ and end at $e(p^*)$, ϕ and ω_p be the angles between the vector $e(p_0)$ and vectors $e(q)$ and $e(p^*)$ respectively (refer to the Figure 5.1).

The angle ω_p can be expressed in terms of p_0 and θ_p as:

$$\cos\omega_p = \frac{|e(p_0)| + r_0\cos\theta_p}{|e(p^*)|}, \sin\omega_p = \frac{r_0\sin\theta_p}{|e(p^*)|}. \tag{5.1}$$

Let θ_{q,p^*} be the angle between the vectors $e(q)$ and $e(p^*)$. With the triangle inequality of angles, we have:

$$|\theta_{q,p*}| \leq |\phi + \omega_p| \tag{5.2}$$

We use the range $[-\pi, \pi]$ as the angle system for all the angle calculation and thus $0 \leq |\theta_{q,p*}| \leq \pi, 0 \leq |\phi + \omega_p| \leq \pi$. We then get:

$$\cos\theta_{q,p*} = \cos|\theta_{q,p*}| \geq \cos|\phi + \omega_p| = \cos(\phi + \omega_p) \tag{5.3}$$

We derive the lower bound as follows:

$$\begin{aligned}
LB(e(q), N) &= \min_{e(p)\in N}\{e(p)^T \cdot e(q)\} \\
&= |e(q)||e(p^*)|\cos\theta_{q,p*} \\
&\geq |e(q)||e(p^*)|cos(\phi + \omega_p) \quad (by\ (5.3)) \\
&= |e(q)||e(p^*)|(\cos\phi\cos\omega_p - \sin\phi\sin\omega_p) \quad (expand\ cos(\phi + \omega_p)) \\
&= |e(q)|(\cos\phi(|e(p^*)|\cos\omega_p) - \sin\phi(|e(p^*)|\sin\omega_p))
\end{aligned}$$

$$
\begin{aligned}
&= |e(q)|(\cos\phi(|e(p_0)| + r_0\cos\theta_p) - \sin\phi(r_0\sin\theta_p)) \quad (by\ (5.1))\\
&\geq |e(q)|\min_{\theta_p}\{\cos\phi(|e(p_0)| + r_0\cos\theta_p) - \sin\phi(r_0\sin\theta_p)\}\\
&= |e(q)|\min_{\theta_p}(|e(p_0)|\cos\phi + r_0(\cos\phi\cos\theta_p - \sin\phi\sin\theta_p))\\
&= |e(q)|\min_{\theta_p}(|e(p_0)|\cos\phi + r_0\cos(\phi+\theta_p))\\
&= |e(q)|(|e(p_0)|\cos\phi - r_0)\\
&\geq |e(q)|(|e(p_0)|\cos\phi - R_0)\\
&= e(p_0)^T \cdot e(q) - R_0|e(q)|
\end{aligned}
$$

That is, the minimum possible inner product between $e(q)$ and any point inside the node N is larger than or equal to $e(p_0)^T \cdot e(q) - R_0|e(q)|$. Thus, we set the lower bound as $LB(e(q), N) = e(p_0)^T \cdot e(q) - R_0|e(q)|$. The proof for the upper bound $UB(e(q), N)$ is similar and could be found in [31]. □

More discussion. Although the ball tree is defined as a binary tree and we also adopt a binary ball tree in the experiments, the ball tree could actually be a more general tree with the number of children of each node being between M_1 and M_2 ($2 \leq M_1 < M_2$). Because the branch-and-bound pruning in a depth first search order does not depend on the number of children of each node, as long as it is tree, such branch-and-bound pruning in depth first search order can work properly.

The advantage of such a more general ball tree is that such a tree could be short and thus could be more efficient when implemented on disks. We could construct such a general ball tree by adopting a similar recursive partition procedure as for binary ball tree. To construct M ($M_1 \leq M \leq M_2$) children of a parent node, we keep partitioning the points in the parent node by performing $M-1$ binary partitions (for the detailed binary partition algorithm, please see [31]). Note the first binary partition splits the parent node into two children. The second binary partition splits one of these two children and adds one more child to the parent node. So every binary partition adds one more child and $M-1$ partitions will split the parent node into M child nodes. Such partitioning will be stopped when the number of points within a node is below a threshold we set (i.e., maximum leaf size). We bring this up just for discussion purpose and this is not the focus of our work here. In this thesis, we simply adopt a binary ball tree as our index.

Chapter 6

Experiments

In this chapter, we report our studies on the feasibility of the encryption schemes proposed in Chapter 4 and the performance of the indexing method in Chapter 5. On one hand, we investigate the performance of the ball tree index including its pruning power measured by examined portion (test ratio) and the real effectiveness measured by average query execution time. We will also compare the average query execution time by ball tree index with the query execution time by linear scan approach.

On the other hand, we intend to give a thorough study and analysis over the encryption scheme including the evaluation on storage cost, encryption cost, query execution cost as well as the communication cost. For the communication cost, it is of no particular interest because our encryption scheme return exact answers and thus has the minimum communication cost among the solutions that guarantee the completeness of answers because all such solutions return all correct records. While for the other costs, we give all concrete experiment results and analysis. That is, storage cost would be measured by the disk space used to store the encrypted database under our encryption schemes. And both encryption cost and query execution cost are measured by the time consumption to perform these operations.

6.1 Experimental Setup

Here we first describe the dataset and the queries we use. And then we also discuss the specific parameters we choose to run the experiments and the reason why we choose some specific ones rather than others. We discuss how and why we design certain experiments as well.

6.1.1 General Setup

For the dataset, we utilized the 2000 Brazil CENSUS data[1] for our studies. This contains 10M records over 9 attributes summarized in Table 6.1. We choose AGE as the attribute K for range queries and we created three samples of sizes 1M, 5M, and 10M in records. The purpose of creating different databases is to study how the query execution time changes as the database size increases.

For the query, we generated a pool of range queries over K of varied selectivity by sampling 50 queries uniformly at random from those with selectivity less than 5%, where the selectivity of a query is defined as the percentage of records that are retrieved by the query. Such queries model real life queries that usually retrieve a small fraction of the whole database. The reported query processing time is the average time of these 50 queries. We implemented these schemes in C++ and run a Intel(R) Xeon(R) CPU 2.53 GHZ PC with 12GB of RAM as Server.

Attributes	Sizes	Attributes	Sizes
STATE	26	AGE	101
SEX	2	MARITAL STATUS	4
STATE OF BIRTH	30	RACE	6
EDUCATION	32	OCCUPATION	511
CLASS OF WORKER	4		

Table 6.1: Attributes and domain sizes

6.1.2 Specific Design

We now discuss the specific parameters for our encryption scheme. We consider four security parameter settings: $\ell = 3, 4, 5, 6$. Recall that $\ell = U + 2 + s$. Since the variance between U and s does not make a big difference here for the same ℓ, we set $U = 1$, so $s = 0, 1, 2, 3$ corresponding to $\ell = 3, 4, 5, 6$, respectively. We say that an encryption is *s-split* if S has s non-zero bits.

Then, the first set of test is to test the performance of the ball tree index over different parameters. This is in the first place because we need to choose a best ball tree to do

[1]`https://international.ipums.org/international/`

comparison with the linear scan approach in next step. The only adjustable parameter which affects the performance of the ball tree is the threshold, max leaf size we defined in Chapter 5 which is the maximum number of points a leaf node could contain.

So we test different performance of the ball tree index by setting different max leaf size as 3, 5, 10, 20, 40. And we basically want to measure two things, one is the pruning power measured by test ratio defined as the percentage of records in the database that are actually tested against the query condition. The other is the real effectiveness of the ball tree measured by average query execution time. The database size does not particularly reflect anything about the ball tree performance, so we will do experiment to test ball tree on the 1M database and use the 50 queries we generated. We test all of the four s-split $s = 0, 1, 2, 3$ encryptions.

Next, we discuss more settings for the experiments on storage cost, encryption cost, query execution cost of the encryption scheme. For the encryption cost, we would like to evaluate total time spent to encrypt the whole databases and the query. Also we want to test the disk space cost for the storage cost evaluation. For these two cost evaluation we collect statistics over the three different databases (1M, 5M, 10M) to see how it changes as the database sizes increases. And we keep the average time to encrypt one query as the measurement of query encryption cost. Here what we need is just a taste on whether the cost to perform our encryption scheme on modern computers is acceptable. So for this second set of test *Encryption time and space overhead*, we just adopt the 3-split encryption with the setting of $\ell = 6, U = 1, s = 3$ which is expected to be the most time consuming and have most space overhead.

The final third set of test *Query execution time* is about the query execution cost. By only checking the linear scan query execution time cost is not that interesting, so we combine the best ball tree index test here to do comparison as we previously discussed. We use the thee encrypted databases to test the average query execution time over the 50 generated queries respectively. And we adopt all the four s-split ($s = 0, 1, 2, 3$) encryptions to check how the query execution time changes. The max leaf size of the ball tree here would be set as 10 since later on we will see that this max leaf size turn out to be the best one.

6.2 Results and Analysis

We collect experiment results and summarize the analysis here.

6.2.1 Ball Tree Performance

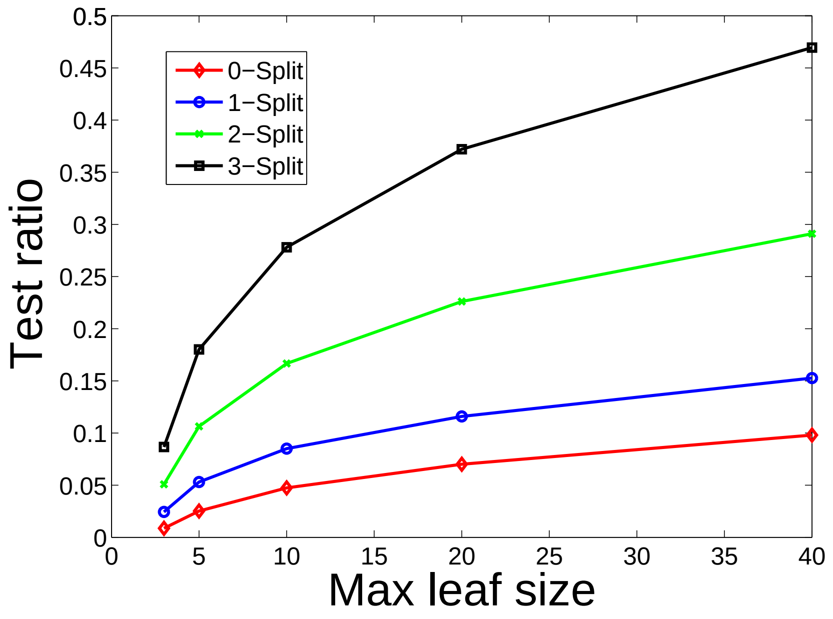

Figure 6.1: Test ratio of the ball tree index for 1M records

Figure 6.1 examines the *test ratio* of the ball tree index. As previously defined, it is the percentage of records in the database that are actually tested against the query condition. As the maximum leaf size increases, the test ratio increases because a larger leaf ball is more likely to intersect with the region of the query, in which case all records in the leaf ball will be tested against the query condition. In addition, as we do more splits in the encryption, the test ratio increases as well. This trend is consistent with the increase in query time in Figure 6.3. The reason is that more splits means a higher dimensionality of the encrypted K, which reduces the effectiveness of the index.

However, as we can see from Figure 6.2, it is not always true that the smaller max leaf size is, the better the real performance (average query execution time) is. This is because the real average query execution time consists of two parts. The first part is the time spent on testing the lower and upper bound $UB(e(q), N), LB(e(q), N)$ of the inner product for each node N and the second part is the time spent on tests of the inner product between each individual record and the query. Smaller max leaf size leads to less tests for the second part but also leads to a deeper ball tree containing more nodes and thus leads to more tests

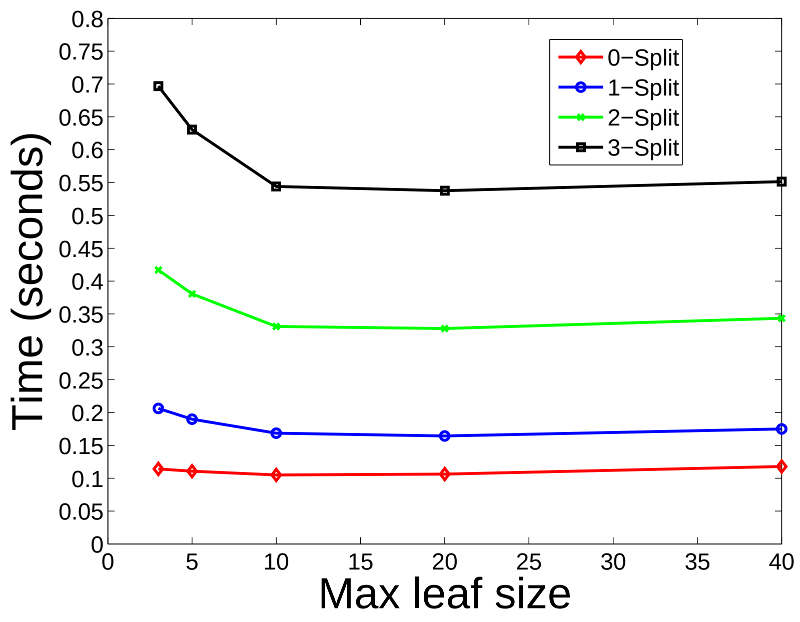

Figure 6.2: Average query execution time of the ball tree index for 1M records

for the first part. So there is a trade-off. Considering both the test ratio and the query execution time results, we think max leaf size = 10 gives a best ball tree which will be used in comparison with linear scan approach in the following experiments.

6.2.2 Encryption Time and Space Overhead for K

In Table 6.2, the last row summarizes the time for 3-split encryption and the first three rows in Table 6.2 show the space overhead of 3-split encryption. The construction time is considered efficient because encryption is done off line and only once. The storage for K after encryption jumps 6 times because the ciphertext is in the space of 6 dimensions, compared to the single dimension K. This ratio remains constant as the database size increases. In general, s-split encryption for a larger s has a higher level of security but at the cost of more storage space. The average time to encrypt one query is 1.76×10^{-5} seconds. These results are quite acceptable for the contemporary computer systems.

Database size (M records)	1	5	10
Original storage for K (MB)	8.3	41.8	83.9
Encrypted storage for K (MB)	51	253	505
Encryption Time (seconds)	5.69	28.48	56.93

Table 6.2: Encryption time and space overhead for K and 3-split encryption

6.2.3 Query Execution Time

This refers to the time for Server to compute the answer to a received query. Table 6.3 collects the average query execution time over the plaintext database as the baseline for query execution time.

Database size (M records)	1	5	10
Query Execution Time (seconds)	0.051	0.253	0.52

Table 6.3: Average query execution time over plaintext database

Figure 6.3 compares the query time for the ball tree index method and the linear scan. Firstly, compared with the baseline, the encryption adds more cost to execute the query. Secondly, as the data size increases, the query time for both methods increases linearly, but the time for the linear scan increases much faster. The query time for 0-split or 1-split is significantly smaller than for 2-split and 3-split. This trend is intuitive because more splits lead to a higher dimensionality of encrypted K, which causes the branch-and-bound strategy of the ball tree index to become less effective. Overall, the indexing method is significantly more efficient than the linear scan.

In summary, we could see from all these experiment results, that less split encryption leads to lower dimensionality of encrypted data, thus, more efficient encryption and query computation, but introduces less randomness in the encryption. This reflects the trade-off between privacy and efficiency. If a high level of privacy is required, the linear scan might be the only choice. In this case, partitioning and distributing the database (on Server) onto multiple machines for parallel linear scans is a promising approach.

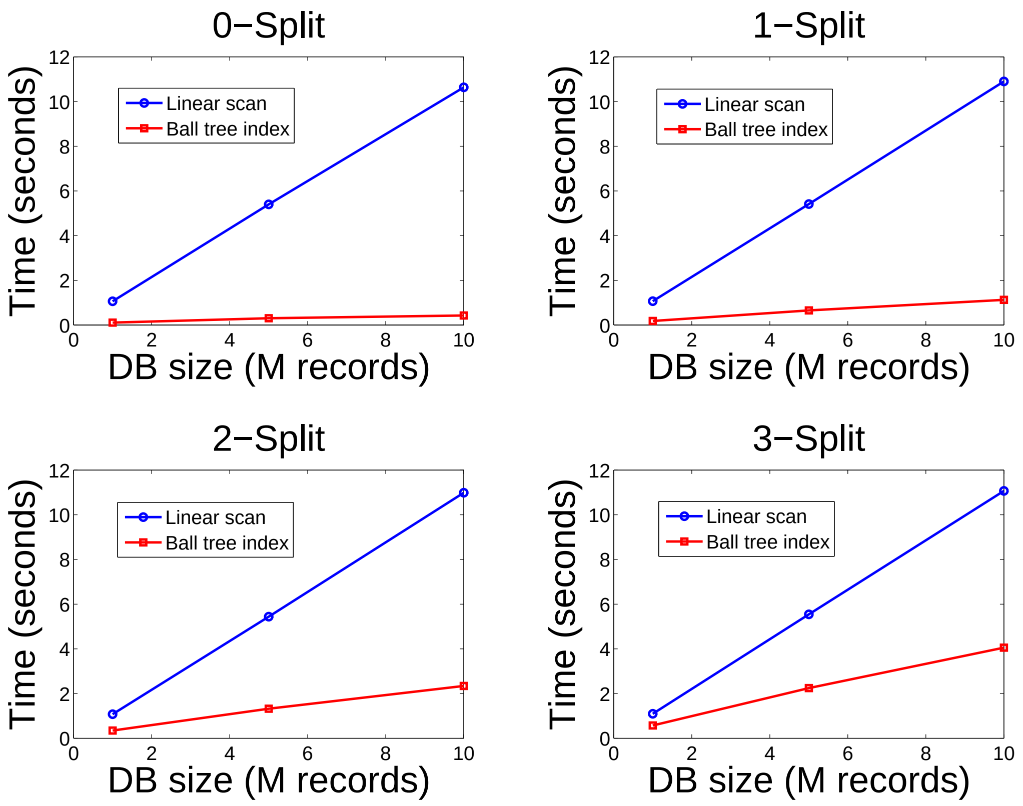

Figure 6.3: Average query execution time for max leaf size 10

Chapter 7

Conclusion

7.1 Summary

A major challenge or concern in the cloud computing paradigm is data privacy. On one hand, there is a demand to leverage the powerful resources of the cloud server to provide services to clients. On the other hand, the cloud server must not learn any sensitive information about the data being managed and the queries being answered. We consider the service of answering the class of range query search over numerical data and propose a data encryption scheme to address these requirements.

We adopt a strong privacy model where the relative order of K values is considered confidential. This problem is challenging essentially because of the following dilemma: a range query requires comparing the values of K, whereas the privacy model requires hiding the order of such values. We propose a probabilistic encryption scheme which does not preserve the order information to address the challenges. We also enable faster search by applying an effective ball tree indexing method directly over encrypted data. Therefore, it is suitable for large scalable applications.

One limitation of our scheme is that the exactness of the returned answers could cause additional privacy leakages and this applies to all the schemes which guarantee the exactness of the returned answers.

7.2 Future Work

Currently in this thesis, we only consider privacy preserving range query search over a single attribute. The first initial step towards the future work could be considering a more complicated and challenging case to enable range query search over multiple attributes (e.g. bank account balance $\leq 10K$ and/or salary $\leq 5K$). Secondly, range query is only a subset of all the operations supported by database management systems although it is a big part. More other spaces could be explored to support more sql queries or operations such as aggregation query, join, group by and so on.

Additionally, because the exactness of the returned answers may open doors for the cloud server to derive useful information, in the future, we could consider this issue in a bit more detail and give a solution to resolve the issue in a different context where false positives or false negatives are allowed in the returned answers.

Bibliography

[1] Amazon web services (aws), http://aws.amazon.com/.

[2] Google app engine, http://code.google.com/appengine/.

[3] Microsoft azure, http://www.microsoft.com/azure/.

[4] Data Encryption Standard (DES). *National Institute of Standards and Technology (NIST)*, 1999.

[5] Advanced Encryption Standard (AES). *National Institute of Standards and Technology (NIST)*, 2001.

[6] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *SIGMOD*, pages 563–574, 2004.

[7] Aderemi A Atayero and Oluwaseyi Feyisetan. Security issues in cloud computing: The potentials of homomorphic encryption. *Journal of Emerging Trends in Computing and Information Sciences*, 2(10):546–552, 2011.

[8] Sumeet Bajaj and Radu Sion. Trusteddb: A trusted hardware based outsourced database engine. *SIGMOD*, 2011.

[9] N. P. Smart C. Gentry, S. Halevi. Homomorphic evaluation of the aes circuit. In *Cryptology ePrint Archive, Report 2012/099*, 2012.

[10] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *INFOCOM*, pages 829–837. IEEE, 2011.

[11] Benny Chor and Niv Gilboa. Computationally private information retrieval. In *ACM symposium on Theory of computing*, pages 304–313, 1997.

[12] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *JACM*, 45(6):965–981, 1998.

[13] Ernesto Damiani, SDCD Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *10th ACM conference on Computer and communications security*, pages 93–102, 2003.

[14] Hans Delfs and Helmut Knebl. *Introduction to cryptography.* 2002.

[15] Fatih Emekci, Divyakant Agrawal, Amr El Abbadi, and Aziz Gulbeden. Privacy preserving query processing using third parties. In *ICDE*, pages 27–27, 2006.

[16] Joan Feigenbaum, Mark Y Liberman, and Rebecca N Wright. Cryptographic protection of databases and software. *Distributed Computing and Cryptography*, 2:161–172, 1991.

[17] Tingjian Ge and Stan Zdonik. Fast, secure encryption for indexing in a column-oriented dbms. In *ICDE*, pages 676–685, 2007.

[18] C. Gentry. Fully homomorphic encryption using ideal lattice. In *STOC*, 2009.

[19] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *JACM)*, 43(3):431–473, 1996.

[20] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, pages 216–227, 2002.

[21] Hakan Hacigümüş, Bala Iyer, and Sharad Mehrotra. Providing database as a service. In *ICDE*, pages 29–38, 2002.

[22] Hakan Hacıgümüş, Bala Iyer, and Sharad Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Database Systems for Advanced Applications*, pages 125–136, 2004.

[23] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. Secure multidimensional range queries over outsourced data. *VLDB Journal*, 21(3):333–358, 2012.

[24] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *VLDB*, pages 720–731, 2004.

[25] Jakob Jonsson and Burt Kaliski. Public-key cryptography standards (pkcs)# 1: Rsa cryptography specifications version 2.1. 2003.

[26] Murat Kantarcıoglu and Chris Clifton. Security issues in querying encrypted data. In *Data and Applications Security XIX*, pages 325–337. 2005.

[27] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. Efficient similarity search over encrypted data. In *ICDE*, pages 1156–1167, 2012.

[28] Peter Mell and Timothy Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

[29] Arup Nanda. Transparent data encryption. *Oracle Magazine*, 2005.

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

[31] Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *SIGKDD*, pages 931–939. ACM, 2012.

[32] Elaine Shi, John Bethencourt, T-HH Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

[33] Erez Shmueli, Ronen Waisenberg, Yuval Elovici, and Ehud Gudes. Designing secure indexes for encrypted databases. In *Data and Applications Security XIX*, pages 54–68. 2005.

[34] Radu Sion and Bogdan Carbunar. On the computational practicality of private information retrieval. In *Network and Distributed Systems Security Symposium*, pages 2006–06, 2007.

[35] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[36] Samuel Madden Nickolai Zeldovich Stephen Tu, M. Frans Kaashoek. Processing analytical queries over encrypted data. In *VLDB*, 2013.

[37] Shiyuan Wang, Divyakant Agrawal, and Amr El Abbadi. A comprehensive framework for secure query processing on relational data in the cloud. In *VLDB Workshp on Secure Data Management*, pages 52–69. 2011.

[38] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *SIGMOD*, pages 139–152, 2009.