View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Sabanci University Research Database

HIDING QUERY ACCESS PATTERNS IN RANGE QUERIES USING PRIVATE INFORMATION RETRIEVAL AND OBLIVIOUS RAM

by

GAMZE TİLLEM

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

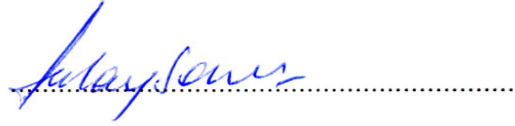
Sabancı University

August 2015

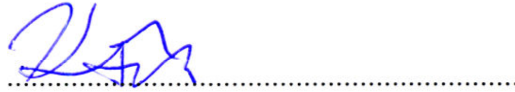
HIDING QUERY ACCESS PATTERNS IN RANGE QUERIES USING PRIVATE INFORMATION RETRIEVAL AND OBLIVIOUS RAM

APPROVED BY:

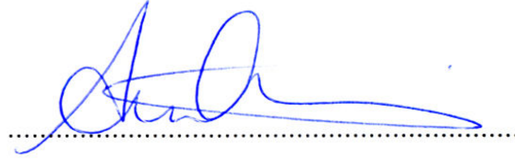
Prof. Dr. Erkay Savaş
(Thesis Supervisor)



Asst. Prof. Dr. Kamer Kaya



Asst. Prof. Dr. Ahmet Onur Durahim



DATE OF APPROVAL: 04/08/2015

© Gamze Tillem 2015

All Rights Reserved

HIDING QUERY ACCESS PATTERNS IN RANGE QUERIES USING PRIVATE INFORMATION RETRIEVAL AND OBLIVIOUS RAM

Gamze Tillem

Computer Science and Engineering, Master's Thesis, 2015

Thesis Supervisor: Erkay Savaş

Abstract

This work addresses the problem of hiding query access patterns in privacy-preserving range queries while guaranteeing data and query confidentiality. We propose two methods, which are based on Private Information Retrieval (PIR) and Oblivious RAM (ORAM) techniques, respectively. For the PIR based search operation, we introduce a new scheme based on Lipmaa's computationally-private information retrieval (CPIR) method. We reduce the computation cost of CPIR by reducing the number of modular exponentiation operations, employing shallow trees and utilizing multi-exponentiation techniques. Furthermore, we improved the performance of CPIR by applying parallel algorithms. For the ORAM based method, we adapted Stefanov's Path ORAM method to the privacy-preserving range search. Our analyses show that, in terms of communication cost, CPIR provides better bandwidth usage especially in large database sizes, while in computational cost, Path ORAM based method performs better due to the negligible cost of server operations. The results imply that, despite some advantageous qualitative aspects of CPIR and its highly parallel implementation, it is still an expensive scheme in terms of computation complexity in comparison with Path ORAM for hiding query access patterns in privacy preserving range queries.

MAHREMİYET KORUMALI ERİM SORGULARINDA MAHREMİYET KORUMALI BİLGİ ERİŞİMİ VE İLGİSİZ BELLEK KULLANARAK SORGU ERİŞİM ÖRÜNTÜSÜNÜN GİZLENİMİ

Gamze Tillem

Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans, 2015

Tez Danışmanı: Erkay Savaş

Özet

Bu çalışma mahremiyet korumalı erim sorgulamalarında veri ve sorgu gizliliğinin yanı sıra, sorgunun erişim örüntüsünün gizlenmesi sorununu işlemektedir. Bu soruna çözüm olarak Mahremiyet Korumalı Bilgi Erişimi (PIR) ve İlgisiz Bellek (ORAM) tekniklerine dayalı iki farklı yöntem önerilmiştir. PIR'e dayalı mahremiyet korumalı erim sorguları için Lipmaa'nın daha önce sunmuş olduğu hesaba dayalı PIR (CPIR) yöntemi üzerinden yeni bir CPIR yöntemi sunulmaktadır. Özgün yöntemin hesaplama süresini düşürmek amacıyla, yeni CPIR yönteminde toplam modüler üst alma işlemi sayısının azaltımı, daha az derinlikli agaçların kullanımı ve üst hesaplamaları için eş zamanlı çoklu üst alma algoritmalarının kullanımı önerilmiştir. Dahası, koşut algoritmalar kullanılarak yeni CPIR yönteminin hesaplama süreleri iyileştirilmiştir. ORAM'a dayalı mahremiyet korumalı erim sorgulama tekniğinde ise Stefanov'un daha önce sunmuş olduğu Path ORAM yöntemi erim sorgularına uyarlanmıştır. Çözümleme sonuçları göstermektedir ki, iletişim maliyeti göz önünde bulundurulduğunda, PIR yöntemi özellikle büyük veritabanlarında daha düşük ağ kullanımı sağlamaktadır. Öte yandan, hesaplama maliyetleri düşünüldüğünde, sunucu tarafındaki maliyetin göz ardı edilebilir olmasından dolayı ORAM temelli yöntem daha iyi sonuçlar sunmaktadır. Bu sonuçlardan yola çıkarak, mahremiyet korumalı erim sorgularında sorgu erişim örüntüsünün gizlenmesinde,nitel açıdan yararları olmasına rağmen, hesaplama maliyetleri açısından CPIR yönteminin ORAM yöntemine göre daha pahalı olduğu söylenebilir.

Acknowledgements

First of all, I would like to thank to my thesis advisor Prof. Dr. Erkay Savaş for his support and guidance throughout my graduate education. His motivation and immense knowledge helped me to complete my research and to write this thesis. Besides my advisor, I would like to thank my thesis jury, Asst. Prof. Dr. Kamer Kaya and Asst. Prof. Dr. Ahmet Onur Durahim for their valuable suggestions. I am extremely grateful for the good advice and help of Asst. Prof. Dr. Kamer Kaya.

My sincere thanks goes to all the members of the Cryptography and Information Security Lab for the great environment they provided me in terms of both research and friendship. Especially, Naim Alperen Pulur, Dilara Akdoğan and Berkay Dinçer deserve infinite thanks for their support and motivation during our undergraduate and graduate studies. Likewise, I have to acknowledge Ecem Ünal for the support she provided whenever I need her.

I also want to mention my gratitude to my friends Ömer Faruk Koru and Melike Kocacık for encouraging me in all parts of my graduate education. Besides, I am grateful to Merve Beydemir, Kardelen Akın and Nazlı Akyurt for their existence.

My special thanks is to The Scientific and Technological Research Council of Turkey, TÜBİTAK for the financial support provided under BİDEB program.

Finally, I would like to thank to my parents Fatma Tillem and Mehmet Tillem, my brother Salih Zeki Tillem and my sisters Merve Tillem and Zeynep Özkılınç Tillem. I am grateful for their unlimited love and support throughout my life.

Contents

Introduction **xii**

1 Background **1**

1.1 Cryptographic Primitives 1

1.1.1 Homomorphic Encryption 2

1.1.2 Damgård - Jurik Cryptosystem 2

1.1.3 Advanced Encryption Standard 3

1.2 Private Information Retrieval 4

1.2.1 Lipmaa's CPIR Scheme 5

1.2.2 Improving Lipmaa's CPIR 7

1.2.3 Lim-Lee Multi-Exponentiation Algorithm 8

1.3 Oblivious RAM 8

1.3.1 Path ORAM 9

1.4 Bucketization Method for Privacy Preserving Range Queries 10

2 A New Method for CPIR **13**

2.1 The New CPIR using BDDs 14

2.1.1 (2, 1) - CPIR 14

2.1.2 (n, 1) - CPIR 16

2.2 Implementing the New CPIR on Octal Trees 18

2.2.1 (8,1)-CPIR for Octal Trees 18

2.2.2 (n,1)-CPIR for Octal Trees 20

2.3 Implementing the New CPIR on Hexadecimal Trees 20

2.3.1 (16,1)-CPIR for Hexadecimal Trees 20

2.3.2 (n,1)-CPIR for Hexadecimal Trees 22

2.4 Utilizing Lim-Lee Multi-Exponentiation Method to Accelerate The New CPIR . . . 23
2.5 A Parallel Implementation for the New CPIR . . . 25
2.5.1 Client Side Parallel Implementation . . . 25
2.5.2 Server Side Parallel Implementation . . . 26
2.6 A Scalable Approach for The Parallel Implementation . . . 29

3 Privacy Preserving Range Queries using PIR and ORAM 32
3.1 CPIR Technique for Privacy Preserving Range Queries . . . 32
3.2 Path ORAM Technique for Privacy Preserving Range Queries . . . 34

4 Communication and Computational Analysis 36
4.1 Analysis of Communication and Computation for CPIR . . . 36
4.1.1 Analysis of Communication Complexity . . . 37
4.1.2 Analysis of Computational Complexity . . . 40
4.2 Analysis of Communication and Computation for Privacy Preserving Range Queries . . . 43
4.2.1 Computational Complexity Analysis . . . 46

5 Implementation Results 48
5.1 Timing results for the new CPIR . . . 48
5.1.1 Timings for Client Side Computations . . . 49
5.1.2 Timings for Server Side Computations . . . 49
5.2 Timing results for the Privacy Preserving Range Queries . . . 56

6 Conclusion 61

List of Figures

1.1 An illustration of Bdd which contains 4 files 6

1.2 An example of (n,1) - CPIR for a database of 4 files 7

4.1 Ratio of exchanged bits to database size in CPIR based technique on octal and hexadecimal trees for different number of buckets 44

4.2 Ratio of exchanged bits to database size in Path ORAM based technique for different number of buckets 45

4.3 The bandwidth usage in the CPIR and ORAM method to retrieve multiple buckets 46

List of Tables

4.1 The total bandwidth usage in number of bits for changing database sizes, where $\lceil \log_2(N) \rceil = 1024$ 38

4.2 Total bandwidth usage in scalable CPIR for octal trees (number of bits), where $\lceil \log_2(N) \rceil = 1024$ 39

4.3 Total bandwidth usage in scalable CPIR for hexadecimal trees (number of bits), where $\lceil \log_2(N) \rceil = 1024$ 39

4.4 Estimated timings of server side computations for different tree types in ms, where $|N| = 1024$ 41

4.5 Estimated timing values for binary trees in ms 42

4.6 Estimated timing values for octal trees in ms 43

4.7 Estimated timing values for hexadecimal trees in ms 43

5.1 Timings of client computation for encryption of selection bits and decryption of results with $|N| = 1024$ 50

5.2 Timings of server computation in serial case for binary, octal and hexadecimal trees with $|N| = 1024$ 51

5.3 Timings of server computation in parallel case for octal and hexadecimal trees 52

5.4 Timings of server computation in scalable case for octal trees 52

5.5 Timings of server computation in scalable case for hexadecimal trees . 53

5.6 Timings of server computation in octal trees for various number of cores in ms 54

5.7 Timings of server computation in hexadecimal trees for various number of cores in ms 54

5.8 Timings of server computation in octal tree with scalable method for various number of cores in sec 55
5.9 Timings of server computation in hexadecimal tree with scalable method for various number of cores in sec 55
5.10 Timings of client computation for CPIR and Path ORAM method in octal case utilizing 4 cores (in ms) 57
5.11 Timings of client computation for CPIR and Path ORAM method in hexadecimal case utilizing 4 cores (in ms) 57
5.12 Timings of client computation for the CPIR method in octal case utilizing multiple cores (in ms) 58
5.13 Timings of client computation for the CPIR method in hexadecimal case utilizing multiple cores (in ms) 58
5.14 Timings of serial server computation for the new CPIR method in octal case in ms 58
5.15 Timings of serial server computation for the new CPIR method in hexadecimal case in ms 59
5.16 Timings of parallel server computation for the new CPIR method in octal case in ms 60
5.17 Timings of parallel server computation for the new CPIR method in hexadecimal case in ms 60

Introduction

While outsourcing storage of data to cloud is beneficial for data owners to reduce the associated costs thereof, ensuring secure and private access to data becomes the next big challenge. The threat is that a curios data holder may try to retrieve information from the stored data, from the queries sent by the data owner or from the results of the queries. Therefore, several approaches are proposed in the literature to securely search over outsourced data for an item or for items in a range. One approach considers preserving the order of plaintext [3], or the order of prefix in plaintext [21], in ciphertext using deterministic encryption methods, known as order-preserving encryption. Boldyreva et al. [4], [5] improve the performance of order-preserving encryption and provides formal security definitions for the method. The second approach is known as predicate encryption [6], or Hidden Vector Encryption (HVE), based on public key encryption. In the setup phase of HVE, based on the characteristics of the plaintext, a vector x is generated and the data is encrypted for storage. When a user wants to query a range from the encrypted data, he needs to create a query token. The token contains a query vector w which is generated for the requested range based on the characterization rules of x vector. Thus, to search for a range, the token should compare the corresponding elements of w and x vectors. If they match, then the token is able to decrypt the corresponding ciphertext [31]. Another solution is utilizing special data structures used to store the encrypted data. For example, Vimercati et al. [26] proposes a privacy-preserving range query method on B+ trees. The method consists of three main parts. The first part involves *cover searches* which hides the actual request within fake requests. The second part is *cached searches* which stores some recent data in a cache on client side. The third part performs *shuffling* on the accessed nodes of B+ tree and rewrites them to the server. The last approach for privacy-preserving range queries is bucketization

methods which partitions the data into buckets according to a predefined rule. An early approach for bucketization, [16], uses simpler methods, such as equi-depth or equi-width partitioning, to partition the data into buckets. According to this model, each data item is assigned a bucket id depending on the partitioning method and stored encrypted in the database along with its bucket id. When a query needs to be performed, the query is first translated into the corresponding bucket id based on the partitioning method. Then, the encrypted data items with the matching bucket ids are retrieved from the database. The main problem of bucketization schemes is the existence of false positives caused by retrieving the data as a bucket instead of one by one. While the existence of false positives benefits the security of the scheme by obfuscating the retrieved data range, it creates an overhead in terms of performance. Hore et al., [17], [18], improve bucketization methods by introducing algorithms for optimized buckets in terms of performance and security.

The proposed methods are generally successful in satisfying two security concerns of privacy-preserving range queries: *data confidentiality* and *query confidentiality*. Since almost all schemes store the data in encrypted form, the confidentiality of data is provided trivially. The security of a query content can be achieved by its transformation into a secure representative such as tokens [6], bucket ids [18] or fake requests [26]. However, apart from those concerns, a good privacy-preserving range query scheme should prevent the disclosure of query access patterns. Especially in precise query protocols, as HVE or order preserving schemes, the observation of query access patterns can reveal useful information about query or data [19], [10]. Although, Vimercati et al.'s B+ scheme achieves hiding the query access patterns by using the shuffling method, the cryptographic primitives of the scheme are weak. Instead, usage of Oblivious RAM (ORAM) or Private Information Retrieval (PIR) methods can provide stronger security for range queries. ORAM and PIR methods are proposed to enable retrieve an item from an encrypted database without leaking any information related to the retrieved data [7], [14]. The high overheads of these methods are an obstacle for PIR and ORAM based techniques, but, both methods provide favorable properties for privacy-preserving range queries provided that their performances are improved.

Indeed, recent advances in the literature such as the Path ORAM method of

Stefanov et al. [28], which is fast and easy to implement, provide almost practical schemes in hiding query access patterns. Similarly, certain acceleration techniques in PIR schemes [24], [30] may also result in acceptable performance results.

The aim of this thesis is to explore the feasibility of hiding access patterns in privacy-preserving range queries. We implement two techniques, one based on the Lipmaa's CPIR [24] method, and the other on the Path ORAM [28] method and compare them in terms of their communication and computation costs. For the CPIR based technique, we propose a new computation method which improves the performance of Lipmaa's CPIR by reducing the total number of modular exponentiation operations, introducing shallow trees using octal and hexadecimal constructions, where internal tree nodes have 8 and 16 children, respectively, and enabling employment of simultaneous modular exponentiation operations. In addition, we present parallel algorithms to further accelerate the computations in the CPIR technique.

The thesis starts with providing necessary preliminary information about the utilized techniques in Chapter 1. Chapter 2 introduces the new method for CPIR technique in detail. Then, in Chapter 3 the application of CPIR and Path ORAM on privacy-preserving range query schemes is explained. Once the methods are introduced, Chapter 4 presents the analysis of communication and computation complexities both for the new CPIR scheme and for the range query schemes which utilize CPIR and Path ORAM. Finally, the actual results of the implementations for the new CPIR scheme and the range query schemes are provided in Chapter 5. Chapter 6 concludes the thesis.

Chapter 1

Background

As it has been already mentioned, this thesis addresses the problem of hiding query access patterns in privacy-preserving range queries. To that end, two different techniques, based on CPIR and Path ORAM, respectively, are employed. This chapter provides the necessary background information for PIR, Oblivious RAM (ORAM) and range queries, which is necessary to follow the discussions in the subsequent chapters. First, we start with introducing the utilized cryptographic preliminaries, which are the concept of homomorphic encryption, the Damgård Jurik homomorphic cryptosystem and the Advanced Encryption Standard (AES). Since the proposed CPIR method is based on the Lipmaa's CPIR method [24], the CPIR method and an extension on the CPIR method [30] are explained. In addition, Lim-Lee's multiexponentiation method, utilized for accelerating the CPIR scheme, is presented. The chapter continues with the detailed explanations of the ORAM and Path ORAM methods as a simple implementation of ORAM. Finally, since the proposed CPIR based range query scheme is constructed using bucketization technique [18], an overview of a bucketization scheme adapted to range queries is provided.

1.1 Cryptographic Primitives

The security of the CPIR methods generally depends on the security of the underlying cryptosystem. The Lipmaa's CPIR method and our new CPIR method are based on additive homomorphism and multiple encryption. Thus, we first explain

the concept of homomorphic encryption, then continue with Damgård-Jurik cryptosystem which satisfies the necessary cryptographic properties for CPIR scheme. Finally, brief information on AES cryptosystem, which is utilized for confidentiality of stored data in range query schemes, is provided.

1.1.1 Homomorphic Encryption

A cryptosystem is homomorphic if it allows to perform operations on encrypted data without decrypting it. The homomorphic property enables the data owners to operate on their data, which is stored on a remote server, without sharing the private key with server. The homomorphic property of most homomorphic cryptosystems is based on a specific operation such as addition or multiplication. RSA [2] and ElGamal [11] cryptosystems are examples of multiplicatively homomorphic cryptosystems while Goldwasser-Micali [15], Paillier [25] and Damgård-Jurik [9] cryptosystems are representatives of additively homomorphic cryptosystems. Furthermore, there are some homomorphic cryptosystems that can perform both addition and multiplication on the ciphertext, which are known as fully homomorphic schemes [12].

1.1.2 Damgård - Jurik Cryptosystem

Retrieving a file from the database using CPIR protocol requires utilizing an additively homomorphic cryptosystem. A possible candidate can be the Paillier cryptosystem. However, Paillier is not a convenient scheme as it cannot accomodate changing block sizes, which is required in CPIR schemes. An alternative is the Damgård - Jurik cryptosystem. It is a generalization of the Paillier cryptosystem, which enables to change the block length of the scheme without losing the homomorphic property [9], and, thus, allows multiple encryptions.

The setting of the Damgård - Jurik cryptosystem is similar to RSA that employs the operations on a modulus N which is the product of two sufficiently large primes, p and q. However, its security assumption differs from the RSA cryptosystem. Rather than relying on the hardness of integer factorization, the security of the Damgård-Jurik scheme is based on the hardness of decisional composite residuosity problem.

The key property of the Damgård - Jurik cryptosystem that makes it useful for

the CPIR protocol is the positive integer s which is used as the power of modulus N. This value allows multiple encryptions in different levels by adjusting the block length for the same public key. The cryptographic operations and homomorphic properties of the Damgård-Jurik cryptosystem are as follows:

Key Generation:

- Choose large primes p and q, and set N as $N = p \cdot q$.
- Choose an element $g \in Z^*_{N^{s+1}}$ such that $g = (1+N)^j x \bmod N^{s+1}$ with j being an integer relatively prime to N and $x \in H$, where the group H is isomorphic to Z^*_N
- Compute $\lambda = lcm(p-1, q-1)$, and choose d such that $d \equiv 1 \bmod N^s$ and $d \equiv 0 \bmod \lambda$.
- Public key : (N, g)
- Private key: d

Encryption:

- $E(m, r) = g^m r^{N^s} \bmod N^{s+1}$, where $m \in Z_N$ is the plaintext and $r \in Z_{N^{s+1}}$ is a random number.

Decryption:

- Compute $c^d \bmod N^{s+1}$, then using the algorithm proposed in [9] find m.

Additive Homomorphic Properties:

- $E(m_1) \cdot E(m_2) = E(m_1 + m_2)$
- $E(m)^c = E(m \cdot c)$

1.1.3 Advanced Encryption Standard

The secure storage of data in a remote server is possible by guaranteeing the confidentiality of data which can be provided by encryption. Advanced Encryption

Standard (AES) is the current standard for data encryption based on Rijndael cipher [1]. It is a symmetric block cipher with 128, 192 and 256 bit key sizes. The cryptosystem is based on repetitive permutation and substitution operations. The size of the message in AES may be larger or smaller than the block size which requires an adaptation of the plaintext to the block size. To that end, the modes of operation are utilized on AES. There are several types of modes of operation to use on block ciphers. The modes may work in parallel blocks or may apply a feedback mechanism such that each block depends on the result of the former method. Such modes of operation require usage of an initialization vector (IV) for the first block which enables to produce distinct ciphertexts for each encryption of a plaintext. Cipher block chaining (CBC) is an example of modes which utilize feedback mechanism using IV. In this method each block encryption depends on the encryption of the previously encrypted blocks. To provide data confidentiality in the proposed privacy preserving range query schemes, we encrypted the database with AES using CBC mode of operation in our experiments.

1.2 Private Information Retrieval

Private Information Retrieval (PIR), introduced by Chor et. al [7], is a method which enables users to retrieve an item from a remote database without leaking any information about the retrieved item to database server. The trivial solution of PIR is downloading the entire data from the server and applying a local retrieve operation. However, it is not a feasible solution considering the size of the database and the access rights of the user. Instead, using PIR methods, an item can be retrieved by exchanging less data than the size of the original database. PIR can be achieved either securing the client against computationally unbounded servers, referred as Information Theoretic PIR (itPIR); or relying the security of the protocol on a computationally difficult problem, known as Computational PIR (CPIR). Existing CPIR schemes benefit from the security assumption of a cryptosystem as the computationally difficult problem. For instance, The CPIR protocol of Kushilevitz and Ostrovsky [20] is developed on the Goldwasser-Micali [15] cryptosystem which depends on the intractability of quadratic residuosity problem. Similarly, Lipmaa

proposed a CPIR method [24] using Damgård-Jurik cryptosystem, the security of which is based on the decisional composite residuosity problem [9]. Since Lipmaa's method has a better communication complexity than other CPIR methods, we select it as the basis of our PIR protocol for our privacy-preserving range query scheme.

1.2.1 Lipmaa's CPIR Scheme

Lipmaa's CPIR method, which is known as Binary Decision Diagram CPIR, or BddCPIR, combines a non-cryptographic data structure with a cryptographic protocol [24]. Binary decision diagrams (Bdd), as the non-cryptographic data structure of BddCPIR, enable to improve the efficiency of the scheme in terms of communication and computation complexity. On the other hand, the security of the scheme is guaranteed by the security assumption of the Damgård-Jurik cryptosystem. The cryptographic properties of Damgård-Jurik cryptosystem are explained previously. Hence, this section continues with the definition of binary decision diagrams and then introduces the Lipmaa's CPIR protocol.

Binary Decision Diagrams

A binary decision diagram (Bdd) is a directed acyclic graph, whose internal nodes have at most two outgoing edges, which are labeled as 0 or 1. Since in the Lipmaa's scheme Bdds always have exactly two outgoing edges, they can be considered as binary trees, as well. For each level i of the binary tree, the jth internal node of the current level is represented as $R_{i,j}$. The data items are stored in the leaf nodes and represented as f_x. The index x is a d-bit string that shows the route taken from the root node to the corresponding sink node, where d is the depth of the tree. A Bdd storing 4 files is demonstrated in Figure 1.1.

Octal and Hexadecimal Trees In this work, we use a slightly different method based on Lipmaa's BddCPIR, which employs octal and hexadecimal trees, instead of binary trees due to performance reasons as explained in the subsequent sections. In general, octal and hexadecimal trees share the same properties with binary trees, but, the main difference is the number of children for each internal node. While octal trees have 8 children in each node, hexadecimal trees have 16. The sink nodes

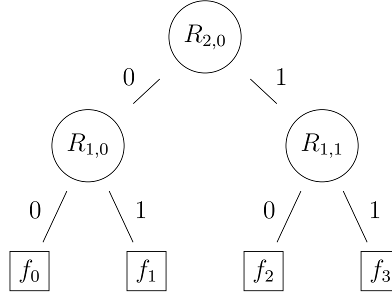


Figure 1.1: An illustration of Bdd which contains 4 files

of octal and hexadecimal trees are represented by $3d$ and $4d$ bits, repectively, where d is the depth of the tree.

(2,1)-CPIR

(2,1)-CPIR is the base protocol of the Lipmaa's 1-out-of-n construction, where n is the number of data items. In this protocol, the client inputs either 0 or 1 to retrieve one of the 2 files -f_0 or f_1- stored in the server without leaking information. The flow of the protocol is as follows:

- Client generates public and private keys (pk, sk). According to the value of input $x \in \{0, 1\}$, client computes the encrypted selection bit $c = E_{pk}(x)$ and sends pk and c to the server.
- Server computes $R = E_{pk}(f_0) \cdot c^{f_1 - f_0}$ and sends R to the client.
- Client decrypts R with his private key sk to find the selected file f_x

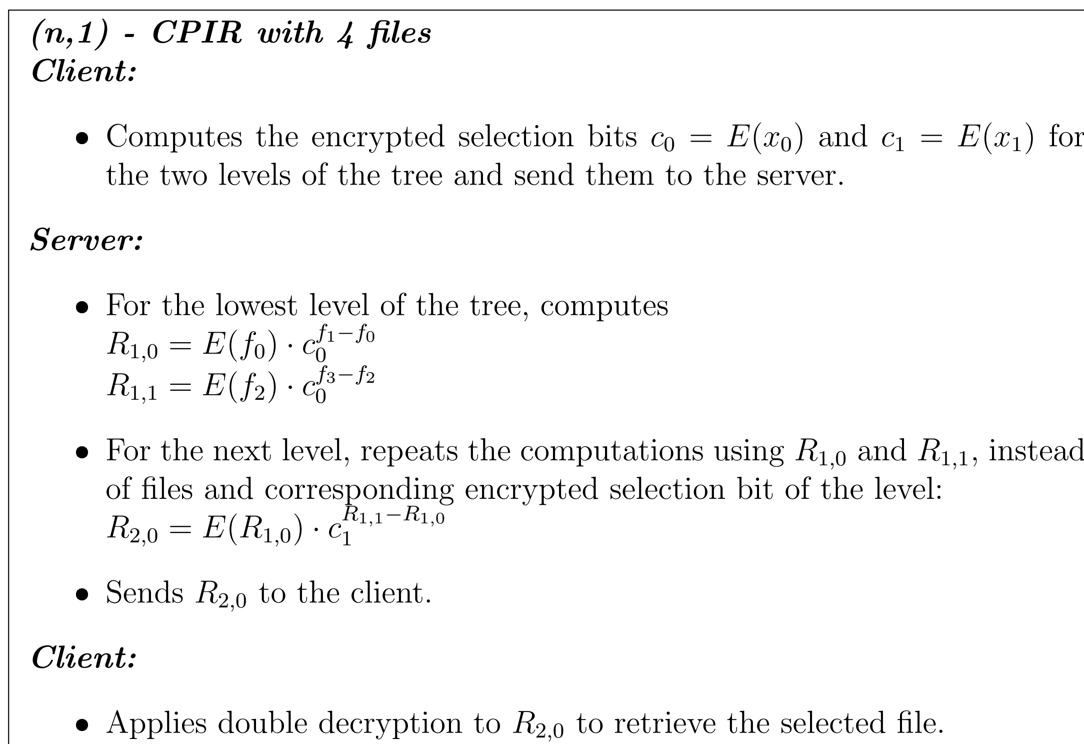
Proof. Based on the homomorphic properties of the Damgård-Jurik cryptosystem, we can show the correctness of the above operation as follows:

$$
\begin{aligned}
R &= E(f_0) \cdot c^{f_1 - f_0} \\
&= E(f_0) \cdot E(x)^{f_1 - f_0} \\
&= E(f_0 + x(f_1 - f_0)) = E(f_x)
\end{aligned}
$$

■

Generalization of (2,1)-CPIR to (n,1)-CPIR

On the basis of Lipmaa's (2,1)-CPIR protocol, a 1-out-of-n method can be constructed by repeating the protocol to 2-file sub-trees in each time. The protocol starts with the sink nodes of the tree and continues towards the root node. For the nodes in each level of the tree, the R values are calculated and used for the subsequent computations in the upper level of the tree. The final result is stored in the root of the tree and the server sends the value of the root node to the client. Different from (2,1)-CPIR, in (n,1)-CPIR protocol, the client needs to send d selection bits as $x = (x_0, x_1, , x_{d-1})$. Furthermore, the result should be decrypted d times to retrieve the requested file. Figure 1.2 illustrates the $(n, 1)-$CPIR protocol for a 4-file case based on the binary tree in Figure 1.1.



(n,1) - CPIR with 4 files
Client:

- Computes the encrypted selection bits $c_0 = E(x_0)$ and $c_1 = E(x_1)$ for the two levels of the tree and send them to the server.

Server:

- For the lowest level of the tree, computes
  $R_{1,0} = E(f_0) \cdot c_0^{f_1 - f_0}$
  $R_{1,1} = E(f_2) \cdot c_0^{f_3 - f_2}$
- For the next level, repeats the computations using $R_{1,0}$ and $R_{1,1}$, instead of files and corresponding encrypted selection bit of the level:
  $R_{2,0} = E(R_{1,0}) \cdot c_1^{R_{1,1} - R_{1,0}}$
- Sends $R_{2,0}$ to the client.

Client:

- Applies double decryption to $R_{2,0}$ to retrieve the selected file.

Figure 1.2: An example of (n,1) - CPIR for a database of 4 files

1.2.2 Improving Lipmaa's CPIR

In [30], Ünal and Savaş propose an improvement on Lipmaa's BddCPIR which employs a method based on octal trees. Although this form of the tree representation does not change the asymptotic complexity, it results in faster implementation by reducing the depth of the tree.

To show the computations in octal tree case, the (8,1)-CPIR is exemplified below:

- Client prepares the encrypted selection bits and sends them to the server:
  $c_0 = E(x_0)$ $c_1 = E(x_1)$ $c_2 = E(x_2)$
  $c_{0,1} = E(x_0 \cdot x_1)$ $c_{0,2} = E(x_0 \cdot x_2)$ $c_{1,2} = E(x_1 \cdot x_2)$
  $c_{0,1,2} = E(x_0 \cdot x_1 \cdot x_2)$

- Server computes:
  $R_{1,0} = E(f_0) \cdot c_0^{f_1 - f_0} \cdot c_1^{f_2 - f_0} \cdot c_2^{f_4 - f_0}$
  $\cdot c_{0,1}^{f_3 + f_0 - f_2 - f_1} \cdot c_{0,2}^{f_5 + f_0 - f_4 - f_1} \cdot c_{1,2}^{f_6 + f_0 - f_2 - f_4}$
  $\cdot c_{0,1,2}^{f_7 - f_6 - f_5 - f_3 - f_0 + f_4 + f_2 + f_1}$
  sends the result $R_{1,0}$ to the client.

- Client decrypts $R_{1,0}$ to retrieve the selected file.

1.2.3 Lim-Lee Multi-Exponentiation Algorithm

The implementation of cryptographic protocols in CPIR requires modular multiplication of several exponentiations, in the form of $x_1^{a_1} \cdot x_2^{a_2} \ldots x_t^{a_t} \bmod N^s$, where s indicates the current level in the tree. As the number of modular exponentiations increases, applying exponentiation in each term separately and then performing multiplication on them become prohibitively time consuming. An algorithm which can perform these exponentiation operations concurrently can help reduce the total cost of operations in CPIR. A candidate is Lim-Lee multi-exponentiation algorithm [22] which is based on precomputation techniques of [23]. The algorithm enables to operate modular exponentiation and multiplication simultaneously for multiple terms which improves the performance up to 4 times [22]. The pseudo-code for Lim-Lee multi-exponentiation algorithm is given in Algorithm 1.

1.3 Oblivious RAM

Private Information Retrieval is not the only solution for achieving hidden retrieval of encrypted data. Another well known method is Oblivious RAM which is based on shuffling and re-encryption operations in each access of data [14]. Different from PIR,

Algorithm 1 LIMLEE: Multi-Exponentiation Algorithm

```
▷ Precomputation
for k ← 0 to h − 1 step 1 do
    for e ← 0 to 2^w − 1 step 1 do
        e ← Σ_{i=0}^{w−1} e_i · 2^i
        Y_{k,e} ← Π_{i=0}^{w−1} y_{kw+i}^{e_i}
    end for
end for
▷ Main Part
Y ← 1
for k ← 0 to h − 1 step 1 do
    e ← Σ_{i=kw}^{kw+w−1} c_{i,t−1} · 2^{i−kw}
    Y ← Y · Y_{k,e}
end for
for j ← t − 2 to 0 step -1 do
    Y ← Y^2
    for k ← 0 to h − 1 step -1 do
        e ← Σ_{i=kw}^{kw+w−1} c_{i,j} · 2^{i−kw}
        Y ← Y · Y_{k,e}
    end for
end for
return Y
```

in ORAM access pattern can be observed but since the location and the encryption of the data item changes after every access to it, the adversary cannot obtain useful information by observing access patterns. Since operating ORAM may require a small client storage and considerable bandwidth usage, several constructions are proposed to achieve a practical ORAM method ([27], [13], [8]). In 2013 Stefanov et al. proposed a method which is claimed as the most practical ORAM scheme by utilizing a small client storage. Thus, in our privacy preserving range query scheme we utilize Stefanov's ORAM method, which is explained in the following section in detail.

1.3.1 Path ORAM

Stefanov's ORAM method which is known as Path ORAM is a simple Oblivious RAM based on shuffling and re-encryption operations by requiring a small client storage. Path ORAM is constructed on binary tree in which a full path of the tree is retrieved in each access. The details of the protocol is as follows:

Server stores the data in a binary tree structure. Each node of the tree is called a bucket. In each bucket, Z blocks of data are stored. If a bucket has less than Z blocks, dummy blocks are added. At the beginning, all buckets are initialized with some dummy values.

Client maintains a local stash, a small and private storage, to perform shuffling and re-encryption operations on the accessed data path and a position map that gives the current location of a data item. At the beginning of the protocol, the stash is empty and the position map assigns data items into some random buckets.

Access Protocol for Read and Write Operations to read or write data a client executes the following steps:

- **Remap block:** Client remaps the position of a data block to a new random position.
- **Read path:** Client reads the path of a data block from the server according to its value before remap operation.
- **Update block:** In write operations, client updates the value of data in the block.
- **Write path:** Client writes the accessed path back to the tree. If the operation is read, it only writes the original accessed path; but if it is a write operation it may add some values from the stash to the path.

1.4 Bucketization Method for Privacy Preserving Range Queries

In our model, to employ CPIR for range queries, we use the bucketization method by Hore et al. [18] as the underlying range query scheme. In bucketization, a secure index tag for each data item is generated using a predefined rule and assign it to a bucket depending on the tag. The query response is retrieved as buckets, instead of single data items which introduces false positives in the scheme. On one hand, retrieving all data items as buckets is a good approach in terms of security, since,

false positives within each bucket obfuscate the retrieved data range. On the other hand, however, the existence of false positives creates overhead on the client side computations, since client needs to clean the false positives to retrieve the target data items. Hore et al. [18] optimizes the buckets such that the existence of false positives helps to hide actual data while the overhead on client computations based on false positives become tolerable.

The optimal performance in the bucketization method is achieved by two algorithms. The first one is a greedy multi-partitioning algorithm, is introduced in [18] (see Algorithm 2). The algorithm minimizes the number of false positives within each bucket to maximize the performance of the scheme. It takes two inputs which are the dataset D and the number of buckets M and returns M buckets, where the cost of each bucket on client side is minimized. Greedy multi-partitioning algorithm assumes each data item as a point, such that the number of its attributes determines the dimensions of the point. To create optimal buckets, firstly, it computes the total cost of the dataset by the following formula:

$$Cost\,(D, R_j) = |D| \cdot \sum_{i=1}^{d} {r_j}^i \tag{1.1}$$

In Equation 1.1 $|D|$ represents the number of data items. d is the dimension of each point which is the number of attributes a data item has. R_j is the bucket candidate which is considered as a rectangle and r_j is the length of the ith edge of rectangle R_j which is actually the range of the bucket for the given attribute. At the beginning, the dataset considered as a bucket and is represented as R_1 in Algorithm 2 which is the largest rectangle that contains all data items. To partition R_1 into M buckets, two data points which forms the largest sub-rectangle based on the cost function in Equation 1.1. The rectangle is assigned as a new bucket and using the same method the remaining dataset is distributed into buckets.

While greedy multi-partitioning algorithm maximizes the performance of the bucketization scheme, the reduced number of false positives causes a vulnerability in the privacy of the scheme. Therefore, a second algorithm, known as *Controlled Diffusion* [18], is applied to the buckets' contents, which aims to redistribute bucket contents based on a pre-defined degradation factor, K. Thus, it optimizes the number of false positives within each bucket. A large K leads to an increase in false

Algorithm 2 Greedy Multidimensional Partitioning

Input Dataset of multidimensional points D, number of buckets M
Output M buckets, Total Cost
$Cost(D, R_1) = |D| * \sum_{i=1}^{d} {r_1}^i$
▷ /*${r_j}^i$ is the length of the ith edge of rectangle R_j*/
for $j \leftarrow 2$ to M **do**
Over all pairs of points in D, choose the pair $({p_1}^*, {p_2}^*)$ and the corresponding rectangle $R*$ s.t. the cost reduction cost $Cost(D, \bigcup_{t-1}^{j-1} R_t) - [Cost(D \backslash D_{R^*}, \bigcup_{t-1}^{j-1} R_t) + Cost(D_{R^*}, R^*)]$ is maximized;
Assign points within R^* to a new cluster R_j and recompute the minimum bounding rectangles ($MBRs$) of all the affected clusters;
Make one pass on D and reassign all points to these j clusters (readjusting $MBRs$ if necessary) to further reduce total cost;
end for
return M clusters and the total cost of the scheme;

positives, while small values of K decrease false positives. Therefore, based on the security and performance concerns, an optimal value for K can be determined. The Controlled Diffusion method is described in Algorithm 3.

Algorithm 3 Controlled-Diffusion (D, M, K)

Input Data set $D = (V, F)$, M = number of CBs (usually same as number opt buckets), K = maximum performance-degradation factor
Output An M-Partition of the data set (i.e. M buckets)
Compute optimal buckets $B_i, ..., B_M$ using QOB algorithm;
Initialize M empty composite buckets $CB_1, ..., CB_M$;
for each B_i **do**
Select $d_i = K * |B_i| \div f_{CB}$ distinct CBs randomly, $f_{CB} = |D| \div M$
Assign elements of B_i equiprobably to the d_i $CB's$;
▷ /*(roughly $|B_i| = d_i$ elements of B_i go into each CB)*/
end for
return the set of buckets $CB_j | j = 1, ..., M$;

Chapter 2

A New Method for CPIR

In BddCPIR proposed by Lipmaa, one factor which dominates the computation cost is modular exponentiation operations. Since the original method requires performing many modular exponentiations, an approach which reduces the number of exponentiation operations can improve the performance of the scheme. Also, as it is addressed in [30], the message expansion property of the Damgård - Jurik cryptosystem affects the computations significantly. Considering the increase in modulus N^{s+1} while proceeding to the upper levels of the tree (s indicating the level in the tree), decreasing the depth of the tree for the same number of data items results in reduced computation cost [30].

In response to the two dominant factors in computational complexity, we propose an accelerated scheme for CPIR in this chapter. Firstly, a new computation method is developed, which reduces the number of modular exponentiation operations per node of the tree. Secondly, octal and hexadecimal trees are utilized for shallow trees in computations. Thirdly, the Lim-Lee multi-exponentiation algorithm [22] is applied to lower the cost of multiple modular exponentiation operations. Furthermore, similar to [30], a non-trivial, efficient parallel algorithm is proposed for the new method. Finally, a hybrid approach for the parallel method is presented which enables the scheme to scale to large database sizes by taking advantage of small subtrees.

2.1 The New CPIR using BDDs

In the Lipmaa's CPIR protocol with binary decision diagrams, for each internal node of the tree, 3 modular exponentiation operations must be performed. Two of the exponentiations are needed in the Damgård-Jurik encryption operation and the third is applied to the encrypted selection bit. Since, modular exponentiation is the most expensive operation in the computations, decreasing the number of modular exponentiation operations will improve the performance of the scheme. To that end, we propose a new protocol which reduces the number of exponentiation operations by one for the internal nodes and by two for the lowest level (leaf) nodes of the binary decision diagram. In the new method, we eliminate the encryption operation in the original method and instead utilize the complements of selection bits, which are obtained through homomorphic computation. The next section explains the new CPIR protocol for 1-out-of-2 construction and the following section extends it to the 1-out-of-n CPIR construction.

2.1.1 (2, 1) - CPIR

The construction of the new scheme is similar to the original CPIR method [24]. Namely, the server stores in a database two files, f_0 and f_1. The client can retrieve one of the files, f_x, from the server by using one selection bit $x \in (0, 1)$. The rest of the new CPIR protocol with binary decision trees works as explained in the following steps:

- Client generates public and private keys (pk, sk). For the selection bit x, client computes $E(x)$ and sends the encrypted selection bit and the public key to the server.
- Server
  - first, finds the complement of the selection bit by the following formula: $C(x) = E(1) \cdot E(x)^{N-1} \mod N^2$
  - Then, computes $R = C(x)^{f_0} \cdot E(x)^{f_1} \mod N^2$ and sends R to the client.
- Client decrypts R to get the file that corresponds to the selection bit.

Proof. To show the correctness of the CPIR operation, first, we need to show the correctness of the complement calculations based on the additive homomorphic property of the Damgård-Jurik cryptosystem:

$$
\begin{aligned}
C(x) &= E(1) \cdot E(x)^{N-1} \mod N^2 \\
&= E(1) \cdot E((N-1) \cdot x) \mod N^2 \\
&= E(1) \cdot E(Nx - x) \mod N^2 \text{ which equals to } E(1 + Nx - x) \mod N^2
\end{aligned}
$$

Since the operations over the plaintext are in mod N, then we have $Nx \equiv 0 \mod N$. Thus, the result is $E(1-x)$. Specifically, if $x = 0$, namely the encrypted selection bit is $E(0)$, then its complement is $C(0) = E(1-0) = E(1)$. And conversely if $x = 1$, namely the encrypted selection bit is $E(1)$, then its complement is $C(1) = E(1-1) = E(0)$.

Now, we can prove the correctness of CPIR operation:

$$
\begin{aligned}
R &= C(x)^{f_0} \cdot E(x)^{f_1} \mod N^2 \\
&= E(1-x)^{f_0} \cdot E(x)^{f_1} \mod N^2 \\
&= E(f_0 - x \cdot f_0) \cdot E(x \cdot f_1) \mod N^2 \\
&= E(f_0 - x \cdot f_0 + x \cdot f_1) \mod N^2 \\
&= E(f_0 - x \cdot (f_0 + f_1)) \mod N^2
\end{aligned}
$$

Consequently, if $x = 0$, then $R = E(f_0 - 0 \cdot (f_0 + f_1)) = E(f_0) \mod N^2$. Then, the decryption of R gives f_0 for the selection bit $x = 0$.

Similarly, if $x = 0$, then $R = E(f_0 - 1 \cdot (f_0 + f_1)) = E(f_1) \mod N^2$. Finally, decrypting R results in f_1 for the selection bit $x = 1$. ■

In the above calculation, $C(x) = E(1) \cdot E(x)^{N-1} \mod N^2$ requires two modular exponentiation operations, i.e. one exponentiation for the encryption of 1 and one exponentiation for the exponent of the encrypted selection bit. However, we can rewrite the equation as $C(x) = g^1 \cdot r_1^N \cdot g^{x \cdot (N-1)} \cdot r_2^{N \cdot (N-1)} \mod N^2$, which is actually equivalent to $C(x) = g^{1-x} \cdot r^N \mod N^2$. Therefore, we obtain the following formula to compute the homomorphic complement of the selection bit $C(x) = g \cdot E(x)^{N-1} \mod N^2$.

Now, we can derive the following formula to compute the value of R

$$
\begin{aligned}
R &= C(x)^{f_0} \cdot E(x)^{f_1} \mod N^2 \\
&= (g \cdot E(x)^{N-1})^{f_0} \cdot E(x)^{f_1} \mod N^2 \\
&= g^{f_0} \cdot E(x)^{(N-1) \cdot f_0} \cdot E(x)^{f_1} \mod N^2
\end{aligned}
$$

$= g^{f_0} \cdot E(x)^{N \cdot f_0 - f_0 + f_1} \mod N^2$

$= g^{f_0} \cdot E(x)^{f_1 - f_0} \mod N^2$

The server has already known the files and g. Therefore, the value of g^{f_0} can be pre-computed and stored on server side, which means the CPIR operation can be handled by only one exponentiation operation for binary decision diagrams.

2.1.2 (n, 1) - CPIR

The new method can be extended to (n,1)-CPIR by using (2,1)-CPIR for each internal node of the tree. Similar to the original CPIR method, it will start from the sink nodes and will continue up to the root node by repeating the operations. The database consists of n files which are represented as $(f_0, f_1, f_2, \cdots, f_{n-1})$. Since the depth of the tree is greater than 1, now, the client needs to send a selection bit for each level of the tree. Therefore, to retrieve a file, f_x, the client should prepare the encryptions for the selection bits $(x_0, x_1, \cdots, x_{d-1})$, where d is the depth of the tree, i.e.i $d = \lceil \log(n) \rceil$. Once he receives the response from the server, the client needs to decrypt it d times. An example (n,1)-CPIR scheme for a database of 4 files is provided as follows:

- ***Client:***
  - Generates public and private keys.
  - Computes encrypted selection bits

    $E(x_0) = g^{x_0} \cdot r^N \mod N^2$

    $E(x_1) = g^{x_1} \cdot r^{N^2} \mod N^3$

    for each level of the tree and sends them to the server.

- ***Server:***
  - Computes the complements of the encrypted selection bits for each level of the tree:

    $C(x_0) = E(1) \cdot E(x)^{N-1} \mod N^2$

    $C(x_1) = E^{(2)}(1) \cdot E(x)^{N^2-1} \mod N^3$

- Computes $R_{i,j}$ values for the lowest level of the tree:
  $R_{1,0} = C(x_0)^{f_0} \cdot E(x_0)^{f_1} \mod N^2$
  $R_{1,1} = C(x_0)^{f_2} \cdot E(x_0)^{f_3} \mod N^2$
- Continues to the computations for the next level:
  $R_{2,0} = C(x_1)^{R_{1,0}} \cdot E^{(2)}(x_1)^{R_{1,1}} \mod N^3$
- Sends $R_{2,0}$ to the client.

- ***Client:***
  - Apply double decryption to $R_{2,0}$ to retrieve the selected file.

Proof. The correctness of the method can be showed by the following operations:

$$R_{2,0} = C(x_1)^{R_{1,0}} \cdot E^{(2)}(x_1)^{R_{1,1}} \mod N^3$$

We know that $C(x_i) = E(1 - x_i)$, then:

$$R_{2,0} = E^{(2)}(E(1-x_0)^{f_0} \cdot E(x_0)^{f_1} \cdot (1-x_1)) \cdot E(E(1-x_0)^{f_2} \cdot E(x_0)^{f_3} \cdot x_1)$$

$$= E^{(2)}(E(f_0+x_0\cdot(f_1-f_0))+x_1\cdot(E(f_2+x_0\cdot(f_3-f_2))-E(f_0+x_0\cdot(f_1-f_0)))$$

Applying double encryption on the resulting $R_{2,0}$ based on the value of x_1 and x_0 gives the requested file f_x. ■

In the above formulation, the superscript of $E^{(2)}(x_1)$ represents the value of s used in encryption operation, e.g., $\pmod N)^3$. Similarly, in the rest of the thesis, $E^{(s)}(x_i) = g^{x_i} r^{N^s} \pmod N)^{s+1}$.

Similar to the (2,1)-CPIR method in the previous section, we can apply an optimization to reduce the cost of exponentiation operations caused by complement operations. For the lowest level of the tree, using the precomputed g^{f_i} values for $i = 0, 2, 4, \ldots$, $R_{i,j}$s are computed as follows:

$$R_{1,0} = g^{f_0} \cdot E(x_0)^{f_1 - f_0} \mod N^2$$

$$R_{1,1} = g^{f_2} \cdot E(x_0)^{f_3 - f_2} \mod N^2$$

Although, the precomputation techniques is utilized for the sink nodes, it is not possible for the internal nodes. The reason is that the value of $R_{2,0}$ depends on the value of encrypted selection bits which are not known by the server beforehand. However, eliminating the extra computations for complement operations is still possible by the following optimizations:

$$\begin{aligned}
R_{2,0} &= C(x_1)^{R_{1,0}} \cdot E(x_1)^{R_{1,1}} \mod N^3 \\
&= (g \cdot E(x_1)^{N^2-1})^{R_{1,0}} \cdot E(x_1)^{R_{1,1}} \mod N^3 \\
&= g^{R_{1,0}} \cdot E(x_1)^{(N^2-1) \cdot R_{1,0}} \cdot E(x_1)^{R_{1,1}} \mod N^3 \\
&= g^{R_{1,0}} \cdot E(x_1)^{N^2 \cdot R_{1,0} - R_{1,0} + R_{1,1}} \mod N^3 \\
&= g^{R_{1,0}} \cdot E(x_1)^{R_{1,1} - R_{1,0}} \mod N^3.
\end{aligned}$$

In summary, while the original scheme by Lipmaa requires three modular multiplication for the same computation, the new technique requires only two.

2.2 Implementing the New CPIR on Octal Trees

As mentioned in the beginning of this chapter, another factor that affects the performance of the CPIR scheme is the expansion of message size in each level of tree due to the construction of the Damgård-Jurik cryptosystem. Therefore, similar to the method proposed in [30], the performance of the new method can be improved with an octal tree implementation since it reduces the depth of the tree. The next sections explain the CPIR method for octal trees first for 1-out-of-8 construction and then, generalizes for 1-out-of-n case.

2.2.1 (8,1)-CPIR for Octal Trees

In 1-out-of-8 construction of CPIR for octal trees, the server stores 8 files represented as $(f_0, f_1, \cdots, f_7)$ in a database. The client retrieves one file f_x from the database, where x corresponds to the selection bits. The operations of octal tree implementation differs from the original BddCPIR in the preparation of the encrypted selection bits, since each node has 8 children instead of 2. The children of a node can be represented by 3 bits as 000, 001, 010, 011, 100, 101, 110, 111.

At the beginning, the client prepares the encrypted selection bits for 7 of the children and sends them to the server:

$$e_{(0,0)} = E(\bar{x}_2\bar{x}_1\bar{x}_0),\ e_{(0,1)} = E(\bar{x}_2\bar{x}_1x_0),\ e_{(0,2)} = E(\bar{x}_2x_1\bar{x}_0)$$

$$e_{(0,3)} = E(\bar{x}_2x_1x_0),\ e_{(0,4)} = E(x_2\bar{x}_1\bar{x}_0),\ e_{(0,5)} = E(x_2\bar{x}_1x_0)$$

$$e_{(0,6)} = E(x_2x_1\bar{x}_0)$$

After receiving the encrypted selection bits, the server performs the complement operation to find the encrypted selection bit for the remaining child f_7:

$$E_0 = e_{(0,0)} \cdot e_{(0,1)} \cdot e_{(0,2)} \cdot e_{(0,3)} \cdot e_{(0,4)} \cdot e_{(0,5)} \cdot e_{(0,6)} \mod N^2$$

$$C_0 = E(1) \cdot E_0^{N-1} \mod N^2$$

Then, server computes $R_{1,0}$:

$R_{1,0} = e_{(0,0)}^{f_0} \cdot e_{(0,1)}^{f_1} \cdot e_{(0,2)}^{f_2} \cdot e_{(0,3)}^{f_3} \cdot e_{(0,4)}^{f_4} \cdot e_{(0,5)}^{f_5} \cdot e_{(0,6)}^{f_6} \cdot C_0^{f_7}$ and sends the result $R_{1,0}$ to the client. The client decrypts $R_{1,0}$ to retrieve the selected file.

The optimization of the complement operations which is proposed for binary trees is valid for octal trees, as well. We can optimize the calculation of $R_{1,0}$ as follows:

$$R_{1,0} = e_{(0,0)}^{f_0} \cdot e_{(0,1)}^{f_1} \cdot e_{(0,2)}^{f_2} \cdot e_{(0,3)}^{f_3} \cdot e_{(0,4)}^{f_4} \cdot e_{(0,5)}^{f_5} \cdot e_{(0,6)}^{f_6} \cdot C_0^{f_7}$$

$$= e_{(0,0)}^{f_0} \cdot e_{(0,1)}^{f_1} \cdot e_{(0,2)}^{f_2} \cdot e_{(0,3)}^{f_3} \cdot e_{(0,4)}^{f_4} \cdot e_{(0,5)}^{f_5} \cdot e_{(0,6)}^{f_6} \cdot g \cdot {E_0}^{N-1^{f_7}}$$

$$= e_{(0,0)}^{f_0} \cdot e_{(0,1)}^{f_1} \cdot e_{(0,2)}^{f_2} \cdot e_{(0,3)}^{f_3} \cdot e_{(0,4)}^{f_4} \cdot e_{(0,5)}^{f_5} \cdot e_{(0,6)}^{f_6} \cdot$$

$$[g \cdot (e_{(0,0)} \cdot e_{(0,1)} \cdot e_{(0,2)} \cdot e_{(0,3)} \cdot e_{(0,4)} \cdot e_{(0,5)} \cdot e_{(0,6)})^{N-1}]^{f_7}$$

$$= e_{(0,0)}^{f_0+(N-1)\cdot f_7} \cdot e_{(0,1)}^{f_1+(N-1)\cdot f_7} \cdot e_{(0,2)}^{f_2+(N-1)\cdot f_7} \cdot e_{(0,3)}^{f_3+(N-1)\cdot f_7} \cdot e_{(0,4)}^{f_4+(N-1)\cdot f_7} \cdot e_{(0,5)}^{f_5+(N-1)\cdot f_7} \cdot e_{(0,6)}^{f_6+(N-1)\cdot f_7} \cdot g^{f_7}$$

$$= e_{(0,0)}^{f_0-f_7} \cdot e_{(0,1)}^{f_1-f_7} \cdot e_{(0,2)}^{f_2-f_7} \cdot e_{(0,3)}^{f_3-f_7} \cdot e_{(0,4)}^{f_4-f_7} \cdot e_{(0,5)}^{f_5-f_7} \cdot e_{(0,6)}^{f_6-f_7} \cdot g^{f_7}$$

The new method for complement computations enables to reduce the number of exponentiation operations to 7 for the sink nodes of octal trees by utilizing precomputed values of g^{f_i} .

2.2.2 (n,1)-CPIR for Octal Trees

Now we can generalize the new CPIR method with octal trees to n files case, where $n = 8^d$. Deriving from the preparation of the selection bits in 8 files case, a general formula can be developed to prepare 7 selection bits for each level of the tree, $s = 1, ..., d$:

$$e_{(s-1,0)} = E(\bar{x}_{3s-1}\bar{x}_{3s-2}\bar{x}_{3s-3}),\ e_{(s-1,1)} = E(\bar{x}_{3s-1}\bar{x}_{3s-2}x_{3s-3}),$$

$$e_{(s-1,2)} = E(\bar{x}_{3s-1}x_{3s-2}\bar{x}_{3s-3}),\ e_{(s-1,3)} = E(\bar{x}_{3s-1}x_{3s-2}x_{3s-3}),$$

$$e_{(s-1,4)} = E(x_{3s-1}\bar{x}_{3s-2}\bar{x}_{3s-3}),\ e_{(s-1,5)} = E(x_{3s-1}\bar{x}_{3s-2}x_{3s-3}),$$

$$e_{(s-1,6)} = E(x_{3s-1}x_{3s-2}\bar{x}_{3s-3})$$

The operations of the server, including the computation of the complement bit for each level of the tree, are explained in Algorithm 4 in detail. The algorithm includes the optimizations for the complement operations such that seven and eight modular exponentiation operations are performed for the sink nodes and the internal nodes, respectively.

2.3 Implementing the New CPIR on Hexadecimal Trees

Considering the bandwidth usage, the depth of the tree for CPIR can be further decreased by using hexadecimal trees. The method is similar to binary and octal tree implementations except for the number of selection bits per level. 16 children of a node are represented with 4 bits, thus, for each level, 15 encrypted selection bits are prepared by client and the 16th bit is calculated on the server side by applying the homomorphic complement operation. The methods for 1-out-of-16 and 1-out-of-n CPIR on hexadecimal trees are explained in the following sections, respectively.

2.3.1 (16,1)-CPIR for Hexadecimal Trees

In (16,1)-CPIR method with hexadecimal trees, the database has 16 files, which are represented as $(f_0, f_1, \cdots, f_{15})$. To request the file f_x from the server, the client prepares the encrypted selection bits x, where $x = (x_3x_2x_1x_0)$, as described below:

Algorithm 4 OCTO-SERIAL: Server computations of the new $(n,1)$-CPIR scheme for the octal tree

Input $\mathcal{E} = \{e_{0,0}, \ldots, e_{d-1,6}\}$ where in $e_{s,i}$ $s = 1 \cdots d$, $i = 0 \cdots 6$ and $\mathcal{F} = \{f_0, \ldots, f_{8^d-1}\}$
Output $R_{d,0}$

```
for i ← 0 to 8^d − 1 do
    R_{0,i} ← f_i
end for

for s ← 1 to d do
▷ Utilize precomputations in the sink nodes
    if s = 1 then
        for j ← 0 to 8^{d−s} − 1 do
            for k ← 0 to 6 do
                t_k ← R_{s−1,8j+k} − R_{s−1,8j+7}
            end for
            R_{s,j} = e_{(s−1,0)}^{t_0} · e_{(s−1,1)}^{t_1} · e_{(s−1,2)}^{t_2} · e_{(s−1,3)}^{t_3} · e_{(s−1,4)}^{t_4}
        ·e_{(s−1,5)}^{t_5} · e_{(s−1,6)}^{t_6} · g^{R_{s−1,8j+7}}
        end for
▷ Computations for internal nodes
    else
        for j ← 0 to 8^{d−s} − 1 do
            for k ← 0 to 7 do
                if k ≠ 7 then
                    t_k ← R_{s−1,8j+k} − R_{s−1,8j+7}
                else
                    t_k ← R_{s−1,8j+7}
                end if
            end for
            R_{s,j} = e_{(s−1,0)}^{t_0} · e_{(s−1,1)}^{t_1} · e_{(s−1,2)}^{t_2} · e_{(s−1,3)}^{t_3} · e_{(s−1,4)}^{t_4}
        ·e_{(s−1,5)}^{t_5} · e_{(s−1,6)}^{t_6} · g^{t_7}
        end for
    end if
end for
return R_{d,0}
```

$$e_{(0,0)} = E(\bar{x}_3\bar{x}_2\bar{x}_1\bar{x}_0),\ e_{(0,1)} = E(\bar{x}_3\bar{x}_2\bar{x}_1x_0),\ e_{(0,2)} = E(\bar{x}_3\bar{x}_2x_1\bar{x}_0),$$

$$e_{(0,3)} = E(\bar{x}_3\bar{x}_2x_1x_0),\ e_{(0,4)} = E(\bar{x}_3x_2\bar{x}_1\bar{x}_0),\ e_{(0,5)} = E(\bar{x}_3x_2\bar{x}_1x_0),$$

$$e_{(0,6)} = E(\bar{x}_3x_2x_1\bar{x}_0),\ e_{(0,7)} = E(\bar{x}_3x_2x_1x_0),\ e_{(0,8)} = E(x_3\bar{x}_2\bar{x}_1\bar{x}_0),$$

$$e_{(0,9)} = E(x_3\bar{x}_2\bar{x}_1x_0),\ e_{(0,10)} = E(x_3\bar{x}_2x_1\bar{x}_0),\ e_{(0,11)} = E(x_3\bar{x}_2x_1x_0),$$

$$e_{(0,12)} = E(x_3x_2\bar{x}_1\bar{x}_0),\ e_{(0,13)} = E(x_3x_2\bar{x}_1x_0),\ e_{(0,14)} = E(x_3x_2x_1\bar{x}_0)$$

The server receives the encrypted selection bits and calculates the complement of the product of the selection bits as the 16th selection bit.

$$E_0 = e_{(0,0)} \cdot e_{(0,1)} \cdot e_{(0,2)} \cdot e_{(0,3)} \cdot e_{(0,4)} \cdot e_{(0,5)} \cdot e_{(0,6)} \cdot e_{(0,7)} \cdot e_{(0,8)} \cdot e_{(0,9)} \cdot e_{(0,10)} \cdot e_{(0,11)} \cdot e_{(0,12)} \cdot e_{(0,13)} \cdot e_{(0,14)} \mod N^2$$

$$C_0 = E(1) \cdot E_0^{N-1} \mod N^2$$

Later, the server performs the CPIR operation and computes $R_{1,0}$:

$R_{1,0} = e_{(0,0)}^{f_0} \cdot e_{(0,1)}^{f_1} \cdot e_{(0,2)}^{f_2} \cdot e_{(0,3)}^{f_3} \cdot e_{(0,4)}^{f_4} \cdot e_{(0,5)}^{f_5} \cdot e_{(0,6)}^{f_6} \cdot e_{(0,7)}^{f_7} \cdot e_{(0,8)}^{f_8} \cdot e_{(0,9)}^{f_9} \cdot e_{(0,10)}^{f_{10}} \cdot e_{(0,11)}^{f_{11}} \cdot e_{(0,12)}^{f_{12}} \cdot e_{(0,13)}^{f_{13}} \cdot e_{(0,14)}^{f_{14}} \cdot C_0^{f_{15}}$

$R_{1,0}$ is sent to the client, which decrypted to retrieve the selected file.

Similar to the octal tree implementation, the computation cost of the complement operations can be eliminated in hexadecimal trees. The method is the same as the octal tree case. The resulting formula for the computation of $R_{1,0}$ is given in the following:

$$R_{1,0} = e_{(0,0)}^{f_0-f_{15}} \cdot e_{(0,1)}^{f_1-f_{15}} \cdot e_{(0,2)}^{f_2-f_{15}} \cdot e_{(0,3)}^{f_3-f_{15}} \cdot e_{(0,4)}^{f_4-f_{15}} \cdot e_{(0,5)}^{f_5-f_{15}} \cdot e_{(0,6)}^{f_6-f_{15}} \cdot e_{(0,7)}^{f_7-f_{15}} \cdot e_{(0,8)}^{f_8-f_{15}} \cdot e_{(0,9)}^{f_9-f_{15}} \cdot e_{(0,10)}^{f_{10}-f_{15}} \cdot e_{(0,11)}^{f_{11}-f_{15}} \cdot e_{(0,12)}^{f_{12}-f_{15}} \cdot e_{(0,13)}^{f_{13}-f_{15}} \cdot e_{(0,14)}^{f_{14}-f_{15}} \cdot g^{f_{15}}$$

2.3.2 $(n,1)$-CPIR for Hexadecimal Trees

In the general form of the new CPIR with hexadecimal trees, the database consists of n files, where $n = 16^d$. The aim is to retrieve file f_x out of n files. The computation of the encrypted selection bits is formulated as below:

$$e_{(s-1,0)} = E(\bar{x}_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4}),\ e_{(s-1,1)} = E(\bar{x}_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}x_{4s-4}),$$

$$e_{(s-1,2)} = E(\bar{x}_{4s-1}\bar{x}_{4s-2}x_{4s-3}\bar{x}_{4s-4}),\ e_{(s-1,3)} = E(\bar{x}_{4s-1}\bar{x}_{4s-2}x_{4s-3}x_{4s-4}),$$

$e_{(s-1,4)} = E(\bar{x}_{4s-1}x_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4})$, $e_{(s-1,5)} = E(\bar{x}_{4s-1}x_{4s-2}\bar{x}_{4s-3}x_{4s-4})$,

$e_{(s-1,6)} = E(\bar{x}_{4s-1}x_{4s-2}x_{4s-3}\bar{x}_{4s-4})$, $e_{(s-1,7)} = E(\bar{x}_{4s-1}x_{4s-2}x_{4s-3}x_{4s-4})$,

$e_{(s-1,8)} = E(x_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4})$, $e_{(s-1,9)} = E(x_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}x_{4s-4})$,

$e_{(s-1,10)} = E(x_{4s-1}\bar{x}_{4s-2}x_{4s-3}\bar{x}_{4s-4})$, $e_{(s-1,11)} = E(x_{4s-1}\bar{x}_{4s-2}x_{4s-3}x_0)$,

$e_{(s-1,12)} = E(x_{4s-1}x_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4})$, $e_{(s-1,13)} = E(x_{4s-1}x_{4s-2}\bar{x}_{4s-3}x_{4s-4})$,

$e_{(s-1,14)} = E(x_{4s-1}x_{4s-2}x_{4s-3}\bar{x}_{4s-4})$

The server computations are similar to the general algorithm for the octal tree case (i.e., Algorithm 4) and are explained in Algorithm 5.

2.4 Utilizing Lim-Lee Multi-Exponentiation Method to Accelerate The New CPIR

The proposed method achieves to reduce the number of modular exponentiation operations and improves the performance of the CPIR. The current version of the protocol requires multiplication of 8 exponentiations for octal trees and multiplication of 16 exponentiations for hexadecimal trees per internal node, as it is explained in Section 2.2 and Section 2.3, respectively. In this section, we propose a new method to reduce the cost of multiple exponentiation operations. To that end, the Lim-Lee's multi-exponentiation algorithm [22] is utilized. The algorithm allows simultaneous execution of multiple exponentiation operations of the form $\prod_{i=1}^{t} a_i^{x_i} = a_1^{x_1} \cdot a_2^{x_2} \cdots a_t^{x_t}$ using pre-computation techniques and outputs the multiplication of exponentiations. In this method, the operations on the client side are not affected, only the server computations are updated. Algorithm 6 illustrates the server computations by employing the Lim-Lee technique on the new CPIR with hexadecimal trees. In the previous sections, for octal and hexadecimal trees the cost of the homomorphic complement bit computations is reduced by some optimizations techniques. These techniques can be applied in the new CPIR with Lim-Lee method, as well. The Lim-Lee algorithm requires a block of t modular exponentiations, where t equals 8 for octal trees and 16 for hexadecimal trees. Therefore, the optimization for the sink

Algorithm 5 HEX-SERIAL: Server computations of the new $(n,1)$-CPIR scheme for the hexadecimal tree

Input $\mathcal{E} = \{e_{0,0}, \ldots, e_{d-1,14}\}$ where in $e_{s,i}$ $s = 1 \cdots d$, $i = 0 \cdots 14$ and $\mathcal{F} = \{f_0, \ldots, f_{16^d-1}\}$
Output $R_{d,0}$

```
for i ← 0 to 16^d − 1 do
    R_{0,i} ← f_i
end for

for s ← 1 to d do
▷ Utilize precomputations in the sink nodes
    if s = 1 then
        for j ← 0 to 16^{d−s} − 1 do
            for k ← 0 to 14 do do
                t_k ← R_{s−1,16j+k} − R_{s−1,16j+15}
            end for
            R_{s,j} = e_{(s−1,0)}^{t_0} · e_{(s−1,1)}^{t_1} · e_{(s−1,2)}^{t_2} · e_{(s−1,3)}^{t_3} · e_{(s−1,4)}^{t_4} · e_{(s−1,5)}^{t_5}
        ·e_{(s−1,6)}^{t_6} · e_{(s−1,7)}^{t_7} · e_{(s−1,8)}^{t_8} · e_{(s−1,9)}^{t_9} · e_{(s−1,10)}^{t_10}
        ·e_{(s−1,11)}^{t_11} · e_{(s−1,12)}^{t_12} · e_{(s−1,13)}^{t_13} · e_{(s−1,14)}^{t_14} · g^{R_{s−1,16j+15}}
        end for
▷ Computations for internal nodes
    else
        for j ← 0 to 16^{d−s} − 1 do
            for k ← 0 to 15 do do
                if k ≠ 15 then
                    t_k ← R_{s−1,16j+k} − R_{s−1,16j+15}
                else
                    t_k ← R_{s−1,16j+15}
                end if
            end for
            R_{s,j} = e_{(s−1,0)}^{t_0} · e_{(s−1,1)}^{t_1} · e_{(s−1,2)}^{t_2} · e_{(s−1,3)}^{t_3} · e_{(s−1,4)}^{t_4} · e_{(s−1,5)}^{t_5}
        ·e_{(s−1,6)}^{t_6} · e_{(s−1,7)}^{t_7} · e_{(s−1,8)}^{t_8} · e_{(s−1,9)}^{t_9} · e_{(s−1,10)}^{t_10}
        ·e_{(s−1,11)}^{t_11} · e_{(s−1,12)}^{t_12} · e_{(s−1,13)}^{t_13} · e_{(s−1,14)}^{t_14} · g^{t_15}
        end for
    end if
end for
return R_{d,0}
```

level, which reduces the number of exponentiation operations to 7 or 15 for octal and hexadecimal trees, respectively, does not work in Lim-Lee method.

Algorithm 6 Server computation for the new $(n,1)$-CPIR scheme for hexadecimal trees using Lim-Lee multi-exponentiation technique

---

Input $\mathcal{E} = \{e_{0,0}, \ldots, e_{d-1,14}\}$ where in $e_{s,i}$ $s = 1 \cdots d$, $i = 0 \cdots 14$ and $\mathcal{F} = \{f_0, \ldots, f_{16^d-1}\}$
Output $R_{d,0}$
  for $i \leftarrow 0$ to $16^d - 1$ **do**
    $R_{0,i} \leftarrow f_i$
  end for

  for $s \leftarrow 1$ to d **do**
    for $j \leftarrow 0$ to $16^{d-s} - 1$ **do**
      for $k \leftarrow 0$ to 15 do **do**
        if $k \neq 15$ **then**
          $e_k \leftarrow R_{s-1,16j+k} - R_{s-1,16j+15}$
          $b_k \leftarrow e_{s-1,k}$
        else
          $e_k \leftarrow R_{s-1,16j+15}$
          $b_k \leftarrow g$
        end if
      end for
      $R_{s,j} = LIMLEE(e, b)$
    end for
  end for
  return $R_{d,0}$

2.5 A Parallel Implementation for the New CPIR

The CPIR method is suitable for parallel implementations since it has several repeating operations which are not dependent on each other. An efficient parallel algorithm for the CPIR on binary and octal trees is proposed by Ünal and Savaş [30]. In this section, we adapt the proposed algorithm to the new CPIR method for the server operations on octal and hexadecimal trees. Before that, a brief explanation for client side parallelization is provided.

2.5.1 Client Side Parallel Implementation

The client is responsible for two main operations: encrypting the selection bits and decrypting the server response. In the decryption of a single response, parallelism

cannot be utilized, since each operation is dependent on the previous one. On the other hand, parallelism is applicable on encrypting the selection bits. Algorithm 7 and Algorithm 8 present the parallelization of encryption operations for octal and hexadecimal trees, respectively.

Algorithm 7 Parallel client side encryptions for the new CPIR on octal trees

---

Input $x = (x_{3d-1}x_{3d-2}\ldots x_0)$, pk
Output $\mathcal{E}$
1: **for** $s \leftarrow 1$ to d **in parallel do**
2: $\quad e_{(s-1,0)} \leftarrow E^{(s)}(\bar{x}_{3s-1}\bar{x}_{3s-2}\bar{x}_{3s-3})$
3: $\quad e_{(s-1,1)} \leftarrow E^{(s)}(\bar{x}_{3s-1}\bar{x}_{3s-2}x_{3s-3})$
4: $\quad e_{(s-1,2)} \leftarrow E^{(s)}(\bar{x}_{3s-1}x_{3s-2}\bar{x}_{3s-3})$
5: $\quad e_{(s-1,3)} \leftarrow E^{(s)}(\bar{x}_{3s-1}x_{3s-2}x_{3s-3})$
6: $\quad e_{(s-1,4)} \leftarrow E^{(s)}(x_{3s-1}\bar{x}_{3s-2}\bar{x}_{3s-3})$
7: $\quad e_{(s-1,5)} \leftarrow E^{(s)}(x_{3s-1}\bar{x}_{3s-2}x_{3s-3})$
8: $\quad e_{(s-1,6)} \leftarrow E^{(s)}(x_{3s-1}x_{3s-2}\bar{x}_{3s-3})$
9: **end parallel for**
10: **return** $\mathcal{E}$

2.5.2 Server Side Parallel Implementation

The private information retrieval operation on server side requires processing of each data item in the database. Thus, applying parallelism can allow significant improvements on the performance of the method. The parallel algorithm in [30] achieves parallelism by dividing the main tree into subtrees, numbers of which are equal to the number of cores. The computations within each sub-tree are performed in serial. Once the cores finish the computations in subtrees, the PIR operations in the remaining (upper) part of the tree are distributed on the available cores in a straightforward manner to utilize parallelizm. The parallel algorithm can be employed for the new CPIR method using both octal and hexadecimal trees including the implementation of the Lim-Lee multi-exponentation method. Parallel server computations for the octal tree case are described in Algorithm 9.

Algorithm 8 Parallel client side encryptions for the new CPIR on hexadecimal trees

Input $x = (x_{4d-1}x_{4d-2}\dots x_0)$, pk
Output $\mathcal{E}$

1: **for** $s \leftarrow 1$ to d **in parallel do**
2: $\quad e_{(s-1,0)} \leftarrow E^{(s)}(\bar{x}_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4})$
3: $\quad e_{(s-1,1)} \leftarrow E^{(s)}(\bar{x}_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}x_{4s-4})$
4: $\quad e_{(s-1,2)} \leftarrow E^{(s)}(\bar{x}_{4s-1}\bar{x}_{4s-2}x_{4s-3}\bar{x}_{4s-4})$
5: $\quad c_{(s-1,3)} \leftarrow E^{(s)}(\bar{x}_{4s-1}\bar{x}_{4s-2}x_{4s-3}x_{4s-4})$
6: $\quad e_{(s-1,4)} \leftarrow E^{(s)}(\bar{x}_{4s-1}x_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4})$
7: $\quad e_{(s-1,5)} \leftarrow E^{(s)}(\bar{x}_{4s-1}x_{4s-2}\bar{x}_{4s-3}x_{4s-4})$
8: $\quad e_{(s-1,6)} \leftarrow E^{(s)}(\bar{x}_{4s-1}x_{4s-2}x_{4s-3}\bar{x}_{4s-4})$
9: $\quad e_{(s-1,7)} \leftarrow E^{(s)}(\bar{x}_{4s-1}x_{4s-2}x_{4s-3}x_{4s-4})$
10: $\quad e_{(s-1,8)} \leftarrow E^{(s)}(x_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4})$
11: $\quad e_{(s-1,9)} \leftarrow E^{(s)}(x_{4s-1}\bar{x}_{4s-2}\bar{x}_{4s-3}x_{4s-4})$
12: $\quad e_{(s-1,10)} \leftarrow E^{(s)}(x_{4s-1}\bar{x}_{4s-2}x_{4s-3}\bar{x}_{4s-4})$
13: $\quad e_{(s-1,11)} \leftarrow E^{(s)}(x_{4s-1}\bar{x}_{4s-2}x_{4s-3}x_{4s-4})$
14: $\quad e_{(s-1,12)} \leftarrow E^{(s)}(x_{4s-1}x_{4s-2}\bar{x}_{4s-3}\bar{x}_{4s-4})$
15: $\quad e_{(s-1,13)} \leftarrow E^{(s)}(x_{4s-1}x_{4s-2}\bar{x}_{4s-3}x_{4s-4})$
16: $\quad e_{(s-1,14)} \leftarrow E^{(s)}(x_{4s-1}x_{4s-2}x_{4s-3}\bar{x}_{4s-4})$
17: **end parallel for**
18: **return** $\mathcal{E}$

Algorithm 9 OCTO-PARALLEL: Parallel server computations for the new CPIR on octal trees

Input $\mathcal{E} = \{e_{0,0}, \ldots, e_{d-1,14}\}$ where in $e_{s,i}$ $s = 1 \cdots d$, $i = 0 \cdots 6$, $\mathcal{F} = \{f_0, \ldots, f_{8^d-1}\}$, 2^κ: number of cores, $\kappa < d$

Output $R_{d,0}$

1: $\alpha = \log_8 2^\kappa$, $\lambda = 8^\alpha$, $\gamma = \lambda/2^\kappa$
2: **for** $p \leftarrow 0$ to $2^\kappa - 1$ **in parallel do**
3:     **for** $y \leftarrow 1$ to $\gamma - 1$ **do**
4:         **for** $i \leftarrow 1$ to $8^d - 1/\lambda$ **do**
5:             $R_{0,i} = f_{p \cdot \gamma \cdot (8^d - 1/\lambda) + y \cdot (8^d - 1/\lambda) + i}$
6:         **end for**
7:         **for** $s \leftarrow 1$ to $d - \alpha$ **do**
8:             **if** $s = 1$ **then**
9:                 **for** $j \leftarrow 0$ to $8^{d-\lambda-s} - 1$ **do**
10:                     **for** $k \leftarrow 0$ to 6 **do**
11:                         $t_k \leftarrow R_{s-1,8j+k} - R_{s-1,8j+7}$
12:                     **end for**
13:                     $R_{s,j} = {e_{(s-1,0)}}^{t_0} \cdot {e_{(s-1,1)}}^{t_1} \cdot {e_{(s-1,2)}}^{t_2} \cdot {e_{(s-1,3)}}^{t_3} \cdot {e_{(s-1,4)}}^{t_4} \cdot {e_{(s-1,5)}}^{t_5} \cdot {e_{(s-1,6)}}^{t_6} \cdot g^{R_{s-1,8j+7}}$
14:                 **end for**
15:             **else**
16:                 **for** $j \leftarrow 0$ to $8^{d-\lambda-s} - 1$ **do**
17:                     **for** $k \leftarrow 0$ to 7 do **do**
18:                         **if** $k \neq 7$ **then**
19:                             $t_k \leftarrow R_{s-1,8j+k} - R_{s-1,8j+7}$
20:                         **else**
21:                             $t_k \leftarrow R_{s-1,8j+7}$
22:                         **end if**
23:                     **end for**
24:                     $R_{s,j} = {e_{(s-1,0)}}^{t_0} \cdot {e_{(s-1,1)}}^{t_1} \cdot {e_{(s-1,2)}}^{t_2} \cdot {e_{(s-1,3)}}^{t_3} \cdot {e_{(s-1,4)}}^{t_4} \cdot {e_{(s-1,5)}}^{t_5} \cdot {e_{(s-1,6)}}^{t_6} \cdot g^{t_7}$
25:                 **end for**
26:             **end if**
27:         **end for**
28:     **end for**
29: **end parallel for**

$\triangleright$ cores sync and continue with the rest of the tree concurrently

30: **for** $s \leftarrow d - \alpha + 1$ to d **do**
31:     **for** $j \leftarrow 0$ to $8^{d-s} - 1$ **in parallel do**
32:         **for** $k \leftarrow 0$ to 7 **do**
33:             $t_k \leftarrow R_{s-1,8j+k} - R_{s-1,8j+7}$
34:         **end for**
35:         $R_{s,j} = {e_{(s-1,0)}}^{t_0} \cdot {e_{(s-1,1)}}^{t_1} \cdot {e_{(s-1,2)}}^{t_2} \cdot {e_{(s-1,3)}}^{t_3} \cdot {e_{(s-1,4)}}^{t_4} \cdot {e_{(s-1,5)}}^{t_5} \cdot {e_{(s-1,6)}}^{t_6} \cdot g^{t_7}$
36:     **end parallel for**
37: **end for**
38: **return** $R_{d,0}$

2.6 A Scalable Approach for The Parallel Implementation

Reducing the number of exponentiation operations, utilizing shallow trees and exploiting parallelism improve the performance of CPIR significantly. However, for large database sizes, the cost of computations are still high due to increase in the size of the modulus in the Damgård-Jurik algorithm. In [30], a scalable method is proposed to overcome the performance issues of large database sizes. In this method, the database is maintained as a set of several reasonable-sized subtrees. Apart from the selection bits for the requested file, the client needs to send additional bits for subtree selection. The main concern is that the number of subtrees should be determined carefully, so that the bandwidth usage should not be adversely affected by large amount of subtrees.

Algorithm 10 demonstrates generation of the subtree selection bits and the regular selection bits for octal tree implementation. The size of the database is $n = 2^m$. The number of the subtrees, $\mu = 2^{m-3l}$, and the size of each subtree 8^l are predetermined values between the server and the client. The selection bits for the subtrees are represented by ς, and the regular selection bits for file operations represented by the set of $\mathcal{E}$.

After the server retrieves the selection bits, first it employs the subtree selection bits on the trees to collapse them into one subtree, as it is shown in Algorithm 11 from step 1 to 10. In step 11 of the algorithm, the retrieval operation for the requested file is operated on a single subtree using Algorithm 9 . The important point is that after collapsing subtrees, the modulus of the regular retrieval computations start from N^3 instead of N^2.

In Algorithm 11, the subtree collapsing operation requires multiple modular exponentiation and multiplication operations repeatedly. Therefore, using the Lim-Lee's multi-exponentiation algorithm, the performance of the scalable method can be improved. Algorithm 12 shows the scalable method which utilizes the Lim-Lee multi-exponentiation algorithm.

Algorithm 10 Client-side computation the scalable new CPIR on octal tree

Input m, l, and $x = x_{l-1} \dots x_1, x_0$

Output $\mathcal{E} = \{e_{0,0}, \dots, e_{l,6}\}$ where in $e_{s,i}$ $s = 1 \cdots l$, $i = 0 \cdots 6$ and $\{\varsigma_0, \dots, \varsigma_{2^{3d-3l}-1}\}$

1: $\mu \leftarrow 2^{3d-3l}$
2: $\zeta \leftarrow x_{3d-1}, \dots, x_l$
3: **for** $i \leftarrow 0$ to $\mu - 1$ **do**
4:     **if** $i \neq \zeta$ **then**
5:         $\varsigma_i \leftarrow E(0)$
6:     **else**
7:         $\varsigma_i \leftarrow E(1)$
8:     **end if**
9: **end for**
10: **for** $s \leftarrow 1$ to l **do**
11:     $e_{(s-1,0)} \leftarrow E^{(s)}(\bar{x}_{3s-1}\bar{x}_{3s-2}\bar{x}_{3s-3})$
12:     $e_{(s-1,1)} \leftarrow E^{(s)}(\bar{x}_{3s-1}\bar{x}_{3s-2}x_{3s-3})$
13:     $e_{(s-1,2)} \leftarrow E^{(s)}(\bar{x}_{3s-1}x_{3s-2}\bar{x}_{3s-3})$
14:     $e_{(s-1,3)} \leftarrow E^{(s)}(\bar{x}_{3s-1}x_{3s-2}x_{3s-3})$
15:     $e_{(s-1,4)} \leftarrow E^{(s)}(x_{3s-1}\bar{x}_{3s-2}\bar{x}_{3s-3})$
16:     $e_{(s-1,5)} \leftarrow E^{(s)}(x_{3s-1}\bar{x}_{3s-2}x_{3s-3})$
17:     $e_{(s-1,6)} \leftarrow E^{(s)}(x_{3s-1}x_{3s-2}\bar{x}_{3s-3})$
18: **end for**
19: **return** $\{e_{0,0}, \dots, e_{l-1,7}\}$ and $\{\varsigma_0, \dots, \varsigma_{\mu-1}\}$

Algorithm 11 Server-side computation the scalable new CPIR on octal trees

Input m, $\mathcal{E} = \{e_{0,0}, \dots, e_{l,6}\}$ where in $e_{s,i}$ $s = 1 \cdots l$, $i = 0 \cdots 6$, $\mathcal{F} = \{f_0, \dots, f_{2^{3d}-1}\}$, $\{\varsigma_0, \dots, \varsigma_{2^{3d-3l}}\}$, l and $\kappa < l$

Output $R_{l,0}$

▷ Collapse the subtrees into one
1: $\mu = 2^{3d-3l}$
2: $\gamma = 2^{3l-\kappa}$
3: **for** $p \leftarrow 0$ to $2^\kappa - 1$ **in parallel do**
4:     **for** $i \leftarrow 0$ to $\gamma - 1$ **do**
5:         **for** $k \leftarrow 0$ to $\mu - 1$ **do**
6:             $f'_k \leftarrow \varsigma_k^{f_{i+p\gamma+k2^{3l}}} \bmod N^2$
7:         **end for**
8:         $R_{0,p\gamma+i} \leftarrow \prod_{k=0}^{\mu-1} f'_k$
9:     **end for**
10: **end parallel for**

▷ Parallel CPIR computation for the remaining subtree
11: $R_{l,0} = OCTO - PARALLEL(\mathcal{E}, R)$
12: **return** $R_{l,0}$

Algorithm 12 Server-side computation the scalable new CPIR on octal trees using LimLee Algorithm

Input m, $\mathcal{E} = \{e_{0,0}, \ldots, e_{l,6}\}$ where in $e_{s,i}$ $s = 1 \cdots l$, $i = 0 \cdots 6$, $\mathcal{F} = \{f_0, \ldots, f_{2^{3d}-1}\}$, $\{\varsigma_0, \ldots, \varsigma_{2^{3d-3l}}\}$, l and $\kappa < l$

Output $R_{l,0}$

▷ Collapse the subtrees into one

1: $\mu = 2^{3d-3l}$
2: $\gamma = 2^{3l-\kappa}$
3: **for** $p \leftarrow 0$ to $2^{\kappa} - 1$ **in parallel do**
4:     **for** $i \leftarrow 0$ to $\gamma - 1$ **do**
5:         **for** $k \leftarrow 0$ to $(\mu/8) - 1$ **do**
6:             **for** $x \leftarrow 0$ to 7 **do**
7:                 $b_x \leftarrow \varsigma_{8k+x}$
8:                 $e_x \leftarrow f_{i+p\gamma+(8k+x)2^{3l}}$
9:             **end for**
10:             $f'_k = LIMLEE(e, b)$
11:         **end for**
12:         $R_{0,p\gamma+i} \leftarrow \prod_{k=0}^{(\mu/8)} f'_k$
13:     **end for**
14: **end parallel for**

▷ Parallel CPIR computation for the remaining subtree

15: $R_{l,0} = OCTO - PARALLEL(\mathcal{E}, R)$
16: **return** $R_{l,0}$

Chapter 3

Privacy Preserving Range Queries using PIR and ORAM

As stated previously, preventing the disclosure of query access patterns is possible by employing ORAM and PIR techniques. So far, we introduced a new method for PIR, based on Lipmaa's CPIR scheme, which enables to improve the performance of CPIR significantly. Further, we present a simple and fast method for ORAM which is known as Path ORAM [28] . Depending on these two techniques, in this chapter we develop two different methods for privacy preserving range queries. In the rest of the chapter, first the range query protocol based on CPIR is explained. Then, it continues with the explanation of the Path ORAM based range query protocol.

3.1 CPIR Technique for Privacy Preserving Range Queries

We introduce a new approach for privacy preserving range queries by implementing Private Information Retrieval protocol on an existing range query scheme. The base scheme for range query is Hore et al.'s [18] bucketization method. In retrieval operations the new CPIR method described in Chapter 2 is employed. The flow of the range query protocol can be summarized as follows:

Setup In the setup phase, initially the data is partitioned into buckets according to the Greedy Multi Partitioning (Algorithm 2) and Controlled Diffusion (Algorithm 3)

algorithms [18]. The number of buckets are assigned as powers of 8 and 16 to utilize octal and hexadecimal trees in CPIR. The bucketization algorithms do not guarantee to create same sized buckets. Thus, once the buckets are generated, some dummy values are inserted to make them equal in size.

Once the buckets are generated, the next step is to send them to server for storage. The buckets are stored on leaf nodes of octal and hexadecimal trees. As the bucket size can be relatively large, generally we need more than one tree to store the buckets. Thus, they are placed in trees in such a way that each leaf node of a tree holds one part of each bucket, which means the number of trees is proportional to the bucket size. Moreover, the size of the trees becomes equal to the number of buckets, since each leaf node maps to one bucket. Here we assume that the data within each bucket is encrypted using a secure symmetric cipher algorithm, therefore, the confidentiality of data is guaranteed.

To exemplify the distribution of a bucket into trees, consider a bucket which contains 16 items as the result of bucketization algorithms including the addition of dummy values. Each data item has 5 integer attributes including the primary key which is equal to 160 bits. Since an encrypted storage is required, the encryption of an item using AES maps to 256 bit of ciphertext for 160 bit plaintext in 256 bit block size. If each node of tree can contain 1024 bits of information, then we can place 4 of the items in a tree. Since we have 16 items in total, we need to distribute the bucket on 4 trees.

Sending Query To perform a range query operation, first the client needs to find the buckets which include the requested data range using a query translation operation. Later, based on the bucket ids, the client needs to prepare the selection bits to retrieve its content from the server. Since the requested bucket is stored on the same leaf node in each tree, the selection bits can generated for once and used on each tree repetitively.

Query Response Based on the selection bits sent by the client, server needs to perform CPIR on each tree to retrieve the corresponding bucket. In our method, we utilized the new CPIR method instead of Lipmaa's original method due to performance reasons. To utilize the parallelism in the best way, when the number of

trees exceeds the number of available cores, we employed a parallelism on tree level. Meanly, each tree operates in parallel, but the CPIR operation in trees is operated serially. Since this case occurs for small tree sizes, such as 8 or 16 nodes for each tree, the serial usage of CPIR method does not affect performance significantly. On the other hand, when there are less trees than the core size, the trees are operated serially while CPIR operations in each tree employ in parallel. The server computations for CPIR based range query method on octal trees are presented in Algorithm 13.

Algorithm 13 RQ-CPIR: Server computations for CPIR based Range Query on octal trees

Input $\mathcal{E} = \{e_{0,0}, \ldots, e_{d-1,6}\}$ where in $e_{s,i}$ $s = 1 \cdots d$, $i = 0 \cdots 6$ and $\mathcal{T} = \{T_1, \ldots, T_t\}$, where t: number of trees, 2^κ: number of cores
Output $R[1:t]_{d,0}$

```
  ▷ Number of trees per core:
  ω = t/2^κ
  ▷ Number of trees ≥ number of cores
  if T ≥ 2^κ then
      for p ← 0 to 2^κ − 1 in parallel do
          for k ← 1 to ω do
              R[p·ω + k]_{d,0} = OCTO-SERIAL(E, T[p·ω + k])
          end for
      end parallel for
  else
  ▷ Number of trees ≤ number of cores
      for k ← 1 to t do
          R[k]_{d,0} = OCTO-PARALLEL (E, T[p·ω + k], 2^κ)
      end for
  end if
  return R[1 : t]_{d,0}
```

3.2 Path ORAM Technique for Privacy Preserving Range Queries

Implementation of Path ORAM for privacy preserving range queries is rather straightforward compared to the CPIR model. The method does not require any change on the server side. The server is only responsible for sending all nodes in the path of the requested bucket id to the client. Similarly, the client side operations do not require any fundamental change to the Path ORAM method.

In the setup phase of the method, the binary tree structure on server which stores

the database is filled with dummy values. To place the data items into binary tree, client performs repetitive write operations using the access protocol. The properties of data item are same with CPIR method. Meanly, each item has several attributes and based on the total size of the item, it is encrypted with AES using a suitable block size. After each retrieval operation, the data is re-encrypted. To enable different ciphertext values for the same data item, in each encryption operation some random values are padded to plaintext.

For query sending step, different from the original algorithm [28], a query processing operation is added. Thus, when client wants to search for a range, the query processor finds the buckets which stores the requested items. Since the query range can map to several buckets, the client may need to perform more than read operation to retrieve the items.

Path ORAM method does not require any computation on server side. Thus, the server is only responsible for sending the path that contains the requested data and writing the path back to the tree, without any additional computations.

Chapter 4

Communication and Computational Analysis

The trivial solution to retrieve a data item or a data range from an outsourced database without leaking any information is to download the entire database and to perform the query in local. However, for large database sizes, this is not a practical solution due to excessive bandwidth usage. Instead, using Private Information Retrieval or Oblivious RAM methods can enable a more bandwidth-efficient retrieval of data by preserving its privacy. However, bandwidth usage is an important concern for both schemes, such that the cost of communication for good PIR and ORAM schemes should not exceed the cost of the trivial solution. To that end, in this chapter, we first analyze the cost of communication for the new CPIR scheme introduced in Chapter 2. Since CPIR requires processing each item of the database an analysis of computation complexity is also provided. Then, the bandwidth requirements of the new CPIR and the Path ORAM methods in privacy-preserving range queries are examined. Finally, the computational analysis of two methods are provided.

4.1 Analysis of Communication and Computation for CPIR

We start our analysis with the cost of communication and computation on the new CPIR method. In the following section, first the bandwidth requirements for octal and hexadecimal trees are explained. Later, the bandwidth usage of the scalable

method on both trees are analyzed. Finally, for the computation cost, the estimated timings of computations are provided. In the tables of this section, to differentiate the methods from each other, specific notations are used for server computations. ‘octo-old’ and ‘binary-old’ display the performance on octal and binary trees for Lipmaa’s CPIR method. ‘binary-new’, ‘octo-new’ and ‘hex-new’ show the timings for the new CPIR method on binary, octal and hexadecimal trees. Finally, ‘octo-LL’ and ‘hex-LL’ show the timing results of the implementation of Lim-Lee technique on the new CPIR method for octal and hexadecimal trees.

4.1.1 Analysis of Communication Complexity

The communication complexity of Lipmaa’s CPIR method based on binary decision diagrams scheme is sub-linear which makes it a promising scheme for PIR. In BddCPIR, the bandwidth usage is determined by two message exchanges. First one is sending the encrypted selection bits from client to server and the second one is the response of server to client as a result of CPIR computations. For a database with n files, the number of selection bits is $\log_2 n$. At the lowest level of the binary tree, when the modulus is N, the size of the encrypted selection bit is $2|N|$ due to message expansion in encryption, where $|N| = \lceil \log_2(N) \rceil$. On the next level, the size increases to $3|N|$ and at the root, it is equal to $(log_2 n+1)\cdot|N|$. Thus, the total size of the encrypted selection bits sent from client to the server is:

$$[2 + 3 + \cdots + (\log_2 n + 1)] \cdot |N|$$

After completing CPIR computations using the encrypted selection bits, the server sends a response of size $(log_2 n + 1) \cdot |N|$.

The bandwidth usage in octal and hexadecimal trees can be computed similarly. In octal trees, for each level of the tree 7 encrypted selection bits need to be sent by the client while for hexadecimal trees it is equal to 15. Since the depth of the trees are $log_8 n$ and $log_{16} n$ respectively, the total amount of bits sent to server can be formulated as follows:

$$\text{Octal}: \; [7 \cdot (2 + 3 + \cdots + (\log_8 n + 1))] \cdot |N|$$

$$\text{Hexadecimal}: \; [15 \cdot (2 + 3 + \cdots + (\log_{16} n + 1))] \cdot |N|$$

The response of the server is $(\log_8 n+1)\cdot|N|$ for octal method and $(\log_{16} n+1)\cdot|N|$ for hexadecimal method. Based on these formulations, Table 4.1 demonstrates the total bandwidth usage for various database sizes, when $|N| = 1024$. From the values in Table 4.1, it is not immediate to identify the best scheme. For smaller datasets, binary tree implementation is advantageous. However, when database size is larger than 4096 items, the communication cost of the octal case becomes advantegous. Hexadecimal tree has the maximum bandwidth usage, but the computational cost of hexadecimal tree can balance its higher bandwidth usage.

n	Database size	binary	octal	hex
2	2048	4096	-	-
4	4096	8192	-	-
8	8192	13312	16384	-
16	16384	19456	-	32768
32	32768	26624	-	-
64	65536	34816	38912	-
128	131072	44032	-	-
256	262144	54272	-	79872
512	524288	65536	68608	-
1024	1048576	77824	-	-
2048	2097152	91136	-	-
4096	4194304	105472	105472	142336
8192	8388608	120832	-	-
16384	16777216	137216	-	-
32768	33554432	154624	149504	-
65536	67108864	173056	-	220160
131072	134217728	192512	-	-
262144	268435456	212992	200704	-
524288	536870912	234496	-	-
1048576	1073741824	257024	-	313344
2097152	2147483648	280576	259072	-

Table 4.1: The total bandwidth usage in number of bits for changing database sizes, where $\lceil \log_2(N) \rceil = 1024$

Bandwidth Usage In Scalable CPIR

The scalable approach requires to send additional selection bits for the subtrees, which adds an overhead in bandwidth usage. In [30] the number of bits sent from client to server in scalable case is formulated as

$$(2\mu + (2^g - 1) \cdot (3 + 4 + \cdots + (l + 2)))|N|$$

where l is the depth of the subtree, $\mu = 2^{m-y\cdot l}$ is the number of subtrees, while 2^m is equal to the number of items in original tree. Also, g is the number of encrypted selection bits prepared by client for each level which is 1, 3, and 4 for binary, octal, and hexadecimal trees, respectively. Depending on the tree used, namely binary, octal, or hexadecimal, y also becomes 1, 3 and 4, respectively. Finally N is the modulus of Damgård-Jurik cryptosystem. Different from the normal case, the size of the response sent from server to the client is $(l + 2) \cdot |N|$.

The costs of overall communication in scalable case for octal and hexadecimal trees are presented in Table 4.2 and Table 4.3, respectively. Accordingly, smaller subtree sizes have significantly greater bandwidth requirements. On the other hand, the advantage of small subtrees in computational cost, as showed in Chapter 5, causes a trade off for the optimal subtree size in the scalable method.

2^m	Database size	l	Bandwidth usage
4096	4194304	2	185344
		3	107520
32768	33554432	2	1102848
		3	222208
		4	151552
16777216	17179869184	2	536925184
		3	67200000
		4	8523776
		5	1234944
		7	326656

Table 4.2: Total bandwidth usage in scalable CPIR for octal trees (number of bits), where $\lceil \log_2(N) \rceil = 1024$

2^m	Database size	l	Bandwidth usage
4096	4194304	2	144384
65536	67108864	2	635904
		3	222208
16777216	17179869184	2	134329344
		3	8578048
		4	806912
		5	423936

Table 4.3: Total bandwidth usage in scalable CPIR for hexadecimal trees (number of bits), where $\lceil \log_2(N) \rceil = 1024$

4.1.2 Analysis of Computational Complexity

In this section, we provide a theoretical analysis to show the improvement in CPIR computations by the improvements presented in Chapter 2. For calculations of estimated timing results of the new scheme, we adopted the formulations given in [30].

Since the computations are dominated by modular exponentiation operations, we measured the time spent for one exponentiation on different levels of the tree which is represented as τ_s, where s indicates the level of the tree. In the original CPIR each node requires 3 modular exponentiations for the binary tree and 9 modular exponentiation for the octal tree. Then the total time spent for one node is $t_s^b = 3 \cdot \tau_s$ and $t_s^o = 9 \cdot \tau_s$ [30]. However, in the new CPIR scheme, since the number of exponentiations are reduced by one for each node, the total time becomes $t_s^b = 2 \cdot \tau_s$, $t_s^o = 8 \cdot \tau_s$ and $t_s^h = 16 \cdot \tau_s$ for binary, octal and hexadecimal trees, respectively. For the sink nodes, the time should be calculated as $t_1^b = \tau_s$, $t_1^o = 7 \cdot \tau_s$ and $t_1^h = 15 \cdot \tau_s$. Based on this information and using the derivations of [30], we can compute the total cost of retrieval operation for the new CPIR method as follows:

$$
\begin{aligned}
T_{2^m} &= \sum_{s=2}^{m} 2^{m-s} t_s^b + 2^{m-1} t_1^b \text{ for } m \geq 1 \\
T_{8^m} &= \sum_{s=2}^{m} 8^{m-s} t_s^o + 8^{m-1} t_1^o \text{ for } m \geq 1 \\
T_{16^m} &= \sum_{s=2}^{m} 16^{m-s} t_s^h + 16^{m-1} t_1^h \text{ for } m \geq 1.
\end{aligned} \tag{4.1}
$$

Similar to this formulation, to compute estimated timing values of the Lim-Lee extension, we measured the time spent for the Lim-Lee operation in each level of the tree, which is represented as τ_s^{LL}. Using this measurement, we compute the total estimated time spent for the new CPIR using the Lim-Lee acceleration technique in the following equations:

$$T^{LL}_{8^m} = \sum_{s=1}^{m} 8^{m-s}\tau^{LL}_s \text{ for } m \geq 1$$

$$T^{LL}_{16^m} = \sum_{s=1}^{m} 16^{m-s}\tau^{LL}_s \text{ for } m \geq 1 \quad (4.2)$$

Using these formulas, the estimated cost of computation for several data sizes is calculated. Table 4.4 demonstrates the results of computations.

No.	binary		octal			hexadecimal		
items	old	new	old	new	LL	old	new	LL
2	4	1	-	-	-	-	-	-
4	26	15	-	-	-	-	-	-
8	98	60	13	10	5	-	-	-
16	280	175	-	-	-	24	21	7
32	703	446	-	-	-	-	-	-
64	1625	1039	155	126	50	-	-	-
128	3560	2284	-	-	-	-	-	-
256	7549	4853	-	-	-	483	432	125
512	15670	10088	1373	1131	432	-	-	-
1024	32069	20663	-	-	-	-	-	-
2048	65057	41938	-	-	-	-	-	-
4096	131248	84631	11239	9274	3526	7980	7152	2036
8192	263884	170188	-	-	-	-	-	-
16384	529436	341489	-	-	-	-	-	-
32768	1060856	684300	90346	74573	28333	-	-	-
65536	2124015	1370135	-	-	-	128153	114880	32654

Table 4.4: Estimated timings of server side computations for different tree types in ms, where $|N| = 1024$

In our method, we improved the performance of the CPIR scheme by utilizing parallel algorithms. Therefore, in theoretical analysis, the effect of parallelism on the new CPIR method is observed. Based on the equations provided in [30], the computation cost of the new CPIR model on binary trees can be computed by following equation:

$$T^p_{2^m} = T_{2^\sigma} + \sum_{s=\sigma+1}^{m} \lceil 2^{\sigma-s} \cdot 2 \rceil \tau_s, \quad (4.3)$$

where

$$\sigma = \begin{cases} m - \kappa & m \geq \kappa \\ 0 & \text{otherwise.} \end{cases}$$

In Equation 4.3, 2^κ is the number of cores where $m \geq \kappa \geq 0$. Based on this equation, Table 4.5 shows the calculated timing values for CPIR on binary trees.

	n	Number of Cores			
		4	8	16	32
binary old	64	449	275	205	180
	128	952	552	378	308
	256	1974	1095	695	521
	512	4037	2165	1286	886
	4096	33059	16840	8858	4964
binary new	64	296	209	179	171
	128	622	399	312	282
	256	1285	765	542	455
	512	2618	1668	956	733
	4096	21347	11395	5971	3545

Table 4.5: Estimated timing values for binary trees in ms

In an octal tree implementation where the number of items in database is 8^m, c is the number of cores and $\lambda = \lceil \log_8 c \rceil$, the formula for estimated timing of the computations can be given as follows

$$T^p_{8^m} = \left\lceil \frac{8^\lambda}{c} \right\rceil T_{8^\sigma} + \sum_{s=\sigma+1}^{m} \left\lceil \frac{8^{m-s} \cdot 8}{c} \right\rceil \tau_s, \tag{4.4}$$

where

$$\sigma = \begin{cases} m - \lambda & m \geq \lambda \\ 0 & \text{otherwise.} \end{cases}$$

Using Equation 4.4, estimated timing values for various database sizes on octal trees are demonstrated in Table 4.6.

Similar to octal case, in hexadecimal tree implementation for a database of 16^m items and c cores,$\lambda = \lceil \log_{16} c \rceil$. Based on these values, the timings can be calculated using Equation 4.5. Table 4.7 presents the estimated cost of server computations on hexadecimal trees for several database sizes.

$$T^p_{16^m} = \left\lceil \frac{16^\lambda}{c} \right\rceil T_{16^\sigma} + \sum_{s=\sigma+1}^{m} \left\lceil \frac{16^{m-s} \cdot 16}{c} \right\rceil \tau_s, \tag{4.5}$$

	n	Number of cores			
		4	8	16	32
octo old	64	43	25	13	10
	512	355	185	95	58
	4096	2831	1429	722	383
octo new	64	32	16	12	9
	512	283	141	78	47
	4096	2318	1159	594	311

Table 4.6: Estimated timing values for octal trees in ms

where

$$\sigma = \begin{cases} m - \lambda & m \geq \lambda \\ 0 & \text{otherwise.} \end{cases}$$

n	Number of Cores			
	4	8	16	32
256	108	54	27	17
4096	1788	894	447	231

Table 4.7: Estimated timing values for hexadecimal trees in ms

4.2 Analysis of Communication and Computation for Privacy Preserving Range Queries

Both Path ORAM and CPIR guarantees the security of privacy-preserving range queries by guaranteeing data confidentiality, query confidentiality and hiding the query access patterns. However, the performance of the two methods differs in terms of efficiency, which is a crucial aspect for any application over outsourced data. Hence, the rest of this section provides a qualitative comparison of CPIR and Path ORAM methods in terms of bandwidth usage and computation cost.

CPIR technique for Privacy Preserving Range Queries: The proposed model for privacy preserving range queries using Private Information Retrieval methods requires employing computations on multiple trees. Although the proposed method achieves good performance results, the size of communication is an important issue for the efficiency of the model which requires a detailed inspection of bandwidth requirements.

In our model, two message exchanges determine the cost of communication. First one is sending the selection bits prepared on client side to server side. Although there are multiple trees, since in each tree the corresponding nodes preserve the order of buckets, generating the selection bits only for one tree and applying them repetitively on each tree can provide the intended selection. However, the size of the selection bits is still an important issue to decide the optimal bucket size for changing database sizes. Furthermore, the size of the server response is determined by the number of trees, since each tree computes a response for the corresponding bucket. Therefore, the server response needs to be multiplied by the number of trees to compute the total bandwidth usage. Section 4.1 provides a detailed analysis on the calculation of the number of bits exchanged for octal and hexadecimal trees in the CPIR method. Based on the provided calculation methods, an analysis of bandwidth usage in CPIR scheme is presented on Figure 4.1 for octal and hexadecimal tree implementations to retrieve one bucket. Since the bandwidth usage of a good CPIR scheme needs to be less than the size of the database, the plots demonstrate the ratio of exchanged bits to database size with respect to increasing database size. The results suggest that increasing database size favors using more buckets.

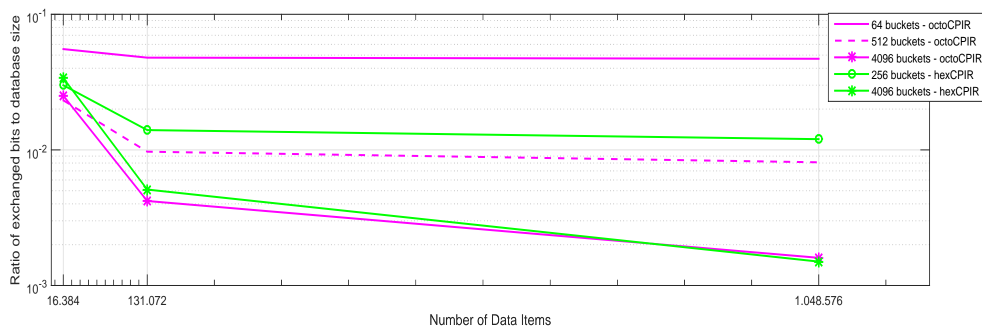


Figure 4.1: Ratio of exchanged bits to database size in CPIR based technique on octal and hexadecimal trees for different number of buckets

Path ORAM technique for Privacy Preserving Range Queries: While Path ORAM is a simple and fast method for the retrieval of encrypted data, since the method requires repetitive path reading and writing, a careful construction is required to optimize the bandwidth usage. Similar to CPIR, the bandwidth usage is determined by two messages. The first one is reading the path of the intended

bucket from the database, which requires server to send $Z \log T$ blocks, where Z is the number of blocks within each bucket, T is the number of buckets and $logT$ is the length of the path from the retrieved node to the root node. The second message is writing the accessed path back to tree, which requires to send $Z \log T$ blocks of data back to the server, as well. Therefore, the total cost of communication for Path ORAM can be summarized as $2Z \log T$. In computations of the bandwidth usage, Z is fixed to 4 to keep in line with original Path ORAM method [28]. Since the size of a block is determined by the number of data items assigned into it, the size of the data item is important for bandwidth computations. In our analysis, each data item is considered as a tuple with 5 integer attributes stored including the primary key. Since each integer is 32-bit, the total size of a tuple becomes 160 bits. Furthermore, the data needs to be stored in encrypted form. Using AES encryption, a 160 bit plaintext value maps to 256 bit ciphertext because of the block size of encryption.

Based on the above explanations, the bandwidth usage in Path ORAM method is presented in Figure 4.2 along with CPIR technique. For compatibility of CPIR and Path ORAM method, the number of buckets utilized in Path ORAM is selected close to the number of buckets in CPIR. Similar to CPIR, a higher number of buckets is advantageous in terms of bandwidth usage for large database sizes.

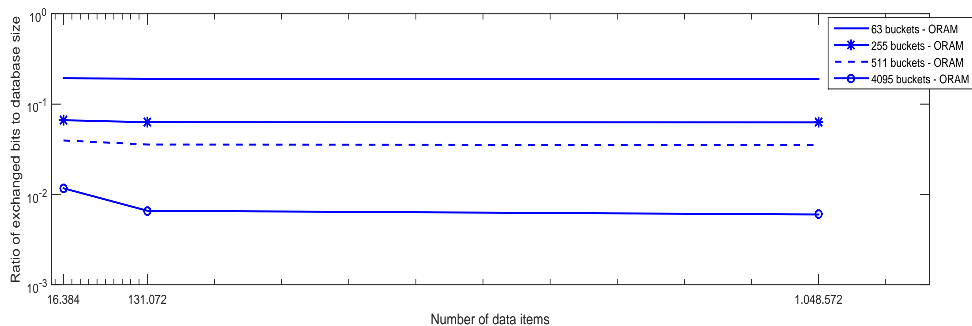


Figure 4.2: Ratio of exchanged bits to database size in Path ORAM based technique for different number of buckets

Based on Figure 4.1 and Figure 4.2, a comparison between CPIR and Path ORAM technique can be made. The results show that as database size increases, the decline of the ratio for bandwidth usage in the CPIR based method is clear for the same number of buckets. An important issue in the provided analyses is that

they demonstrate bandwidth usage to retrieve one bucket from the trees. However, querying for a range may require to retrieve more than one buckets. Especially, utilizing a method to introduce false positives in buckets for privacy concerns, which is Controlled Diffusion mechanism in our case, causes the increase of the number of retrieved buckets for range queries. Thus, in Figure 4.3, an analysis of the bandwidth usage in CPIR and Path ORAM based range query scheme for increasing number of buckets is presented. In figure, the bandwidth usage is represented as the ratio of exchanged bits to database size. To compare the performance of hexadecimal and octal tree implementation of CPIR, the results are provided separately for the two implementations. According to the results, there is not a significant difference in bandwidth consumption between octal tree and hexadecimal tree CPIR technique for increasing number of buckets. However, the bandwidth usage of the Path ORAM based technique clearly are much higher compared to the CPIR methods.

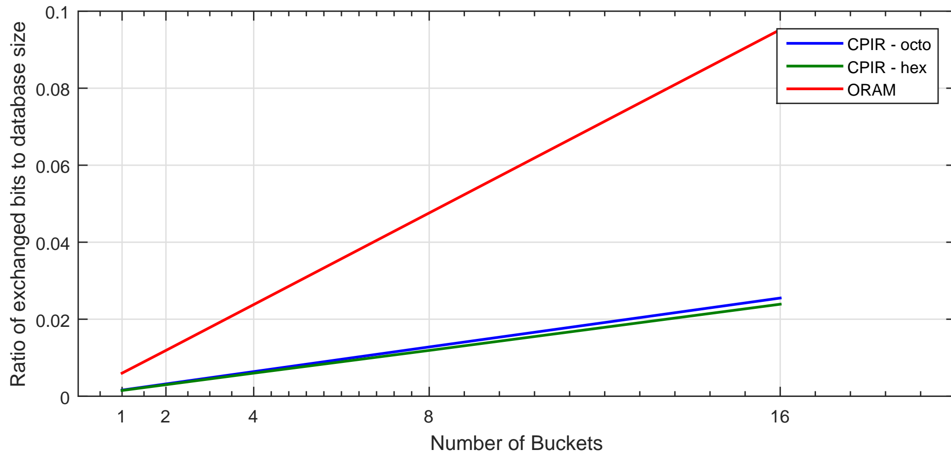


Figure 4.3: The bandwidth usage in the CPIR and ORAM method to retrieve multiple buckets

4.2.1 Computational Complexity Analysis

Apart from the simplicity of implementation, Path ORAM is a fast method in encrypted data retrieval; since the server does not perform any further computation, but only returns the requested path to the client. Therefore, there is no burden on the server in terms of computational complexity. The cost of computation on the client side for one access is $O(logK) \cdot \omega(1)$, where K is the number of total blocks

outsourced to the server. The cost contains decryption, shuffling and re-encryption operations on $log(K)$ blocks in each access.

On the other hand, Lipmaa's CPIR requires $O(n)$ computation cost on server side and $O(log^2 n)$ computation cost on client side. Furthermore, since each bucket is partitioned into several trees, the cost of computation on the server side will be increased by a constant magnitude. However, although in terms of asymptotic complexity CPIR scheme is slower, by changing the structure of the tree and adding parallelism into the implementation [30], CPIR can be a practical scheme for privacy preserving range queries.

Based on the analysis of communication and computation, in terms of asymptotic complexity, Path ORAM gives better results. However carefully selection of bucket sizes and utilizing parallellization techniques makes CPIR a practical scheme for range queries, as well. Apart from the concerns of bandwidth and computation performance, the requirement of additional storage, the private stash, on the client side makes Path ORAM less advantageous against CPIR. On the CPIR method, the client does not need a storage but only responsible for computation of encrypted selection bits once. The encrypted selection bits can be used on each tree on the server side without a re-computation. Furthermore, the existence of a private stash may create difficulties in a multi-client scenario. It requires a separate stash for each client. When a client performs a read or write operation, he needs to write back the retrieved items to a new path. Thus, in each access operation, the clients need to inform each other for the new locations, which causes additional cost on operations. On the other hand, a multi-client implementation of CPIR scheme does not affect the cost of operations.

Chapter 5

Implementation Results

To show the correctness of the improvements on the BddCPIR method, all of the proposed schemes, serial, parallel, Lim-Lee technique and scalable technique, are implemented. Furthermore, these schemes adapted to range query scheme to compare its performance with Path ORAM. The programming language used for all implementations is C++. For big integer operations GMP, The GNU Multiple Precision Arithmetic Library, is utilized. Parallel operations are handled by OpenMP API. The experiments are employed on two different computers. The first machine runs 64 bit Ubuntu 12.04 operating system. It is a 6-core platform where each core is an Intel Xeon CPU E5-1650 v2 operating at 3.50 GHz. We used 4 of the cores in the experiments which show the improvements on the BddCPIR method. The second machine has 64 bit CentOS 6.5 as operating system. It has 30 cores where each one is an Intel Xeon CPU E7-4870 v2 operating at 2.30 GHz. This machine is utilized to show the scalability of the new CPIR scheme on large database sizes and to compare the performance of CPIR and Path ORAM.

5.1 Timing results for the new CPIR

The implementation includes binary, octal and hexadecimal approaches for the new CPIR method. The experiments for the usage of the Lim-Lee technique is only applied on octal and hexadecimal trees. For the scalable method, the results of the new CPIR method and the new method with Lim-Lee technique are demonstrated separately. To show the improvement in performance the results on binary and

octal trees for the original method [30] are included in each table. To clarify the notation in tables for server computations, 'octo-old' and 'binary-old' display the performance on octal and binary trees for Lipmaa's CPIR method. 'binary-new', 'octo-new' and 'hex-new' show the timings for the new CPIR method on binary, octal and hexadecimal trees. Finally, 'octo-LL' and 'hex-LL' show the timing results of the implementation of Lim-Lee technique on the new CPIR method for octal and hexadecimal trees.

The proposed methods in this work change the computations on the server side, but the computations on client side is the same as the original method. Therefore, the performance measurements for the client side operations are categorized based on the type of the tree used in implementation, as binary, octal and hexadecimal.

5.1.1 Timings for Client Side Computations

On the client side, two operations are important in measuring the performance of the CPIR. These are encryption of the selection bits and iterative decryption of server response. As proposed in Section 2.5, the encryptions are performed using parallel algorithms. Since decryption operations are dependent on each other, it is performed serially.

The timings for the client operations of CPIR on binary, octal and hexadecimal trees for various data sizes are presented in Table 5.1. The results show that the message expansion property of the Damgård-Jurik cryptosystem benefits shallower trees in encryption and decryption operations. Namely, increasing the number of items in database, hexadecimal tree has the best performance results and octal tree, obviously, performs better than binary tree implementation.

5.1.2 Timings for Server Side Computations

The server operations are the dominant part of the CPIR method in terms of performance since it requires employment on each file of the database. However, this performance drawback can be handled by utilization of parallel methods in implementation. The CPIR method is suitable for parallelism since it involves repetitive independent operations. In our experiments, we tested the proposed CPIR method both for serial and parallel versions including the scalable case. Further, we observed

No. of Items	Client Encryption (ms)			Client Decryption (ms)		
	binary	octo	hex	binary	octo	hex
2	2	-	-	2	-	-
4	7	-	-	5	-	-
8	19	5	-	11	2	-
16	34	-	8	19	-	2
32	55	-	-	30	-	-
64	78	19	-	41	5	-
128	114	-	-	58	-	-
256	151	-	40	78	-	5
512	200	48	-	102	10	-
1024	257	-	-	130	-	-
2048	324	-	-	163	-	-
4096	416	93	93	200	18	10
32768	-	197	-	-	28	-
65536	-	-	176	-	-	18

Table 5.1: Timings of client computation for encryption of selection bits and decryption of results with $|N| = 1024$

the change in performance by increasing the number of cores.

Serial Case

In the experiments, we first tested the performance of the new method with the original BddCPIR method. Since the original scheme is a serial implementation, Table 5.2 lists the timing values without utilizing any parallel method. Since the message size increases in each level, the performance of binary tree implementation becomes prohibitively poor for large data sizes. On the other hand, octal and hexadecimal trees gain significant advantage on the original BddCPIR scheme.

As mentioned in Chapter 2, another dominant factor on the performance of CPIR is the number of exponentiation operations. Comparing the values of 'binary-old' and 'binary-new' in Table 5.2, we can observe the improvement by reducing the number of exponentiations. Furthermore, the effect of simultaneous exponentiation in computation time by utilization of the Lim-Lee multi-exponentiation algorithm is apparent in timings. For example, in octal case a speed up of $96,808/35,691 = 2.71$ is achieved for a database with 4096 files compared to the performance of Lipmaa's CPIR method on octal trees.

The overall change in the performance by usage of shallow trees and the new CPIR method is remarkable. For a database size of 4096, while BddCPIR method's

cost is 135249 ms, the hexadecimal tree implementation by utilizing Lim-Lee multi-exponentiation has a cost of 2002 ms which is equal to 68 times acceleration.

No of.	Server Computations - Serial (ms)						
items	binary-new	binary-old	octo-LL	octo-new	octo-old	hex-LL	hex-new
2	2	5	-	-	-	-	-
4	16	28	-	-	-	-	-
8	62	102	5	11	16	-	-
16	181	292	-	-	-	7	25
32	460	730	-	-	-	-	-
64	1,067	1,682	61	135	182	-	-
128	2,358	3,683	-	-	-	-	-
256	4,999	7,786	-	-	-	128	484
512	10,359	16,167	535	1,205	1,580	-	-
1,024	21,260	33,053	-	-	-	-	-
2,048	43,124	67,141	-	-	-	-	-
4,096	87,111	135,249	4,368	9,869	12,052	2,002	7,703
32,768	-	-	35,691	79,484	96,808	-	-
65,536	-	-	-	-	-	32,385	123,540

Table 5.2: Timings of server computation in serial case for binary, octal and hexadecimal trees with $|N| = 1024$

Parallel Case

The structure of CPIR algorithms is suitable to employ computations in parallel. Therefore, using the parallel algorithms defined in Section 2.5, we utilized parallelism on the new CPIR method. As the results of Table 5.3 imply, parallel algorithms do not only benefit the new CPIR method, but it also improves the original BddCPIR method. Especially for larger databases, the effect of parallel implementation against serial implementation is obvious. For example, in a database with 64 files, the speedup of the parallel implementation of the new CPIR method using octal trees (octo-new) over its serial implementation is approximately 135 / 47 = 2.87. However, when the number of items in database are increased to 4096, the acceleration increases up to 9,869 / 2,603 = 3.79.

As the actual timing results show, adding parallelism into the new CPIR method makes it a more feasible scheme compared to Lipmaa's BddCPIR scheme. To support this claim we can analyze the performance of the BddCPIR and the new CPIR on hexadecimal tree by utilizing multi-exponentiation technique. It requires 135249 ms to complete computations of 4096 files for BddCPIR in serial computation while

the new CPIR on hexadecimal tree can finish it in 573 ms, which is equal to an improvement by 135249 / 573 = 236, approximately.

No of.	Server Computations - Parallel (ms)						
items	binary-new	binary-old	octo-LL	octo-new	octo-old	hex-LL	hex-new
2	2	5	-	-	-	-	-
4	13	23	-	-	-	-	-
8	34	61	6	4	6	-	-
16	84	138	-	-	-	7	8
32	164	289	-	-	-	-	-
64	338	566	35	47	58	-	-
128	676	1,138	-	-	-	-	-
256	1,390	2,282	-	-	-	56	139
512	2,814	4,551	186	329	407	-	-
1,024	5,689	9,063	-	-	-	-	-
2,048	11,444	18,076	-	-	-	-	-
4,096	22,938	36,039	1,219	2,603	3,199	573	2,031
32,768	-	-	9,336	20,817	25,409	-	-
65,536	-	-	-	-	-	8,448	32,370

Table 5.3: Timings of server computation in parallel case for octal and hexadecimal trees

Scalable Case

For larger data sizes, we conducted experiments using the scalable approach for CPIR proposed in Section 2.6. To compare the performance of octal and hexadecimal trees we used a database of 4096 files with subtree sizes 8, 64 and 512 for octal and 16 and 256 for hexadecimal case. Furthermore, to observe larger data sizes, we tested databases with 32768 and 65536 files for octal and hexadecimal trees respectively. The results of experiments are displayed on Table 5.4 and Table 5.5.

No. of	Size of	Server Computation (ms)		
items	Subtree	octo-LL	octo-new	octo-old
4096	64	799	1,755	1,803
	512	1,256	2,688	2,956
32768	64	5,588	13,146	13,259
	512	6,048	14,127	13,347
	4096	9,324	21,534	23,528

Table 5.4: Timings of server computation in scalable case for octal trees

Analyses of Table 5.4 and Table 5.5 suggest that when the size of subtree is smaller, the performance of the scalable scheme is better. However, when subtree

No. of items	Size of Subtree	Server Computation (ms)	
		hex-LL	hex-new
4096	16	440	1,665
	256	567	2,094
65536	16	6,699	26,250
	256	6,832	26,699
	4096	8,515	33,404

Table 5.5: Timings of server computation in scalable case for hexadecimal trees

size gets small, the number of subtrees increases which adversely affects the bandwidth of the scheme. Another important issue in scalable case is the performance of the new CPIR with Lim-Lee multi exponentiation method. Applying Lim-Lee's technique in subtree collapsing operations improves the performance of scalable method significantly. Using Lim-Lee's method, the performance of the new CPIR method in octal trees for 4096 files is improved by 2956 / 1256 = 2.35 times approximately. The speed up value increases to 2094 / 567 = 3.69, when the method is implemented on hexadecimal trees for the same database size.

Timings on 30-Core Computer Platform

Finally, in order to demonstrate the scalability of the new CPIR scheme on large database sizes, we conducted experiments on a machine which utilizes 30 cores. The observations include both the parallel and scalable approaches of the new CPIR on octal and hexadecimal trees. Table 5.6 and Table 5.7 list the timings of parallel implementation for various number of cores. The improvement in timings are in line with the change in the number of cores. For example, in Table 5.6 for 32768 items, the computation time of the new CPIR method utilizing Lim-Lee technique is 49671 ms for 1 core. When the number of cores doubled, the time decreases to 25031 ms, almost half of the former one. Similarly, for 4 cores, the computation time reduces to 12656 ms which is again half of the result with 2 cores. Table 5.8 and Table 5.9 show the results of computations for the scalable method on octal and hexadecimal trees with large database sizes.

No. of items	Number of Cores																	
	1			2			4			8			16			30		
	octo LL	octo new	octo old	octo LL	octo new	octo old	octo LL	octo new	octo old	octo LL	octo new	octo old	octo LL	octo new	octo old	octo LL	octo new	octo old
8	7	15	19	7	8	11	7	4	6	7	3	4	7	3	4	7	3	4
64	85	190	234	57	95	121	42	47	65	35	24	37	35	27	52	35	27	53
512	748	1,692	2,065	407	847	1,044	237	423	534	152	213	278	123	153	262	123	143	248
4096	6,106	13,880	16,868	3,117	6,938	8,472	1,625	3,471	4,257	878	1,744	2,154	537	980	1,369	447	723	1,117
32768	49,225	111,759	135,738	24,630	55,584	68,022	12,461	27,800	34,114	6,339	13,927	17,088	3,350	7,204	9,070	2,307	4,697	6,330
262144	*	*	*	195,484	446,293	542,879	98,421	223,470	271,456	49,372	112,311	136,266	24,919	56,751	68,977	15,910	33,829	44,487
2097152	*	*	*	*	*	*	*	*	*	400,844	971,242	1091,510	199,177	485,164	555,415	113,385	274,862	317,615

Table 5.6: Timings of server computation in octal trees for various number of cores in ms

No. of items	Number of Cores											
	1		2		4		8		16		30	
	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL
16	32	9	17	9	8	9	5	9	6	9	6	9
256	653	171	326	102	164	67	82	50	41	43	48	43
4096	10,775	2,818	5,399	1,449	2,708	762	1,359	419	680	250	541	217
65536	173,469	45,241	86,540	22,765	43,510	11,399	21,770	5,794	10,969	2,974	6,696	2,004
1048576	2,752,900	719,958	1,383,160	361,099	697,282	181,716	348,686	90,749	174,412	45,579	101,279	28,561

Table 5.7: Timings of server computation in hexadecimal trees for various number of cores in ms

No. of items	Subtree size	Number of Cores																	
		1			2			4			8			16			30		
		octo old	octo new	octo LL	octo old	octo new	octo LL	octo old	octo new	octo LL	octo old	octo new	octo LL	octo old	octo new	octo LL	octo old	octo new	octo LL
4096	64	9.14	9.12	8.98	4.63	4.6	4.52	2.42	2.36	1.05	1.22	1.19	0.56	0.66	0.64	0.33	0.57	0.56	0.29
	512	11.88	11.71	10.42	6.02	5.92	5.21	3.96	3.60	1.64	2.01	1.81	0.88	1.28	1.01	0.54	1.07	0.83	0.46
32768	64	70.44	70.62	70.49	35.46	35.36	35.35	17.87	17.72	7.53	8.87	8.86	3.76	4.51	4.48	1.95	3.29	3.02	1.40
	512	73.26	73.1	71.65	36.82	36.66	36.05	19.34	19.07	8.16	9.64	9.52	4.12	5.11	4.84	2.17	3.72	3.53	1.62
	4096	95.44	93.78	82.75	48.03	47.01	41.03	31.72	28.88	12.52	15.93	14.51	6.37	8.48	7.43	3.36	5.86	5.31	2.51
262144	64	*	*	*	282.83	280.89	117.44	140.35	140.06	59.55	70.17	70.41	29.57	35.05	35.04	14.82	23.81	23.27	9.77
	512	*	*	*	282.16	283.74	118.47	141.26	142.38	59.91	71.14	71.35	28.88	35.74	35.46	15.08	23.12	23.20	9.74
	4096	*	*	*	293.12	292.90	127.77	153.73	152.92	64.31	77.13	76.06	32.20	39.10	38.89	16.32	26.13	24.16	10.55
	32768	*	*	*	383.80	375.76	197.14	252.98	231.40	98.88	126.65	115.89	49.57	64.14	58.57	25.17	42.93	38.11	16.05

Table 5.8: Timings of server computation in octal tree with scalable method for various number of cores in sec

Number of Items	Subtree Size	Number of Cores											
		1		2		4		8		16		30	
		hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL
65536	256	143.75	36.23	70.82	18.17	35.51	9.06	17.85	4.60	8.90	2.35	6.32	1.81
	4096	179.86	45.09	89.79	22.40	45.70	11.34	22.59	5.73	11.37	3.00	7.71	2.17
1048576	256	*	*	*	*	558.41	143.05	280.75	71.48	139.79	35.77	94.11	23.42
	4096	*	*	*	*	567.29	144.64	284.46	72.30	142.25	36.38	92.14	23.43
	65536	*	*	*	*	721.67	180.33	360.36	90.42	180.91	45.43	115.26	29.41

Table 5.9: Timings of server computation in hexadecimal tree with scalable method for various number of cores in sec

5.2 Timing results for the Privacy Preserving Range Queries

For the range query experiments, we implemented the new CPIR model both in serial and parallel versions and the Path ORAM technique based on the algorithm in [28]. As data set, to keep in line with the experiments of the bucketization method in [18], the Lineitem table of TPCH benchmark is used [29] which is a common benchmark for the evaluation of database management systems. Since the table contains more than 6 million data entries, random subsets of data are created for experiments. Each data set contains 128, 1024 and 16384 entries respectively. To evaluate multi-dimensional range queries, four integer attributes of the Lineitem table - Quantity, Linenumber, ExtendedPrice and Tax- are selected with primary key PartKey-SuppKey. Further, to apply multi-dimensional range queries, query sets are generated within the boundaries of each dataset. Since the confidentiality of data is a requirement in privacy preserving range queries, each data tuple is encrypted for server storage using AES with 256-bit block size. For the operations of Damgård-Jurik cryptosystem, 1024-bit modulus is used to provide 80-bit security.

Our first experiment compares the performance of the new CPIR and Path ORAM in terms of client computation cost. For the CPIR method we measured the performance on octal and hexadecimal trees by utilizing the new CPIR method and its Lim-Lee extension. Table 5.10 and Table 5.11 give the measurements of client side computations for octal and hexadecimal case, respectively. To make Path ORAM scheme consistent with the CPIR scheme, we tested the schemes on the same bucket sizes. Considering the current capabilities of a clients machine, in CPIR timings 4 threads are utilized to compute client side operations. The results show the average time for processing a query. AES encrypt and decrypt operations comprise majority of computation in Path ORAM. In the CPIR technique, exponentiation of large numbers during encryption of selection bits and decryption of query response put burden on computation cost. As the computational analysis of the two techniques suggests, Path ORAM performs better than CPIR. According to results, in CPIR method, for large datasets hexadecimal implementation is advantageous. The better performance of hexadecimal tree is a result of decryption operations. Meanly,

for example, for 16384 items with 4096 buckets, octal tree implementation requires decryption of 512 trees on average while in hexadecimal tree this number is around 256 trees. Therefore, in that case while the total cost of client is 208 ms for octal tree, it reduces to 107 ms for hexadecimal trees.

n	Number of buckets							
	CPIR				Path ORAM			
	8	64	512	4096	7	63	511	4095
128	5	23	-	-	0.17	0.20	-	-
1024	16	22	62	-	0.52	0.30	0.40	-
16384	208	96	71	129	6.20	2.10	1.00	0.40

Table 5.10: Timings of client computation for CPIR and Path ORAM method in octal case utilizing 4 cores (in ms)

n	Number of buckets					
	CPIR			Path ORAM		
	16	256	4096	15	255	4095
128	9	-	-	0.17	-	-
1024	13	39	-	0.51	0.30	-
16384	107	53	112	6.40	1.00	0.40

Table 5.11: Timings of client computation for CPIR and Path ORAM method in hexadecimal case utilizing 4 cores (in ms)

Although the measurements of client side operations are performed on 4 cores for CPIR method, the development in computation technologies may lead to utilize computers with more cores for users. Thus, we, also, measured the cost of client side operations using larger core sizes. The results for encryption and decryption operations are provided in Table 5.12 and Table 5.13 separately.

The second experiment is based on the cost of the server side operations. Since the Path ORAM method does not require any computation on server side, we conducted experiments only for the CPIR method. Our measurements include the timing results for serial and parallel implementation of the new CPIR method on octal and hexadecimal trees. Table 5.14 and Table 5.15 demonstrate the average time spent to retrieve one bucket for a query in serial method utilizing octal and hexadecimal trees in ms, respectively. However, a range query may require to retrieve multiple buckets which means spending more time for server side operations. The results show that while the data size increases, the hexadecimal implementation gains advantage over the octal tree implementation. Furthermore, usage of the

		Client Encryption					Client Decryption				
		Number of cores					Number of cores				
n	No. buckets	1	2	4	8	16	1	2	4	8	16
128	8	15	9	4	2	2	9	4	2	2	2
	64	76	41	22	11	8	7	7	7	7	7
1024	8	15	9	4	2	2	68	34	17	9	4
	64	76	41	22	11	9	27	13	7	7	7
	512	227	128	65	32	30	14	14	14	14	14
16384	8	15	9	4	2	2	1099	549	274	138	69
	64	76	41	22	11	9	424	213	106	53	27
	512	227	128	65	32	30	113	56	28	14	14
	4096	512	281	144	73	62	25	25	25	25	25

Table 5.12: Timings of client computation for the CPIR method in octal case utilizing multiple cores (in ms)

		Client Encryption					Client Decryption				
		Number of cores					Number of Cores				
n	No. buckets	1	2	4	8	16	1	2	4	8	16
128	16	32	17	9	4	2	4	2	2	2	2
1024	16	32	17	9	4	2	34	17	9	4	2
	256	163	85	44	22	11	7	7	7	7	7
16384	16	32	17	9	4	2	549	274	137	69	34
	256	163	85	44	22	11	106	53	27	13	7
	4096	487	258	130	65	32	17	17	17	15	15

Table 5.13: Timings of client computation for the CPIR method in hexadecimal case utilizing multiple cores (in ms)

multi-exponentiation algorithm reduces the cost significantly. For example, for a data set of 16384 items with 4096 buckets, Lim-Lee method provides a speed up by $10781/2823 = 3.81$.

n	Number of buckets							
	8		64		512		4096	
	octo-new	octo-LL	octo-new	octo-LL	octo-new	octo-LL	octo-new	octo-LL
128	60	28	189	85		-		-
1024	479	227	759	338	1689	743	-	-
16384	7656	3647	12117	5425	13518	5944	13819	6067

Table 5.14: Timings of serial server computation for the new CPIR method in octal case in ms

Table 5.16 and Table 5.17 show the timings for parallel implementation of CPIR technique to retrieve one bucket on octal and hexadecimal trees, respectively. The results are similar with serial implementation. Increasing data size benefits hexadecimal tree implementation. To show the consistency between the serial and parallel

n	Number of buckets					
	16		256		4096	
	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL
128	64	17	-	-	-	-
1024	513	139	652	172	-	-
16384	8192	2245	10457	2760	10781	2823

Table 5.15: Timings of serial server computation for the new CPIR method in hexadecimal case in ms

implementation, the results of parallel implementation utilizing 1 core is added, as well.

n	No. buckets	Number of cores 1		2		4		8		16	
		octo-new	octo_LL	octo-new	octo-LL	octo-new	octo-LL	octo-new	octo-LL	octo-new	octo-LL
128	8	60	28	30	14	15	7	15	7	15	7
	64	189	84	130	56	100	42	85	35	85	35
1024	8	479	227	239	114	120	57	60	29	30	15
	64	759	338	379	170	190	85	95	53	65	41
	512	1692	743	933	405	554	236	364	152	303	124
16384	8	7678	3649	3827	1827	1916	914	959	454	479	230
	64	12204	5149	6071	2712	3043	1358	1519	680	762	341
	512	13561	5946	6757	2976	3384	1488	1700	748	942	407
	4096	13870	6089	7109	3103	3713	1614	2020	873	1260	537

Table 5.16: Timings of parallel server computation for the new CPIR method in octal case in ms

n	No. buckets	Number of cores 1		2		4		8		16	
		hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL	hex-new	hex-LL
128	16	64	17	32	9	32	9	32	9	32	9
1024	16	513	140	257	72	129	36	65	19	32	11
	256	653	171	396	102	269	67	204	50	173	43
16384	16	8218	2235	4104	1128	2052	574	1026	296	514	156
	256	10441	2750	5223	1386	2615	702	1312	353	663	183
	4096	10770	2813	5568	1441	2971	757	1658	418	1005	247

Table 5.17: Timings of parallel server computation for the new CPIR method in hexadecimal case in ms

Chapter 6

Conclusion

To address the problem of hiding query access patterns in privacy preserving range query scheme, we proposed two methods using PIR and ORAM techniques. Our methods aim to prevent the disclosure of access patterns, in addition to providing the confidentiality of data and query. For Private Information Retrieval, we introduced an improved version of Lipmaa's BddCPIR. To that end, we propose a new method which reduces the number of modular exponentiation operations in each node of the tree. We implemented our CPIR method on octal and hexadecimal trees to utilize shallow trees in implementation. Furthermore we benefit from a multi-exponentiation algorithm [22] which enables to operate multiplication of several exponentiation terms simultaneously. The new CPIR method is applied on an existing scheme based on bucketization method [18] for range query operations. For ORAM, we adapted Stefanov et al.'s [28] Path ORAM to range query scheme. We analyzed two methods for the cost of communication and computation. The results of bandwidth analysis to retrieve one bucket show that for large database sizes the communication cost of the CPIR method is less than the Path ORAM method. We analyzed the bandwidth usage for retrieval of multiple buckets, since a range query may map to more than one bucket. The results of communication for multiple buckets, also, benefits the usage of CPIR compared to Path ORAM. On the other hand, in terms of computation cost, although the method we proposed for CPIR improves the performance of Lipmaa's BddCPIR scheme significantly, it cannot make CPIR an advantageous scheme against Path ORAM. While Path ORAM scheme has insignificant server cost, the cost of computations on server side dom-

inates the computations in CPIR. Apart from communication and computational comparison, the qualitative aspects of two methods is, also, important in privacy-preserving range queries. For example, CPIR method is suitable for multi-client applications. However, on Path ORAM method utilizing a multi-client application requires an additional cost to inform each client for each access to database due to the change in data path.

In conclusion, depending on the bandwidth usage and the qualitative aspects, CPIR based privacy-preserving range query scheme is more advantageous than Path ORAM based privacy-preserving range query scheme for hiding query access patterns. However, in terms of computational cost, CPIR is still an expensive scheme compared to Path ORAM.

References

[1] "Announcing the Advanced Encryption Standard", *Federal Information Processing Standards Publication 197*. United States National Institute of Standards and Technology (NIST). November 26, 2001.

[2] Adleman,L.M., Rivest, R.L., Shamir, A. "Cryptographic communications system and method", *U.S. Patent No. US4405829 A*. 1983.

[3] Agrawal, R., Kiernan, J., Srikant R., and Xu, Y. "Order preserving encryption for numeric data", In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD '04)*, pp. 563-574. ACM, New York, NY, USA, 2004

[4] Boldyreva,A., Chenette,N., Lee, Y., O'Neill,A. "Order Preserving Symmetric Encryption", In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques (EUROCRYPT '09)*, pp.224-241. Springer-Verlag, Berlin, Heidelberg, 2009.

[5] Boldyreva,A., Chenette,N., O'Neill,A. "Order Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions", In *Proceedings of the 31st annual conference on Advances in cryptology (CRYPTO'11)*, pp. 578-595. Springer-Verlag, Berlin, Heidelberg, 2011.

[6] Boneh,D., Waters,B. "Conjunctive, Subset, and Range Queries on Encrypted Data", In *Proceedings of the 4th conference on Theory of cryptography (TCC'07)*, pp.535-554. Springer-Verlag, Berlin, Heidelberg, 2007.

[7] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M., "Private Information Retrieval", In *FOCS 95: Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, pp. 41-50, 1995.

[8] Chung,K.M., Pass,R. "A Simple ORAM", *IACR Cryptology ePrint Archive 2013: 243*, 2013.

[9] Damgård, I., and Jurik, M. "A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System", In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography (PKC '01)*, pp.119-136. Springer-Verlag, London, UK, 2001.

[10] Dautrich, J., and Ravishankar,C.V. "Compromising privacy in precise query protocols", In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT '13)*, pp.155-166. ACM, New York, NY, USA, 2013.

[11] El Gamal, T. "A Public Key Cryptosystem and A Signature Scheme Based on Discrete Logarithms", In *Proceedings of CRYPTO 84 on Advances in cryptology*, pp.10-18. Springer-Verlag New York, New York, NY, USA, 1985.

[12] Gentry, C. "Fully homomorphic encryption using ideal lattices", In *Proceedings of the forty-first annual ACM symposium on Theory of computing (STOC '09)*, pp. 169-178, ACM, New York, NY, USA, 2009.

[13] Gentry, C., Goldman,K., Halevi,S., Julta,C., Raykova,M., Wichs,D. "Optimizing ORAM and Using it Efficiently for Secure Computation", In *Privacy Enhancing Technologies 2013: 1-18*, 2013.

[14] Goldreich, O., Ostrovsky, R. "Software Protection and simulation on oblivious RAMs", *J. ACM 43*, pp. 431-473. 1996.

[15] Goldwasser, S., Micali, S. "Probabilistic Encryption", In *Journal of Computer and System Sciences*, 28(2), pp.270-299, 1984.

[16] Hacıgümüş, H., Iyer,B., Li,C., Mehrotra, S. "Executing SQL over Encrypted Data in the Database-Service-Provider Model", *SIGMOD '02Proceedings of the*

2002 ACM SIGMOD international conference on Management of data, pp. 216-227. ACM, New York, NY, USA, 2002.

[17] Hore, B., Mehrotra, S., Tsudik, G. “A privacy-preserving index for range queries”, In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30 (VLDB ’04)*, Vol. 30. VLDB Endowment pp. 720-731. 2004

[18] Hore, B., Mehrotra S., Canım, M. Kantarcıoğlu, M. “Secure multidimensional range queries over outsourced data”, *The VLDB Journal*, 21(3):333?58, 2012.

[19] Islam, M.S., Kuzu,M., Kantarcıoğlu, M. “Inference attack against encrypted range queries on outsourced databases”, In *Proceedings of the 4th ACM conference on Data and application security and privacy (CODASPY ’14)*, pp. 235-246. ACM, New York, NY, USA, 2014.

[20] Kushilevitz, E., Ostrovsky, R., ”Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval”, *FOCS ’97*, 1997.

[21] Li, J., Omiecinski, E.R. “Efficiency and security trade-off in supporting range queries on encrypted databases”, In *Proceedings of the 19th annual IFIP WG 11.3 working conference on Data and Applications Security (DBSec’05)*, pp. 69-83. Springer-Verlag, Berlin, Heidelberg, 2005.

[22] Lim,C.H. “Efficient Multi-Exponentiation and Application to Batch Verification of Digital Signatures”,Manuscript, 2000.

[23] Lim,C.H., Lee, P.J. “More Flexible Exponentiation with Precomputation”, In *Advances in Cryptology-CRYPTO’94,LNCS389*, pp. 95–107. Springer-Verlag, 1994.

[24] Lipmaa, H. “First CPIR protocol with data-dependent computation”, In *Information, Security and Cryptology ICISC 2009*, pp. 193-210. Springer Berlin Heidelberg, 2009.

[25] Paillier, P. “Public-key cryptosystems based on composite degree residuosity classes”, In *Advances in cryptology, EUROCRYPT’99*, pp. 223-238. Springer Berlin Heidelberg, 1999.

[26] Vimercati,S., Foresti,S., Paraboschi,S., Pelosi,G., Samarati, P. “Efficient and Private Access to Outsourced Data”, In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems (ICDCS ’11)*,pp. 710-719. IEEE Computer Society, Washington, DC, USA, 2011.

[27] Shi, E., Chan, T.H.H., Stefanov,E., Li, M. ”Oblivious RAM with $O((logN)^3)$ worst-case cost”, In *Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security (ASIACRYPT’11)*, pp. 197-214. Springer-Verlag, Berlin, Heidelberg, 2011.

[28] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L. “Path ORAM: An Exteremely Simple Oblivious RAM Protocol”, In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security (CCS ’13)*, pp. 299-310. ACM, New York, NY, USA, 2013.

[29] TPC-H, Decision Support Benchmark, http://www.tpc.org/tpch

[30] Ünal, E. and Savaş, E., “On Acceleration and Scalability of Number Theoretic Private Information Retrieval”, To appeared in *IEEE, Transactions on Parallel and Distributed Systems.*

[31] Wen, M., Lu, R., Zhang,K., Lei,J., Liang, X., Shen,X. ”PaRQ: A Privacy Preserving Range Query Scheme Over Encrypted Metering Data for Smart Grid”, In *IEEE Transactions on Emerging Topics in Computing*, Vol. 1, No. 1, pp. 178-191, 2013.