MONASH University

Symmetric Searchable Encryption Supporting Rich Functionality and Enhanced Security

Shabnam Kasra Kermanshahi

A thesis
submitted for the degree of Doctor of Philosophy
at Monash University in (2019)
Information Technology (Cyber Security)

Abstract

This thesis presents three Searchable Symmetric Encryption (SSE) schemes supporting rich functionality and enhanced security. Searchable encryption (SE) is a cryptographic primitive which allows a secure search over encrypted data while protecting the confidentiality of data and queries. SE became an interesting research area due to its wide range of applications as well as open problems. The aim of this research is to address three major challenges in this area which are adapting multi-user settings into an SSE scheme, supporting ranked search without revealing information to the server on database content, and enabling secure range query over encrypted spatial data. Among the different settings of SE schemes, Multi-user ones are more appealing in practice. Multi-user settings have extra functionality requirements as well as the security requirements which makes it challenging. The three main functionality requirements of Multi-user settings are Data sharing, Write/Read capability, and Revocation. Data sharing refers to the ability to share data between several users securely. Write/Read capability is to allow the user to perform both read and write over the encrypted database. Key revocation is required to prevent any illegitimate access to the encrypted database by a revoked user. To satisfy the security requirements, the risks associated with Multi-user settings must be minimised. That is, the privacy of database content must be preserved even if some of the users lose their private keys. Moreover, the privacy of each users' data against the other users must be preserved. Last but not least, there should be a solution for a fast update of the encryption key to protect the stored data on the server in case of lose/leakage of the master key. Although SE schemes have been studied very well in both Symmetric and Asymmetric structures, finding a balance between security, functionality, and performance is still a big challenge. The first contribution of this study is an efficient Multi-user SSE protocol which satisfy these mentioned requirements. Another challenge is the relevance of search results and the queried keywords. Although searchable encryption schemes allow secure search over the encrypted data, they mostly support conventional Boolean keyword search, without

capturing any relevance of the search results. This leads to a large amount of post-processing overhead to find the most matching documents and causes unnecessary network traffic between the servers and end-users. Such problems can be addressed efficiently using a ranked search system that retrieves the most relevant documents. However, most of the current solutions in the context of searchable symmetric encryption suffer from either (a) security breaches due to revealing information to the server on database content via ranking or (b) usability concerns caused by some assumptions like using the two non-colluding servers. In this research, we present a generic filtering solution for multi-keyword ranked search over the encrypted cloud data. Then, an SSE scheme is designed which uses the proposed technique and filters the results and returns the most relevant documents; it can resist all ranking related attacks while guaranteeing the security using somewhat homomorphic encryption; and it can use a single cloud server. Finally, in order to support geometric range search, two dynamic SSE schemes are presented. Our constructions are the first to provide forward/backward security in the context of SSE-based schemes supporting geometric range search. In addition, we define a security notion called content privacy. This security notion captures the leakages that are critical in the context of geometric range search but not considered by forward/backward security. Content privacy eliminates the leakage on the updated points of the database during both search and update. Due to the inherent leakages associated with range queries, none of the existing related works can support content privacy whereas the design of our constructions avoids such leakages. When compared to the state-of-the-art schemes our constructions provide a higher level of security and practical efficiency supported by our experimental results.

Declaration

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: .....

Print Name: .....

Date: .....

Publications

This thesis includes three original papers published in peer reviewed conferences and journals and one submitted publication. The core theme of the thesis is symmetric searchable encryption. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology under the supervision of Joseph K. Liu, Ron Steinfeld, and Surya Nepal.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research. In particular, in terms of evaluation of the proposed schemes via implementations.

List of publications included as part of the thesis The following publications arise from this thesis.

- S. Kasra Kermanshahi, J. Liu, R. Steinfeld, S. Nepal. "Generic Multi-keyword Ranked Search on Encrypted Cloud Data" in The European Symposium on Research in Computer Security-2019 (Accepted-Core A).

- S. Kasra Kermanshahi, J. Liu, R. Steinfeld "Multi-user cloud-based secure keyword search" in 22nd Australasian Conference on Information Security and Privacy -2017 (Best paper award).

- S. Kasra Kermanshahi, S. Sun, J. Liu, R. Steinfeld, S. Nepal, F. Lau, M. Ho."Geometric range search on encrypted data with Forward/Backward security", submitted to IEEE Transactions on Dependable and Secure Computing (Submitted-Core A).

- S. Kasra Kermanshahi, J. Liu, R. Steinfeld, S. Nepal, S. Lai, R. Loh, C. Zhou."Multi-client Cloud-based Symmetric Searchable Encryption". in IEEE Transactions on Dependable and Secure Computing (Under minor revision-Core A).

Additional publications

- B. Yu, S. Kasra Kermanshahi, A. Sakzad, S. Nepal. "Privacy preserving payment channel network", the 13th international conference on provable and practical security: ProvSec 2019 (Accepted-Core B)

Acknowledgements

I would like to begin by thanking my advisors Joseph K. Liu, Ron Steinfeld, and Surya Nepal for supporting me through these years with guidance and inspiration. I would also like to thank my committee members Carsten Rudolph and Amin Sakzad for all their valuable feedback that improved this work. I want to especially thank the authority of Monash University and Data 61 for providing me with a good environment and facilities to complete this research.

Table of Contents

List of Tables xv

List of Figures xvi

1 Introduction 1

1.1 Motivation 5

1.1.1 Research Objectives 7

1.2 Main Contributions Summary 8

1.2.1 Multi-client symmetric searchable encryption 8

1.2.2 Multi-keyword ranked symmetric searchable encryption 9

1.2.3 Geometric range search on encrypted data with forward/backward privacy 10

1.3 Thesis Structure 10

2 Background 12

2.1 Cryptographic Background 12

2.1.1 Threshold secret sharing 12

2.1.2 One-more one-way function 13

2.1.3 Decisional Diffie-Hellman . . . 14
2.1.4 Ring-LWE problem . . . 14
2.1.5 Homomorphic Encryption from RLWE . . . 14
2.1.6 Homomorphic Encryption from Learning with Errors (LWE) . . . 15
2.1.7 Homomorphic Encryption from Ring-GSW . . . 16
2.1.8 Pseudorandom Functions (PRFs) . . . 17
2.1.9 Shen, Shi and Waters Encryption (SSW) . . . 18
2.2 Alternative approaches for search on encrypted data . . . 19
2.2.1 Public Key Encryption with keyword Search (PKES) . . . 19
2.2.2 Order Preserving Encryption (OPE) . . . 20
2.2.3 Oblivious RAM . . . 21
2.3 Searchable Encryption . . . 21
2.3.1 Preliminaries . . . 21
2.3.1.1 Notations and assumptions . . . 21
2.3.1.2 Syntax of Symmetric Searchable Encryption . . . 22
2.3.1.3 Syntax of Dynamic Symmetric Searchable Encryption 22
2.3.2 Oblivious Cross Tags (OXT) . . . 23
2.3.3 RSA-SSE . . . 25
2.3.4 Geometric Range Searchable Encryption (GRSE) . . . 27
2.4 Security . . . 29
2.4.1 SSE Leakage Profile . . . 29
2.4.2 DSSE Leakage Profile . . . 30
2.4.3 Forward Privacy . . . 30

2.4.4 Backward privacy . . . 31
2.5 Summary . . . 32

3 Multi-client symmetric searchable encryption 34
3.1 Overview . . . 34
3.1.1 Motivation . . . 34
3.1.2 Contributions . . . 39
3.2 Preliminaries . . . 41
3.2.1 Notations . . . 41
3.3 Background . . . 42
3.4 Syntax of multi-client SSE . . . 43
3.5 Security definitions of multi-client SSE . . . 45
3.5.1 Privacy against server . . . 45
3.5.2 Query privacy against other key share holders . . . 47
3.5.3 Database content privacy against active collusion . . . 47
3.5.4 Database content privacy after update . . . 48
3.5.5 Database content privacy after revocation . . . 48
3.6 Randomizable Distributed Key Homomorphic PRFs . . . 50
3.6.1 Definition . . . 50
3.6.2 PRF Evaluation protocol . . . 51
3.6.2.1 Correctness. . . . 51
3.6.2.2 PRF security definition of RDPRF. . . . 52
3.6.2.3 RDPRF Active Collusion Security Definition. . . . 52

3.6.2.4 Query privacy security of RDPRF 53
3.6.3 Concrete construction of RDPRF 53
3.7 Multi-client symmetric searchable encryption 58
3.7.1 Construction 59
3.7.2 Update, Revocation and Enrollment 60
3.7.2.1 Update 62
3.7.2.2 Revocation 63
3.7.2.3 Enrolment 65
3.8 Security analysis 66
3.9 Security, Functionality and Performance Comparison 71
3.10 Summary 75

4 Multi-keyword ranked symmetric searchable encryption 76

4.1 Overview 76
4.1.1 Motivations 78
4.1.2 Contributions and technique 79
4.2 Preliminaries 80
4.3 Our threshold-based filtering approach 82
4.3.1 Homomorphic operations 82
4.3.2 Homomorphic search algorithm 86
4.3.3 Homomorphic filter algorithm 87
4.4 Our multi-keyword ranked searchable symmetric encryption scheme . 89
4.4.1 Modes of operation 92

4.5 Evaluation . . . 92
4.5.1 Computation complexity . . . 92
4.5.2 Communication complexity . . . 93
4.6 Summary . . . 98

5 Geometric range search on encrypted data with forward/backward privacy 99

5.1 Introduction . . . 99
5.2 Preliminaries . . . 102
5.2.1 Additive homomorphic encryption . . . 102
5.2.1.1 Symmetric additive homomorphic encryption. . . . 103
5.2.2 Security definition . . . 104
5.3 Security notions . . . 105
5.3.1 Content privacy . . . 105
5.3.1.1 Example of content privacy against the existing dynamic SSE . . . 107
5.4 Syntax of DSSE with geometric range query . . . 109
5.5 SSE schemes for geometric range search . . . 111
5.5.1 Overview . . . 112
5.5.1.1 Setup . . . 112
5.5.1.2 Search . . . 113
5.5.1.3 Update . . . 115
5.5.2 The naive solution . . . 117
5.5.3 Construction-I . . . 121

5.5.3.1 Setup . . . 121
5.5.3.2 Search . . . 124
5.5.3.3 Update . . . 125
5.5.4 Construction-II . . . 126
5.6 Comparison . . . 128
5.7 Security analysis . . . 132
5.7.1 Range search leakage functions . . . 132
5.7.2 Construction-I . . . 133
5.7.3 Construction-II . . . 136
5.8 Summary . . . 137

6 Conclusions and Future Work **139**
6.1 Summary . . . 139
6.2 Future Work . . . 140

References **141**

APPENDICES **154**

A Experimental Results **155**
A.1 Multi-client symmetric searchable encryption . . . 155
A.2 Geometric range search on encrypted data with forward/backward privacy . . . 160

List of Tables

3.1 Notations and terminologies 41

3.2 Comparison of a subset of existing related works 43

3.3 Computational and communication cost between client and server . . 72

3.4 Security analysis 74

3.5 Functionality analysis 75

4.1 Summary of comparison 79

4.2 Notations and terminologies 81

4.3 Communication cost improvement 97

4.4 Parameter settings 97

5.1 Security comparison 130

5.2 Performance comparison 132

A.1 RDPRF Performance 160

List of Figures

1.1 Searchable encryption framework 2

2.1 PEKS framework 20

2.2 Overview of GRSE scheme 28

3.1 Sample database 35

3.2 Forward index 36

3.3 Inverted index 37

3.4 Single Keyword Search (SKS) 38

4.1 Example of Filter algorithm 87

4.2 Overall communication cost 98

5.1 Security notions similarity 106

5.2 Sample spatial dataset 113

5.3 Converted dataset 114

5.4 Sample indexed binary tree 115

5.5 Sample Node-Inverted index for x-axis 116

5.6 Sample Node-Inverted index for y-axis 117
5.7 Sample Range search 118
5.8 Binary tree for range search over x-axis and y-axis 118
5.9 Example of update over x-axis 119
5.10 Example of update over y-axis 120

A.1 Stag generation of the encrypted database with a fixed threshold . . . 157
A.2 xtoken generation of the encrypted database with a fixed threshold . 157
A.3 xtoken selectivity 158
A.4 Search time vs. dimension size (20K points) 161
A.5 Search time vs. number of data points ($D = 2^{15}$) 162
A.6 Update time vs. dimension size (20K points, 10 points per update) . 162
A.7 Update communication vs. dimension size 163
A.8 Update time vs. number of updating points (20K points, $D = 2^{15}$) . . 163
A.9 Update communication vs. number of updating points 164

Chapter 1

Introduction

The last decade has seen a growing trend towards the use of off-site hosts commonly referred to as the Cloud. This economic information technology paradigm provides ubiquitous access to storage and computing resources. However, confidentiality of sensitive data stored on the cloud is a major concern as the data is visible to the cloud [1, 2]. Although, standard encryption prevents information disclosure to the cloud server, it is impractical in many applications. That is, to search for specific information through the encrypted database, the user has to download and decrypt the data as the data is not readable by the cloud server. Therefore, it incurs massive communication and computation overheads to the system. Searchable Encryption (SE) is a cryptographic technique that can address this problem, as it allows searching of particular data in encrypted content without decryption. There is a large volume of published studies on SE (e.g. [3–6]). These studies can be categorised into two major categories based on the underlying encryption technique employed: Symmetric Searchable Encryption (SSE) (e.g. [7]) and Public key Encryption with Keyword Search (PEKS) (e.g. [8]). PEKS is far less efficient than SSE; likewise, the standard encryption. The focus of this research is on symmetric searchable encryption. We consider three main roles in an SSE scheme as follows:

- Data owner: who provides the data and defines the access permissions to the data.
- Client/user: who queries the data.
- Server: who stores the data and process it in order to response to the queries (e.g. Cloud server).

The main challenge faced by many researchers in developing SSE schemes is to construct the scheme to be comparable with unencrypted databases. More precisely, it is challenging to provide similar functionality and performance while the security is the main concern to be addressed. Figure 1.1 demonstrates the SE framework. From this figure we can see different aspects of SE to be considered in designing a new searchable encryption scheme.

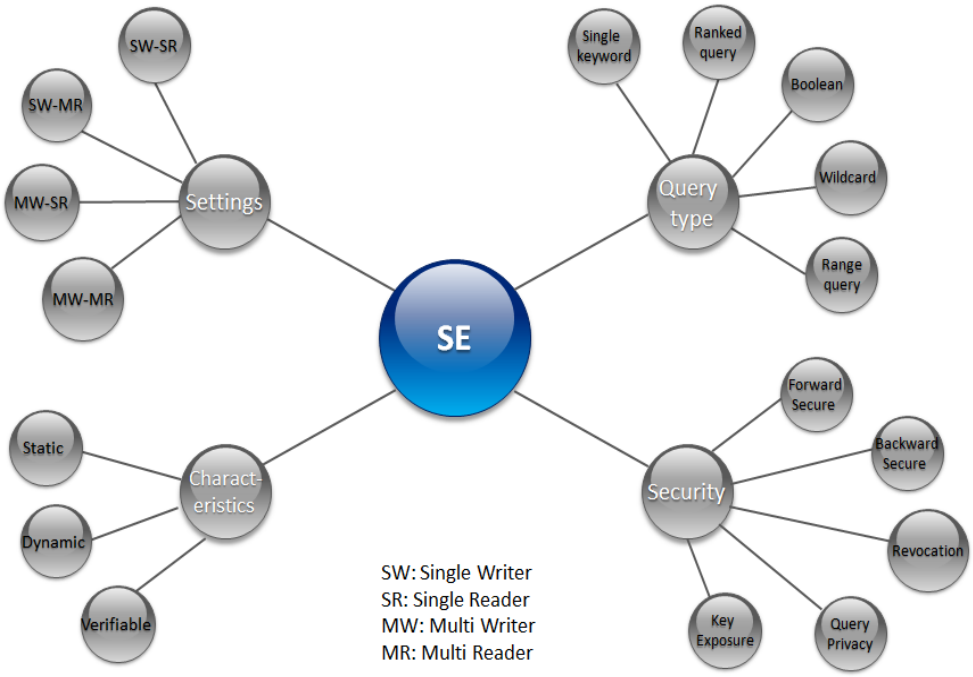


Figure 1.1: Searchable encryption framework

- **Settings:** To issue search queries to the server, one or more clients should be able to generate the search tokens. Thus, SE schemes can be differentiated based on the utilized settings as follows [9];
  - ***Single-Writer/Single-Reader (SW-SR):*** a single user acts as both the data owner and the client. Song et al. [5] proposed the first SSE scheme in this setting. The majority of the existing works are designed in this setting as it is basic and less challenging; a) there is no concern about collusion, malicious user, revocation, etc. b) there is no additional communication/computational overhead to the system for generation/distribution of the search tokens or retrieving the data afterwards.
  - ***Single-Writer/Multi-Reader (SW-MR):*** the data owner provides the data and clients only allowed to read the data. To support this setting, either the secret should be shared with the clients or the data owner must provide the search token for them which in turn introduces new challenges regarding the security and efficiency. For instance, in the proposed scheme by Curtmola et al. [4] one key is shared among all users. Thus, the key lose/exposure of a single user might lead to the leakage of the whole encrypted database (refer to Chapter 3).
  - ***Multi-Writer/Single-Reader (MW-SR):*** more than one entity (data owner and client) have the write permission (provide the data) while only one entity reads the data. This setting is mainly supported by PEKS where the application scenarios are retrieving emails or documents from a server. That is, those who know the public key generate the ciphertexts and the owner of the private key can read the data. Most of the existing works (e.g. [10–14]) in this setting are focused on the security of the scheme where they rely on heavy computations (mainly using pairing operations). Therefore, they are considered as theoretical solutions and the feasibility of them in practice is an open question.

  - ***Multi-Writer/ Multi-Reader (MW-MR):*** the data can be read and write by more than one entity (data owner and client). Like SW-MR schemes, MW-MR are challenging to be designed using the symmetric encryption. However, MW-MR can be achieved by combining public key encryption with key distribution.

- **Query types:** there exists various types of search queries and combinations of them. The following query types are the most common ones.

  - ***Single keyword:*** this query type supports only a single keyword for searching and does not allow for Boolean combinations of keywords.
  - ***Boolean:*** this query type allows a combination of "AND", "OR", and "NOT" over the search keywords. A trivial approach to support Boolean queries is to perform single keyword search for all of the search keywords and then find the intersection. However, it reveals the relation between documents and search keywords, hence the server can learn more about the database by performing statistical analysis.
  - ***Ranked:*** this query type returns the matching documents in a ranked order according to a certain relevance criteria. This type of query reduces the post processing of the results for finding the most relevant documents.
  - ***Range:*** returns documents matching a conditional value between an upper and lower boundary. The range query can be one dimensional (mostly over ordered set of elements) or multi dimensional (like geometric dataset).

- **Characteristics:**

  - ***Static:*** documents can not be added, deleted, or modified once uploaded to the server. Therefore, any changes on the outsourced database is not possible or requires re-encryption and re-upload of the whole database.
  - ***Dynamic:*** database can be updated at any time. This updates includes insertion, deletion, and modification of document(s) of the database.

- ***Verifiable:*** the client can detect whether server returns a correct data as the response to a query or not.

- **Security:**
  - ***Forward privacy:*** An SSE scheme is "forward private", if there is no relation between an update and previous search results.
  - ***Backward privacy:*** ensures that search queries do not leak matching documents after they have been deleted.
  - ***Query privacy:*** The privacy of issued queries by the users must be preserved against all of the involving entities including Data owner and cloud server.
  - ***User revocation:*** is required to prevent any illegitimate access to the data by a revoked client.
  - ***Key exposure:*** key exposure of a user should not lead to disclosure of all of the users' information.

1.1 Motivation

Among the different settings of SE schemes, Multi-client (Multi-user) ones are more appealing in practice. Multi-client settings have extra functionality requirements as well as the security requirements which makes it challenging. The three main functionality requirements of Multi-client settings are Data sharing, Write/Read capability, and Revocation. Data sharing refers to the ability of sharing data between several users securely. Write/Read capability is to allow the user to perform both read and write over the encrypted database. Key revocation is required to prevent any illegitimate access to encrypted database by a revoked user. To satisfy the security requirements, the risks associated with Multi-client settings must be minimised. That is, the privacy of database content must be preserved even if some

of users lose their private keys. Moreover, privacy of each users' data against the other users must be preserved. Last but not least, there should be a solution for fast update of the encryption key to protect the stored data on the server in case of lose/leakage of the master key. There are two common approaches for supporting the single-writer/multi-reader setting, sharing the encryption key and providing the search tokens. The former requires re-distribution of the key and re-encryption of the database for each user revocation. Moreover, it poses major risks associated with users' key lose/exposure. In the multi-client schemes that utilized this approach like [4, 15, 16] if a user being compromised or collude with an adversary or the cloud server it would result to disclosure of all of the users' information. The latter does not suffer from these issues however, it does not support users' privacy against the data owner.

Another challenge is preserving query functionality. Although searchable encryption schemes allow secure search over the encrypted data, they mostly support conventional Boolean keyword search, without capturing any relevance of the search results. This leads to a large amount of post-processing overhead to find the most matching documents and causes an unnecessary network traffic between the servers and end-users. Such problems can be addressed efficiently using rich queries such as range, ranked, substring, wildcard, and phrase queries. This research aims to present solutions for ranked and range queries. Most of the current solutions for ranked search in the context of searchable symmetric encryption suffer from either (a) security breaches due to revealing information to the server on database content via ranking [17, 18] or (b) usability concerns caused by some assumptions like using the two non-colluding servers [19–21].

On the other hand, there are limited number of studies on range searches, in particular geometric range search. Range query is a primary database operation to meet the practical data retrieval need. Range search plays a vital role in supporting common applications using Location-Based Services (LBS) such as Uber. Nevertheless, the leakages from the range query enable an attacker to reconstruct the dataset.

Recently, a considerable amount of literature has been published on the problem of reconstructing encrypted databases from range query leakage [22–25]. Moreover, due to the objects' movements in LBS applications, support of dynamic setting is essential. However, dynamic setting introduces additional leakages to the server about the update undertaken. Supporting forward and backward privacy enables the SSE scheme to mitigate such attacks [26, 27]. However, none of the existing solutions for geometric range search using SSE schemes considered forward and backward privacy.

1.1.1 Research Objectives

The major objectives of this study in order to present SSE schemes supporting rich functionality as well as enhanced security are as follows:

1. To propose novel SSE scheme that support Single-writer/Multi-reader settings with following features;
   (a) minimum presence of the data owner after the setup phase
   (b) support of user privacy by removing the need to take the search token from the data owner
   (c) resilience against exposure of users' keys as well as passive and active collusion
   (d) achieve the query privacy property (against 'helping users') by introducing a new security primitive.
   (e) support user revocation
   (f) support update of the encrypted database using new key
2. To propose SSE scheme that supports ranked search
   (a) secure against all attacks related to ranking leakage while uses a single cloud server

(b) support multi-keyword search over Boolean, ranked and limited range queries

3. To propose an SSE scheme for spatial data that satisfies the following requirements;

   (a) support of dynamic settings

   (b) support of geometric range search

   (c) forward/backward privacy

1.2 Main Contributions Summary

The contributions made in this thesis on Cloud storage security are as follows:

- Multi-client symmetric searchable encryption;
- Multi-keyword ranked symmetric searchable encryption
- Geometric range search on encrypted data with forward/backward privacy

In the following we summarize each main contribution.

1.2.1 Multi-client symmetric searchable encryption

This research presents a Single-writer/Multi-reader SSE scheme that minimizes the presence of the data owner, fully supports user privacy, and preserves the privacy of database content even if some clients lose their private keys. Our multi-client SSE scheme offers two approaches for user revocation. The first solution enables the data owner to revoke key shares from clients by generating new PRF key shares and distributing them between non-revoked clients while leaving the master key and

EDB unchanged to save the costly re-encryption operation. The second approach has minimal communication overhead but requires EDB re-encryption (update).

Moreover, a solution for fast and efficient re-encryption of the database using the new key is provided. This solution enables the data owner to perform one-time update of the encryption key as well as EDB (Encrypted Database) by sending the corresponding key-material to the server. That is, a substantial amount of cost in terms of computations performed locally at data owner side for encryption of the database using the new key, and bandwidth required for uploading the updated EDB as well as transferring information to the considered clients would be avoided.

1.2.2 Multi-keyword ranked symmetric searchable encryption

A generic solution for multi-keyword ranked search over the encrypted cloud data is presented. The proposed solution can be applied over different symmetric searchable encryption schemes. To demonstrate the practicality of our technique, the Oblivious Cross Tags (OXT) protocol of Cash et al. [3] is leveraged due to its scalability and remarkable flexibility to support different settings. Our proposed scheme supports the multi-keyword search on Boolean, ranked and limited range queries while keeping all of the OXT's properties intact. The key contribution of our solution is that it enables our scheme to resist against all common attacks that take advantage of OPE leakage while only a single cloud server is used. Moreover, the results indicate that using the proposed solution the communication overhead decreases drastically when the number of matching results is large.

1.2.3 Geometric range search on encrypted data with forward/backward privacy

Two dynamic symmetric searchable encryption schemes for geometric range search are presented. Our constructions are the first to provide forward/backward privacy in the context of SSE-based schemes supporting geometric range search. In addition, we define a security notion called content privacy. This security notion captures the leakages that are critical in the context of geometric range search but not considered by forward/backward privacy. Content privacy eliminates the leakage on the updated points of the database during both search and update. Due to the inherent leakages associated with range queries, none of the existing related works can support content privacy whereas the design of our constructions avoids such leakages. When compared to the state-of-the-art schemes our constructions provide a higher level of security and practical efficiency supported by our experimental results.

1.3 Thesis Structure

The remaining of the thesis is organized as follows:

Chapter 2: provides fundamental concepts required for understanding the subsequent chapters. It also presents the state-of-the-art techniques and tools to search the encrypted data.

Chapter 3: presents the detailed explanation of the first contribution of this research. That is, a multi-client SSE scheme which addresses several challenges in this domain.

Chapter 4: gives the detailed explanation of the second contribution of this research; a generic solution to support multi-keyword ranked search in SSE schemes.

Chapter 5: presents a dynamic SSE scheme for spatial databases. In addition, a new security notion is introduced which is critical for dynamic SSE scheme supporting

geometric range searches.

Chapter 6: concludes this thesis and presents the open problems as well as future works.

Chapter 2

Background

This chapter presents required background related to the thesis and discusses the state-of-art on search on encrypted data. This chapter is organized as follows: Section 2.1 briefly reviews the cryptographic concepts required in the subsequent chapters. Existing alternative approaches for searching on the encrypted data are described in Section 2.2. Since the main focus of this research is on symmetric searchable encryption, some of the related well-known SSE schemes are reviewed in this chapter.

2.1 Cryptographic Background

2.1.1 Threshold secret sharing

This subsection reviews the secret sharing scheme proposed by Shamir [28], briefly. The idea is to divide the secret $k \in \mathbb{Z}_p$ (for a prime $p > N$) into N pieces such that the knowledge of at least θ pieces (threshold) is required to recover k. More precisely, this scheme consists of two main algorithms *Share* and *Recon* as followed.

- Share: This algorithm takes a secret $k \in \mathbb{Z}_p$ as an input and outputs the corresponding key shares $k_1 \in \mathbb{Z}_p, ..., k_N \in \mathbb{Z}_p$.

- Recon: This algorithm reconstructs the secret using any subset $W = \{i_1, ..., i_\theta\} \subset [N]$ of size θ by computing $k = \sum_{j=1}^{\theta} \lambda_{ij}.k_{ij} \pmod p$, where $\lambda_{ij} \in \mathbb{Z}_p$ are reconstruction coefficients.

2.1.2 One-more one-way function

The security of utilized distributed PRFs in our construction relies on the hardness of the $one - more\ one - way\ function$ assumption that can be defined as Definition 1. The notion of "one-more" problem was introduced by Bellare et al. [29] in order to prove the security of Chaum's blind signature scheme [30].

Definition 1 (one-more one-way function assumption)**.** *A computable function f in polynomial time is $one - more\ one - way$ if the advantage of any adversary $\mathcal{A}$ (probabilistic polynomial-time algorithm) which can win the following game by having an access to "Inversion" and "Challenge" oracles be negligible [29, 31]:*

- *$\mathcal{A}$ inputs the definition of f.*

- *In order to win the game, $\mathcal{A}$ must invert n points output by the challenge oracle while strictly less than n queries issued to the inversion oracle.*

Inversion Oracle. *inputs y in f's codomain and outputs x in f's domain such that $f(x) = y$.*

Challenge Oracle. *there is no input; it outputs a random challenge point from f's codomain.*

Note. *For any integer $n > 1$, solving the one-more problem with access to the inversion oracle up to n times cannot be reduced to the resolution of this problem with access to inversion oracle limited to $n + 1$ queries [31].*

2.1.3 Decisional Diffie-Hellman

Definition 2 (DDH problem)**.** *Let $\mathbb{G}$ be a group of prime order q with generator g. Decisional Diffie-Hellman (DDH) problem can be described as follows; given two probability distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) where, $a, b, c \in \mathbb{Z}_q$, there is no probabilistic polynomial time algorithm to distinguish the mentioned probability distributions (in κ, the advantage of DDH distinguisher is negligible).*

2.1.4 Ring-LWE problem

Definition 3 (Ring-LWE problem.)**.** *For a security parameter λ let*

- *$\Phi_\mu(X)$ be μ-th cyclotomic polynomial for an integer $\mu = \mu(\lambda)$*
- *$\mathbb{A} = \mathbb{Z}[X]/ < \Phi_\mu(X) >$ and $\mathbb{A}_q := \mathbb{A}/q\mathbb{A}$ for an integer $q = q(\lambda) \geq 2$*
- *$\chi = \chi(\mu)$ be a distribution over $\mathbb{A}$*

The $RLWE_{\mu,q,\chi}$ problem is to distinguish two samples from polynomially many samples. In the first distribution, one samples (a_i, b_i) uniformly from $(\mathbb{A}_q)^2$. In the second distribution, one first picks $s \xleftarrow{\$} \mathbb{A}_q$ and then outputs $(a_i, b_i) \in (\mathbb{A}_q)^2$ by sampling $a_i \in \mathbb{R}_q$ uniformly, sampling $e_i \in \chi$ according to the distribution, and setting $b_i = a_i.s + e_i$. The $RLWE_{\mu,q,\chi}$ assumption states that the $RLWE_{\mu,q,\chi}$ problem is unfeasible [32].

2.1.5 Homomorphic Encryption from RLWE

Ring-LWE encryption scheme is associated with a number of parameters [32]:

- λ: Security parameter
- $R = \mathbb{Z}[X]/(x^d + 1)$: polynomial ring of degree d

- $R_q \coloneqq R/qR$: ring mod q for an integer q (ciphertexts are pairs of R_q elements)
- $R_\tau \coloneqq R/\tau R$: message space for $\tau = 2$, R_τ can be represented by polynomials $p(x) = \sum_{i<mu} p_i x^i$ for $p_i \in \mathbb{Z}_\tau$.
- χ: a distribution of polynomials over R_q with 'small' coefficients (with standard deviation σ).

Ring-LWE encryption can be described by the following algorithms:

KeyGen(): This algorithm samples $t \leftarrow \chi$, $e \leftarrow \chi$ and defines the secret key $sk = \vec{s} \leftarrow (1, -t)$ and computes the public key $pk = (a, b)$. Here, $a \xleftarrow{\$} R_q$ and $b = at + e$.

Enc():Given the public key and a message $m \in R_\tau$, the encryption algorithm chooses a small polynomial $v \leftarrow \chi$ and two polynomials e_0 and e_1 and computes the ciphertext $\vec{c} = (c_0, c_1)$ where $(c_0, c_1) = (m.q/\tau, 0) + (bv + e_0, av + e_1)$ (note that $c_0 = a't + (e_0 + ev) + m.q/\tau$ and $c_1 = a' + e_1$, where $a' = av$).

Dec(): Given a ciphertext $\vec{c} = (c_0, c_1)$, this algorithm outputs $m' = \vec{c} \cdot \vec{s}$ and rounds m' coefficient to nearest multiple of q/τ.

Eval(): Given two ciphertexts, this algorithm outputs the ciphertext obtained through the considered operation over the given ciphertexts (for detailed operations refer to section 4.3).

2.1.6 Homomorphic Encryption from Learning with Errors (LWE)

In the design of our filtering approach presented in Chapter 4, we use LWE-based Regev's encryption scheme [33] rather than Ring-LWE, denoted by Enc_{LWE}. In Regev's encryption, the public key consists of a matrix $\mathbb{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ and an LWE sample $\vec{a} \in \mathbb{Z}_q^m$ (q, m, and n are integers). Let $\vec{a} = \mathbb{A}^T \vec{s} + \vec{e}$ where $\vec{s} \xleftarrow{\$} \mathbb{Z}_q^n$ is the

secret and $\vec{e} \in \mathbb{Z}^m$ is the error. For the encryption, one chooses uniformly random vectors $\vec{y} \in \{0,1\}^m$ and computes $c = \mathbb{A}\vec{y} \mod q$, $c' =< \vec{a}, \vec{y} > +m.\lfloor q/2 \rfloor \mod q$. For the decryption, $c' - \vec{s}^T c \mod q$ must be computed to remove the common part and recover the message by rounding the "error" to nearest multiple of $\lfloor q/2 \rfloor$.

2.1.7 Homomorphic Encryption from Ring-GSW

Let $\mathbf{G} = [\mathbf{I}, 2\mathbf{I}, 4\mathbf{I}...2^{l-1}\mathbf{I}]^t \in R_q^{2l\times 2}$ be the gadget matrix. Homomorphic Encryption from Ring-GSW can be described by the following algorithms [34]:

KeyGen(): This algorithm samples $\vec{t} \leftarrow \mathbb{Z}_q$ and defines the secret key $sk = \vec{s} \leftarrow (1, -t)$ and sets $\vec{v} = \mathbf{G}\vec{s}$. To define public key $pk = A$, this algorithm first generates a $m \times 1$ matrix $B \xleftarrow{\$} R_q^{m\times i}$ (where $R_q = \mathbb{Z}_q(x)/(x^d + 1)$) and a vector $\vec{e} \leftarrow \chi^m$; it sets $\vec{b} = B.\vec{t} + \vec{e}$, and A to be the 2-column matrix consisting of $\vec{b}$ followed by the n columns of B ($A.\vec{s} = \vec{e}$).

Enc(): To encrypt the message $\mu \in \{0,1\}$, this algorithm computes a ciphertext in the form of $C = \mathsf{Flatten}(\mathsf{C}')$ where $C' = \mu.I_\eta + \mathsf{BitDecomp}(RA)$. Here, $\eta = 2 \times l$ and $\mathsf{BitDecomp}(a) = (a_0, ..., a_{l-1}) \in R_q^l$ for $a = \sum_i a_i.2^i$ where each a_i is an element of R that when represented as a polynomial of degree $d-1$ has coefficients that are all in $\{0,1\}$.

Let $C'' = \mathsf{BitDecomp}^{-1}(C')$, thus $C = \mathsf{Flatten}(C') = \mathsf{BitDecomp}(C'')$. $\mathsf{Flatten}$ ensures that the coefficients of C are small; therefore C has the proper form of a ciphertext that permits our homomorphic operations [34].

Dec(): This algorithm computes $C.\vec{v} = \mu.\vec{v} + \mathsf{BitDecomp}(C'').\vec{v} = \mu.\vec{v} + C''.\vec{s} = \mu.\vec{v} + e'$ where e' is a small noise of C.

The other utilised homomorphic operations are NAND and Mult as follows.

$$\mathsf{NAND}(C_1, C_2) = \mathsf{Flatten}(I_N - C_1.C_2)$$

$$\mathsf{Mult}(C_1, C_2) = \mathsf{Flatten}(C_1.C_2)$$

2.1.8 Pseudorandom Functions (PRFs)

In this subsection, we review the Pseudorandom Functions.

Key Homomorphic Pseudorandom Functions. Let $\mathcal{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be an efficiently computable function where $(\mathcal{K}, \oplus)$ and $(\mathcal{Y}, \otimes)$ are groups. Tuple $(\mathcal{F}, \oplus, \otimes)$ is a key homomorphic PRF if the following properties hold [35]:

1. $\mathcal{F}$ is a secure pseudorandom function.
2. For every $k_1, k_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$, $\mathcal{F}(k_1, x) \otimes \mathcal{F}(k_2, x) = \mathcal{F}(k_1 \oplus k_2, x)$.

Constructing key homomorphic PRFs in the random oracle model is straightforward. Let G be a finite cyclic group of prime order q and let $H_1 : \mathcal{X} \to G$ be a hash function modeled as a random oracle. We can define the function $F_{DDH} : \mathbb{Z}_q \times \mathcal{X} \to G$ as $F_{DDH}(k, x) \leftarrow H_1(x)^k$ and observe that $F_{DDH}(k_1+k_2, x) = F_{DDH}(k_1, x).F_{DDH}(k_2, x)$.

Distributed PRF. A distributed PRF mainly consists of five algorithms; S*etup*, K*ey sharing*, P*artial evaluation*, C*ombine*, and E*valuation* with the following properties [35].

- S*etup*: This algorithm takes an input security parameter κ and outputs the public parameters pp.
- K*ey sharing* $\mathcal{K} \to \mathcal{K}^{\mathcal{N}}$: This algorithm generates key shares $(k_1, ..., k_N) \in \mathcal{K}^{\mathcal{N}}$ of the considered master key $(k_0 \in_r \mathcal{K})$ using $S = (\theta, N)$, a threshold secret sharing scheme proposed by Shamir [28], which is explained in subsection 2.1.1 .
- P*artial evaluation*: The function $\mathcal{F} : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ takes a key share and an input point, and outputs a partial evaluation of the E*valuation* function f.
- C*ombine*: The algorithm $G : 2^{[N]} \times \mathcal{Y}^{\theta} \to \mathcal{Y}$ takes inputs θ and a subset $W \subset [N]$ of size θ , partial evaluations on key shares in the set W , and outputs a value in $\mathcal{Y}$.

- E*valuation*: The function $f : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ maps a key and an input to the space of outputs.

The distributed PRF is initialized by a trusted third party who runs $\mathsf{S}etup(1^\kappa)$ to obtain the public parameters pp, samples the master secret key $mk = k_0$ uniformly from $\mathcal{K}$, and performs $\mathsf{K}ey\ sharing(k_0)$ to obtain a tuple $(k_1, ..., k_N)$. The key share k_i is distributed as the secret key for each key share holder i along with public parameters pp. A client who wants to compute the evaluation function using k_0 on input x sends x to $\theta - 1$ key share holders $(i_1, ..., i_{\theta-1})$. Each key share holder i responds to the client with $\mathcal{F}(k_i, x)$. then the client locally computes $f(k_0, x)$ by computing $G(W, \mathcal{F}(k_{i_1}, x), ..., \mathcal{F}(k_{i_\theta}, x))$.

Correctness. By considering pp as the output of $\mathsf{S}etup(1^\kappa)$, k_0 sampled uniformly from $\mathcal{K}$, and $(k_1, ..., k_N)$ the key share output by $\mathsf{K}ey\ sharing(k_0)$; for every subset $W = i_1, ..., i_\theta \subset [N]$ of size θ, and for every input x, a distributed PRF is correct if $f(k_0, x) = G(W, \mathcal{F}(k_{i_1}, x), ..., \mathcal{F}(k_{i_\theta}, x))$.

2.1.9 Shen, Shi and Waters Encryption (SSW)

Let $\Sigma = \mathbb{Z}_N^n$ to be the class of plaintexts and the class of predicates to be $\mathcal{F} = \{f_{\boldsymbol{v}} | \boldsymbol{v} \in \mathbb{Z}_N^n\}$ with $f_{\boldsymbol{x}}(\boldsymbol{v}) = 1$ iff $\langle \boldsymbol{x}, \boldsymbol{v} \rangle = 0 \mod N$. The symmetric-key predicate encryption scheme of Shen et al. [36] consists of following algorithms:

$\mathsf{Setup}\big(1^\lambda\big)$: This algorithm inputs 1^λ and generates $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$ where $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r \times \mathbb{G}_s$. It picks the corresponding generators g_p, g_q, g_r, g_s. It chooses $h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i} \in \mathbb{G}_p$ uniformly at random for $i = 1, ..., n$. The secret key is $SK = \big(g_p, g_q, g_r, g_s, \{h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i}\}_{i=1}^n\big)$.

$\mathsf{Encrypt}(SK, \boldsymbol{x})$: To encrypt $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_N^n$ using the secret key SK, this algorithm picks random $y, z, \alpha, \beta \in \mathbb{Z}_N$, $S, S_0 \in \mathbb{G}_s$, and $R_{1,i}, R_{2,i} \in \mathbb{G}_r$ for $i = 1, ..., n$. Then outputs the ciphertext

$$CT = \begin{pmatrix} C = S \cdot g_p^y, & C_0 = S_0 \cdot g_p^z \\ \{C_{1,i} = h_{1,i}^y \cdot u_{1,i}^z \cdot g_q^{\alpha x_i} \cdot R_{1,i} & C_{2,i} = h_{2,i}^y \cdot u_{2,i}^z \cdot g_q^{\beta x_i} \cdot R_{2,i}\}_{i=1}^n \end{pmatrix}$$

$\mathsf{GenToken}(SK, \boldsymbol{v})$: Let $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{Z}_N^n$. This algorithm chooses $f_1, f_2 \in \mathbb{Z}_N$, $r_{1,i}, r_{2,i} \in \mathbb{Z}_N$, $R, R_0 \in \mathbb{G}_r$, and $S_{1,i}, S_{2,i} \in \mathbb{G}_s$ at random, for $i = 1$ to n. It outputs the token

$$TK_{\boldsymbol{v}} = \begin{pmatrix} K = R \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}, & K_0 = R_0 \cdot \prod_{i=1}^n u_{1,i}^{-r_{1,i}} \cdot u_{2,i}^{-r_{2,i}} \\ \{K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_1 v_i} \cdot S_{1,i}, & K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 v_i} \cdot S_{2,i}\}_{i=1}^n \end{pmatrix}$$

$\mathsf{Query}(TK_{\boldsymbol{v}}, CT)$:
Let $CT = \left(C, C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n\right)$ and $TK_v = (K, K_0, \{K_{1,i}, K_{2,i}\}_{i=1}^n)$ as above. The query algorithm outputs 1 iff

$$e(C, K) \cdot e(C_0, K_0) \cdot \prod_{i=1}^n e(C_{1,i}, K_{1,i}) \cdot e(C_{2,i}, K_{2,i}) \stackrel{?}{=} 1$$

2.2 Alternative approaches for search on encrypted data

Several approaches have been proposed that offer "search on encrypted data", however SSE has better trade off between security and efficiency among them. Thus, the main focus of this research is on SSE and the other approaches are briefly reviewed in this section for the sake of completeness.

2.2.1 Public Key Encryption with keyword Search (PKES)

Public Key Encryption with keyword Search is a cryptographic primitive which enables searching on data that is encrypted using a public key cryptosystem. The notion of PKES was first introduced by Boneh et al. [37] in 2004. They proposed two constructions, one using bilinear pairing and another is based on general trapdoor permutations where the former is more efficient.

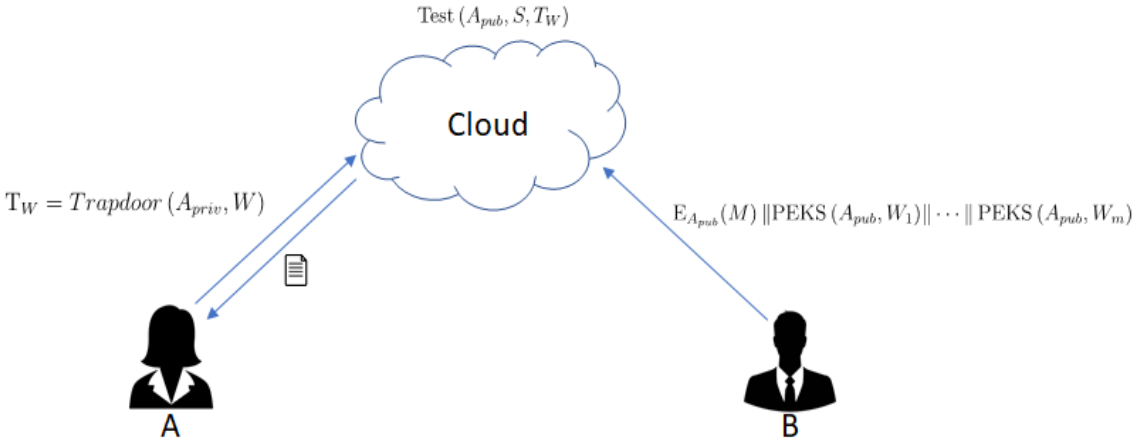


Figure 2.1: PEKS framework

As shown in Figure 2.1, this system consists of three major roles: data sender (B), data receiver (A), and the server (Cloud). The data sender uses the public key of the data receiver (A_{pub}) to encrypt the a message M with keywords $W_1, \ldots, W_m$ and then sends it to the server. The data receiver uses its private key to generate a trapdoor of the search keyword (T_W). Given A's public key, a searchable encryption $S = \text{PEKS}\,(A_{pub}, W')$ and a trapdoor $T_W = \text{Trapdoor}\,(A_{priv}, W)$, the server tests if $W = W'$ and then sends the matching documents to the data receiver. However, their solution reveals partial information which leads to the *access pattern* leakage. To overcome this problem Boneh et al. [38] proposed a PKES scheme using Bloom filter which hides the access pattern. Nevertheless, this solution is much less efficient than SSE based solutions as the search complexity is proportional to the square root of the database size.

2.2.2 Order Preserving Encryption (OPE)

Order Preserving Encryption (OPE) is a deterministic encryption scheme where the numerical order of the plaintexts are preserved by the encryption function. That is, for any secret key k, $Enc_k(m_1) < Enc_k(m_2)$ if $m_1 < m_2$. Boldyreva et al. [39] presented the first formal cryptographic treatment of OPE. Although OPE allows

efficient range queries, it cannot achieve the conventional notions of security for symmetric encryption, IND-CPA. Indeed, OPE reveals the relative order of elements in the database, and therefore does not meet the data owner's data privacy.

2.2.3 Oblivious RAM

Goldreich and Ostrovsky [40] introduced the concept of Oblivious RAM (ORAM) which is recently being used in outsourcing data [41,42]. ORAM conceals the memory access pattern. This is achieved by randomizing and re-encrypting data whenever it is accessed. Although ORAM enables "search on encrypted data" with data privacy protection via hiding the access pattern from the untrusted server, it is impractical due to its overhead. For the ORAM with maximum capacity N, the server stores $O(N \log N)$ items and each oblivious data access request requires $O(\log^3 N)$. A considerable amount of literature has been published on ORAM [41,43–48] to address its overwhelming overhead and make it practical. However, it is still inefficient for use in large-scale cloud storage.

2.3 Searchable Encryption

This section reviews the SSE schemes which are used for the comparisons in the subsequent chapters.

2.3.1 Preliminaries

2.3.1.1 Notations and assumptions

Let $\mathsf{DB} = \{(ind_i, \mathrm{W}_i) : 1 \leq i \leq D\}$ be a database with $ind_i \in \{0,1\}^\ell, W_i \subseteq \{0,1\}^*$. Here, ind_i are document indices and W_i is a set of keywords matching document ind_i. We denote the set of keywords in DB with $W = \cup_{i=1}^{D}$ where $K = |W|$. We define

$N = \sum_{i=1}^{D} |W_i|$ as the number of document/keyword pairs. We denote $\mathsf{DB}(w) = \{ind_i | w \in \mathrm{W}_i\}$ as the set of documents containing keyword w.

2.3.1.2 Syntax of Symmetric Searchable Encryption

In general, a searchable symmetric encryption scheme consists of an algorithm Setup and a protocol Search as follows;

$\mathsf{Setup}(\lambda, \mathsf{DB})$: Given security parameter λ and a database DB, this algorithm generates the encrypted database EDB.

$\mathsf{Search}(q, \mathsf{EDB})$: inputs the search query q and the encrypted database EDB, then outputs the search result.

2.3.1.3 Syntax of Dynamic Symmetric Searchable Encryption

This section briefly reviews general Dynamic Symmetric Searchable Encryption (DSSE) since it is required for understanding the security notions discussed in Section 5.3. A DSSE scheme Π consists of a Setup algorithm and two protocols Search and Update between a client and a server [26]:

- $\mathsf{Setup}(\mathsf{DB}, \lambda) \rightarrow (\mathsf{EDB}, K, \sigma)$: Given the security parameter λ and the database DB, this algorithm outputs the encrypted database EDB with the master key K, and σ as the client's state.
- $\mathsf{Search}\ (q, \sigma, \mathsf{EDB}) \rightarrow (ER)$: This protocol is performed between the client and the server. Given the search query q by the client, the server searches the encrypted database EDB and outputs the set of the encrypted matching results, ER.
- $\mathsf{Update}\ (K, \sigma, op, in, \mathsf{EDB}) \rightarrow (\mathsf{EDB}', \sigma')$: The client inputs K, σ, and an operation op with its input $in = (ind, w)$ (an index and a set of keywords to

be modified). The server inputs EDB. Update outputs the new version of the encrypted database and the updated client's state.

2.3.2 Oblivious Cross Tags (OXT)

The proposed protocol by Cash et al. [3] called Oblivious Cross Tags (OXT) is the first searchable symmetric encryption (SSE) scheme that goes beyond a single-keyword search. This scalable scheme supports boolean queries over the encrypted database in sublinear time. OXT consists of an algorithm EDBSetup and a protocol Search as follows;

EDBSetup(λ, DB): Given a security parameter λ and a DB $= (id_i, \mathrm{W}_i)_{i=1}^{d}$, this algorithm generates the encrypted database EDB which is given to the server and a secret key for the user[1]. This phase is given in Algorithm 1, where the data owner runs this algorithm and uploads the EDB to the server. Note that EDB consists of two data structures TSet and XSet. The former allows one to associate a list of fixed-sized data tuples with each keyword in the database, and later issues the keyword-related tokens to retrieve these lists [3]. The latter contains elements computed from each keyword-document pair, called $Xtag$.

The protocol Search running between the user and server consists of following algorithms;

TokenGeneration$((q(\bar{w}) = (w_1, \ldots, w_n),$ EDB$))$: If a user wants to make a query $q(\bar{w})$ over EDB, the search tokens are required. This algorithm (as shown in Algorithm 1) generates the search tokens $Tok_{\mathbf{q}}$ based on the given query.

Search$(Tok_{\mathbf{q}},$ EDB$)$: The algorithm inputs the search token $Tok_{\mathbf{q}} =$ (stag, xtoken[1], xtoken[2], $\cdots$) and outputs the encrypted search result(s) $ERes$.

DecResult $(ERes, K)$: This algorithm takes the encrypted search result $ERes$ and

[1] In single-writer/single-reader setting like OXT and our scheme, data owner and the client/user are the same entity

the utilized secret key as inputs and outputs the corresponding document identifier(s) id(s).

Algorithm 1 OXT: Oblivious Cross-Tags Protocol

EDBSetup

Initialize $\mathbf{T}$ to an empty array indexed by keywords W.
Select key K_S for PRF F. Select keys $K_X,\ K_I,\ K_Z$ for PRF F_p with range $\mathbb{Z}_p^*$ and parse DB as $(id_i, \mathrm{W}_{id_i})_{i=1}^{d}$.
$\mathsf{XSet} \leftarrow \{\}$
for $w \in \mathsf{W}$ **do**
    Initialize $\mathbf{t} \leftarrow \{\}$; and let $K_e \leftarrow F(K_S, w)$.
    for $\mathsf{id} \in \mathsf{DB}(w)$ **do**
        Set a counter $c \leftarrow 1$
        Compute $\mathsf{xid} \leftarrow F_p(K_I, \mathsf{id})$, $z \leftarrow F_p(K_Z, w||c)$; $y \leftarrow \mathsf{xid}z^{-1}$, $e \leftarrow \mathsf{Enc}(K_e, \mathsf{id})$.
        Set $\mathsf{xtag} \leftarrow g^{F_p(K_X,w)\cdot \mathsf{xid}}$ and $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathsf{xtag}\}$
        Append (y, e) to $\mathbf{t}$ and $c \leftarrow c + 1$.
    end for
    $\mathbf{T}[w] \leftarrow \mathbf{t}$
end for
Set $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.Setup}(\mathbf{T})$ and let $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$.
return $\mathsf{EDB}, K = (K_S, K_X, K_I, K_Z, K_T)$

Token Generation $(q(\bar{w}), K)$

Client's input is K and query $q(\bar{w} = (w_1, ..., w_n))$.
Computes $\mathsf{stag} \leftarrow \mathsf{TSet.GetTag}(K_T, w_1)$.
Client sends stag to the server.
for $c = 1, 2, \ldots$ until the server stops **do**
    for $i = 2, \ldots, n$ **do**
        $\mathsf{xtoken}[c, i] \leftarrow g^{F_p(K_Z, w_1||c)\cdot F_p(K_X, w_i)}$
    end for
    $\mathsf{xtoken}[c] \leftarrow (\mathsf{xtoken}[c, 2], \ldots, \mathsf{xtoken}[c, n])$
end for
$Tok_{\mathbf{q}} \leftarrow (\mathsf{stag}, \mathsf{xtoken})$
return $Tok_{\mathbf{q}}$

Search $(Tok_{\mathbf{q}}, \mathsf{EDB})$

ERes $\leftarrow \{\}$
$t \leftarrow TSetRetrieve(TSet, stag)$
for $c = 1, ..., |t|$ **do**
    Retrieve (e, y) from the $c-$th tuple in t
    if $\mathsf{xtoken}[c, i]^y \in \mathsf{XSet}$ for all $i = 2, ..., n$ **then**
        $ERes \leftarrow ERes \cup \{e\}$
    end if
end for
return ERes

Retrieve

Client sets $K_e \leftarrow F(K_S, w_1)$; for each $e \in ERes$ received, computes $id \leftarrow Dec(K_e, e)$ and outputs id.

T-Set Instantiation: Cash et al. in [3] instantiate a T-set as a hash table with B buckets of size S. The $\mathsf{TSetSetup}(\mathsf{T})$ procedure sets the parameters B and S depending on the total number $N = \Sigma_{w \in W}|T[w]|$ of tuples in T in such a way so that (1) the probability of an overflow of any bucket after storing N elements in this hash

table is a sufficiently small constant; and (2) the total size $B.S$ of the hash table is O(N). T-set instantiation $\Sigma = (\mathsf{TSetSetup}, \mathsf{TSetGetTag}, \mathsf{TSetRetrieve})$ given by Cash et al. [3] as shown in Algorithm 2.

Algorithm 2 T-Set Instantiation

TSetSetup

1- Initialize an array TSet of size B whose every element is an array of S records of type record.
2- Initialize an array Free of size B whose elements are integer sets, initially all set to $\{1, ..., S\}$.
3- Choose a random key K_T of PRF F.
4- Let W be the set of keywords in DB . For every $w \in W$ do the following:
- Set $stag \leftarrow F(K_T, w)$ and $t \Leftarrow T|w|$.
- For each $i = 1, ..., |t|$, set s_i as the i-th string in t, and perform the following steps:

- Set $(b, L, K) \leftarrow H(F(stag, i))$.
- If Free[b] is an empty set, restart TSetSetup(T) with fresh key K_T.
- Choose $j \in_r Free[b]$ and remove j from set Free[b].
- Set bit β as 1 if $i < |t|$ and 0 if $i = |t|$.
- Set $TSet[b, j].label \leftarrow L$ and $TSet[b, j].label \leftarrow (\beta|s_i) \oplus K$.

Output $(TSet, K_T)$.

TSetRetrieve$(TSet, stag)$

Output $stag \leftarrow F(K_T, w)$

TSetGetTag(K_T, w)

1- Initialize t as an empty list, bit β as 1, and counter i as 1.
2- Repeat the following loop while $\beta = 1$:

- Set $(b, L, K) \leftarrow H(F(stag, i))$ and retrieve an array $B \leftarrow TSet[b]$
- Search for index $j \in \{1, ..., S\}$ s.t. $B[j].lable = L$.
- Let $v \leftarrow B[j].value \oplus K$. Let β be the first bit of v, and s the remaining $n(\lambda)$ bits of v.
- Add string s to the list t and increment i.

Output t.

2.3.3 RSA-SSE

The proposed scheme by Sun et al. [49] is an enhanced version of OXT which could achieve multi-user setting using RSA and Attribute Base Encryption (ABE).

$\mathsf{EDBSetup}(1^\kappa, \mathsf{DB}, \mathrm{RDK}, \mathcal{U})$: Similar to OXT the main goal of this algorithm is to generate the encrypted database, EDB. Since, it involves ABE, beside of the security parameter κ, a database DB, a retrieval decryption key array RDK and an attribute universe $\mathcal{U}$ need to be considered as inputs. After performing $\mathsf{ABE.Setup}(1^\kappa, \mathcal{U})$ and few other computations, it generates EDB and XSet as shown in Algorithm 3 [49].

Algorithm 3 RSA-SSE: EDB Setup Algorithm

Require: MK, PK, DB, RDK
Ensure: $\mathsf{EDB}, \mathsf{XSet}$

function EDBGen(MK, PK, DB, RDK)
  $\mathsf{EDB} \leftarrow \{\}$; $\mathsf{XSet} \leftarrow \emptyset$
  for $w \in \mathrm{W}$ **do**
    $c \leftarrow 1$; $\mathsf{stag}_w \leftarrow F(K_S, g_1^{1/w} \mod n)$
    for $id \in \mathsf{DB}[w]$ **do**
      $\ell \leftarrow F(\mathsf{stag}_w, c)$; $e \leftarrow \mathsf{ABE.Enc}(mpk, id||k_{id}, \mathbb{A})$
      $\mathrm{xind} \leftarrow F_p(K_I, id)$; $z \leftarrow F_p(K_Z, g_2^{1/w} \mod n||c)$
      $y \leftarrow \mathrm{xind} \cdot z^{-1}$; $\mathsf{xtag} \leftarrow g^{F_p(K_X,\, g_3^{1/w} \mod n)\cdot \mathrm{xind}}$
      $\mathsf{EDB}[\ell] = (e, y)$; $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup \{\mathsf{xtag}\}$
      $c \leftarrow c + 1$
    end for
  end for
  return $\mathsf{EDB}, \mathsf{XSet}$
end function

$\mathsf{TokenGen}(sk, Q)$: This algorithm inputs the private key sk generated by the data owner and search query Q and then outputs the search token as shown in algorithm 4 [49].

Algorithm 4 RSA-SSE: Token Generation Algorithm

Input: sk, Q
Output: st

function TokenGen(sk, Q)
  $st, \mathsf{xtoken} \leftarrow \{\}$; $\bar{\mathbf{s}} \leftarrow \emptyset$
  $\bar{\mathbf{s}} \leftarrow \bar{\mathbf{s}} \cup \{w_1'\}$
  $\mathbf{x} \leftarrow \bar{\mathbf{w}} \setminus \bar{\mathbf{s}}$
  $\mathsf{stag} \leftarrow F\big(K_S, (sk_{\mathbf{w}}^{(1)})^{\prod_{w\in\mathbf{w}\setminus\{w_1'\}} w} \mod n\big) = F(K_S, g_1^{1/w_1'} \mod n)$
  for $c = 1, 2, \ldots$ until the server stops **do**
    for $i = 2, \ldots, m$ **do**
      $\mathsf{xtoken}[c, i] \leftarrow g^{F_p\left(K_Z, (sk_{\mathbf{w}}^{(2)})^{\prod_{w\in\mathbf{w}\setminus\{w_1'\}} w} \mod n||c\right)\cdot F_p\left(K_X, (sk_{\mathbf{w}}^{(3)})^{\prod_{w\in\mathbf{w}\setminus\{w_i'\}} w} \mod n\right)}$
      $= g^{F_p(K_Z, g_2^{1/w_1'} \mod n||c)\cdot F_p(K_X, g_3^{1/w_i'} \mod n)}$
    end for
  end for
  $st \leftarrow (\mathsf{stag}, \mathsf{xtoken})$
  return st
end function

$\mathsf{Search}(st, \mathsf{EDB}, \mathsf{XSet})$: This algorithm (given in algorithm 5 [49]) inputs the search

token st for the search over the encrypted database, and outputs the search result R which is an encrypted form of document indexes containing the search keyword. Although this protocol could achieve multi-user setting, the user revocation and key update have not been addressed.

Algorithm 5 RSA-SSE: Search Algorithm

Input: $st = (\text{stag}, \mathsf{xtoken}[1], \mathsf{xtoken}[2], \cdots), \mathsf{EDB}, \mathsf{XSet}$
Output: R

function SEARCH($st, \mathsf{EDB}, \mathsf{XSet}$)
  R $\leftarrow \{\}$
  for stag $\in$ **do**
    $c \leftarrow 1; \ell \leftarrow F(\text{stag}, c)$
    while $\ell \in \mathsf{EDB}$ **do**
      $(e, y) \leftarrow \text{EDB}[\ell]$
      if $\mathsf{xtoken}[c, i]^y \in \mathsf{XSet}$ for all i **then**
        $R \leftarrow R \cup \{e\}$
      end if
      $c \leftarrow c + 1; \ell \leftarrow F(\text{stag}, c)$
    end while
  end for
  return R
end function

2.3.4 Geometric Range Searchable Encryption (GRSE)

The proposed work by Wang et al. [50] enables geometric range queries on encrypted spatial data. This scheme uses SSW (Shen, Shi and Waters) encryption as the building block. As shown in Figure 2.2 their probabilistic scheme first adds the points to the Bloom filter (BF) and then encrypts all bits of the BF. To generate the search token, user enumerates all of the possible points inside the geometric range query and then adds the corresponding ciphertexts to a Bloom filter. To check whether a point is inside the queried range, server checks whether an encrypted data record is an element contained in the BF given as the search token. GRSE is linear search regarding to the number of points in a dataset.

GRSE consists of the following algorithms:

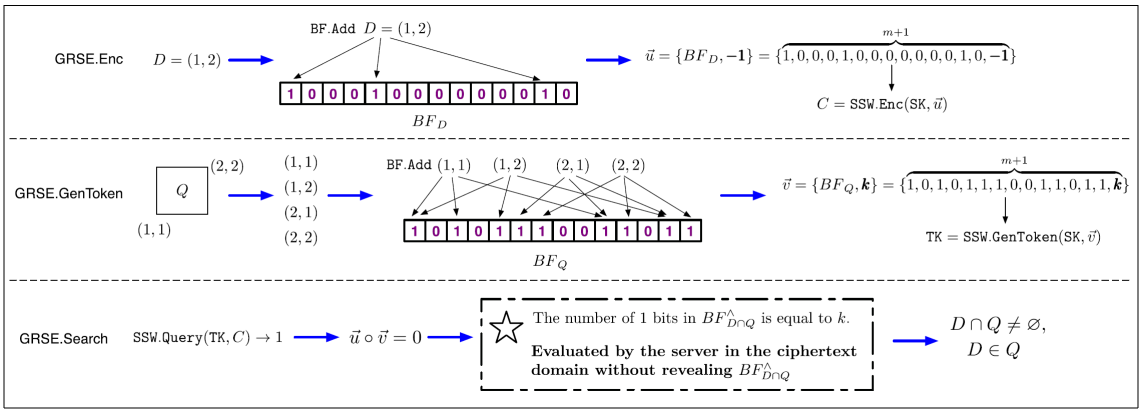


Figure 2.2: Overview of GRSE scheme

$\mathsf{GenKey}(1^\lambda)$: Given (1^λ) as an input, the data owner runs SSW to generate the secret key SK. This algorithm outputs $\{m, h_1, \ldots, h_k\}$ as the public parameters, where m is the length of the BF and the rest are hash functions.

$\mathsf{Enc}(SK, D)$: On the input of the secret key and a dataset $D = (D_1, \ldots, D_n)$ where $D_i \in \Delta_T^w$, for $1 \leq i \leq n$, the data owner computes the corresponding Bloom filter $BF_{D_i} := \mathrm{BF.Init}(m)$, $BF_{D_i} := \mathrm{BF} \cdot \mathrm{Add}\,(D_i, BF_{D_i})$. Here, $BF_{D_i} = (b_{D_i,1}, \ldots, b_{D_i,m})$ and $b_{D_i,j} \in \{0,1\}$, for $1 \leq j \leq m$. Then, it pads $\vec{u}_i = (BF_{D_i}, -1) \in \{0,1\}^m \times \{-1\}$, and calculates the ciphertext C_i of $\vec{u}_i$ using SSW encryption. Finally it outputs $\mathbf{C} = (C_1, \ldots, C_n)$.

$\mathsf{GenToken}(SK, Q)$: This algorithm inputs the secret key and a geometric search query $Q \subseteq \Delta_T^w$ and computes $\mathbf{S} = \{S_1, \ldots, S_t\} :=$ EnumerateInsidePoints (Q), $BF_Q := \mathrm{BF} \cdot \mathrm{Init}(m)$, and $BF_Q := \mathrm{BF.Add}\,(S_i, BF_Q)$, for $1 \leq i \leq t$. Here t is the number of possible points in the query, $BF_Q = (b_{Q,1}, \ldots, b_{Q,m})$ and $b_{Q,j} \in \{0,1\}$, for $1 \leq j \leq m$. It pads $\vec{v} = (BF_Q, k) \in \{0,1\}^m \times \{k\}$ and outputs the search token TK which is generated by running SSW.

$\mathsf{Search}(TK, \mathbf{C})$: On the input of the search token TK and the encrypted dataset $\mathbf{C} = (C_1, \ldots, C_n)$, server runs the SSW query algorithm $\mathrm{Flag}_i \leftarrow \mathrm{SSW} \cdot \mathit{query}\,(\mathrm{TK}, C_i)$, for $1 \leq i \leq n$, if the $\mathrm{FIag}_i = 1$, it returns the corresponding ciphertext.

2.4 Security

2.4.1 SSE Leakage Profile

In general, the leakage function of an SSE scheme, $\mathcal{L}(\mathsf{DB}, \mathbf{q})$, for $\mathsf{DB} = (\mathsf{id}_i, \mathsf{W}_i)_{i=1}^n$ and $\mathbf{q} = (s[i], x[i])$, can be defined as a tuple $(N, \bar{\mathbf{s}}, \mathrm{SP}, \mathrm{RP}, \mathrm{IP})$ formed as follows [3]:

- $N = \sum_{i=1}^{n} |\mathrm{W}_{id_i}|$ is the number of keyword-document pairs, which is the size of EDB.
- $\bar{\mathbf{s}} \in \mathbb{N}^T$ is the equality pattern of the sterms $\mathbf{s}$, indicating which queries have the same sterms. It is calculated as an array of integers, such that each integer represents one sterm. For instance, if we have $\mathbf{s} =$ (a, b, c, a, a), then $\bar{\mathbf{s}} =$ (1, 2, 3, 1, 1).
- SP is the size pattern of the queries, which is the number of matching results returned for each stag.
- $\mathrm{RP}[t, \alpha] = \mathsf{DB}[\mathbf{s}[t]] \cap \mathsf{DB}[\mathbf{x}[t, \alpha]]$, where $\mathbf{s}[t] \neq \mathbf{x}[t, \alpha]$, reveals the intersection of the sterm with any other xterm in the same query.
- $\mathrm{SRP}[t] = \mathsf{DB}[\mathbf{s}[t]]$ is the search result pattern corresponding to the stag of the t-th query.
- $\mathrm{IP}[t_1, t_2, \alpha, \beta] = \begin{cases} \mathsf{DB}[\mathbf{s}[t_1]] \cap \mathsf{DB}[\mathbf{s}[t_2]], \\ \text{if } \mathbf{s}[t_1] \neq \mathbf{s}[t_2] \text{ and } \mathbf{x}[t_1, \alpha] = \mathbf{x}[t_2, \beta] \\ \emptyset, \\ \text{otherwise} \end{cases}$ is the conditional intersection pattern, which is a generalization of the IP structure in [3].
- $\mathrm{XT}[t] = |\mathbf{x}[t, \cdot]|$ is the number of xterms in the t-th query.

2.4.2 DSSE Leakage Profile

In this section, we define the general leakage functions, $\mathcal{L}$, associated with dynamic searchable symmetric encryption schemes [26].

- $sp(w) = \{u : (u, w) \in Q\}$ is the search pattern which shows two search queries pertain to the same keyword, w. This leakage function records the list Q of every search query, in the form (u, w), where u is the timestamp (increases with every query).
- $UpHist(w)$ is a history which outputs the list of all updates on keyword w. Each element of this list is a tuple (u, op, ind), where u is the timestamp of the update, op is the operation, and ind is the updated index.
- $TimeDB(w)$ is the list of all documents matching w, excluding the deleted ones, together with the timestamp of when they were inserted in the database.
- $Updates(w)$ is the list of timestamps of updates on w.
- $DelHist(w)$ is the deletion history of w which is the list of timestamps for all deletion operations, together with the timestamp of the inserted entry it removed.

2.4.3 Forward Privacy

Forward privacy plays a critical role in preventing leakage-abuse attacks in dynamic SSE schemes. An SSE scheme is "*forward private*", if there is no relation between an update and previous search results. That is, there is no leakage to the server about the updated keywords or the updated document matching the previous search queries. Stefanov et al. first introduced the notion of forward privacy in [51]; later Bost in [27] presented the formal definition of that and proposed a forward private SSE scheme. Their formal definition of the forward privacy is as follows;

Definition 4. *A $\mathcal{L}$-adaptively-secure SSE scheme Σ is forward private if the update leakage function $\mathcal{L}^{Updt}$ can be written as $\mathcal{L}^{\mathrm{Updt}}(\mathrm{op}, \mathrm{in}) = \mathcal{L}'(\mathrm{op}, \{(\mathrm{ind}_i, \mu_i)\})$, where $\{(ind_i, \mu_i)\}$ is the set of modified documents paired with the number μ_i of modified keywords for the updated document ind_i. Here,* $\mathrm{op} \in \{Add, Del, Mod\}$.

2.4.4 Backward privacy

Backward privacy (Security) with three types of leakages, introduced by [26], is ordered from the most to least secure (Type-I to Type-III). Backward privacy limits the information on the updates affecting a keyword that the server can learn upon a search query on it. In the following definitions, $TimeDB(w)$ is the list of all documents matching w, excluding the deleted ones, $Updates(w)$ is the list of timestamps of updates on w, and $DelHist(w)$ is the list of timestamps for all deletion operations, together with the timestamp of the inserted entry it removed. Type-I leakage (Backward privacy with Insertion Pattern): This leakage revealed the document identifiers matching the issued search keyword when they were inserted, and the total number a_w of updates over the search keyword.

Definition 5 (Backward privacy with Insertion Pattern)**.** *A $\mathcal{L}$-adaptively-secure SSE scheme is insertion pattern revealing backward-private iff the search and update leakage functions $\mathcal{L}^{Srch}$, $\mathcal{L}^{Updt}$ can be written as:*

$\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op)$

$\mathcal{L}^{Srch}(w) = \mathcal{L}''(TimeDB(w), a_w, sp(w))$,

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

Type-II leakage (Backward privacy with Update Pattern): This leakage reveals all of the information contained in Type-I and also reveals when all updates over the search keyword happened without their content.

Definition 6 (Backward privacy with Update Pattern)**.** *A $\mathcal{L}$-adaptively-secure SSE scheme is update pattern revealing backward-private iff the search and update leakage functions $\mathcal{L}^{Srch}$, $\mathcal{L}^{Updt}$ can be written as:*

$\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op, w)$

$\mathcal{L}^{Srch}(w) = \mathcal{L}''(TimeDB(w), Updates(w), sp(w))$,

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

Type-III leakage (Weak Backward privacy): This leakage reveals all of the information contained in Type-II and also reveals which deletion update canceled which previous insertion.

Definition 7 (Weak Backward privacy)**.** *A $\mathcal{L}$-adaptively-secure SSE scheme is update pattern revealing backward-private iff the search and update leakage functions $\mathcal{L}^{Srch}$, $\mathcal{L}^{Updt}$ can be written as:*

$\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op, w)$

$\mathcal{L}^{Srch}(w) = \mathcal{L}''(TimeDB(w), DelHist(w), sp(w))$,

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

Note. Although the definition presented by Bost et al. in [26] omitted the search pattern sp, in all of the above mentioned three types of the backward privacy, the search pattern sp is also leaked by $\mathcal{L}^{Srch}$.

2.5 Summary

In this chapter the required cryptographic background has been reviewed. Different approaches can be used to address the issue of searching on encrypted data. However, searchable symmetric encryption offers more efficient and practical solutions in comparison to the other cryptographic approaches. Thus, this chapter introduced the

alternative approaches, briefly. From the state-of-the-art SSE schemes, we reviewed OXT [3], RSA-SSE [49], and GRSE [50] which are highly referred in the subsequent chapters. Finally, we reviewed the essential security properties for SSE schemes.

Chapter 3

Multi-client symmetric searchable encryption

In this chapter, we present our multi-client symmetric searchable encryption scheme. The initial version of this work was published in Australasian Conference on Information Security and Privacy (ACISP-2017). The extended version is under minor revision in the IEEE Transactions on Dependable and Secure Computing (TDSC).

3.1 Overview

3.1.1 Motivation

Consider Figure 3.1 as a simple example of a database that we want to perform search on it. A naive mode of indexing for search might be the use of Forward indexing as shown in Figure 3.2. In this mode a record points to its keywords. However, this is not an efficient mode as the search complexity is linear to the number of documents $O(d)$ (where d is the number of documents).

Document id	Keywords	Colour	Number plates
1	BMW	White	BA 435
202	Toyota	Blue	GH 949
300	Audi	Gray	KS 305
410	BMW	Black	QP 816
567	Bugatti	Silver	UN 428
610	Honda	White	EL 360

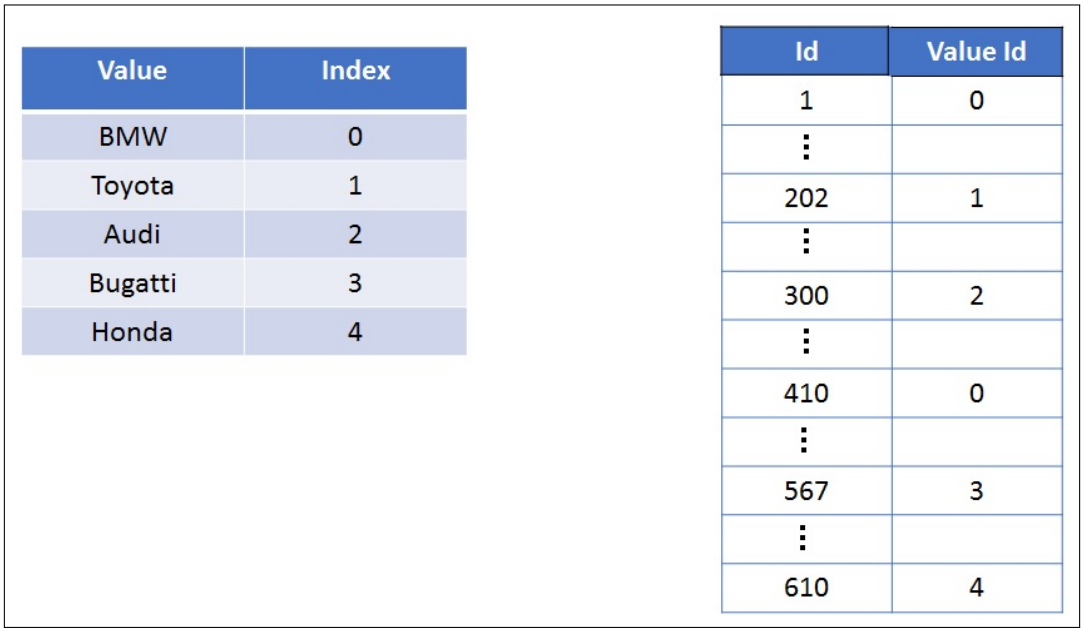
Figure 3.1: Sample database

Figure 3.3 shows a more efficient indexing mode called inverted index. In this mode the search complexity relies on the number of keyword $O(w)$, (here w indicates the number of keywords). In this technique, a keyword points to the records that contain the considered keyword. More precisely, the query is the output of the hash of the keyword and there is a hash table of keywords where the search algorithm look up to retrieve the pointer. However, this approach is insecure due to the excessive leakage to the server.

Cash et al. in [3] proposed Single Keyword Search (SKS) protocol as the basis of the other protocols proposed in the same paper. Figure 3.4 illustrates this protocol. In this protocol they first built an inverted index and instead of using hash function, they utilized PRF to encrypt the keyword using the considered selected key. This approach provides a higher level of security while the same level of search complexity is preserved.

The main goal of this research is to propose Multi-user SE protocols which can overcome the security challenges introduced by Multi-user settings while an acceptable level of efficiency and functionality is considered.

Although Multi-reader setting is more appealing in practice, adapting this setting to SE schemes is left unconsidered due to its challenging security requirements. In this section, we introduce two different approaches for adapting Multi-reader setting with their security drawbacks.



Value	Index
BMW	0
Toyota	1
Audi	2
Bugatti	3
Honda	4

Id	Value Id
1	0
⋮	
202	1
⋮	
300	2
⋮	
410	0
⋮	
567	3
⋮	
610	4

Figure 3.2: Forward index

- **Approach1: Key Sharing**

  This straightforward approach uses a Master-key which is distributed among all legitimate users by the data owner to extend Single-user setting to Multi-user. An example of utilizing this approach is the proposed scheme by Curtmola et al. [4], which applied broadcast encryption to adapt Multi-user setting. However, the authors could not overcome the following drawbacks introduced by this approach;

  - **Key exposure/Collusion:**

    The use of a naive approach of sharing a single key among all of authorized users to transform Single-user setting to multi-user ones, pose a major risk of key exposure of one user leading to making the whole system security. That is, whole DB key exposure risk may multiply by N, where N indicates the number of authorized users.

    In the multi-user schemes that utilized this approach like [4, 15, 16] if a

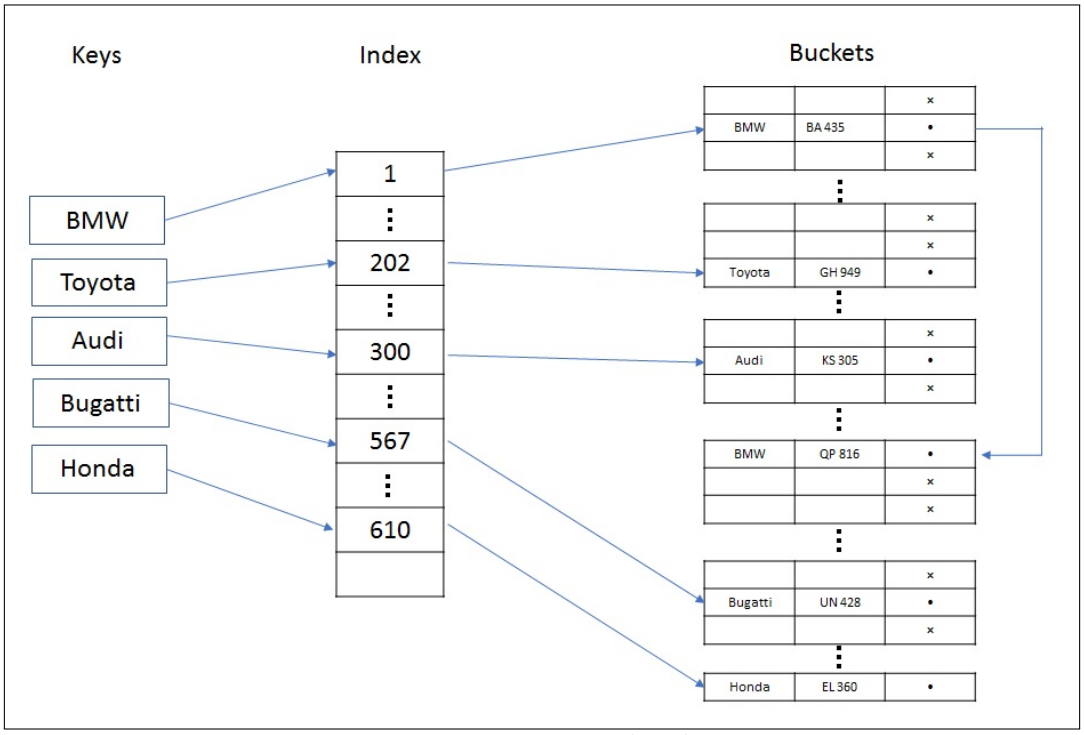


Figure 3.3: Inverted index

user being compromised or collude with an adversary or the cloud server it would result to disclosure of all of the users' information.

- **User Revocation:**

  Due to the use of a single key by all of the authorized users, each revocation requires a new key to be distributed to the remaining users, which is not practical [9].

- **Approach2: Sending Search token**

  In this approach users/readers refer to the data owner to obtain the search token(s). Although this approach can solve key exposure risk, it poses new problems:

  - **Query Privacy:**

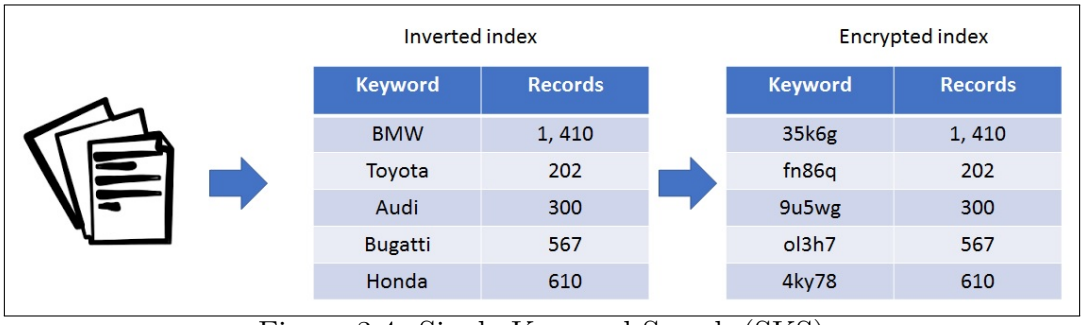


Figure 3.4: Single Keyword Search (SKS)

This approach does not support query privacy against the data owner as the search tokens are generated by data owner then delivered to the users. Thus, the data owner learns about the queries and query privacy is not supported.

– **Functionality Problem:**

Per query interaction with data owner requires online presence of the data owner.

Our approach: θ-threshold key distribution and randomizable PRF

In our approach we applied t-threshold key distribution which facilitate Key revocation by the data owner. That is, data owner distributes the key shares along with the list of authorized users to the authorized users in the setup phase. Thus, to revoke a key share data owner would update the list and the key shares.

Any $\theta-1$ authorized users (here θ is the considered threshold) can help the reader to generate the search token without requiring communication with the data owner. This can solve the functionality problem of Approach2.

In order to overcome the query privacy problem, we proposed a new cryptographic primitive called Randomizable Distributed PRF (RDPRF). More precisely, RDPRF hides the search keyword from the helpers during the token generation process.

Finally, our approach is secure against key exposure by the size of the considered

threshold. That is, low key exposure risk is preserved as key exposure will happen only if the key share of all of t readers be exposed.

3.1.2 Contributions

Our multi-client SSE achieves query privacy against all involving entities: cloud server, data owner and the other key share holders. In addition, once the data owner performed the Encrypted DataBase (EDB) setup phase, it is not necessary for the data owner to be involved in the protocol unless for key update and revocation. At the same time, our protocol is resilient against key exposure of up to $\theta - 1$ client keys. A search is performed by a client via interaction with the server and $\theta - 1$ 'helping' users. Furthermore, the proposed protocol can be applied to a wide range of applications in modern distributed computing platforms such as cloud computing, Internet-of-Things and social computing. This collaborative approach is particularly useful in applications/use cases where a large number of users are always available such as hospitals and police departments or agent-based applications where the agents could be present all the time on-behalf of users [52]. As mentioned earlier, our generic design is inspired by the OXT protocol of Cash et al. [3]. The contributions of our multi-client SSE can be summarized as followed;

- **Privacy and Security:**

    1. **Query privacy:** Our proposed scheme preserves query privacy against both data owner and helping users. This goal is achieved by designing a new primitive named RDPRF (Randomizable Distributed key- homomorphic PRF) which enables the data owner to distribute the PRF key Shares with desirable clients (key share holders) without further interactions. More precisely, clients would be able to carry out the search without per-query interaction with the data owner, which in turn leads to two nice features. First, online presence of the data owner at all the time is not

required since online presence of $\theta - 1$ helping users would be sufficient for the successful operation. Second, queries performed by clients are not monitored by the data owner.

2. **Revocation:** Another desirable feature of our scheme is the key revocation property which can be used to prevent any illegitimate access to EDB by revoked client(s). In the area of SSE, key revocation problem is usually left unaddressed. In this paper we propose two approaches for revocation. First solution enables the data owner to revoke key shares from clients by generating new PRF key shares and distributing them between non-revoked clients while leaving the master key and EDB unchanged to save the costly re-encryption operation. The second approach has minimal communication overhead but requires EDB update.

   Our threat model considers an honest but curious and malicious helping users, we consider the following collusions.

3. **Passive Collusion:** In this condition, the server tries to compromise a user's key to expose the EDB content. In our scheme, the privacy of database content is resilient against the exposure of up to $\theta - 1$ users' keys. More precisely, under a realistic assumption that the data owner does not collude with the server, our multi-client SSE is secure against a passive attack by a server colluding with any coalition of less than θ helping users.

4. **Active Collusion:** Active collusion is a stronger assumption where the server exposes at most $\theta - 1$ key shares (for a threshold parameter θ) and can control those $\theta - 1$ key share holders (non-initiator) in the token generation process. In this active server attack, the server should not learn anything about EDB beyond the search tokens for the search queries (which are trivially learnt by the attacker in this scenario). This protects privacy of a database content not queried by the active attack.

- **Efficiency:** In the current related works [3, 49, 53], if the data owner decides to update the encryption key, a substantial amount of cost would be added to the system in terms of computations performed for encryption of the database using the new key, and bandwidth required for uploading the updated EDB (Encrypted Database) as well as transferring information to the considered clients. We achieve a more efficient solution which enables the data owner to perform one-time update of the encryption key as well as EDB by sending the corresponding key-material to the server.

3.2 Preliminaries

In this section, the required preliminaries are provided.

3.2.1 Notations

Frequently used notations and terminologies in this paper are listed in Table 4.2.

Table 3.1: Notations and terminologies

Notation	Description
id_i	the document identifier of the i-th document
W_{id_i}	a list of keywords contained in the i-th document
$\mathsf{DB} = (id_i, \mathrm{W}_{id_i})_{i=1}^{d}$	a database consisting of a list of document identifier and keyword-set pairs
$\mathsf{DB}[w] = \{id : w \in \mathrm{W}_{id}\}$	the set of identifiers of documents that contain keyword w
$\mathrm{W} = \bigcup_{i=1}^{d} \mathrm{W}_{id_i}$	the keyword set of the database
θ	threshold
TSet	an array that presents equal sized lists for each keyword in the database
XSet	a lookup table that contains elements computed from each keyword-document pair
sterm	the least frequent term among queried terms
xterm	other queried terms (excluding sterm)

3.3 Background

Followed by the first Symmetric Searchable Encryption (SSE) proposed by Song et al. [5], several SSE schemes have been proposed (e.g. [4,6,16,36,54–62]). In the early stage, most of the works were in a single-writer / single-reader setting. However, the single-user setting is not beneficial for cloud storage as usually enterprise cloud servers serve multiple users. Multi-client searchable encryption was pioneered by Curtmola et al. [4] for a symmetric setting. In their scheme, the single-user searchable encryption is transformed to the multi-client one by sharing the key for encryption of the database. This work supports a single keyword boolean search that is very simple in terms of functionality [63].

In order to manage the search ability of users, the proposed scheme by Bao et al. [15] deployed a trusted third party, called User Manager (UM). Although it seems that their scheme could handle the key revocation issue, using such a trusted party is not common in the cloud storage. The proposed schemes in [56, 64, 65] suffer from the similar problem. The proposed work by Popa et al. [66] could overcome the need of a trusted administration. However, it is inefficient due to the unavoidable network bandwidth and storage overhead. Although the proposed scheme by Tang et al. [67] could improve the Popa's scheme from the security perspective, it did not explain how to revoke a user. The problem is similar to multi-client searchable encryption schemes proposed in [68, 69]. Several searchable encryption schemes have been proposed using the public key structure in order to support a multi-client setting [10, 70–73]. However, they have limited applications in practice due to the cost of the public key solutions. Thus, a further investigation of the cons and pros of these works is out of the scope of this research.

Motivated by the SSE protocol of Cash et al. [3], Jarecki et al. [53] and Sun et al. [49] proposed a SSE scheme in the multi-client setting. Although Sun's scheme could improve the communication overhead by avoiding per-query interaction between the data owner and clients as in the Jarecki's scheme, the data owner is still involved in

the token generation process. In addition, the search keywords are predetermined and restricted by the data owner. More importantly, the user enrollment and revocation remained unaddressed in these works. The notion of threshold privacy preserving keyword search (TPPKS) was first introduced by Wang et al. [52]. This TPPKS scheme is based on the Shamir's secret sharing [28] and Boneh and Franklin's ID-based cryptosystem [74]. Although TPPKS has some attractive properties such as share verification, there is a substantial computational overhead due to the use of Bilinear pairings. Moreover, the query privacy against the helping users is not considered in this scheme.

Table 3.2: Comparison of a subset of existing related works

References	Multi-user	Revocation	Query Privacy	Write capability	Key exposure	Key update
[4]	✓	✓	X	✓	X	X
[56]	✓	✓	✓	✓	X	X
[16]	X	N/A	N/A	✓	X	X
[75]	✓	X	X	✓	✓	X
[15]	✓	✓	✓	✓	X	X
[49]	✓	X	Semi	X	✓	X
[3]	X	N/A	N/A	✓	X	X
[53]	✓	X	✓	X	✓	✓
[76]	✓	✓	Semi	X	✓	X

3.4 Syntax of multi-client SSE

Our multi-client SSE construction, Π_{mu}, consists of seven phases $\Pi_{mu} = (EDBSetup_{mu}$, $KeySharing_{mu}$, $TokenGen_{mu}$, $Search_{mu}$, $Retrieve_{mu}$, $Update_{mu}$, $Revoke_{mu})$ as defined below. We point out the following main differences from the syntax of the single-user SSE [3]. To achieve the resilience against the client key loss, the $KeySharing_{mu}$ algorithm allows the Master key k to be split into client key shares $k_1, \ldots, k_N$ using a threshold secret sharing scheme, such that any θ key shares can reconstruct k, for a threshold parameter θ. The $TokenGen_{mu}$ protocol allows a client to compute a search token for a query by interacting with other $\theta - 1$ 'helping' key share holders.

- $EDBSetup_{mu}$: A data owner runs the algorithm $EDBSetup_{mu}$ that takes the security parameter λ and the database DB as inputs and outputs the encrypted database EDB along with the master key k and the set of public parameters pp.

- $KeySharing_{mu}$: The data owner executes this algorithm which takes pp, k, θ and the number of desired key share holders N as inputs, and outputs the generated key shares $(k_1, ..., k_N) \in \mathcal{K}^{\mathcal{N}}$ for the given master key k (here, $\mathcal{K}$ is the key space).

- $TokenGen_{mu}$: The $TokenGen_{mu}$ protocol is run by θ key share holders $(i_1, ..., i_\theta) \in [N]$. The key share holder i_1 who aims to do the search on EDB starts the protocol by taking the query q, public parameters pp and the key share k_{i_1} as inputs. The $\theta - 1$ other key share holders i_j where $j \in [2, \theta]$ input their key shares k_{i_j} during the protocol run. By the end of the protocol, the Key share holder i_1 outputs the search token Tok_q whereas the remaining key share holders output $\perp$.

- $Search_{mu}$: This is a protocol run between the key share holder $i_1 \in [N]$ and the server, where i_1 provides the search token Tok_q along with pp and the server provides pp and EDB. By the end of this protocol, the key share holder i_1 outputs the encrypted result $ERes$ whereas the server outputs $\perp$.

- $Retrieve_{mu}$: This is a protocol run between the key share holder i_1 and the $\theta - 1$ other key share holders $(i_2, ..., i_\theta) \in [N]$, where $(pp, ERes, k_{i_1})$ are the inputs of i_1 and the $\theta - 1$ other key share holders i_j input (pp, k_{i_j}) (where $j \in [2, \theta]$). Finally, the key share holder i_1 outputs Res which is the identifiers of the documents containing the issued query whereas the remaining involved key share holders output $\perp$.

- $Update_{mu}$: This is a protocol between the data owner and the server. The data owner initiates this protocol by choosing two new keys K'_S, K'_T for updating

TSet and generating a keying material Δ_E for the update of XSet. Then, the data owner updates TSet and delivers TSet' to the server and shares of K'_S, K'_T to the clients. Given Δ_E along with the encrypted database as inputs, the server outputs the updated EDB whereas the data owner outputs the updated TSet which is delivered to the server and the shares of K'_S, K'_T.

- $Revoke_{mu}$: The data owner runs this algorithm which takes pp, k, i_R (the identity of the client to be revoked), and the number of desired key share holders N as inputs, and outputs the generated new key shares $\vec{Sh} = (k_1, ..., k_N) \in \mathcal{K}^{\mathcal{N}}$ for the given master key k along with the updated list of legitimate clients which excludes i_R. This algorithm can be performed in two different ways. Approach 1 is more efficient in terms of computational cost required for the update of the encrypted database after a certain number of user revocation. Approach 2 allows key update with minimal communication overhead via a broadcast message without requiring peer to peer interactions between the data owner and the non-revoked clients.

3.5 Security definitions of multi-client SSE

In this section, we give security definitions of our multi-client searchable encryption based on different viewpoints.

3.5.1 Privacy against server

The given semantic security definitions is similar to [3]. The security definition of our multi-user searchable encryption, here Π, is parametrized by a leakage function $\mathcal{L}$ (refer to Section 2.4.1).

Indeed, security shows how the server's view in an adaptive attack (database and queries are selected by the server) can be simulated using only the output of $\mathcal{L}$. For

algorithms $\mathcal{A}$ and $\mathcal{S}$, we define a real experiment $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$ and an ideal experiment $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$ as follows:

$\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$: $\mathcal{A}(1^{\lambda})$ chooses a database DB and a subset of corrupted clients $(j_1, ..., j_{\theta-1})$. However, this non-adaptive attacker is not allowed to interact with the clients. The experiment samples a k_0 uniformly from $\mathcal{K}$ and runs $\mathsf{K}ey\ sharing(k_0)$ to obtain $(k_1, ..., k_N)$. Then, responds with the corresponding key shares $k_{j_1}, ..., k_{j_\theta-1}$ as the exposed key shares to the attacker.

Afterwards, the experiment runs $(mk, pp, \mathsf{EDB}, \mathsf{XSet}) \leftarrow \mathsf{EDBSetup}(1^{\lambda}, \mathsf{DB})$ and returns $(pp, \mathsf{EDB}, \mathsf{XSet})$ along with C (the list of key share holders, $|C| = N$) to $\mathcal{A}$. Then $\mathcal{A}$ repeatedly chooses a query $q[i]$. Then, the experiment runs the algorithm $TokenGen$ on input $k_{i_1}, ..., k_{i_\theta}$, and returns Search tokens to $\mathcal{A}$. Eventually, the experiment outputs the bit that $\mathcal{A}$ returns.

$\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$: By setting a counter $i = 0$ and an empty list $\mathbf{q}$ the game is initialized. $\mathcal{A}(1^{\lambda})$ chooses a DB , a query list q and a subset of corrupted clients $(j_1, ..., j_{\theta-1})$. The experiment responds with simulated key shares $k_{j_1}, ..., k_{j_\theta-1}$ as the exposed key shares to the attacker. Afterwards, the experiment runs $(pp, \mathsf{EDB}, \mathsf{XSet}) \leftarrow \mathcal{S}(\mathcal{L}(\mathsf{DB}))$ and gives $(pp, \mathsf{EDB}, \mathsf{XSet})$ to $\mathcal{A}$. $\mathcal{A}$ then repeatedly chooses a search query q. To respond, the experiment records this query as $\mathbf{q}[i]$, increments i and gives the output of $\mathcal{S}(\mathcal{L}(\mathsf{DB}, \mathbf{q}))$ to $\mathcal{A}$, where $\mathbf{q}$ consists of all previous queries in addition to the latest query issued by $\mathcal{A}$. Eventually, the experiment outputs the bit that $\mathcal{A}$ returns.

Definition 8 (Security)**.** *The protocol* Π *is called* $\mathcal{L}$*-semantically-secure against adaptive attacks if for all adversaries* $\mathcal{A}$ *there exists an efficient algorithm* $\mathcal{S}$ *such that* $|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)]| \leq \mathrm{negl}(\lambda)$.

Definition 9 (Leakage)**.** *The leakage,* $\mathcal{L}$*, is similar to [3] as presented in Section 2.4.1.*

3.5.2 Query privacy against other key share holders

Query privacy is a new property for hiding the search keywords of a client from the other $\theta-1$ 'helper' key share holders (in our construction, this means that the search query input of PRF is hidden from the 'helper' key share holders).

Indeed, this security shows the compromised clients' view in an adaptive attack (key share holders and queries are selected by the compromised 'helper' clients). For algorithms $\mathcal{A}$ and $\mathcal{S}$, we define a *Query Privacy Game* as follows:

Query Privacy Game : $\mathcal{A}(1^\lambda)$ chooses a set of $i_1, ..., i_\theta$ of key share holders, where i_1 is the client making a query and $i_2, \ldots, i_\theta$ are the 'helping' key share holders. Then $\mathcal{S}$ samples a K_0 uniformly from $\mathcal{K}$ and runs $\mathsf{K}ey\ sharing(K_0)$ to obtain $(K_1, ..., K_N)$, and returns $K_{i_2}, ..., K_{i_{\theta-1}}$ to $\mathcal{A}$. Then $\mathcal{A}$ chooses a pair of keyword queries (x_0, x_1).

In order to respond, $\mathcal{S}$ chooses a random bit $b \in_r \{0,1\}$ and runs $TokenGen_{mu}$ protocol with searching client's input $(q = x_b, pp, K_{i_1})$ and 'helper' client inputs $(K_{i_2}, \ldots, K_{i_\theta})$. $\mathcal{S}$ returns the protocol view of the 'helping' share holders $i_2, \ldots, i_\theta$ to $\mathcal{A}$. Then, $\mathcal{A}$ outputs a bit b' which is also outputs by the algorithm $\mathcal{S}$. As a result, $Adv(A) = Pr(b = b') - \frac{1}{2}$.

Definition 10 (Query Privacy)**.** *The protocol* Π *is called Query private if for all adversaries* $\mathcal{A}$ *in Query Privacy Game* $Adv(A) \leq \text{negl}(\lambda)$.

3.5.3 Database content privacy against active collusion

This security shows the server's view in an adaptive attack (DB and queries are chosen by the server) where the server colludes with $\theta - 1$ of helping users (non-initiator). This security is the same as that of "Privacy against server" except the adversary controls the behaviour of these compromised clients in *TokenGen* and *Retrieve* protocols. Note that the leakage function is the same as Definition 9.

Definition 11 (Active Collusion Resilience)**.** *The protocol* Π_{mu} *is called Active Collusion Resilient if for all adversaries* $\mathcal{A}$ *in Active Collusion game as defined above,* $Adv(\mathcal{A}) \leq \text{negl}(\lambda)$.

Remark. There might be an active collusion where a server colludes with a client as an initiator. Ideally, we would like this adversary unable to get any knowledge about database beyond whatever is queried by the initiator. We leave the security of this type of active collusion open for a future extension.

3.5.4 Database content privacy after update

This security shows the attacker's view in an attack against the privacy of the database content after an update. This security is similar to that of "Privacy against Server" except that beside a snapshot of the updated encrypted database, the value of old master key and public parameters are known to the adversary. It is worth to note that the attacker is not allowed to make TokenGen queries. Therefore, there is no leakage; $\mathcal{L} = \emptyset$.

Definition 12 (Database content privacy after update)**.** *The protocol* Π_{mu} *is called database content private after an update if for all adversaries* $\mathcal{A}$ *there exists an efficient algorithm* $\mathcal{S}$ *such that* $|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)]| \leq \text{negl}(\lambda)$.

3.5.5 Database content privacy after revocation

This security shows the attacker's view against the privacy of database content after user revocation. For Revocation-Approach 1, if the revoked clients from different time periods (time between two key refreshment) form a coalition, they cannot distinguish the search token of the non-queried keywords from a random value even when the number of members on such coalition is larger than the considered threshold (here θ). Note that this adversary is not allowed to communicate with the non-revoked key share holders.

For algorithms $\mathcal{A}$ and $\mathcal{S}$, we define *Revocation Privacy Game* as follows.

Revocation Privacy Game: $\mathcal{A}(1^\lambda)$ specifies the time t for the last key refreshment and the revoked clients $\left\{u_1^i, ..., u_{\theta-1}^i\right\}_{i=1}^t$ for each time period before t, then a simulator $\mathcal{S}$ returns with their corresponding key shares $\left\{k_1^i, ..., k_{\theta-1}^i\right\}_{i=1}^t$.

$\mathcal{A}$ sends TokenGen queries for some keywords $w_1, ..., w_Q \in \mathcal{W}$ to $\mathcal{S}$, and for each query w_j, the simulator $\mathcal{S}$ responds with Tok_j. Once $\mathcal{A}$ decides to challenge, in new time period t' (where $t' \neq t$ and $t' > t$), it submits a pair of fresh keywords (w_0^*, w_1^*) where $w_0^*, w_1^* \notin \mathcal{W}$. In response, $\mathcal{S}$ chooses a random bit $b \in_r \{0, 1\}$. If $b = 0$, $\mathcal{S}$ samples Tok_0 at random. Otherwise, it runs the $TokenGen_{mu}$ protocol using the key of the time t' and returns the output to $\mathcal{A}$. Then, $\mathcal{A}$ outputs a bit b' which is also an output by the algorithm $\mathcal{S}$. As a result, $Adv(\mathcal{A}) = Pr(b = b') - \frac{1}{2}$.

Definition 13 (Security after Revocation-Approach 1)**.** *The protocol* Π_{mu} *is secure after Revocation-Approach 1 if for all adversaries* $\mathcal{A}$ *in Revocation Privacy Game ,* $Adv(\mathcal{A}) \leq \text{negl}(\lambda)$.

Remark. As mentioned in Definition 13, the security is defined for non-queried keywords. The search tokens of the previously queried keywords by the revoked users (at the time that they were still legitimate) would remain available to them. Thus, there is a chance of replay attack. In order to mitigate this attack, other security mechanisms might be applied such as providing a list of legitimate clients to the server or performing challenge-response authentication by the server.

For Revocation-Approach 2, revoked clients (any coalition smaller than the threshold) should not be able to generate any valid search token(s). This security follows the rules of *Revocation Privacy Game* except here the public information (the broadcast message) is given to the adversary. Moreover, an adversary can specify only $\theta - 1$ of the revoked clients.

Definition 14 (Security after Revocation-Approach 2)**.** *The protocol* Π_{mu} *is secure after Revocation-Approach 2 if for all adversaries* $\mathcal{A}$ *in Revocation Privacy Game (with the above mentioned modifications)* $Adv(\mathcal{A}) \leq \text{negl}(\lambda)$.

3.6 Randomizable Distributed Key Homomorphic PRFs

As a tool for our multi-client SSE, we define a special type of distributed PRF which makes the input point blind from key share holders and unblind it while executing the E*valuation* algorithm. More precisely, PRF evaluation protocol plays a major role in our scheme. It enables our scheme to support multi-client setting while it prevents the leakage of the search keyword when the users collaborate with each other in token generation and result retrieval.

3.6.1 Definition

Our Randomizable Distributed key-homomorphic PRF (RDPRF) mainly consists of seven algorithms; S*etup*, K*ey sharing*, R*and*, P*artial evaluation*, C*ombine*, U*nRand*, and E*valuation* with the following properties. Note that except R*and* and U*nRand*, the rest of the mentioned algorithms are similar to the ones proposed by Boneh et al. [35].

- S*etup*: This algorithm takes a security parameter κ as an input and outputs the master-key mk and public parameters pp.
- K*ey sharing* $(pp, (K, N)) \rightarrow \mathcal{K}^{\mathcal{N}}$: This algorithm generates key shares $(k_1, ..., k_N) \in \mathcal{K}^{\mathcal{N}}$ of the considered master key. Here, N is the number of key share holders, $\mathcal{K}$ is the key domain and $\mathcal{K}^{\mathcal{N}}$ is the domain of key shares.
- R*and*: The function $Rand(x, r, pp) \rightarrow z$ randomizes an input point $x \in \mathcal{K}$ by a uniformly random value $r \in \mathbb{Z}_p^*$.
- P*artial evaluation*: The function $\mathcal{F} : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ takes a key share and an input point, and outputs a partial evaluation of the E*valuation* function f.

- C*ombine*: The algorithm $G : 2^{[N]} \times \mathcal{Y}^{\theta} \rightarrow \mathcal{Y}$ takes θ (threshold), a subset $W \subset [N]$ of size θ and partial evaluations on key shares in the set W as inputs, and outputs a value in $\mathcal{Y}$.
- U*nRand*: The function $UnRand(\mathcal{F}(k, z), r)$ takes the randomized distributed PRF (RDPRF) under the key k and the utilized random value r as inputs and outputs the unradomized DPRF using the inverse of r.
- E*valuation*: The function $f : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ maps a key and an input to the space of outputs.

3.6.2 PRF Evaluation protocol

Algorithm 6 must be performed for all of the utilized PRFs. The idea is to make the clients able to search through the database without interaction with the data owner. More precisely, online presence of the data owner at all the time is not required. In addition, Algorithm 6 enables the client to hide the considered keyword for search from the other key share holders. Assume that a client wants to evaluate a PRF such as $PRF(k_0, x)$. The client first sends a short message containing x to each key share holder. After receiving at least $\theta - 1$ responses from them, the client would be able to evaluate the mentioned PRF using the PRF Evaluation Protocol.

Algorithm 6 is inspired from the proposed key homomorphic PRFs by Boneh et al. [35] (refer to supplementary materials). Note that our algorithm does not leak any information such as the considered keyword to other key share holders.

3.6.2.1 Correctness.

Consider pp as the output of the S*etup*(1^{λ}), k_0 is sampled uniformly from $\mathcal{K}$, and $(k_1, ..., k_N)$ is the key share output by K*ey sharing*(k_0). For every subset $W = i_1, ..., i_{\theta} \subset [N]$ of size θ, and for every input x, RDPRF is correct if $f(k_0, x) = UnRand(G(W, \mathcal{F}(k_{i_1}, z), ..., \mathcal{F}(k_{i_{\theta}}, z))$.

Algorithm 6 PRF Evaluation protocol

Input:
For client i_c: key share holders list $C = i_1, i_2, ..., i_N$, key share k_{i_c}, RDPRF input x, and threshold θ
For Helping users i_j: key share holders list $C = i_1, i_2, ..., i_N$, key shares k_{i_j} for $j = 1, ..., \theta - 1$
Output: $Y = PRF(k_0, x)$
function PRFEVAL(k_0, x)
Y $\leftarrow \{\}$
client i_c picks random $r \in \mathbb{Z}_p^*$
client i_c computes $z \leftarrow Rand(x, r, pp)$
$j \leftarrow 1$
for $j = 1, ..., \theta - 1$ **do**
while $i_j \in C$ **do**
client i_c send z to i_j
i_j computes $y_{i_j} = \mathcal{F}(k_{i_j}, z)$
i_j sends y_{i_j} to i_c
$j \leftarrow j + 1$
end while
client i_c performs $y \leftarrow Combine(W, y_{i_1}, ..., y_{i_\theta})$
end for
client i_c compute $y \leftarrow Unrand(y, r)$
return Y
end function

3.6.2.2 PRF security definition of RDPRF.

The E*valuation* function f should remain pseudorandom even when the adversary is given $\theta - 1$ key shares $(k_{i_1}, ..., k_{i_{\theta-1}})$ for indices $i_1, ..., i_{\theta-1}$ of its choice. The adversary is also given an oracle $\mathcal{O}$ that performs arbitrary partial evaluations: it takes (i, x) as input and returns $\mathcal{F}(k_i, x)$. The adversary should be unable to distinguish the function from random at point x.

3.6.2.3 RDPRF Active Collusion Security Definition.

We present a stronger security model for RDPRF which is an extension to "PRF security definition of RDPRF". This model for active collusion provides randomisation for same inputs. More precisely, the adversary is given an oracle $\mathcal{O}'$ that takes x as input and returns $(z, (y_{i_1}, ..., y_{i_\theta - 1}), y)$ which first evaluates $z = Rand(x, r)$ (for uniformly random r), then partial evaluations $(y_{i_j} = \mathcal{F}(k_{i_j}, z))_{j=1}^{\theta-1}$ and finally full evaluation $y = \mathcal{F}(k_0, x)$.

The adversary should be unable to distinguish two games; REAL and RAND.

REAL is the above mentioned game with $\mathcal{F}$, the real PRF, whereas RAND is the above mentioned game with $\mathcal{F}$, a truly random function.

3.6.2.4 Query privacy security of RDPRF

The query privacy security of the Randomizable Key Homomorphic Distributed PRF shows that for each $x \in \mathcal{X}$, the output of $Rand(x, r)$ is uniform on the domain of the distributed PRF, $\mathcal{X}$, when r is uniform in the input space $\mathbb{Z}_p^*$.

3.6.3 Concrete construction of RDPRF

- S*etup*: This algorithm first chooses prime a p, $\mathcal{K} = \mathbb{Z}_p^*$ as the key domain, a cyclic group G of prime order p, and the master key $k_0 \in_r \mathcal{K}$. Then, it defines $S = (\theta, N)$, a threshold secret sharing scheme proposed by Shamir [28] (refer to section 2.1.1). Finally, it defines an output $mk = k_0$, $pp =< G, p, S, \mathcal{K}, H >$ where $H : \mathcal{X} \to G|\{1\}$ is a cryptographic hash function which maps the PRF domain $\mathcal{X} : \{0,1\}^*$ to its range, $G|\{1\}$, using a randomizer $r \in_r \mathcal{R}$ where $\mathcal{R} = \mathbb{Z}_p^*$.

- K*ey sharing* $(pp, (mk, N)) \to \mathcal{K}^{\mathcal{N}}$: On inputs $mk = k_0$ and an integer N, the threshold secret sharing scheme S outputs the key shares $k_1, ..., k_N$.

- R*and*(x, r, pp): Picks up a uniformly random value $r \in_r \mathbb{Z}_p^*$ and outputs r and $z = H(x)^r$.

- P*artial evaluation* (k_i, z): Given a key share $k_i \in \mathbb{Z}_p^*$ and an input point $z \in G$, this algorithm returns $y_i = \mathcal{F}(k_i, z)$ where $y_i = z^{k_i}$ is an element of the group G.

- C*ombine* $(W, \{y_{i_1}, ..., y_{i_\theta}\})$: For any $W = \{i_1, ..., i_\theta\} \subset [N]$ of size θ and the corresponding partial evaluations $\{y_{i_1}, ..., y_{i_\theta}\}$, this algorithm outputs $f(k_0, z) = z^{k_0} = z^{\sum_{j=1}^{\theta} \lambda_{i_j} . k_{i_j}} = \Pi_{j=1}^{\theta} (y_{i_j})^{\lambda_{i_j}}$.

- $\mathsf{U}nRand\ (\mathcal{F}(k_0, z), r)$: this algorithm takes $y_r = \mathcal{F}(k_0, z) = H(x)^{rk_0}$ and the related random value r as inputs and then outputs the unrandomized value $y = y_r^{r^{-1} \pmod p}$.

- $\mathsf{E}valuation(k, x)$: given a secret k and an input x, this algorithm outputs $f(k, x) = H(x)^k$.

Theorem 1. *Suppose the hash function H is a random oracle, then RDPRF is a secure randomizable key homomorphic distributed PRF that satisfies PRF security if the DDH assumption holds and the query privacy is satisfied unconditionally.*

Proof. This proof consists of two parts; PRF Security and query privacy.

PRF Security. To prove the theorem, consider $Expt_b{}^{RDPRF'}$ for $b \in \{0, 1\}$. Now we construct a simulator $\mathcal{B}$ that uses an adversary $\mathcal{A}$ to gain advantage against PRF F in $Expt_b{}^{DDH}$. The game between the challenger and $\mathcal{B}$ starts with generating the public parameters pp by the challenger and transferring them to $\mathcal{B}$.

Algorithm $\mathcal{B}$ simulates the challenger for $\mathcal{A}$ in the following game;

Setup: Given the public parameters pp and the DDH tuple $< g, g^\alpha, g^\beta, C >$, the algorithm works as follows:

The simulator $\mathcal{B}$ gives the public parameters pp to an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ specifies a set $C^* = i_1, ..., i_{\theta-1}$ of $\theta - 1$ key share holders to the simulator $\mathcal{B}$. The simulator $\mathcal{B}$ samples the corresponding key shares $k_{i1}, ..., k_{i\theta-1}$ uniformly from $\mathbb{Z}_p^*$ and returns to the adversary.

H-Query: at any time the adversary $\mathcal{A}$ can query the random oracle H. To respond these queries, the simulator $\mathcal{B}$ maintains the $H-list$ containing tuples $< x_j, r_j, z_j, coin >$ as described below. Note that the $H-list$ is initially empty.

1. If the query x_j already appears in the $H - list$, then the simulator $\mathcal{B}$ responds with the corresponding value of $z_j = H(x_j)$

2. Otherwise, the simulator $\mathcal{B}$ predetermines $k \in [1, m]$.

3. The simulator $\mathcal{B}$ picks a random $r_j \in \mathbb{Z}_p^*$.
   If $j \neq k$, compute $z_j = g^{r_j}$. If $j = k$, compute $z_j = g^{\beta . r_j}$.

4. The simulator $\mathcal{B}$ adds the tuple $< x_j, r_j, z_j, coin >$ to the $H - list$ and responds to the adversary $\mathcal{A}$ by z_j.

Query-1: Given $x_j \in \mathcal{X}$ for $j \in [1, Q_1]$ issued by the adversary $\mathcal{A}$, the simulator $\mathcal{B}$ maintains $rdPRF - list$ containing tuples $< flag, z_j, f_\alpha(z_j) >$ as described below. Note that the $rdPRF - list$ is initially empty.

Step 1: The simulator $\mathcal{B}$ checks the flag; it responds to the query if $flag = 0$ ($flag = 0$ means the query is made in the query phase, otherwise in the Challenge phase). Otherwise, the simulator $\mathcal{B}$ reports a failure and aborts.

Step 2: the simulator $\mathcal{B}$ runs the H-queries algorithm to find the corresponding tuple $< x_j, r_j, z_j, coin >$ in $H - list$. If $coin = 1$, then the $\mathcal{B}$ reports a failure and aborts. Otherwise, the $\mathcal{B}$ performs the Step 3.

Step 3: The $\mathcal{B}$ extracts z_j from the $H - list$ and computes $f_\alpha(z_j) = g^{\alpha_i . r_j}$ where $j \in \{1, ..., q-1\}$ and returns $f_\alpha(z_j)$ to the adversary $\mathcal{A}$.

Step 4: The $\mathcal{B}$ adds tuples $< flag, z_j, f_\alpha(z_j) >$ to the $rdPRF - list$.

Challenge: Given $x^* \in \mathcal{X}$ $\{x_1, ..., x_Q\}$ submitted by the adversary $\mathcal{A}$, this algorithm acts as follows.

Step 1: The adversary $\mathcal{A}$ submits a challenge query $(x_1^*, ..., x_m^*) \in \mathcal{X} \setminus \{x_1, .., x_Q\}$ to (here $Q = Q_1 + Q_2$) the challenger.

Step 2: The simulator $\mathcal{B}$ checks the flag, and then responds to the query if $flag = 1$ ($flag = 0$ means the query is made in the query phase; otherwise in the Challenge phase). Otherwise, the $\mathcal{B}$ reports a failure and aborts.

Step 3: The simulator $\mathcal{B}$ picks a random bit $b \in \{0,1\}$. If $b = 0$, the simulator $\mathcal{B}$ samples $\{< z_i, f_\alpha(z_i) >\}_{i=1}^m$ and randomly chooses $k \in [1,m]$; it then returns to the adversary except that the value of $< z_k, f_\alpha(z_k) >$ is replaced with $< g^\beta, C >$.

If $b = 1$, $\forall i \in \{1, ..., m\}$ the simulator $\mathcal{B}$ responds with $\{< z_i, f_\alpha(z_i) >\}_{i=1}^m$.

Query-2: Given $x_j \in \mathcal{X}$ for $j \in [Q_1 + 1, Q_2]$ issued by the adversary $\mathcal{A}$, the simulator $\mathcal{B}$ acts similar to that of Query-1 .

Guess: The adversary $\mathcal{A}$ outputs a bit b' which is also an output by the simulator $\mathcal{B}$.

As a result, $\mathbf{Adv}^{DDH}[F, \mathcal{B}] = \mathbf{Adv}^{RDPRF}[\Pi_{RDPRF}, \mathcal{A}] = \frac{1}{m}$.

Query Privacy. The query privacy shows that for any $x \in \{0,1\}^*$ and a uniformly random value $r \in_r \mathbb{Z}_p^*$, the distribution of the output of $\mathsf{R}and(x, pp) = z$ (here $z = H(x)^r$) is uniform in $G|\{1\}$. That is, $H(x) \in G|\{1\}$ has the prime order of p. Thus, different values of r results in different group elements as the output of the $\mathsf{R}and$ function.

Theorem 2. *Our randomized distributed PRF (RDPRF) is Active Collusion Resilient, as defined by "RDPRF Active Collusion Security Definition".*

Proof. The proof is conducted via a sequence of games. The game G_0 and G_2 are designed to have the same distribution as REAL and RAND game of "RDPRF Active Collusion Security Definition", respectively. By showing that the distributions of games (Game0 and Game1, Game1 and Game2) are indistinguishable to each other, we can see that the simulator $\mathcal{S}$ meets the requirements of the security definition, therefore completing the proof of the theorem.

Game G_0. This is the real game of the definition. This game is the same as REAL game of "RDPRF Active Collusion Security Definition".

Game G_1. This game is similar to G_0 except here instead of computing the value of y_i's as before, it is selected uniformly at random.

Game G_2. This game is modification of G_1 where we replace y_{i_j}'s by uniformly random value.

To show the indistinguishability between G_0 and G_1, a reduction to DDH assumption can be conducted. Let A be an adversary which can distinguish G_0 and G_1. The view of A has the form

$$View(G_0) = (h_i = H(x_i), z_i = H(x_i)^{r_i}, y_i = H(x_i)^k, (y_{i,i_j} = H(x_i)^{r_i k_{i_j}})_{j=1}^{\theta-1})_{i=1}^{N}$$

$$View(G_1) = (h_i = H(x_i), z_i = H(x_i)^{r_i}, y_i \in_r \mathbb{Z}_p^*, (y_{i,i_j} = H(x_i)^{r_i k_{i_j}})_{j=1}^{\theta-1})_{i=1}^{N}$$

Because H is random oracle and the r_i's are independent uniformly random in $\mathbb{Z}_p^*$, we can rewrite these views in the form of following distributions where D_0 and D_1 represent the view of adversary A in the games G_0 and G_1, respectively. To simplify the proof, we assume x_i's are distinct.

$D_0 = (h_i = g^{\alpha_i}, z_i = g^{\bar{\alpha}_i}, y_i = g^{\delta_i}, (y_{i,i_j} = g^{\bar{\alpha}_i k_{i_j}})_{j=1}^{\theta-1})_{i=1}^{N}$ for α_i, $\bar{\alpha}_i k$, and k_{i_j} uniformly random at $\mathbb{Z}_p^*$ and $\delta_i = \alpha_i k$.

$D_1 = (h_i = g^{\alpha_i}, z_i = g^{\bar{\alpha}_i}, y_i = g^{\delta_i}, (y_{i,i_j} = g^{\bar{\alpha}_i k_{i_j}})_{j=1}^{\theta-1})_{i=1}^{N}$ for α_i, $\bar{\alpha}_i k$, k_{i_j} and δ_i uniformly random at $\mathbb{Z}_p^*$.

In order to show that the mentioned distributions are indistinguishable, we present a reduction from DDH through a hybrid argument. Using A, we build N adversaries $(\mathcal{B}_s)_{s=0}^{N-1}$ against DDH assumption where $\mathcal{B}_s$ distinguishes $hybrid_S$ from $hybrid_{S+1}$.

$$Pr[G_0 = 1] - Pr[G_1 = 1] \leq \sum_{s=0}^{N-1} Adv_{B_s}^{DDH}(\lambda)$$

We define N intermediate hybrid distributions $(hybrid_s)_{s=0}^{N-1}$ between D_0 and D_1

as follows where $hybrid_0 = D_0$ and $hybrid_{N-1} = D_1$;

$$(h_i = g^{\alpha_i}, z_i = g^{\bar{\alpha_i}}, y_i = \begin{cases} \in_r \mathbb{Z}_p^* & i \leq s \\ \alpha_i k & i > s \end{cases}, (y_{i,i_j} = g^{\bar{\alpha_i} k_{i_j}})_{j=1}^{\theta-1})_{i=1}^{N}$$

We need to show each of these hybrids are indistinguishable for all hybrids $hybrid_0$ to $hybrid_{n-1}$. Thus, through the following reduction we show $hybrid_S$ is indistinguishable from $hybrid_{S+1}$, which can be generalised to all mentioned hybrids. $\mathcal{B}_s$ is a given DDH tuple $(g^\alpha, g^\beta, g^\gamma)$, where γ is either $\alpha\beta$ or a uniformly random value at $\mathbb{Z}_p^*$.

Let $g^k = g^\beta$. $\mathcal{B}_s$ runs A on inputs;

$$\Big(h_i = \begin{cases} g^{\alpha_i} & i \leq s \\ g^{\alpha} & i = s+1 \\ \in_r \mathbb{Z}_p^* & i > s+1 \end{cases}, z_i = g^{\bar{\alpha_i}}, y_i = \begin{cases} (g^k)^{\alpha_i} & i \leq s \\ g^{\gamma} & i = s+1 \\ \in_r \mathbb{Z}_p^* & i > s+1 \end{cases}, (y_{i,i_j} = g^{\bar{\alpha_i} k_{i_j}})_{j=1}^{\theta-1}\Big)_{i=1}^{N}$$

We remark this distribution represent $hybrid_{S+1}$ if $\gamma = \alpha\beta$, and $hybrid_S$ if γ is uniformly random. Thus, the advantage of $\mathcal{B}_s$ against DDH is equal to the advantage of A in distinguishing $hybrid_S$ from $hybrid_S + 1$. As a result,

$$\sum_{s=0}^{N-1} Adv_{B_s}^{DDH}(\lambda) \geq Adv_A(hybrid_n, hybrid_0) = Adv_A(G_0, G_1)$$

Similar approach can be used to show the indistinguishably between G_1 and G_2. We omit the details here.

3.7 Multi-client symmetric searchable encryption

In this section, we present our multi-client symmetric searchable encryption inspired by OXT [3]. The proposed extension supports EDB update as well as key revocation while preserving the full functionality of OXT.

3.7.1 Construction

Our multi-client SSE construction consists of five main algorithms $\Pi = (EDBSetup_{mu}, KeySharing_{mu}, TokenGen_{mu}, Search_{mu}, Retrieve_{mu})$, and two supporting algorithms $Update_{mu}$ and $Revoke_{mu}$.

In the proposed construction, the $EDBSetup_{mu}$ algorithm as defined in Algorithm 14 is similar to the one in OXT [3] except the utilized PRFs which are randomizable distributed key-homomorphic PRFs (refer to Section 3.6). In order to enable several clients to access the same database, the data owner runs the $KeySharing_{mu}$ algorithm (defined in Algorithm 8) which takes the utilized PRF keys as inputs and outputs the corresponding key shares by performing the Shamir's scheme [28] explained in subsection 2.1.1. The generated key shares are then distributed to the desired clients along with the list of legitimate key share holders' identity. Let D to be a data owner who outsources an encrypted database EDB to a remote server S such that S cannot learn anything more than what is predicted as the leakage profile. Moreover, authorized clients (key Share holders) such as $i_1, i_2, ..., i_N$ are allowed to carry out the search through EDB.

Algorithm 7 EDB Setup

Input: Security parameter λ, Database DB
Output: Encrypted database EDB, *Master key K and public parameters pp*

function EDBSetup(λ, DB)
  Initialize $\mathbf{T} \leftarrow \emptyset$ indexed by keywords W.
  Runs $RDPRFSetup$ for RDPRF F which outputs key K_S and pp.
  Runs $RDPRFSetup$ for RDPRF F_p with range $\mathbb{Z}_p^*$ for each of
  K_X, K_I, K_Z keys separately.
  EDB$\leftarrow$ {}
  for $w \in$ W **do**
    Initialize $\mathbf{t} \leftarrow \{\}$; and let $K_e \leftarrow F(K_S, w)$.
    for id $\in$ DB(w) **do**
      Set a counter $c \leftarrow 1$
      Compute xid $\leftarrow F_p(K_I, \mathsf{id})$, $z \leftarrow F_p(K_Z, w||c)$; $y \leftarrow \mathsf{xid} z^{-1}$, $e \leftarrow \mathsf{Enc}(K_e, \mathsf{id})$.
      Set xtag $\leftarrow g^{F_p(K_X, w) \cdot \mathsf{xid}}$ and XSet $\leftarrow$ XSet $\cup$ {xtag}
      Append (y, e) to $\mathbf{t}$ and $c \leftarrow c + 1$.
    end for
    $\mathbf{T}[w] \leftarrow \mathbf{t}$
  end for
  Set (TSet, K_T) $\leftarrow$ TSet.Setup($\mathbf{T}$) and let EDB(1) = (TSet, XSet).
  return EDB = (EDB(1), XSet), $K = (K_S, K_X, K_I, K_Z, K_T), pp$
end function

Algorithm 8 Key Sharing

Input: Public parameter pp, Master key $K = (K_S, K_X, K_I, K_Z, K_T)$, Number of users N, Threshold θ
Output: List of all key shares $\vec{Sh}$.
function KeySharing($(pp, (K, N))$)
$\vec{Sh} \leftarrow \{\}$
for $i = 1, 2, ..., N$ Data owner **do**
Run $K_{S_i} \leftarrow Share(K_S)$
Run $K_{X_i} \leftarrow Share(K_X)$
Run $K_{I_i} \leftarrow Share(K_I)$
Run $K_{Z_i} \leftarrow Share(K_Z)$
Run $K_{T_i} \leftarrow Share(K_T)$
Set $K_i \leftarrow (K_{S_i}, K_{X_i}, K_{I_i}, K_{Z_i}, K_{T_i})$
$\vec{Sh} \leftarrow K_i \cup \vec{Sh}$
end for
return $\vec{Sh} = (K_{S_i}, K_{X_i}, K_{I_i}, K_{Z_i}, K_{T_i})_{i=1}^{N}$
end function

To perform a search over EDB, a client needs to run the $TokenGen_{mu}$ protocol as defined in Algorithm 9. Given the query q, this algorithm first performs *PRF Evaluation Protocol* and then generates the search tokens, Tok_q . One of the main contributions of this scheme is hiding the searched keywords from other key share holders when the search token is generated collaboratively. This is achieved by defining another primitive called randomizable distributed key-homomorphic PRFs (refer to section 3.6).

The $Search_{mu}$ protocol is similar to the $Search_{OXT}$ as defined in [3]. Once the client sent the search token Tok_q to the server, the server performs the search over EDB and outputs the encrypted result $ERes$. Finally, the $Retrieve_{mu}$ protocol as described in Algorithm 10 should be performed in order to extract the identifiers of the documents containing the searched keyword.

3.7.2 Update, Revocation and Enrollment

In this subsection, we are going to discuss about the extensions of our work as described below.

Algorithm 9 TokenGen Protocol

Input:
For client i_1: Search query $\mathbf{q} = (w_1 \wedge \cdots \wedge w_n)$ and key share K_{i_1}
For Helping users i_j: key shares K_{i_j} for $j = 2, ..., \theta$
For Server: Encrypted database EDB.
Output: Search token $Tok_{\mathbf{q}}$.
function $\mathsf{TokenGen}((\bar{w} = (w_1, \ldots, w_n),\ \mathsf{EDB}))$
Client's input is $(K_S, K_X, K_I, K_Z, K_T)$ and $\bar{w}$.
performs PRF Evaluation Protocol to computes
$\mathsf{stag} \leftarrow \mathsf{TSet.GetTag}(K_T, w_1)$.
Client sends stag to the server.
for $c = 1, 2, \ldots$ until the server stops client **do**
for $i = 2, \ldots, m$ **do**
performs PRF Evaluation Protocol on inputs $(i_1, ..., i_\theta)$ *and* $(K_{i_1}, ..., K_{i_\theta})$
to compute $X = F_p(K_Z, w_1||c)$ *and* $Y_i = F_p(K_X, w_i)$
$\mathsf{xtoken}[c, i] \leftarrow g^{X \cdot Y_i}$
end for
$\mathsf{xtoken}[c] \leftarrow (\mathsf{xtoken}[c, 2], \ldots, \mathsf{xtoken}[c, m])$
end for
$Tok_{\mathbf{q}} \leftarrow (\mathrm{stag}, \mathsf{xtoken})$
return $Tok_{\mathbf{q}}$
end function

Algorithm 10 Client Search Result Retrieval Protocol

Input:
For client i_1: Encrypted result $ERes$ as the output of Search algorithm and sterm w_1 and key share K_{i_1}
For Helping users i_j: key shares K_{i_j} for $j = 2, ..., \theta$
For Server: Encrypted database EDB.
Output: Result id.
function $\mathsf{Retrieval}((e, id))$
$id \leftarrow \{\}$
Client performs PRF Evaluation Protocol for
$k_e \leftarrow PRF(k_s, w_1)$
for $i = 1, 2, \ldots$ until the server stops **do**
if $e_i \in R$ **then**
compute $id_i \leftarrow Dec(k_e, e_i)$
$id \leftarrow id \cup id_i$
end if
end for
return id_i
end function

3.7.2.1 Update

Lets consider the condition that the master key is leaked, the data owner should update the encrypted database using a new key to prevent any extraction of the information by an unauthorized entity. In our approach, the data owner sends a keying material to the server for updating the encrypted database. It is assumed that the server does the update honestly and removes the keying material after finishing the update. The data owner runs *Update* algorithm as described below.

$Update(\mathsf{EDB}, \Delta_E)$: The data owner initiates this protocol by choosing two new keys K'_S, K'_T for updating TSet and generating a keying material Δ_E for the update of XSet. Then, the data owner updates TSet and delivers TSet' to the server and shares of K'_S, K'_T to the clients.

The server takes the encrypted database, EDB, and the keying material Δ_E as inputs for the update of XSet. To update XSet for each $\mathsf{xtag}_i \in \mathsf{XSet}$ (for $i \in [1, |\mathsf{XSet}|]$), this algorithm computes $\mathsf{xtag}'_i = G((g^{F_p(K_X, w)\cdot \mathsf{xid}})^{\Delta_E})$ where G is a new hash function. It is worth to note that, using this approach the data owner is able to update the encrypted database one time only. We leave the multiple updates as an open problem for future research.

Although it is possible that the server updates the TSet, it causes more leakage to the server. Since the size of TSet is much smaller than XSet, the better solution might be to perform the update of the TSet by the data owner and delivers it to the server. This solution might add a little overhead but it has less leakage to the server. It is worth to note that, the EDB update happens rarely and this amount of overhead is therefore reasonable. Moreover, in comparison with traditional techniques such as re-encryption by the data owner and re-uploading it to the server, our solution is much more efficient.

Algorithm 11 Update Algorithm

Input:
Data owner: $TSet$ and new keys K'_S, K'_T
Server: Encrypted database EDB and Keying material Δ_E
Output: Updated encrypted database EDB'
function UPDATE(EDB, Δ_E)
TSet' $\leftarrow \{\}$
XSet' $\leftarrow \{\}$
for $w \in \mathsf{W}$ **do**
Initialize $\mathbf{t}' \leftarrow \{\}$; and let $K'_e \leftarrow F(K'_S, w)$.
for $\mathsf{id} \in \mathsf{DB}(w)$ **do**
Data owner computes $y' \leftarrow y\Delta_E$, $e' \leftarrow \mathsf{Enc}(K'_e, \mathsf{id})$.
Append (y', e') to $\mathbf{t}'$
end for
$\mathbf{T}'[w] \leftarrow \mathbf{t}'$
end for
Set $(TSet', K'_T) \leftarrow \mathsf{TSet.Setup}(\mathbf{T}')$
Data owner delivers $TSet'$ to the server and shares of K'_S, K'_T to the clients
for $i = 1, ..., |XSet|$ **do**
if $\mathsf{xtag}_i \in \mathsf{XSet}$ for all i **then**
$xtag'_i \leftarrow G((g^{F_p(K_X, w)\cdot \mathsf{xid}})^{\Delta_E})$
$XSet' \leftarrow XSet' \cup xtag'_i$
end if
end for
Set $\mathsf{EDB}' = (\mathsf{TSet}', \mathsf{XSet}')$.
return EDB'
end function

3.7.2.2 Revocation

Some key share holders might have to be deleted from the system due to being corrupted or other reasons. In this subsection, we introduce two approaches for user revocation.

Approach 1. In our scheme, it is possible to revoke up to $\theta - 1$ key share holders (in each time period between two key share refreshing) simply using the revocation algorithm as shown in Algorithm 12. Therefore, the new key shares of the same master key must be generated by the data owner and become known to all other non-revoked key share holders. Note that, a coalition of all the $\theta - 1$ revoked users does not have any information about the master key. This property follows immediately from the security of Shamir's secret sharing scheme. Moreover, if the revoked clients from different time intervals form a coalition to rebuild the master key, even if the coalition has more than θ members, they will not succeed (refer to security

definitions). Its communication overhead is $O(N-(number\ of\ revoked\ users))$ and might be very large in some scenarios. It is possible to minimise this overhead by just broadcasting the updated list of legitimate clients without delivering the new key shares to them. However, giving new key shares to the non-revoked users can minimise the risks associated with key loss and collusion of revoked clients. Moreover, it can guarantee the freshness of the key shares. We may also use Approach 2 which is more efficient in terms of communication overhead during the revocation process.

Algorithm 12 Revocation Algorithm

Input: key share holders list $C = i_1, i_2, ..., i_N$, Public parameters pp, Master key K, and Identifier of the client to be revoked i_R
Output: C' and $\vec{Sh'}$
function REVOKE(k_R, i_R)
$\vec{Sh'} \leftarrow \{\}$
C' $\leftarrow \{\}$
Data owner runs key sharing $(pp, (K, N))$
$\vec{Sh'} \leftarrow (k'_1, ..., k'_N)$
Data owner updates key share holders list
$C' \leftarrow C \setminus i_R$
$j \leftarrow 1$
for $j = 1, ..., N$ **do**
while $i_j \in C'$ **do**
Data owner sends k'_j and C' to i_j
$j \leftarrow j + 1$
end while
end for
return C', $\vec{Sh'}$
end function

Approach 2. This approach adapts the revocation scheme proposed by Naor et al. [77] to our multi-client SSE scheme which applies the idea of doing Shamir's secret sharing in the exponents. In order to support the assumptions considered in Naors' scheme [77], we need to slightly modify our scheme as follows. Lets consider that the Decisional Diffie-Hellman assumption holds for a group $\mathbb{Z}_q$ of prime order q and generator g. Here, $\mathbb{Z}_q$ is a subgroup of $\mathbb{Z}_p^*$ where p is prime and $q|p-1$. Let $C_i \in \mathcal{F}$ be an arbitrary identifier of the client i where $\mathcal{F}$ is a field. In this approach, the data owner has to define the master key k_0, to be used after the revocation, in EDBSetup phase. Here, $k_0 = P(0)$ where P is a random polynomial of the degree $\theta - 1$ generated by the data owner. Each user would receive a pair $< C_i, k_{C_i} >$ (here $k_{C_i} = P(C_i)$) by the end of $KeySharing_{mu}$ algorithm. Note that, the users'

identifiers, p, and q are publicly known.

In order to revoke at most $\theta - 1$ clients, the data owner picks a random $r \in_r \mathbb{Z}_q$ and distributes g^r along with the updated list of legitimate clients. Then, each non-revoked key share holder such as C_i, updates its share to $g^{rk_{C_i}}$. The equation, $g^{rk_0} = g^{r\Sigma_{i=0}^{\theta}\Lambda_i k_{C_i}}$, shows how the key can be reconstructed using θ shares. Here, the Λ_i's are Lagrange coefficients that depend only on C_i's. However, in our protocol the clients do not need to reconstruct the key. Thus, the data owner does not require to reveal the shares of the revoked clients. Note that, in our scheme the clients just need to perform PRF evaluation collaboratively.

Remark. When the number of clients is huge and the size of the database is relatively small, Approach 1 is not suitable as it requires peer to peer interaction between the data owner and the non-revoked clients. Thus, the communication overhead might be very large. In this scenario, Approach 2 can be used. It does not require any interactions between the data owner and the clients, as it uses a broadcast message. In addition, Approach 2 is fast and efficient to perform batch revocation. Approach 1 is more applicable if a single user revocation or revocation of a few clients is required in a scenario where the database is very large and the number of clients is small. In this scenario, using Approach 2 can cause additional computational overhead as it requires EDB update for each revocation.

3.7.2.3 Enrolment

The data owner can add new clients even if they join the group after the key sharing phase. User enrolment can be done just by providing the pair $<$ *identity*, *key share* $>$ to the corresponding key share holders and updating the clients' list. Note that the number of clients is limited to the considered N at the beginning of the protocol.

3.8 Security analysis

In this section, we state the security of our multi-client SSE protocol against a dishonest server and the dishonest/compromised key share holders, respectively.

Theorem 3. *Let $\mathcal{L}$ be the leakage function (as defined in Section 3.5.1). Then, our multi-client SSE protocol is $\mathcal{L}$-semantically-secure against adaptive server (Definition 8), if the OXT [3] is secure and the DDH assumption holds.*

Proof. Let $\mathcal{A}$ be a dishonest server who performs an adaptive attack against our multi-client SSE protocol. Then we can construct an algorithm $\mathcal{B}$ that breaks the server privacy of OXT protocol [3] by running $\mathcal{A}$ as a subroutine with non-negligible probability.

- Algorithm $\mathcal{B}$ passes the selected DB by $\mathcal{A}$ to the OXT challenger.
- The OXT challenger runs $(\mathsf{K},\mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB})$ and returns EDB to the algorithm $\mathcal{B}$. Then, the algorithm $\mathcal{B}$ sets $\mathsf{EDB}_{mu} = \mathsf{EDB}$.
- The algorithm $\mathcal{B}$ sends EDB_{mu} along with the public parameters pp to an adversary $\mathcal{A}$.
- The adversary $\mathcal{A}$ specifies a set of key share holders $C^* = i_1, ..., i_{\theta-1}$.
- The algorithm $\mathcal{B}$ responds with the corresponding key shares $k_{i_1}, ..., k_{i_\theta-1}$ at random.
- For each $q[i]$ query issued by the adversary $\mathcal{A}$, the algorithm $\mathcal{B}$ defines $TokenGen_{mu}(K, q[i])$ which outputs the output of the $TokenGen$ oracle of OXT.
- Finally, the adversary $\mathcal{A}$ outputs a bit that the algorithm $\mathcal{B}$ returns.

Since in our multi-client SSE protocol, the *Setup* is the same as the Real world of OXT protocol [3], it simulates the Real world application of our protocol. Thus, the only thing that we need to show is the condition where a set of key shares of size $\theta - 1$ (θ indicates the threshold) are corrupted. Therefore, we refer to the property of the utilized secret sharing scheme in [28]; that is, any number of key shares less than the considered threshold is uniform and independent of the secret.

Simulator. By considering $\mathcal{A}$ as a dishonest server against our multi-client SSE protocol, Π_{mu}, we construct an algorithm $\mathcal{B}$ that breaks the server privacy of Π_{OXT} protocol [3] by running $\mathcal{A}$. Let $\mathcal{S}_{OXT}$ be the simulator for OXT; then we construct a simulator $\mathcal{S}_{mu}$ for our multi-client SSE. The algorithm $\mathcal{B}$ uses $\mathcal{S}_{OXT}$ to construct the simulator $\mathcal{S}_{mu}$ in order to answer the queries issued by $\mathcal{A}$. Since the Real of our multi-client SSE is the same as Real of OXT for the Ideal case, we just need to use the simulator of OXT for $\mathcal{A}_{OXT}$, to construct the simulator of our multi-client SSE for $\mathcal{A}_{mu}$. By running $\mathcal{S}_{OXT}$ for $EDBSetup$ and $TokenGen$ queries, we can construct a simulator $\mathcal{S}_{mu}$ for $EDBSetup$ and $TokenGen$ queries of our multi-client SSE.

$Pr(Real_{\mathcal{A}}^{\Pi_{\text{mu}}} = 1) - Pr(Ideal_{\mathcal{A},\mathcal{S}_{mu}}^{\Pi_{\text{mu}}} = 1) \leq neg\ Pr(Real_{\mathcal{B}}^{\Pi_{\text{OXT}}} = 1) - Pr(Ideal_{\mathcal{B},\mathcal{S}_{OXT}}^{\Pi_{\text{OXT}}} = 1) \leq neg$

As a result, $\mathcal{A}$'s view is the same for both simulators; $\mathcal{S}_{OXT}$ and $\mathcal{S}_{mu}$.

Theorem 4. *Our multi-client SSE protocol Π_{mu} is query-private, as defined in Definition 10.*

Proof. Let $\mathcal{A}$ be a collusion of $\theta - 1$ key share holders who perform a query privacy attack against our multi-client SSE protocol. In our query privacy attack game, the view of the adversary $\mathcal{A}$ is the same as view of key share holders $i_1, ..., i_{\theta-1}$ in TokenGen protocol. The only value which is potentially related to the keyword x_b is z which is $Rand(x_b, r, pp)$ using a randomizer r. By "Query privacy of RDPRF", z is uniform in $\mathcal{X}$ with respect to the choice of $r \in_r \mathcal{R}$, independent of x_b. Thus, the view of the adversary $\mathcal{A}$ is independent of b. As a result, $Adv(\mathcal{A}) = 0$.

Theorem 5. *Our multi-client SSE protocol* Π_{mu} *is active collusion resilient, as defined in Definition 11.*

Proof. The proof is conducted via a sequence of Games. In all games, the adversary supplies a database DB and the keyword queries w_i at the beginning of the game. The first game Game 0 is designed to have the same distribution as the real game of our multi-client SSE protocol, $Real_{\mathcal{A}}^{\Pi_{\text{mu}}}$. Two other games Game 1 and Game 2 are designed to be easily simulated by an efficient simulator $\mathcal{S}$. By showing that the distributions of games (Game 0 and Game 1, Game 1 and Game 2) are indistinguishable to each other, we can see that the simulator $\mathcal{S}$ meets the requirements of the security definition, therefore completing the proof of the theorem.

Game 0. This game is the same as the real game of our multi-client SSE protocol, $Real_{\mathcal{A}}^{\Pi_{\text{mu}}}$.

Game 1. This game is the same as Game 0, except that here we replace $z_i = \mathsf{R}and(w_i, r)$ with a uniformly random value drawn from $\mathbb{Z}_p^*$.

Game 2. This game is almost identical to Game 1, except here we change y_{i_c} from $\mathcal{F}(k_{i_c}, z_i)$ to a uniformly random value on the range of $\mathcal{F}$. Similarly, we change $\mathcal{F}(k, z_i)$ to a uniformly random value over the range of $\mathcal{F}$.

Game 0 and Game 1 are indistinguishable under partial evaluation queries due to the randomness property of RDPRF. Whereas, Game 1 and Game 2 are indistinguishable for either partial evaluation queries or full queries under the RDPRF security definition.

Note that here for the sake of simplicity we consider only a single keyword query; it can be easily extended to multi keywords queries.

Given this information and controlling the behaviour of helping users, an adversary $\mathcal{A}$ at most would learn the partial evaluation of the chosen keyword with respect to the initiator key share, k_{i_c}. However, based on the "Security definition of RDPRF", $\mathcal{A}$ is unable to extract k_{i_c}.

Theorem 6. *Suppose the hash function G is a random oracle, then our multi-client SSE protocol Π_{mu} is secure after Update as defined in Definition 12.*

Proof. The proof is similar to the proof of Theorem2 with the following differences: 1) the adversary is an outsider not a dishonest server and it is assumed that the server performs the update honestly; 2) the old master key is given to the adversary instead of a set of key shares of the corrupted clients; 3) an snapshot of the updated encrypted database (EDB') is given to the adversary instead of the encrypted database; 4) the adversary is not allowed to communicate with the clients; and 5) the simulator simulates the Search and Retrieve algorithms instead of TokenGen.

Let $\mathcal{A}_{mu}$ be an adversary against Π_{mu} and $\mathcal{A}_{up}$ be an adversary against Π_{mu} after update. Let $\mathcal{S}_{mu}$ be the simulator for Π_{mu} and $\mathcal{S}_{up}$ be the simulator for Π_{mu} with update. We can transform $\mathcal{A}_{up}$ to $\mathcal{A}_{mu}$. Moreover, $\mathcal{S}_{up}$ can act the same as $\mathcal{S}_{mu}$ for simulation of TSet' by simulating y' at random. Thus, $\mathcal{A}_{up}$ would not be able to distinguish them from a random value.

For XSet', $\mathcal{S}_{up}$ simulates XSet' by the random elements. For queries to the random oracle G, as long as the queries are not made at the $BP = (g^{F_p(K_X,w)\cdot \mathsf{xid}})$ points, it returns random elements. In fact, once the adversary query the random oracle at one of those points, then $\mathcal{A}_{up}$ can compute Δ_E uniquely given w, xid, and K_X. Thus, the chance of this bad event happening is going to be at most;

$$Pr[\mathcal{A}_{up}\ queries\ at\ BP] \leq \frac{|XSet'| \times No.\ of\ queries\ on\ G}{q}$$

Here, q is the size of the considered group.

In fact, the simulator $\mathcal{S}_{up}$ is unable to answer correctly if $\mathcal{A}_{up}$ makes a query at those points. $\mathcal{A}_{up}$ has no information about Δ_E (Δ_E is simulated uniformly and independently and it is hidden from the adversary). As long as the bad event does not happen, the simulation of XSet' with random elements will be indistinguishable to $\mathcal{A}_{up}$ from the real one. Although the value of Δ_E is used in generation y' of TSet', it is encrypted under a new key. Thus, for TSet' the simulator works in the same

way.

Theorem 7. *Our multi-client SSE protocol* Π_{mu} *is secure after Revocation-Approach 1 as defined in Definition 13.*

Proof. The proof is the same as proof of Theorem1, except that here instead of simulated $\theta - 1$ key shares at random in Theorem1, for each time period $\theta - 1$ key shares must be simulated. More precisely, in each time period independent randomness is used for the coefficients of the polynomial which in turn results in $\theta - 1$ uniformly random and independent key shares from key shares generated in previous time periods.

Theorem 8. *Our multi-client SSE protocol* Π_{mu} *is secure after Revocation-Approach 2 as defined in Definition 14.*

Proof. For Approach 2, a coalition of revoked clients (maximum of $\theta-1$) cannot distinguish the new key from a random value. By considering $\mathcal{A}$ as a coalition of revoked clients, $i_1, ..., i_{\theta-1}$, against our multi-client SSE protocol, Π_{mu}, we construct an algorithm $\mathcal{B}$ that breaks the DDH assumption by running $\mathcal{A}$.

Setup: Given the public parameters pp and the DDH tuple $< g, g^a, g^b, C >$, the algorithm works as follows:

The simulator $\mathcal{B}$ gives the public parameters pp to an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ specifies a set $C^* = i_1, ..., i_{\theta-1}$ of $\theta - 1$ key share holders to the simulator $\mathcal{B}$. The simulator $\mathcal{B}$ samples the corresponding key shares $k_{i1}, ..., k_{i\theta-1}$ uniformly from $\mathbb{F}_q$ and returns to the adversary.

Revocation Query: at any time the adversary is allowed to issue a revocation query R_j. That is, for the revocation query of a user i_j for $j \in [1, \theta - 1]$ is issued by the adversary. The simulator $\mathcal{B}$ picks a random value $r_j \in_r \mathbb{Z}_q$ and maintains a $Revoke - list$ containing tuples $< R_j, i_j, r_j, g^{r_j a}, g^{r_j b} >$. It then returns $g^{r_j a}$ as the broadcast message for the user revocation. Note that the $Revoke - list$ is initially empty.

TokenGen Query: $\mathcal{A}$ sends TokenGen queries for some keywords $w_1, ..., w_Q \in \mathcal{W}$ to $\mathcal{B}$, and for each query (R_j, w_j), the simulator $\mathcal{B}$ refers to $Revoke - list$ and responds with $Tok_j = f(g^{r_j b}, w_j)$.

Challenge: $\mathcal{A}$ submits a pair of fresh keywords (w_0^*, w_1^*) where $w_0^*, w_1^* \notin \mathcal{W}$. In order to respond, $\mathcal{B}$ chooses a random bit $b \in_r \{0, 1\}$ If $b = 0$, $\mathcal{B}$ samples Tok_0 at random. Otherwise, $\mathcal{B}$ returns C to $\mathcal{A}$.

Guess: The adversary $\mathcal{A}$ outputs a bit which is also an output by the simulator $\mathcal{B}$.

As a result, $\mathcal{B}$'s success probability in breaking DDH assumption is the same as $\mathcal{A}$'s probability of breaking our scheme after revocation using the Approach 2.

3.9 Security, Functionality and Performance Comparison

Since our multi-client protocol, MC-OXT [53] and the proposed protocol by Sun et al. [49] are under the framework of OXT [3], in this section we compare our protocol with theirs in terms of communication and computation overhead. The communication overhead between the data owner and the server during EDBSetup phase is the same in all of them as well as the communication between the client and the server. However, in our multi-client SSE client who wants to search over EDB requires to communicate with at least $\theta - 1$ (θ is the threshold) key share holders in order to generate the search token. Due to the use of ABE, the Sun's protocol has storage overhead and some computational costs to the data owner. Moreover, the data owner should compute an extra exponentiation for the PRF calculation during the setup phase, totally introducing $(2\sum_{w \in \mathrm{W}} |\mathsf{DB}[w]| + |\mathrm{W}|)$ exponentiation operations for the whole database. For a conjunctive query, e.g., $Q = (w_1 \wedge w_2 \wedge \cdots \wedge w_m)$ performed by a client, we assume that the associated keywords belong to the client's authorized keyword set $\mathbf{w}$, i.e., $w_i \in \mathbf{w}$ for $i \in [m]$. Table 3.3 summarizes the computation and

Table 3.3: Computational and communication cost between client and server

Conjunctive query $Q = (w_1 \wedge w_2 \wedge \cdots \wedge w_m)$, where $w_i \in \mathbf{w}$			
Reference	Data owner's computation cost	Clients' computation cost	Communication cost between client and server
OXT [3]	$\lvert\mathsf{DB}[w_1]\rvert(m-1)\cdot\text{exp}$	N/A	$l(1+(m-1).\lvert DB(w_1)\rvert)$
RSA-OXT [49]	$3\cdot\text{exp}$	$(\lvert\mathsf{DB}[w_1]\rvert(m-1)+(m+1))\cdot\text{exp}$	$l(1+(m-1).\lvert DB(w_1)\rvert)+3\,l_{RSA}$
TPPKS [52]	N/A	$(m+1)2p$	$(m+1)\lvert G_0\rvert$
MC-OXT [53]	$(m-1)\cdot\text{exp}$	$(\lvert\mathsf{DB}[w_1]\rvert(m-1))\cdot\text{exp}$	$l(1+(m-1).\lvert DB(w_1)\rvert)+2\mathcal{T}+n\lvert\mathbb{Z}_p^*\rvert$
Our multi-client SSE	N/A	$\lvert\mathsf{DB}[w_1]\rvert m\cdot\text{exp}+m\cdot inv$	$l(m.(1+\lvert DB(w_1)\rvert))$

exp: the exponentiation operation on the group; $\lvert\cdot\rvert$: the size of a finite set or group, e.g., $\lvert\mathbb{G}\rvert$; $\mathbf{w}$: the authorized keyword set for a client; l: the length of Xtoken; p: the pairing operation; inv: the inversion operation; n: number of blinding factors; $\mathcal{T}$:size of the output of the PRF F_T.

communication of our multi-client SSE protocol in comparison with some relevant related works.

To perform the search as described above, the client in our multi-client SSE does not require to refer to the data owner. However, it has to interact with key share holders to generate the search token where the user needs to compute $((m-1)+|DB(w_1)|)(2exp+1inv)$. Note that two exponentiations and an inversion here are required for *Rand* and *UnRand* functions in the PRF Evaluation protocol. In contrast, the user in OXT [3] requires per-query interaction with the data owner to get the search token. Thus, the data owner needs to compute $((m-1).|\mathsf{DB}(w_1)|)$ exponentiations. In Sun's protocol, the client needs to get the secret information from the data owner at the beginning, where the data owner needs to compute 3 exponentiations and generates an attribute-related secret key for each client. Then the client is able to perform the searches at the cost of $(m+1)$ additional exponentiations to the generation of xtoken. In order to generate a search token in the proposed protocol by Wang et al. [52], each user must compute a share of the search token using its key share and communicates with at least $\theta-1$ other collaborating users. By comparing the outputs of two pairings, these users can verify each others' token shares. Afterwards, they combine the token shares to create the search token.

Since our multi-client SSE is an enhanced version of OXT [3], we can estimate the extra cost of computation and communication over OXT in the mentioned scenario

(by assuming $1inv \leq 1exp$) as followed;

$$
\begin{aligned}
&Overhead\ ratio\ for\ computation \\
&= 1 + \frac{(2exp + 1inv)((m-1) + |\mathsf{DB}(w_1)|)}{(1exp)((m-1).|\mathsf{DB}(w_1)|)} \\
&\leq 1 + 3(\frac{((m-1) + |\mathsf{DB}(w_1)|)}{((m-1).|\mathsf{DB}(w_1)|)}) \\
&\leq 1 + 3(\frac{1}{(m-1)} + \frac{1}{|\mathsf{DB}(w_1)|})
\end{aligned}
$$

And the communication overhead for the client is;

$$
\begin{aligned}
&Overhead\ ratio\ for\ communication \\
&= 1 + \frac{l((m-1)|\mathsf{DB}(w_1)|)}{l(1 + (m-1).|\mathsf{DB}(w_1)|)} \\
&= 1 + (\frac{1}{(m-1)} + \frac{1}{|\mathsf{DB}(w_1)|})
\end{aligned}
$$

Here, l indicates the length of Xtoken.

There is no computational overhead for revocation using Approach 1. However, its communication overhead is linear to the number of non-revoked clients. As mentioned earlier, it is possible to minimise this overhead by just broadcasting the updated list of legitimate clients without delivering the new key shares to them. The communication overhead associated to the user revocation using Approach 2 is relatively small as the revocation message just contains one group element, g^r. Since the clients do not require to reconstruct the new key in our scheme, each user just performs one exponentiation to switch to the share of the new key. That is, in our scheme, the users just need to compute the output of PRF evaluation of the search keyword collaboratively.

The EDB is critical and the trivial approach of re-uploading the re-encrypted database has high communication overhead. In our proposed scheme, to update the encrypted database, EDB, two sets must be updated, XSet and TSet. The update algorithm raises each element of these sets to the key material. Thus, we require $|t|$

Table 3.4: Security analysis

Reference	Query Privacy			Key exposure/ Collusion	Revocation	Security	
	Server	Data owner	Helping users			Definition	Assumption
OXT [3]	✓	×	N/A	N/A	×	IND-CKA2	DDH
RSA-OXT [49]	✓	Semi	N/A	×	×	IND-CKA2	DDH and RSA
TPPKS [52]	✓	✓	×	✓	✓	IND1-CKA	DL, DDH and CDH
MC-OXT [53]	✓	×	N/A	✓	×	IND-CKA2	OM-GDH
our multi-client SSE	✓	✓	✓	✓	✓	IND-CKA2	OM-OWF and DDH

IND1-CKA:indistinguishability against chosen keyword attacks (nonadaptive); IND-CKA2:indistinguishability against chosen keyword attacks (adaptive)

exponentiations for the update of TSet and |XSet| exponentiations for the update of XSet. Table 3.4 gives a brief security analysis of our multi-client SSE protocol and a comparison with the other closely related works. In order to prove the security of OXT, Cash et al. [3] generalized IND-CKA2 for conjunctive queries, which is parameterized by the leakage functions. RSA-OXT, MC-OXT and our proposed protocol all follow the utilized security definition by cash et al. [3] (generalized IND-CKA).

To prove the security of TPPKS [52], the authors used ICLR (indistinguishability of ciphertext from limited random string) game introduced by Golle et al. [57]. This game is the extended version of IND1-CKA [16] and guarantees that an adversary cannot recover the contents of a document from its secure index and the indices of other documents.

As shown in Table 3.5, all of the considered schemes support boolean queries. Our protocol, RSA-OXT and MC-OXT, are designed based on the OXT protocol proposed by cash et al. [3] in order to support a multi client setting. MC-OXT requires online presence of the data owner to generate the search tokens for the clients whenever it is needed. Although Sun et al. [49] could avoid per-query interaction with the data owner, it requires the engagement of the data owner in token generation process. Similar to our protocol, TPPKS does not require an online presence of the data owner as it is not involved in the Token generation, search and even in the decryption phase.

Table 3.5: Functionality analysis

Reference	Query Type	Setting	Online presence of Data owner	Notes
OXT [3]	Boolean	S/S	✓	Highly scalable implementation
RSA-OXT [49]	Boolean	S/M	Initialization phase	Attribute-based encryption
TPPKS [52]	Boolean	S/M	Not required	Shares verification
MC-OXT [53]	Boolean	S/M	✓	homomorphic signature by data owner on search tokens
our multi-client SSE	Boolean	S/M	Not required	Hides search keywords

S/S: Single writer / Single Reader; S/M: Single writer / Multi Reader

3.10 Summary

A multi-client symmetric searchable encryption scheme is proposed. The proposed construction is query private against all entities involved including the data owner, server, and key share holders. In order to make the search keyword hidden from the other share holders during the token generation process, we defined a new distributed key homomorphic PRF. In addition, we gave a concrete construction of our randomizable key homomorphic distributed PRF. We designed an update algorithm which enables the data owner to update the encrypted database efficiently. We presented two approaches for user revocation considering different scenarios. Finally, the security, functionality and efficiency of our multi-client SSE have been analysed from different aspects.

Chapter 4

Multi-keyword ranked symmetric searchable encryption

This section presents the details of the second contribution of this research, multi-keyword ranked symmetric searchable encryption, which is published in the European Symposium on Research in Computer Security (ESORICS 2019).

4.1 Overview

In contrast to Boolean queries, which rely on appearance of the queried keywords in the database and return the matching documents, the ranked search captures the most relevant documents for a query. In the potentially huge result space, ranked search systems minimize the post processing of data for end-users. Moreover, it has a great impact on the system usability and performance when dealing with the massive amounts of data stored in the cloud. Ranked search has been widely studied by Information Retrieval (IR) and database communities. Top-$\mathscr{K}$ query processing techniques such as TA [78] and FA [79] are well-known examples of such

systems. However, such techniques do not preserve the privacy of the data stored on the database. That is, they require direct access to the relevance scores as well as various modes of access to data which makes them inapplicable in the context of encrypted data search.

The first multi-keyword ranked search scheme was proposed by Cao et al. [80], where both documents and queries are represented as vectors of dictionary size. This scheme sorts documents using the score based on Inner Product Similarity (IPS), where a document score is simply the number of matches of queried keywords in each document, which is not accurate [20]. In general, for ranked search the following techniques are proposed in the literature.

- **Fully Homomorphic Encryption (FHE)**: Although FHE [81] supports arbitrary computations over the encrypted data, due to the high performance overheads its not suitable for practical database queries [21].
- **ORAM**: Similar to FHE, ORAM (Oblivious Random-Access Machine) [40] is computationally expensive to be used in practice [21]. Although some works tried to improve ORAM efficiency [41, 82–84], its application in symmetric encryption for execution of top-$\mathscr{K}$ queries is limited [85].
- **OPE**: Order Preserving Encryption (OPE) [86] allows efficient range queries over the encrypted data. However, OPE reveals the relative order of elements in the database, and therefore does not meet the data owner's data privacy.
- **Using two (or more) non-colluding servers**: The authors of [20] justified the assumption of non-colluding servers; these parties are usually supplied by different companies hence have also commercial interests not to collude. This model would be a solution to avoid multiple rounds of user-to-server interactions. However, it requires the server-to-server interactions instead. Moreover, this assumption is less appealing in practice compared to the traditional single-server model [50].

4.1.1 Motivations

Among different methods to support ranked query for searchable encryption, OPE is the most popular one due to its efficiency. However, the leakage associated with OPE makes it vulnerable to several attacks. Naveed et al. [87] presented two attacks against OPE as follows:

Sorting attack: This attack decrypts the OPE-encrypted columns. That is, adversary sorts the ciphertext and the message space, and outputs a function that maps each ciphertext to the element of message space with the same rank.

Cumulative attack: OPE reveals the frequency of the data and its relative order at the same time which helps an adversary to find out what fraction of encrypted data is smaller than each ciphertext. This is known as cumulative attack. This attack recovers plaintext from OPE with high probability using auxiliary information[1].

Durak et al. [88] showed that the above attacks did not take advantage of the additional leakage that is present in OPE constructions. They discussed additional two types of attacks *Inter-column correlation-based attacks* and *Inter+Intra-column correlation-based attack*. The former takes the advantage of correlation between OPE columns where the adversary knows a bounding box for the plaintext. That is, the columns of data in a table are usually correlated because a row of a table usually corresponds to an individual record. The latter attack uses both inter and intra column correlations.

Table 4.1 provides a summary of some related works that support ranked search over an encrypted database. They either used OPE or cryptographic primitives over two servers. The former suffers from serious leakages and the latter from usability (i.e., issues to use in practice). Shen et al. [17] built an OPE on the top of Oblivious Cross Tags (OXT) protocol of [3]. Although their scheme is efficient, it is vulnerable against aforementioned attacks due to the OPE leakage. To avoid reveal-

[1] Auxiliary information are publicly-available information such as application details, public statistics, and prior versions of the database (possibly achieved by a prior data breach)

Table 4.1: Summary of comparison

Protocol	Encryption method	Single/twin server	No OPE leakage
Meng et al. [21]	Encrypted hash list using Paillier	Twin server	?
Shen et al. [17]	OPE	Single server	×
Baldimtsi and Ohrimenko [20]	Paillier	Twin server	?
Jiang et al. [19]	Paillier	Twin server	?
Wang et al. [18]	One-to-many OPE	Single server	×
Our MRSSE	SWHE	Single server	✓

ing the distribution of scores in OPE, Wang et al. [18] proposed one-to-many OPE. Their construction conceals the distribution of scores using a probabilistic encryption. However, Li et al. [89] presented a differential attack over one-to-many OPE which reveals the leakage of distribution by exploiting the difference between ciphertexts. It is assumed that the attacker has some background information which helps him to infer the encrypted keywords using differential attacks. On the other hand, if two servers are located in the same place, this contradicts with the assumption that they do not collude. There would be server-to-server communication overhead if they are located in different places. This cancels out the major advantage of ranked query, minimizing the unnecessary network traffics. Therefore, an effective solution to support ranked query over symmetric searchable encryption schemes is still a challenge. Our proposed approach in this research aims to address this challenge.

4.1.2 Contributions and technique

The key contribution of this work is a generic solution for supporting effective multi-keyword ranked search over an encrypted database. We demonstrate the application of this solution through the proposed Multi-keyword Ranked Searchable Symmetric Encryption scheme (MRSSE). MRSSE is secure against all of the attacks related to OPE without relying on a two server assumption, and hence overcomes the limitations of existing approaches. More precisely, MRSSE uses somewhat homomorphic encryption within our proposed filtering techniques instead of OPE to provide a ranked search. In terms of functionality, MRSSE supports the multi-keyword search over Boolean, ranked and limited range queries without adding any extra leakage. It

reduces the communication overhead when the number of search results is large (we give examples in Section 4.5.2) while the security is guaranteed. The effectiveness of MRSEE is proven via security and efficiency analyses. It is important to note that though our approach is generic, in this scheme we leverage OXT protocol[2] as an example to demonstrate the applicability of the proposed approach.

It is worth to note that when performing rank search, the server learns a set of ciphertexts which are satisfying the ranking condition, this is an inherent leakage in ranked search. Moreover, in the most of the current solutions for ranked-search the relative order of the importance of the document (ranking) is also leaked to the server. However, the proposed solution in this chapter avoids this leakage as the server returns always a fixed size of unsorted results (the actual results are padded with the encryption of 0s) and the client performs sorting locally.

Our technique We used various homomorphic encryption tools and techniques to efficiently filter the search results. We considered using BGV-type homomorphic encryption but it resulted in high depth for equality check on integers (j and P)[3]. Hence, we reduced this depth by using unary encoding (we also have document scores encoded in unary which are small). However, the conditional increment of the pointer "P" involves a multiplication. Therefore, we switched to Ring-GSW homomorphic encryption which allows us to do the repeated multiplications with low noise growth (refer to Section 4.3.1 for details).

4.2 Preliminaries

In this section, we present notations and definitions needed in our construction.

Cryptographic primitives. The utilized cryptographic primitives in this chapter are presented in details in Section 2.1.

[2]For detailed explanations of OXT refer to section 2.3.2

[3]($f_j(.)$) in line 18 and line 19 of Algorithm 13 where j is the counter for candidate list and P is the pointer of the output buffer

Table 4.2: Notations and terminologies

Notation	Description
id_i	the document identifier of the i-th document
W_{id_i}	a list of keywords contained in the i-th document
$\mathsf{DB} = (id_i, \mathrm{W}_{id_i})_{i=1}^{\mathscr{D}}$	a database consisting of a list of document identifier and keyword-set pairs
$\mathsf{DB}[w] = \{id : w \in \mathrm{W}_{id}\}$	the set of identifiers of documents that contain keyword w
$\mathrm{W} = \bigcup_{i=1}^{\mathscr{D}} \mathrm{W}_{id_i}$	the keyword set of the database
EDB	the encrypted database
SEDB	the scored encrypted database
sterm	the least frequent term among queried terms (or keywords) in a search query
xterm	other queried terms in a search query (i.e., the queried terms excluding sterm)
TSet	an array that presents equal sized lists for each keyword in the database
XSet	a lookup table that contains elements computed from each keyword-document pair
P	pointer to the output buffer
j	counter for the candidate list
t	threshold
$\hat{t}$	encrypted threshold
N	size of the output buffer
L	number of search keyword
l	bit length of the score
n	number of search results
z	a bit showing whether the threshold condition holds
s	score
$\hat{s}$	encrypted score
λ	security parameter
n_b	breaking point

Notations. Notations frequently used in this chapter are listed in Table 4.2.

Scoring approach. We use a common method of evaluating a relevance score, called $TF \times IDF$ (term frequency times inverse document frequency) [90]. However, it should not be regarded as the name suggest. It is defined as $Score(w_j, id_i) = \frac{1}{|id_i|}.(ln(1 + \frac{N}{f_{w_j}})).(1 + ln(f_{id_i,w_j}))$. This score consists of two main components; term weight $tw = (ln(1 + \frac{N}{f_{w_j}}))$ and relative term frequency $r_{d,t} = (ln(1 + \frac{N}{f_{w_j}}))$. Here, f_{w_j} is the number of documents that contain the keyword/term w_j, N is total number of documents in the collection, f_{id_i,w_j} is the frequency of w_j in the document id_i and, $|id_i|$ is a normalization factor to discount the contribution of long documents and obtained by counting the number of indexed terms in a document. Note that $Score(w_j, id_i)$ is set to zero if the keyword w_j does not appear in document with identifier id_i. Finally, the similarity measure is the sum of the products of the weights of query terms (Q) and the corresponding document terms [90]: $Score(Q, id_i) = \sum_{w_j \in Q} Score(w_j, id_i)$.

4.3 Our threshold-based filtering approach

We solve the problem of multi-keyword ranked search by introducing threshold-based filtering on the ciphertexts. This generic approach can be applied on symmetric searchable encryption schemes. Assume that the database DB is going to be outsourced to a honest-but-curious cloud server. The first step for the data owner is to generate the scored encrypted database. In this phase the data owner can apply any desired scoring technique (such as $TF \times IDF$). Whenever, the data owner wants to search through the scored encrypted database, he needs to generate a search token and choose a threshold value. Thus, the server would be able to find the matching results, then using homomorphic operations aggregate the scores and filter them according to the threshold and return the most relevant results (with the scores higher or equivalent to the threshold). Since the size of output buffer, N, is independent of total number of search results, n, we can achieve a homomorphic computation and communication independent of n.

4.3.1 Homomorphic operations

We define the following homomorphic operations which are used in the homomorphic search and homomorphic filter presented in Section 4.3.2 and Section 4.3.3, respectively.

- **Component-wise homomorphic Operations:** We represent the encryption of a ℓ-bit integer such as $s = s_{\ell-1}s_{\ell-2}...s_0$ using Ring-GSW (see appendix) as $\hat{s} = Enc(s_{\ell-1})\ Enc(s_{\ell-2})...Enc(s_0)$, where the plaintext space of encryption is $(\mathbb{Z}_2, (+, .))$. We denote addition and multiplication over ciphertext with $\boxplus$ and $\boxdot$, respectively. It extends to vectors of encrypted bits by just doing $\boxplus$ and $\boxdot$ operations component-wise. Thus, the multiplication of an encryption of a bit z with a vector of the encryption of bits like $\hat{s} = Enc(s_{\ell-1})Enc(s_{\ell-2})...Enc(s_0)$, is defined to be $(Enc(z) \boxdot Enc(s_{\ell-1}))(Enc(z) \boxdot Enc(s_{\ell-2}))...(Enc(z) \boxdot Enc(s_0))$.

Remark. All of the following algorithms are applying the same operation component-wise to each bit ciphertext of the score except $Convert_{GSW,RLWE}(.)$.

- **Greater-Than Comparison ($\hat{\geq}$):** For two encrypted unsigned ℓ-bit integers $\hat{s_i}$ and $\hat{t}$, the operation $(\hat{s_i} \ \hat{\geq} \ \hat{t})$ outputs 1 if $s_i \geq t$ and 0 otherwise. We adapt the greater-than comparison circuit of Cheon et al. [32] by defining a NOT operation. That is, $NOT(b)$ outputs 1 if $b = 1$ and 0 otherwise.This operation can be defined as $NOT(b) = 1 - b$. Thus, $\hat{\geq}$ operation can be defined as $\hat{z_i} = \hat{s_i} \ \hat{\geq} \ \hat{t}$ for $\hat{z_i} = 1 \boxplus (1 \boxplus \hat{s}_{i,\ell-1}) \boxdot \hat{t}_{\ell-1} \boxplus \sum_{j=0}^{\ell-2} (1 \boxplus \hat{s}_{i,j}) \boxdot \hat{t}_j \boxdot d_{j+1}, ..., d_{\ell-1}$.
  Here, $d_j = (1 \boxplus \hat{s}_{i,j} \boxplus \hat{t}_j)$. The depth of this circuit is $log(\ell + 1)$ and, to evaluate this circuit, $2\ell - 2$ homomorphic multiplication is required.

- **Integer Addition ($\dot{+}$):** We use $\dot{+}$ to denote the homomorphic integer addition operation. For addition of two ν-bit integer, x and y, first we make them ℓ-bit by padding with zeros on the left (here, $\ell > \nu$). Then, the sum $\hat{s_i} = \hat{x} \dot{+} \hat{y}$ can be computed efficiently using SIMD operations as introduce in [32]. For $i \in [1, \ell - 1]$ with initial values $\hat{s_0} = \hat{x_0} \boxplus \hat{y_0}$ and $\hat{Carry_0} = \hat{x_0} \boxdot \hat{y_0}$, where the $\hat{s_i}$s are written as $\hat{s_i} = \hat{x} \boxplus \hat{y} \boxplus \sum_{j=0}^{i-1} t_{ij}$ where, $t_{ij} = (\hat{x_i} \boxdot \hat{y_i}) \Pi_{j+1 \leqslant k \leqslant i-1} (\hat{x_k} \boxplus \hat{y_k})$ for $j < i - 1$ and $t_{i,i-1} = \hat{x}_{i-1} \boxdot \hat{y}_{i-1}$. The circuit has $log(\ell - 2) + 1$ depth and using SIMD and parallelism, it can be evaluated just by $3\ell - 5$ homomorphic multiplications.

- **Unary encoding** We represent the unary encoding of a number such as $p \in [0, N)$ (we assume $N < d - 1$, where d is the ring dimension) as $p(x) = 0x^0 + 0x^1 + ... + 1x^p + ... + 0x^{N-1}$. The RLWE encryption of $\hat{p}(x)$ is in the form of $(c = \psi_0 u + \vec{t}g + p(x), c' = \psi_1 u + \vec{t}f)$ where (ψ_0, ψ_1) is the public key, u, f, g are small random noises and $\vec{t}$ is the plaintext space (here $\{0, 1\}$). Note that, $\psi_0 = (\psi_1 . \vec{s} + \vec{te})$ where $\vec{s}$ is the secret. We may represent n-dimension encryption of $\vec{p}$ using matrices as follows:

$$\begin{bmatrix} c_0 \\ \\ \vdots \\ \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} \psi_{0,0} & -\psi_{0,n-1} & \cdots & -\psi_{0,1} \\ \\ \psi_{0,1} & \psi_{0,0} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{0,n-1} & \psi_{0,n-2} & \cdots & \psi_{0,0} \end{bmatrix} \begin{bmatrix} u_0 \\ \\ \vdots \\ u_{n-1} \end{bmatrix} + t \begin{bmatrix} g_0 \\ \\ \vdots \\ g_{n-1} \end{bmatrix} + \begin{bmatrix} \psi_0 \\ \\ \vdots \\ \psi_{n-1} \end{bmatrix}$$

$$\begin{bmatrix} c'_0 \\ \\ \vdots \\ \\ c'_{n-1} \end{bmatrix} = \begin{bmatrix} \psi_{0,0} & -\psi_{0,n-1} & \cdots & -\psi_{0,1} \\ \psi_{0,1} & \psi_{0,0} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{0,n-1} & \psi_{0,n-2} & \cdots & \psi_{0,0} \end{bmatrix} \begin{bmatrix} u_0 \\ \\ \vdots \\ u_{n-1} \end{bmatrix} + t \begin{bmatrix} f_0 \\ \\ \vdots \\ f_{n-1} \end{bmatrix}$$

- **Increment by 1:** The operation $Inc(\hat{p})$ increases the value of $p \in [0, N)$ by 1, unconditionally: $Inc(\hat{p}) = x.\hat{p}(x)$ Here, $\hat{p}(x)$ is the unary encryption of p.

- **Conditional Increment by 1:** The operation $Inc(\hat{p}, \hat{z}_i)$ increases the value of $\hat{p}(x)$ (which is the unary encoding of $p \in [0, N)$) by 1 if $\hat{z}_i = 1$. This function can be defined as $Inc(\hat{p}, \hat{z}_i) = \hat{z}_i(x).(x.\hat{p}(x)) + (1 - \hat{z}_i(x)).\hat{p}(x)$. Here, $\hat{z}_i = \begin{cases} 0 \\ 1 \end{cases}$ is a constant polynomial. This point to the line 10 of Filter in Algorithm 13. To efficiently compute the iterated homomorphic multiplications of the increment function in the loop (line 10 of algorithm 13), we applied GSW encryption based on ring-LWE. More precisely, for $i = 0, ..., n-1$, we can rewrite the increment as $\hat{P}_{i+1} = \hat{z}_i.x\hat{P}_i + (1 - \hat{z}_i).\hat{P}_i = (x-1)\hat{z}_i\hat{P}_i + \hat{P}_i$ which can be simplified to $\hat{P}_{i+1} = [(x-1)\hat{z}_i + 1]\hat{P}_i$. Then, if we expand this down to $i = 0$:

$$\hat{P}_{i+1} = \prod_{j=1}^{i} \hat{z}'_j \hat{P}_0$$

  where $\hat{z}'_j = (x-1)\hat{z}_i + 1$. Note that there is no multiplicative depth in these computations due to the use of GSW encryption. One of the advantages of using GSW is the additive noise growth for iterative homomorphic multiplications [34]. Thus, the noise growth for $\hat{z}_i$ and $\hat{z}'_j$ is equivalent to i additions.

- **Equality ($f_j(.)$):** To evaluate the equality of j and p in **Filter**(S, t) (line 18/19 of Algorithm 13), we define a function $f_j(p) = Enc_{LWE}(\langle \vec{p}, \vec{j} \rangle)$. Here, $f_j(p)$ is equal to "$\hat{1}$" if $j = p$, and "$\hat{0}$" otherwise. Note that both p and j are unary encoded. Therefore, $f_j(p) = \vec{j}^T\vec{p}$. Given the RLWE ciphertext $\hat{p} = (c, c')$ for unary encoded $\vec{p}$ and the plaintext $\vec{j}$, we implement $f_j(p)$ operation homomorphically as follows, which outputs LWE ciphertext of $f_j(p)$:

$$c_{LWE} = \vec{j}^T \vec{c} = \vec{j}^T rot(\psi_0)\vec{u} + \vec{t}.\vec{j}^T \vec{g} + \vec{j}^T \vec{p}$$
$$c'_{LWE} = \hat{c'}^T = \vec{j}^T rot(c') = \vec{j}^T rot(\psi_0) rot(u) + \vec{t}\vec{j}^T rot(f)$$

By substituting the $rot(\psi_0)$ for $rot(\psi_1)rot(\vec{s}) + \vec{t}rot(\vec{e})$, we can rewrite this operation in terms of the secret as follows:

$$c_{LWE} = (\vec{j}^T rot(\psi_1) rot(u))\vec{s} + \vec{t}.(\vec{j}^T (rot(\vec{e})\vec{u} + \vec{g})) + \vec{j}^T \vec{p}$$
$$c'_{LWE} = \vec{j}^T (rot(\psi_1) rot(u))\vec{s} + \vec{t}.(\vec{j}^T (rot(\vec{e}) rot(u) + rot(f)))$$

To do the decryption using the secret s, we proceed as follows:

$$\hat{c'}^T \vec{s} = (\vec{j}^T rot(\psi_1) rot(u))\vec{s} + \vec{j}^T rot(f)\vec{s}$$
$$\vec{j}^T \vec{c} - \hat{c'}^T \vec{s} = \vec{t}\vec{j}^T (rot(\vec{e})\vec{u} + \vec{g} - rot(f)\vec{s}) + \vec{j}^T \vec{p}$$

By reducing this result modulo t, the message which is $\vec{j}^T \vec{p}$ can be obtained.

- **Convert GSW to RLWE** We define $Convert_{GSW,RLWE}(.)$ function to convert a GSW ciphertext of kth bit $s_{i,k}$ of score into RLWE ciphertext of $s_{i,k}$ with message multiplied by $\frac{q}{2^{k+1}}$ for $k = 0, ..., 6$. We define the convert function as $Convert_{GSW,RLWE}(C_{GSW}, i)$ which picks the row $\ell - (r - i)$ of C_{GSW} and outputs the RLWE ciphertext of form $c_0 = a.\vec{t} + \vec{e} + \frac{q}{2^r}.2^i.s_i$, $c_1 = a$ which is an encryption of the plaintext $2^i.s_i$ with plaintext space modulo 2^r. For the bits $s_0, ..., s_{r-1}$ that at the end we want to pack into one ciphertext of the integer $s = \sum 2^i.s_i$, we use $Convert_{GSW,RLWE}(C_{GSW}, i)$ for the bit s_i to encode s_i as $2^i.s_i$ and at the end we add the ciphertext i to get the ciphertext for $\sum 2^i.s_i = s$.

- **Convert RLWE to LWE:** We define $Convert_{RLWE,LWE}(.)$ function to convert a RLWE ciphertext of a bit to the LWE ciphertext. More precisely, in Algorithm 13 line 11, the generated value y_i is a GSW ciphertext, however,

the homomorphic operation in the line 18 is done using LWE (similarly for y'_i). Therefore, we define the convert function as follows over the plaintext:

$$Convert_{RLWE,LWE}(y_i) = < \Big[1, 0, \ldots, 0\Big], \begin{bmatrix} y_i \\ 0 \\ \vdots \\ 0 \end{bmatrix} >= y_i$$

Here, $y_i(x) = y_i x^0 + 0x^1 + ... + 0x^{n-1}$. Similarly, for the ciphertext for $\hat{y}_i = (c, c')$, the conversion function $Convert_{RLWE,LWE}(\hat{y}_i)$ operates as follows:

$$c_{LWE} =< \Big[1, 0, \ldots, 0\Big], rot(c') >$$
$$c'_{LWE} =< \Big[1, 0, \ldots, 0\Big], \vec{c} >$$

4.3.2 Homomorphic search algorithm

The search algorithm only requires one homomorphic operation, an integer addition when the overall score using the aggregation function is computed. For instance, homomorphic aggregation of the scores in **Server Search**$(Tok_{\mathbf{q}},)$ of Algorithm14 can be written as follows:

function Score − Cipher((XSet))
  for $i = 1, ..., L$ **do**
    $\hat{S}^{(GSW)} \leftarrow \hat{S}^{(GSW)} \dot{+} \hat{csc\!ore}_{x_i}$ and $\hat{S}^{(GSW)} \leftarrow \hat{S}^{(GSW)} \dot{+} \hat{csc\!ore}_s$
  end for

Here, $\hat{csc\!ore}_s$ and $\hat{csc\!ore}_{x_i}$ are the ciphertexts of the scores of sterm and xterms, respectively. $\hat{S}^{(GSW)}$ denotes the GSW ciphertext of the aggregated score. The details of this homomorphic integer addition operation is given in Section 4.3.1.

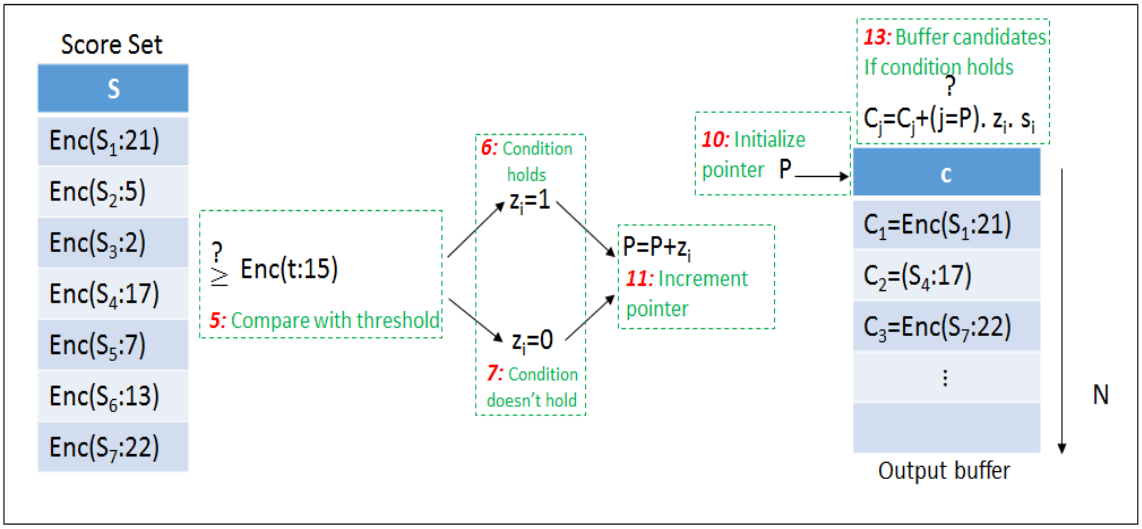


Figure 4.1: Example of Filter algorithm

4.3.3 Homomorphic filter algorithm

Algorithm 13 shows how the filtrating is performed on the ciphertexts. This algorithm reflects **Filter**(S, t) while the homomorphic computations are used. Figure 4.1 demonstrates the functionality of the Algorithm 13 (which is also used in the plaintext form in Filter of Algorithm 14). First, it compares the overall scores (output of the monotone aggregation function in the search algorithm) with the considered threshold value received from the user to set z_is. Whenever, $s_i \geq t$ it sets $z_i = 1$; otherwise it is set to zero. Whenever $z_i = 1$, it copies the corresponding score to the buffer. Finally, it returns the scores stored in the buffer which are in fact equal or greater than the threshold value.

Algorithm 13 Filter-ciphertext

Input: Score set $\hat{S} = \{(\hat{s}_1^{(GSW)}, e_1), ..., (\hat{s}_n^{(GSW)}, e_n)\}$, Threshold $\hat{t}^{(2)}$
Output: two lists of score-ciphertext where the scores are higher than the threshold; $\hat{C}_f$ and $\hat{C}'_f$

1: **function** Filter($\hat{S}, \hat{t}$)
2: $i \leftarrow 1$
3: $j \leftarrow 1$
4: Initialize pointer $\hat{P} = 0$ (beginning of output)
5: $\hat{C}_f \leftarrow Enc^{(LWE)}(0)$
6: $\hat{C}'_f \leftarrow Enc^{(LWE)}(0)$
7: **for** $i = 1, ..., n$ **do**
8: $\hat{e_i}^{(GSW)} = Enc^{(GSW)}(e_i)$
9: $\hat{z_i}^{(GSW)} = \hat{s_i}^{(GSW)} \boxed{\geq} \hat{t}^{(GSW)}$
10: $\hat{P}^{(GSW)} = Inc(\hat{P}^{(GSW)}, \hat{z_i}^{(GSW)})$
11: $\hat{y_i}^{(GSW)} = \hat{z_i}^{(GSW)} \boxdot \hat{s_i}^{(GSW)}$
12: $\hat{y'_i}^{(GSW)} = \hat{z_i}^{(GSW)} \boxdot \hat{e_i}^{(GSW)}$
13: $\hat{y_i}^{(RLWE)} = Convert_{GSW,RLWE}(\hat{y_i}^{(GSW)})$
14: $\hat{y'_i}^{(RLWE)} = Convert_{GSW,RLWE}(\hat{y'}_i^{(GSW)})$
15: $\hat{y_i}^{(LWE)} = Convert_{RLWE,LWE}(\hat{y_i}^{(RLWE)})$
16: $\hat{y'_i}^{(LWE)} = Convert_{RLWE,LWE}(\hat{y'}_i^{(RLWE)})$
17: **for** $j = 1, ..., N$ **do**
18: $\hat{c_j}^{(LWE)} = \hat{c_j}^{(LWE)} \boxplus (f_j(\hat{P})) \boxdot \hat{y_i}^{(LWE)}$
19: $\hat{c'_j}^{(LWE)} = \hat{c'_j}^{(LWE)} \boxplus (f_j(\hat{P})) \boxdot \hat{y'_i}^{(LWE)}$
20: **end for**
21: **end for**
22: $\hat{C}_f = \hat{c}_1^{(LWE)}, ..., \hat{c}_N^{(LWE)}$
23: $\hat{C'}_f = \hat{c'}_1^{(LWE)}, ..., \hat{c'}_N^{(LWE)}$
24: **return** $\hat{C}_f$ and $\hat{C}'_f$
25: **end function**=0

In Algorithm 13, we start homomorphic computations with GSW with the modulus q, where we have $\vec{s}$ as the secret key (line 9 to 11). Then we do the conversion to LWE with the modulus q using the secret key $\vec{s}$. The last multiplication (in line 18/19) involves three steps as defined in [91]; Mult, Scale, and SwitchKey. After the first step, in the LWE with modulus q, the secret key becomes $\vec{s}''$. Then, we do the modulus switching in the second step. We do the conversion to LWE with modulus q' using the secret key $\vec{s}''$. In the last step, we switch the key back to s while the modulus is still q'. Note that to support the above mentioned chain, we need to publish the public key of GSW as well as the key switching public key of LWE which

is $\tau_{\vec{s}' \to \vec{s}}$.

4.4 Our multi-keyword ranked searchable symmetric encryption scheme

In this section, we present our multi-keyword ranked searchable symmetric encryption scheme (MRSSE). By leveraging OXT [3] in a novel way, MRSSE can support multi-keyword ranked search as well as conjunctive and limited range queries. Let us consider the database DB consists of $\mathscr{D}$ documents where each keyword-document pair has a score which shows their relevance. MRSSE first encrypts the scored database using **Setup** algorithm. In order to perform a ranked search over the encrypted database, the **Search** protocol must be run between the client and the server. Then, the server performs **Filter** algorithm to narrow down the results by comparing them with the given threshold in order to return the most relevant documents to the client. Afterwards, the client can sort the received results using **Sort** algorithm. Finally, the client retrieves the top-$\mathscr{K}$ most relevant documents using **Retrieve** algorithm.

Our construction given in Algorithm 14 consists of the following algorithms.

- **Setup**(λ, DB): This algorithm is similar to the one in OXT [3] except that here the scores of keywords are also encrypted and inserted to XSet (the differences are highlighted in red). The score of each keyword-document pairs is computed using the scoring approach introduced in Section 4.2.
- **Search**: This protocol consists of two algorithms:
  - **Client Search**$(K\ , q(\bar{w} = (w_1, ..., w_L)))$: This algorithm inputs the PRF's keys K and the search keywords $(w_1, ..., w_L)$ then generates the search token and outputs it alongside of the chosen threshold.

Algorithm 14 Our MRSSE

Setup(λ, DB)
Input Security parameter λ, Database DB, *Data owner's public key*
output *Encrypted scored database*

Initialize $\mathscr{T} \leftarrow \emptyset$ indexed by keywords W.
Select key K_S for PRF F. Select keys K_X, K_I, K_Z for PRF F_p with range $\mathbb{Z}_p^*$. $\mathsf{XSet} \leftarrow \emptyset$
$\leftarrow \{\}$
for $w \in \mathsf{W}$ **do**
    Initialize $\vartheta \leftarrow \{\}$
$K_e \leftarrow F(K_S, w)$
    for $\mathsf{id} \in \mathsf{DB}(w)$ **do**
        Set a counter $\mathscr{C} \leftarrow 1$
        Compute $\mathsf{xid} \leftarrow F_p(K_I, \mathsf{id})$,
        $z \leftarrow F_p(K_Z, w||\mathscr{C})$; $y \leftarrow \mathsf{xid}z^{-1}$,
        $e \leftarrow \mathsf{Enc}(K_e, \mathsf{id})$
        Compute $s_w = Score(w, \mathsf{id})$
        Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w)\cdot \mathsf{xid}}$
        Compute $cscore = Enc(, s_w)$
        Set $\mathsf{XSet} \leftarrow \mathsf{XSet} \cup (\mathsf{xtag}, cscore)$
        Append $(y, e, cscore)$ *to* ϑ
        $\mathscr{C} \leftarrow \mathscr{C} + 1$
    end for
    $\mathscr{T}[w] \leftarrow \vartheta$
end for
Set $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSet.Setup}(\mathscr{T})$ and let $= (\mathsf{TSet}, \mathsf{XSet})$.
return , $K = (K_S, K_X, K_I, K_Z)$

Search

Client Search$(K$, $q(\bar{w} = (w_1, ..., w_L)))$:
Input K , $q(\bar{w} = (w_1, ..., w_L))$
Output $Tok_{\mathbf{q}}, t$

Client's input is K and query q.
Computes $\mathsf{stag} \leftarrow \mathsf{TSet.GetTag}(K_T, w_1)$.
Client sends stag to the server.
for $\mathscr{C} = 1, 2, \ldots$ until the server stops **do**
    for $i = 2, \ldots, L$ **do**
        $\mathsf{xtoken}[\mathscr{C}, i] \leftarrow g^{F_p(K_Z, w_1||\mathscr{C})\cdot F_p(K_X, w_i)}$
    end for
    $\mathsf{xtoken}[\mathscr{C}] \leftarrow (\mathsf{xtoken}[\mathscr{C}, 2], \ldots, \mathsf{xtoken}[\mathscr{C}, L])$
end for
$Tok_{\mathbf{q}} \leftarrow (\mathrm{stag}, \mathsf{xtoken}[\mathscr{C}])$
Client chooses a threshold t
return $Tok_{\mathbf{q}}, t$

Server Search$(Tok_{\mathbf{q}},)$:
Input $Tok_{\mathbf{q}}$,
Output S

$\mathrm{S} \leftarrow \{\}$
$\vartheta \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathrm{stag})$
for $\mathscr{C} = 1 : |\vartheta|$ **do**
    Retrieve $(e_{\mathscr{C}}, y_{\mathscr{C}}, cscore_{s,\mathscr{C}})$ from the $\mathscr{C}$-th tuple in ϑ
    if XSet contains $(\mathsf{xtoken}[\mathscr{C}, i]^{y_{\mathscr{C}}}, cscore_{x_i})$ for some $cscore_{x_i}$ for all $i = 1, ..., L$ **then**
        $s_{\mathscr{C}} \leftarrow \mathsf{Score} - \mathsf{Cipher}(\mathsf{XSet})$
    // that is $s_{\mathscr{C}} = \sum_{i=1}^{L} cscore_{x_i} + cscore_{s,\mathscr{C}}$ in the plaintext form
        $S \leftarrow S \cup \{(s_{\mathscr{C}}, e_{\mathscr{C}})\}$
    end if
end for
return S

Filter$(\hat{S}, \hat{t})$
Input Score set S, Threshold t
Output two lists of score-ciphertext where the scores are higher than the threshold; $\hat{C}_f$ and $\hat{C'}_f$

$(\hat{C}_f, \hat{C'}_f) \leftarrow Filter(\hat{S}, \hat{t})$
    // the following demonstrate the filter in plaintext form
$i \leftarrow 1$
$j \leftarrow 1$
for $i = 1, ..., n$ **do**
    if $s_i \geq t$ **then**
        Set $z_i = 1$
    else
        Set $z_i = 0$
    end if
    Initialize pointer $P = 0$ to the output buffer
    $P = P + z_i$
    for $j = 1, ..., N$ **do**
        $c_j = c_j + (j \stackrel{?}{=} P).z_i.s_i$
        $c'_j = c'_j + (j \stackrel{?}{=} P).z_i.e_i$
    end for
end for
$C_f = (c_1, ..., c_N)$
$C'_f = (c'_1, ..., c'_N)$
return C_f, C'_f

Sort$(\hat{C}_f, \hat{C'}_f)$
Input $\hat{C}_f, \hat{C'}_f$
Output C_s

for $j = 1, 2, \ldots N$ **do**
    $s \leftarrow Dec(\mathsf{sk}, c_j)$
    $e \leftarrow Dec(\mathsf{sk}, c'_j)$
    $C \leftarrow C \cup \{(s, e)\}$
    $C_s \leftarrow Sort(C)$ // sort the set using any well-known sorting algorithm
end for
return C_s

Retrieve$(C_s, \mathscr{K})$
Input
Sorted encrypted results C_s and the number $\mathscr{K}$ of top documents to be retrieved
Output Result id

Client computes $k_e \leftarrow PRF(k_s, w_1)$
for $\mathscr{P} = 1, ..., \mathscr{K}$ **do**
    compute $id \leftarrow Dec(k_e, e_{\mathscr{P}})$
    return id
end for

– **Server Search**($Tok_{\mathbf{q}}$,): This algorithm inputs the search token $Tok_{\mathbf{q}}$ and the scored database . Next, finds the match for the least frequent keyword in TSet and then tests the membership of corresponding document Ids in XSet for the other searched keywords. Moreover, in parallel this algorithm computes the aggregation function over the collected scores homomorphically. It is worth to note that when we perform ranking, we might not want to ignore the documents that do not contain sterm with the intersection of all of xterms. That is, there might be some documents which contains a few of xterms with high scores. Thus, if we want to return only documents matching all queried keyword, we keep the XSet as it is in the **Server Search**($Tok_{\mathbf{q}}$,) algorithm; otherwise we can simply remove it.

- **Filter**(S, t): Given the score set S and the threshold t, this algorithm compares the overall scores (output of the monotone aggregation function in **Server Search**($Tok_{\mathbf{q}}$,)) with the threshold value. It returns the scores which are equal or greater than the considered threshold and the corresponding ciphertext[4].

- **Sort**(C_f, C'_f): This algorithm inputs two list with corresponding elements (score/ciphertext). First, decrypts the score list and then uses any well-known sorting algorithm to sort the received list of candidates. Finally, it outputs the ordered set of the score-ciphertext pairs.

- **Retrieve**$(C_s, \mathscr{K})$: To retrieve the top-$\mathscr{K}$ documents, this algorithm first computes the decryption key and then decrypts the first $\mathscr{K}$ documents from the ordered set of the score-ciphertext pairs C_s.

For the sake of readability, in this section, the homomorphic computations are not considered in **Search** protocol and **Filter**(S, t) algorithm. That is, we show only operations performed on plaintext values. Section 4.3 presented the actual algorithms with the required homomorphic computations on ciphertexts.

[4]We are assuming the size of output buffer (here, N) is big enough to contain all of the results which has equal/higher score than the considered threshold. If not, our protocol returns last N results higher than the threshold since the pointer P is incremented modulo N.

4.4.1 Modes of operation

To minimize the communication overhead, we define two modes for our scheme: trivial and filtered. The former refers to the condition where the number of results in **Search** is smaller than the breaking point[5]. In this mode, both **Filter**(S, t) and **Sort**(C_f, C'_f) algorithms are not required. Otherwise, the server performs in the filtered mode by running the **Filter**(S, t) algorithm to narrow down the results to the most relevant ones. This leads to the lower communication overhead and network traffic. These two modes are illustrated in Figure 4.2 where we discus the communication overhead of our scheme. That is, the communication overhead in trivial mode (black line) is linear to the number of matching result whereas it is constant in the filtered mode (red line). Note that the communication cost in the filtered mode is independent of number of matching results.

4.5 Evaluation

In this section we discuss the cost evaluation of MRSSE from computation and communication complexity viewpoints (refer to the full version for the security analysis).

4.5.1 Computation complexity

The costs that our design added on the top of OXT to support ranked queries are related to the homomorphic computations in **Server Search**$(Tok_{\mathbf{q}},)$ and **Filter**(S, t) algorithm. In **Server Search**$(Tok_{\mathbf{q}},)$, we need to compute the score aggregation function using the integer addition operation $(\dot{+})$ in a loop that costs $L \times \dot{+}$. The evaluation of this circuit in the loop has multiplicative depth of $logL(log(\ell - 2) + 1)$. **Fil-**

[5] breaking point is the point that the number of filtered results is the same as unfiltered results- refer to section 4.5.2.

ter(S, t) algorithm requires more homomorphic computations which are performed over $(\mathbb{Z}_N, (+, .))$ as follows:

- $n\times$ $\boxed{\geq}$ for the loop in line 9 of Algorithm 13,
- $2 \times n \times \boxdot$ for the loop in line 11 of Algorithm 13,
- $2 \times n \times N \times (\ell \boxplus + f_j(\hat{P}) + (\ell + 1)\boxdot)$ for line 18 and line 19 of Algorithm 13,

where the computation cost of $f_j(\hat{P})$ is negligible. Finally, the overall multiplicative depth of **Filter**(S, t) would be $Depth(Filter) = 2+(log\ell+1)$. Therefore, the overall multiplicative depth would be:

$$Depth(Search) + Depth(Filter) = logL(log(\ell - 2) + 1) + (2 + (log\ell + 1))$$

4.5.2 Communication complexity

In order to determine the communication complexity, we should set the parameters in such way that the following conditions hold. We denote as $\|.\|_\infty$ standard norm for scalars and vectors.

- *Aggregation noise growth.* Let D_{Agg} be the depth of the aggregation function in $Search_{mr}$ algorithm. The noise growth of this function is at most $(\eta d + 1)^{\frac{1}{2}D_{Agg}}.\sigma(CScore)$ where $\sigma(CScore) = \sqrt{m.d}\sigma(\vec{e})$ is the noise for the fresh ciphertext from GSW encryption. By assuming $\sigma(\vec{e}) = 2$, the noise growth of aggregation function is at most $(\eta d + 1)^{\frac{1}{2}D_{Agg}}.2\sqrt{m.d}$.
- NAND *gate homomorphic noise growth.* We use NAND operation to restrict the message space to $\{0, 1\}$ in order to avoid blowup of error. The noise of NAND of ciphertexts C_1, C_2 for the message $\mu \in \{0, 1\}$ is:

$$|noise(\mathsf{NAND}(C_1, C_2))| = \mu.noise(C_1) + \|C_1.noise(C_2)\|_\infty$$

$$|noise(\mathsf{NAND}(C_1, C_2))| \leq \|noise(C_1)\|_\infty + \|C_1.noise(C_2)\|_\infty$$

$$|noise(\mathsf{NAND}(C_1, C_2))| \leq (\eta.d + 1) \times max(\|noise(C_1)\|_\infty, \|noise(C_2)\|_\infty)$$

We analyze the standard deviation growth using independence heuristic assumption (that indicates the coordinates of noise vector are independent of Gaussian samples), for each coordinates of noise (similar to the assumption in [92]):

$Var(noise(\mathsf{NAND}(C_1, C_2)) \leq (\eta.d + 1) \times max(\, Var(noise(C_1))\, Var(noise(C_2))$

$\sigma = sd(noise(\mathsf{NAND}(C_1, C_2)) \leq \sqrt{(\eta.d + 1)} \times max(sd(noise(C_1)), sd(noise(C_2))$

$\sigma_{\mathsf{NAND}}(out) \leq \sqrt{(\eta.d + 1)}\sigma_{\mathsf{NAND}}(in)$ .Therefore, for the depth $D - D'$, the noise growth is $\sigma_{\mathsf{NAND}}(out) \leq (\eta.d + 1)^{\frac{D-D'}{2}} \sigma_{\mathsf{NAND}}(in)$.

- *Homomorphic noise growth for conditional increment.* The conditional increment of the encrypted pointer $\hat{P}$ can be implemented using Mult operation: $\hat{P}_{i+1} = \hat{P}_i \hat{z}'_i = \mathsf{Mult}(\hat{P}_i, \hat{z}'_i) = \mathsf{Flatten}(\hat{P}_i \hat{z}'_i)$. Here, $\hat{z}'_i = \mathsf{Flatten}((x-1).I.\hat{z}_i + I)$,which has the noise $\|noise(\hat{z}'_i)\| = noise(\hat{z}_i).\|(x-1)\|$

  $sd(\hat{z}'_i) = \|(x-1)\|.sd(\hat{z}_i)$

  Therefore, the conditional increment has the noise growth as follows:

  $noise(\hat{P}_{i+1}) = \hat{z}'_i.noise(\hat{P}_i) + \hat{P}_i.noise(\hat{z}'_i)$

  $Var(\hat{P}_{i+1}) \leq \|\hat{z}'_i\|^2.\, Var(\hat{P}_i) + \eta d\, Var(\hat{z}'_i)$

  Here, $\|\hat{z}'_i\| = 1$because $\hat{z}'_i \in \{x^0, x'\}$, and $Var(\hat{z}'_i) \leq 2 \times Var(\hat{z}_i)$. Therefore,

  $Var(\hat{P}_{i+1}) \leq Var(\hat{P}_i) + 2\eta d\, Var(\hat{z}_i)$

  $\forall i = 0, ..., n-1$

  $Var(\hat{P}_i) \leq Var(\hat{P}_n) \leq Var(\hat{P}_0) + (n-1) \times 2Nd(max\, Var(\hat{z}_i))$

  $\forall i\ sd(\hat{P}_i) \leq \sqrt{2n\eta d} \times max\ sd(\hat{z}_i)$

- *Overall noise growth condition.* We denote the depth of $\boxed{\geq}$ homomorphic operation by $D_{\boxed{\geq}}$. The overall noise noise growth condition for Ring-GSW is

$\sigma_{out} \leq \sigma_{in} \times (\eta d+1)^{\frac{1}{2}D_{\boxed{\geq}}} . \sqrt{2n\eta d} . \sqrt{\eta d+1}$where σ_{in} indicates the input noise to the **Filter**(S,t) algorithm. This noise is generated by the aggregation function in **Server Search**$(Tok_{\mathbf{q}},)$. Note that here, $(\eta d+1)^{\frac{1}{2}D_{\boxed{\geq}}}$, $\sqrt{2n\eta d}$, and $\sqrt{\eta d+1}$ correspond to lines 9, 10, and 11 of Algorithm 13, respectively. More accurately, the noise for $\hat{P}_i$ is at most $\sigma_{\hat{P}_i} = 2\sqrt{md}(\eta d+1)^{\frac{1}{2}(D_{\boxed{\geq}}+D_{agg})} . \sqrt{2n\eta d}$ whereas, the noise for $\hat{y}_i$ is at most $\sigma_{\hat{y}_i} = 2\sqrt{md}(\eta d+1)^{\frac{1}{2}(D_{\boxed{\geq}}+D_{agg})} . \sqrt{\eta d+1}$.

When we perform $f_j(\hat{P})$, $\hat{P}$ will be converted to LWE. Hence, we switch from modulus q to q'. Therefore, for the last multiplication in **Filter**(S,t) algorithm $\sigma^{(LWE),q'}_{f_j(\hat{P})\boxdot\hat{y}_i}$ is at most $\sigma^{(LWE),q}_{f_j(\hat{P})}$ if the condition: $2 \times (\sigma^{(LWE),q}_{\hat{P}})\gamma_{\mathbb{Z}} \leq \Delta$ is satisfied. Here, $\Delta = \frac{q}{q'}$ is the ratio between q and q'. The noise in each $\hat{c}_j$ (with modulus q') would be at most $\sqrt{n} \times \sigma^{(LWE),q'}_{f_j(\hat{P})}$. That is, each $\hat{c}_j$ is computed as a sum of n intermediate ciphertexts thus, the standard deviation gets multiplied by $\sqrt{n}$.

The BGV-type LWE multiplication consists of three steps; Mult, Scale, and SwitchKey [91]. After the first step, the noise has length at most $\gamma_{\mathbb{Z}}B^2$ (here B is a bound on the noise length). Then, we apply the Scale function (as defined in Lemma 4 in [91]), after which the noise length is at most $(q'/q)\gamma_{\mathbb{Z}}B^2 + \eta_{Scale}$ where $\eta_{Scale} \leq (\tau/2)\sqrt{d}\gamma_{\mathbb{Z}}h$. After the last step (the SwitchKey is defined in Lemma 9 in [91]), the noise become at most $(q'/q)\gamma_{\mathbb{Z}}B^2 + \eta_{Scale} + \eta_{SwitchKey}$ where $\eta_{SwitchKey} \leq 2\gamma_{\mathbb{Z}}\binom{d+1}{2}(logq')^2$. At the end, we want the noise to be smaller than $\frac{1}{2}(q'/\tau)$ to decrypt correctly. Note that, in BGV-type LWE, the actual bound on the noise length is used, whereas, in our case, we use the standard deviation but the relation is similar. In fact, the last multiplication has the standard deviation of the noise of input $\sigma_{\hat{P}_i}$ and $\sigma_{\hat{y}_i}$. We are assuming these are heuristically independent of Gaussian noises. Therefore, if the ratio between the threshold q'/τ and the standard deviation is more than $\sqrt{2ln(2/\epsilon)} \leq \frac{q'}{2\tau}$, then the probability, that the noise crosses this threshold, is less than ϵ.

- *Decryption correctness condition.* For the correctness of decryption, at the end

the condition: $\frac{\sigma_{\hat{P}_i}.\sigma_{\hat{y}_i}}{\Delta} + \eta_{Scale} + \eta_{SwitchKey} \leq \frac{q'}{\tau\sqrt{2ln(2/\epsilon)}}$must hold. Here, $\epsilon = 2^{-20}$. For the security, the hardness of RLWE 2^λ security for GSW keys must be hold. Thus, we need to show that the RLWE with noise $\sigma(\vec{e}) = 2$, dimension d and modulus q is hard. The condition $\alpha = 2/q$ ensures this assumption.

Parameter setting: We can determine the parameters based on the above mentioned conditions and the security of RLWE. To determine d, we need to choose it based on the complexity of best lattice attack against underlying Ring-LWE problem in dimension d with modulus q and noise σ. Once we determined α, q and d, we can find the security against lattice attack, T_{attack}. That is, we set the security parameter λ so that $T_{attack} \geq 2^\lambda$. Table 4.4 provides a few examples of parameter setting on the security parameter $\lambda = 80$. We used an attack estimator for the Learning with Errors Problem [93, 94]. The output of **Filter**(S, t) algorithm are LWE ciphertexts encrypted with the modulus, q'. For kth bit of each score s_i, its LWE ciphertext is in the form of $(a_{i,k}, b_{i,k} = a_{i,k}.\vec{s} + \vec{e} + \frac{q'}{2^i}.s_{i,k})$. Each ciphertext is $(d+1) \times logq'$ bits, for 7 bits score and 64 bits document identifier, we would have $71 \times N \times (d+1) \times logq'$ bits of N filtered results to be delivered to the client. This can be reduced by packing each $\log \tau$ bits ciphertext into one ciphertext by homomorphic addition of the ciphertexts. This results the following overall communication cost:

$$Communication\ cost = \frac{71N}{\log \tau} \times (d+1) \times (logq')$$

Although the length of ciphertext in symmetric cipher of OXT is relatively small, the overall communication overhead is directly related to the number of matching results. This is evident in the case of cloud storage where the result space is potentially huge. The following example clarifies the significance of this issue. Let us assume that the scores are 7 bits and document Ids are 64 bits, which results in about 200 bits of ciphertext in OXT assuming AES CBC mode is used. Thus, the communication cost would be $CC_{OXT} = 200n$ (where n is the number of matching results for all queried keywords). Let us assume the total number of matching results $n = 10^6$, then the communication cost of OXT would be 25MB. However, this can be

reduced to just about 6.5MB by performing our scheme in the filtered mode. More precisely, the server can compute the breaking point $n_b = \frac{\frac{71N}{\log\tau}\times(d+1)\times(logq')}{200}$ which in this case is about 263055 matching results (for $N = 2^6$, $d = 4750$, $\log\tau = 32$), and then compares it with the total results $n = 10^6$. Since the breaking point is smaller than n, the server runs filtered mode which has the communication cost of $CC_{our} = \frac{71N}{\log\tau} \times (d+1) \times (logq') = 6.55$MB. Table 4.3 shows the communication cost improvement when the filtration approach is used versus the trivial mode that returns all of the matching documents. It is apparent from this table that the proposed approach has a significant impact on the communication cost.

Table 4.3: Communication cost improvement

Number of matching results	400000	600000	800000	1000000
Communication cost of Trivial mode	8×10^7 Bits	12×10^7 Bits	16×10^7 Bits	2×10^8 Bits
Communication cost of Filter mode	52611×10^3 Bits	52611×10^3 Bits	52611×10^3 Bits	52611×10^3 Bits
Communication cost improvement	34.23%	56.15%	67.11%	73.69%

Figure 4.2 illustrates the overall communication cost of our scheme. Note that the trivial mode introduced in section 4.4.1 acts the same way as OXT. It is apparent from this figure that our scheme (the green line) reduces the communication cost significantly by filtering the results to the most relevant ones.

Table 4.4: Parameter settings

Parameters									T_{attack}			Breaking point (n_b)
L	ℓ	N	d	D	α	q	q'	$\log\tau$	usvp	dec	dual	
4	7	2^6	4750	8	$2^{(-216)}$	2^{217}	2^{78}	32	$2^{80.8}$	$2^{81.4}$	$2^{82.2}$	263055

L:number of queried keyword; l: bit length of scores; N: output buffer size; d: ring dimension
D: multiplicative depth of MRSSE; usvp: unique shortest vector problem; dual:dual-lattice attack

Figure 4.2: Overall communication cost

4.6 Summary

We have presented a generic solution for efficient multi-keyword ranked searchable symmetric encryption. The proposed threshold-based filtering solution enables the honest-but-curious server to refine the encrypted search results and returns only the most relevant ones to the user. The proposed scheme supports multi-keyword ranked search as well as Boolean and limited range queries. Our scheme resists all attacks associated with OPE leakage. In comparison with the conventional searchable symmetric encryption schemes, our solution decreases the communication overhead between the client and server significantly without adding any additional leakage to the server.

Chapter 5

Geometric range search on encrypted data with forward/backward privacy

This section presents the details of the third contribution of this research, geometric range search on encrypted data with forward/backward privacy, which is submitted to IEEE Transactions on Dependable and Secure Computing.

5.1 Introduction

Location-Based Services (LBS) have a wide range of applications from transportation (e.g., Uber) to social media (such as "Nearby friends" on Facebook or "People Nearby" on WeChat). The LBS providers mostly deal with a large scale datasets and outsource such datasets to a third party server (e.g., Cloud server), and provides great benefits to database owners such as on-demand access to the data. However, the confidentiality of the outsourced sensitive data and users' privacy are the major

concerns.

Searchable Symmetric Encryption (SSE) is an efficient cryptographic technique, which enables secure storage an untrusted/semi-trusted server while the searchability is preserved. There is a growing body of literature on SSE with varying trade-offs between security, efficiency, and practicality [3, 95, 96]. Range query is a primary database operation to meet the practical data retrieval need. Range search plays a vital role in supporting LBS.

Nevertheless, the following leakages from the range query enable an attacker to reconstruct the dataset.

- Access pattern: the encrypted records matching the query are leaked to the server.
- Communication volume: the number of ciphertexts returned as a matching response to a query.

Recently, a considerable amount of literature has been published on the problem of reconstructing encrypted databases from range query leakage [22–25]. So far, two types of attacks have been discovered based on the leakage from range queries: Full Database Reconstruction (FDR) and Approximate Database Reconstruction (ADR). FDR recovers the exact value for every record of the database, whereas ADR aims to reconstruct the plaintext database with a bounded approximation error. Kellaris et al. in [24] showed that a passive adversary (does not choose the queries) could perform FDR without requiring auxiliary information. Moreover, several works (e.g., [23, 25]) performed ADR using *access pattern* leakage.

On the other hand, dynamic setting (in which the data is updatable) is a necessity in LBS applications due to users' movements. Dynamic setting introduces additional leakages to the server about the update undertaken. These leakages are subject to devastating adaptive attacks. Supporting forward and backward privacy enables the SSE scheme to mitigate such attacks [26, 27]. Forward privacy ensures that new

updates cannot be related to the previous search results [27]. Moreover, forward private searchable encryption schemes are not vulnerable to file injection attacks. Backward privacy is another important security notion for searchable encryption schemes. Backward privacy ensures the privacy of the database and its updates during search queries. Roughly speaking, search queries should not leak matching documents after they have been deleted. However, backward privacy does not hide the document identifiers matching a search query (access pattern) although it hides them in updates. In the existing definitions of forward/backward privacy as given above, some leakages have not been captured. Such leakages might be critical for SSE-based geometric range search schemes but may not be a serious issue in the other contexts. For example, in the case of LBS application, the privacy of the dataset content (locations/ points of interest) is critical. As mentioned earlier, one of the popular applications of LBS is finding nearby people on social media. For instance, a user wants to learn the location of his friends which are at a particular distance from him. In this scenario, access pattern leakage reveals the identities of the users in a region (database content) to the server.

As the main focus of this is on geometric search, we introduce a new security notion, called content privacy. Content privacy hides the access pattern both in search and update. That is, a content private SSE-based scheme supporting geometric range search would not reveal to the server the identities of points in the query range as well as update queries but it allows leakage on the query. Therefore, in the given scenario, if the utilized SSE-based scheme support content privacy, then in the worst case, the server learns that some of the friends are in a particular location but will not learn their identities.

It is worth to note that in addition to SSE, there exists some other approaches for range search over encrypted data. They might be applicable for supporting geometric range search such as OPE[1] [97], and ORAM[2] [98]. SSE appears to be

[1]Order-Preserving Encryption
[2]Oblivious RAM

the most suitable solution as it achieves a stronger notion of security than OPE and better efficiency than ORAM. Therefore, the rest of this paper focuses only on SSE-based schemes for geometric range search.

Our Contributions. There are several important areas where this study makes an original contribution, namely:

- We first formulate the new definition of content privacy and show its importance in the context of spatial search over encrypted datasets.
- We propose two new constructions that support range searches over the dynamic encrypted spatial dataset. Our first construction supports backward privacy while the second one is forward secure. Both of the proposed constructions are content private.
- In comparing with the existing SSE-based scheme which support geometric range search, our constructions are the first of this type that could avoid access pattern leakage; hence they are secured against FDR and ADR attacks.
- Our design offers a higher level of security and practical efficiency. Our experimental results show that for the worst case of 20K data points the maximum time required for search by our constructions is about 1.3 seconds.

5.2 Preliminaries

In this section, we present the definitions needed in our construction and security analysis.

5.2.1 Additive homomorphic encryption

An additive homomorphic encryption scheme works as follows: *Key generation:* Given the security parameter λ, this algorithm generates the key k. *Encryption:*

This algorithm takes the message m and the key k as inputs, and outputs the corresponding ciphertext C where $C = Enc_k(m)$. *Decryption:* On the inputs C and k, this algorithm outputs the plaintext message $m = Dec_k(C)$. *Homomorphic addition:* The homomorphic addition of two ciphertexts will decrypt to the sum of their corresponding plaintexts; $Dec_k(C_1 + C_2) = Dec_k(Enc_k(m_1) + Enc_k(m_2)) = m_1 + m_2$ The scheme mentioned above is symmetric; for the asymmetric setting, the private-public key pair should be used. In the following subsection, we introduce a variation of the additively symmetric homomorphic encryption (ASHE) scheme of [99]. To be much more efficient, we replaced the addition modulo n with the exclusive-or operation which can satisfy the requirements in our proposed scheme.

5.2.1.1 Symmetric additive homomorphic encryption.

Let $K = \{K_1, ..., K_N\}$ be the set of the secret keys; for the plaintext space $\{0,1\}^\lambda$, we define a pseudo-random function (PRF) $F_K : I \rightarrow \{0,1\}^\lambda$. Here, $i \in I$ is an identifier (which is used as the counter in our constructions). We define two additive homomorphic encryption $\mathbf{E} : (Enc, Dec)$ and $\mathbf{E^{Updt}} : (Enc^{Updt}, Dec^{Updt})$ as follows;

$$(C, i) = \mathrm{Enc}_K(m, i) := \left(\left(m \oplus \sum_{j=1}^{N} F_{K_j}(i) \right), i \right) \tag{5.1}$$

$$m = \mathrm{Dec}_K(C, i) := C \oplus \sum_{j=1}^{N} F_{K_j}(i) \tag{5.2}$$

$$(C, i) = \mathrm{Enc}_K^{Updt}(m, i) := \left((m \oplus F_K(i-1) \oplus F_K(i)), i \right) \tag{5.3}$$

$$m = \mathrm{Dec}_K^{Updt}(C, i) := C \oplus F_K(i-1) \oplus F_K(i) \tag{5.4}$$

Let $(C_1, i) = Enc_K(m_1, i)$ and $(C_2, i) = Enc_{K'}(m_2, i)$, we define the additive homomorphic operation for $\mathbf{E}$ as shown in Equation 5.5;

$$(C_1, i) \oplus (C_2, i) = (C_1 \oplus C_2, i) := \mathrm{Enc}_{K''}((m_1 \oplus m_2), i) \tag{5.5}$$

Here, $K'' = (K \cup K')$. Similarly, for $(C_3, i-1) = Enc_K(m_3, i-1)$ and $(C_4, i) = Enc_K^{Updt}(m_4, i)$, the additive homomorphic operation works as shown in Equation 5.6;

$$(C_3, i-1) \oplus (C_4, i) = (C_3 \oplus C_4, i) := \mathrm{Enc}_K\left((m_3 \oplus m_4), i\right) \tag{5.6}$$

Remark. Note that we use this additive operation in two different ways in our construction. That is, in Algorithm 23, to perform the update, the client sends a bit string encrypted using $\mathbf{E^{Updt}}$ which enables the server to update the leaf nodes using the homomorphic addition given in Equation 5.6. Then, to update the intermediate nodes to the root, the server uses Equation 5.5. More precisely, for the update of the leaf nodes, the key would be the same as the old key for each node while the counter is different whereas for the intermediate nodes is the opposite.

5.2.2 Security definition

The security definition of the proposed constructions is formulated using two games; $\mathtt{REAL}_{\mathcal{A}}^{\Sigma}(\lambda)$ and $\mathtt{IDEAL}_{\mathcal{A},\mathcal{S}}^{\Sigma}(\lambda)$. The former is executed using our scheme, whereas the latter is simulated using the leakage of our scheme as defined in Section 5.7.1. If an adversary such as $\mathcal{A}$ cannot distinguish these two games, then we can say that there is no leakage beyond what is defined in the leakage function. These games can be formally defined as followed;

- $\mathtt{REAL}_{\mathcal{A}}^{\Sigma}(\lambda)$: On input a dataset chosen by the adversary $\mathcal{A}$, it outputs EDB by using $\mathsf{Setup}(DB, \lambda)$ to $\mathcal{A}$. The adversary can repeatedly perform a search and update query. The game outputs the results generated by running Search (q, σ, EDB) and Update (K, σ, op, in, EDB) to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a bit.
- $\mathtt{IDEAL}_{\mathcal{A},\mathcal{S}}^{\Sigma}(\lambda)$: On input a database chosen by $\mathcal{A}$, it outputs EDB to the adversary $\mathcal{A}$ by using a simulator $\mathcal{S}(\mathcal{L}^{Stp})$. Then, it simulates the results for the search query using the leakage function $\mathcal{S}(\mathcal{L}^{Srch})$ and uses $\mathcal{S}(\mathcal{L}^{Updt})$ to simulate the results for update query. Eventually, $\mathcal{A}$ outputs a bit.

Definition 15. *The scheme Σ is $\mathcal{L}$-adaptively-secure if for every PPT adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ such that* $|Pr[\mathtt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda) = 1] - Pr[\mathtt{IDEAL}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq negl(\lambda)$.

5.3 Security notions

Several studies showed that the leakage of SSE schemes could lead to revealing the search queries and/or reconstructing of the plaintext of the encrypted database [100–103]. In this section, we review the original definitions of forward and backward privacy. We then extend them to cover the SSE schemes with spatial datasets. Afterward, we introduce the notion of "Content privacy", which is useful for SSE schemes where the access pattern should not be leaked.

In this research, we are discussing the forward privacy for the encrypted spatial dataset. Thus, we define a variation of the Definition 4 for the searchable encryption with the spatial dataset as introduced in Definition 16.

Definition 16. *A $\mathcal{L}$-adaptively-secure spatial-SSE scheme Σ is forward secure if the update leakage function $\mathcal{L}^{Updt}$ can be written as $\mathcal{L}^{\mathrm{Updt}}(\mathrm{op}, \mathrm{in}) = \mathcal{L}'(\mathrm{op}, \mathrm{P}_i)$, where the input "*in*" consists of an updated point label and its updated location and the operation* op *can be insertion, deletion, or modification.*

5.3.1 Content privacy

By content privacy, we model the leakage which is not considered in the previous definitions. This model is not a stronger/weaker security definition than the backward privacy, but a different notion to capture the other aspects of the leakage. Content privacy limits the information on the updates affecting document *id* that the server can learn upon a search query on w. That is, update queries do not leak any information about the updated document.

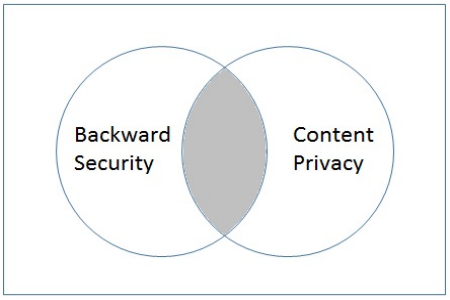


Figure 5.1: Security notions similarity

As shown in Figure 5.1, the content privacy has some similarity with Type-II backward privacy as both protect the content and do not leak about the documents' identifiers in the update queries. However, in Type-II backward privacy, it is still assumed that the access pattern is revealed and this, in turn, reveals the content information on the documents' identifiers. More precisely, Backward privacy leaks about the content in the search queries via access pattern ($TimeDB(w)$ in Definition 6 and Definition 7), whereas our proposed content privacy definition (Definition 17) does not leak about the content during the search. On the other hand, it allows more leakage on the keyword from the backward privacy. In particular, the previous models for SSE schemes were designed for document retrieval. Thus, the access pattern leakage was inherent. Our model does not leak access pattern as the data retrieval is not required and only checking the presence of the points of interest is desired.

Definition 17 (Content Privacy)**.** *A $\mathcal{L}$-adaptively-secure SSE scheme is content-private iff the search and update leakage functions $\mathcal{L}^{Srch}$, $\mathcal{L}^{Updt}$ can be written as: $\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op, w)$ $\mathcal{L}^{Srch}(w) = \mathcal{L}''(w)$ where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.*

To apply this notion to spatial datasets, we define a variant of the Definition 17 as presented in Definition 18. In general, update queries do not leak which point labels are involved in the coordinates that are being updated.

Definition 18 (Content privacy for spatial dataset)**.** *A $\mathcal{L}$-adaptively-secure SSE scheme is content-private iff the search and update leakage functions $\mathcal{L}^{Srch}$, $\mathcal{L}^{Updt}$*

can be written as: $\mathcal{L}^{Updt}(op, r, P) = \mathcal{L}'(op, r)$ $\mathcal{L}^{Srch}(r) = \mathcal{L}''(r)$ *where* $\mathcal{L}'$ *and* $\mathcal{L}''$ *are stateless. Here,* r *represents a range of coordinate and a point identifier is denoted by* P*.*

5.3.1.1 Example of content privacy against the existing dynamic SSE

In this section, we show how a lack of content privacy makes dynamic SSE schemes prone to a leakage abuse attack and a file injection attack. In the dynamic setting, the data owner can update the files (it also includes the addition of the new files) after the initial setup. We investigate the vulnerabilities in the dynamic SSE proposed by Cash et al. in [104]. To support the dynamic setting, they maintain a revocation list for deletions and an extra dictionary "D^+" for the addition of new documents. This strategy does not support forward privacy and content privacy. More precisely, the server can reuse the previously issued search tokens and learn about the updated document and the new search queries. For the sake of simplicity, we demonstrate the issue for the *add* updates. For the static database D and the extra dictionary "D^+", let us consider the following example;

- The state $\sigma = 1$: the database is not updated. The client issues the search token ST_i for the keyword w_i; the server returns the set of the matching results ID from the static database D.
- The state $\sigma = 2$: the client sends the update token $UT_{\gamma 1}$ to add the document $id_{\gamma 1}$ which contains w_i to the server. The server adds $id_{\gamma 1}$ to the extra dictionary "D^+".
- The state $\sigma = 3$: to search for the keyword w_i, the client generates two search token ST_i and ST_i^+ for searching D and "D^+", respectively. The server returns the matching results ID and ID^+ from D and "D^+", respectively.
- The state $\sigma = 4$: the client sends the update token $UT_{\gamma 2}$ to add the document $id_{\gamma 2}$ which contains w_i to the server. The server adds $id_{\gamma 2}$ to the extra

dictionary "D^{+}".

- The state $\sigma \geq 4$: the client generates two search token ST_i and ST_i^{+} to search for the keyword w_i. The server returns the matching results ID and ID^{+}, where ID^{+} contains $id_{\gamma 1}$ and $id_{\gamma 2}$.

In the above scenario, at the state $\sigma = 4$, the server can reuse the ST_i^{+} to detect the presence of w_i in $id_{\gamma 2}$. The search pattern and the access pattern enables the server to reveal the search queries as well as the updated documents and when they are updated.

The dynamic SSE of [104] assumes Q as a list of the all of the issued queries, and ID the set of all identifiers to which w_i was ever added, $\mathcal{L}$ the leakage for *add* updates over the database $DB = (\mathrm{id}_i, \mathrm{W}_i)_{i=1}^{d}$. The *add* pattern can be defined as follows:

$\mathrm{AP}(w_i, Q, \mathrm{ID}) = \{(\mathrm{id}, \mathrm{ap}(\mathrm{id}, w_i, Q)) : \mathrm{id} \in \mathrm{ID}, \mathrm{ap}(\mathrm{id}, w_i, Q) \neq \emptyset\}$

Then, the leakage $\mathcal{L}$ produces outputs as follows:

- saves the state; for each search query over w_i, appends $(\sigma, srch, w_i)$ to Q and increments σ; then outputs the corresponding search pattern and *add* pattern.
- for *add* query, appends $(\sigma, add, id, W_{id})$ to Q, adds id to ID, and increments σ; then, outputs $|W_{id}|$ and the set of the search patterns, $\{sp(w_i, Q) : w_i \in W_{\mathrm{id}}\}$; for non-empty search patterns, it outputs id.

For the previously searched keywords server such as w_i learns that a keyword with search pattern $sp(w_i, Q)$ was added via the set of search patterns in the update-leakage. Moreover, the server can find the id being updated because it can search for any of its keywords.

5.4 Syntax of DSSE with geometric range query

A dataset is composed of a collection of N points; each consists of two dimensional coordinate value (x_i, y_i) for $x_i, y_i \in [0, D-1]$ where D is the size of each dimension. A dataset $\Delta = (P_i, (x_i, y_i))_{i=1}^{N}$ is a set of points' label/coordinates pairs, where P_i is the label of the point (x_i, y_i). For the sake of simplicity, our description considers integer ranges. Both of the proposed constructions consist of a pre-processing phase **Setup** run by the data owner and two protocols between the data owner and server: **Search**, and **Update**. **Setup** phase consists of following algorithms:

- **Build Binary Tree**$(t) \rightarrow (BT)$**:** The data owner inputs the tree parameter t where the dimension size $D = 2^t$ and $t+1$ is the hight of the binary tree. Then, he runs this algorithm two times to generate the binary trees, BT, for each dimension (BT_x and BT_y for x-axis and y-axis, respectively).

- **Build Inverted Index**$(t, N, \Delta) \rightarrow (ST)$**:** Given the tree parameter t, the number of the data points N, and the dataset Δ, the data owner builds the inverted index of the database and then converts each entry of it to bit strings. The data owner runs this algorithm two times to generate the set of the inverted index bit strings, ST, for each dimension (ST_x and ST_y for x-axis and y-axis, respectively).

- **Build Node Inverted Index**$(ST, BT) \rightarrow (S)$**:** This algorithm assigns each of the inverted index bit strings ST to the corresponding leaf nodes in the tree BT. Then, it generates the bit stings for the parent nodes by adding (modulo 2) the bit string of its child nodes. The data owner runs this algorithm two times to generate the node inverted index S_x and S_y for the dimensions x and y, respectively.

- **EDS Setup**$(\lambda, t, S_x, S_y, BT) \rightarrow (EDS, K_s, K_x, K_y)$**:** This algorithm is run by the data owner who inputs the security parameter λ, the tree parameter t, and

the node inverted indexes S_x and S_y. Then, It outputs the encrypted dataset EDS and the utilized keys (K_s, K_x, K_y).

Search is a protocol between a data owner and the server which consists of the following algorithms:

- **Client Search** $(q, t, C, C') \rightarrow (R)$: This algorithm is run by the data owner who inputs the desirable range query (such as $q = [a, b]_x, [a', b']_y$), the tree parameter t, and the counters[3] C and C' (for the dimensions x and y, respectively). To generates the search token Tok (which consists of the tags[4] for each dimension), the data owner runs **Build Binary Tree**(t) for each dimension and finds the minimum set of suitable nodes that cover the search query. Then, he sends the search token Tok to the server. Once, he received the corresponding results ER, he decrypts them. To find the points located in the range search, the data owner first computes the addition modulo 2 of the nodes over each dimension. Then, gets the intersection between the two dimensions by computing "AND" of the bit string of x-axis and y-axis. Finally, this algorithm outputs the search results R.

- **Server Search**$(EDS, Tok) \rightarrow (ER)$**:** Given the encrypted dataset EDS and the search token Tok, it outputs ER which contains the ciphertext of the bit strings corresponding to the issued search token.

Update : The data owner and the server perform **Update** protocol jointly which consists of the following algorithms:

For Construction-I:

- **Client Update**$(K_s, K_x, K_y, t, P_i, P_i', C, C') \rightarrow (LU_x, LU_y)$**:** To change a point $P_i' = (x_j, y_j)$ to $P_i = (x_i', y_i')$, the data owner inputs the point to be updated

[3]These counters indicate the number of updates undertaken
[4]This tags enable the server to retrieve the ciphertext without decryption

and its new coordinates values, the tree parameter t, and the counters C and C'. Then, generates the new ciphertexts for the corresponding leaf nodes in the binary tree as well as the nodes on the path from those leaf nodes to the root. Finally, this algorithm outputs two update lists LU_x and LU_y, for the x-dimension and y-dimension, respectively.

- **Server Update(**$(EDS, LU_x, LU_y) \rightarrow (Updated\ EDS)$**):** Given the current dataset EDS and the update lists LU_x and LU_y, server replaces the old ciphertext in EDS with the updated ones from LU_x and LU_y.

For Construction-II:

- **Client Update**$(K_s, K_x, K_y, t, P_i, P'_i, C, C') \rightarrow (LU_x, LU_y)$**:** To change a point $P_i = (x_i, y_i)$ to $P'_i = (x_j, y_j)$, the data owner inputs the point to be updated and its new coordinates values, the tree parameter t, and the counters C and C'. Then, he generates the update bit string S_{ux} where the position i is set to "1" and the rest are "0"s. Then, encrypts update bit string to e_u for the update of the corresponding leaf nodes on the binary tree of each dimension and send it along with the encryption of "0"s for the other leaf nodes to the server using the update lists LU_x and LU_y.

- **Server Update(**$(EDS, LU_x, LU_y) \rightarrow (Updated\ EDS)$**):** Given the current dataset EDS and the update lists LU_x and LU_y, server adds the given bit strings using additive homomorphic encryption to the bit string of the leaf nodes. Then, updates the parent nodes by homomorphic addition of the bit strings of their child nodes.

5.5 SSE schemes for geometric range search

This section, first gives an overview of our scheme using a simple example. Then, two constructions for dynamic symmetric searchable encryption are presented in details.

5.5.1 Overview

We introduce our constructions via a sample dataset as shown in Figure 5.2. It can be seen from this figure that there is a two dimensional environment which contains eight points. The first step is to build the encrypted dataset, "EDS". Once the EDS is outsourced to the server (honest-but-curious) the client would be able to perform range search over it with the assistance of binary trees. These binary trees also facilitate the update of EDS.

5.5.1.1 Setup

The data owner performs the following steps to build EDS.

1. Data owner builds two binary trees where in each of them the leaf nodes are indexed by the x-axis and y-axis values, respectively. For the considered sample dataset these trees are in the form of Figure 5.4. To avoid redundancy we considered the same size for x-axis and y-axis ($D = 8$, 0 to 7). In this figure, the red numbers represent the values on a dimension.

2. The data owner parse the points' labels for each dimension in the inverted index. Then, converts these inverted indexes to the bit strings as shown in Figure 5.3. That is, the data owner sets a fixed size bit string (mostly bigger than the number of existing nodes to enable further addition) for each value on each dimension. If the x-coordinate of a point such as P_i is equal to the value over the x-axis then the ith bit of the bit string is set to 1, and 0 otherwise. Figure 5.3 illustrates this procedure over the sample dataset. For instance, in this figure there are only two points which have $x = 1$ (points 1 and 2). Thus, the data owner sets the bits to "1" in the position 1 and 2 and the rest to "0" (highlighted).

3. The next step is to assign these bit strings to the nodes in the binary trees. Thus, the bit strings generated in the previous step will be assigned to the leaf

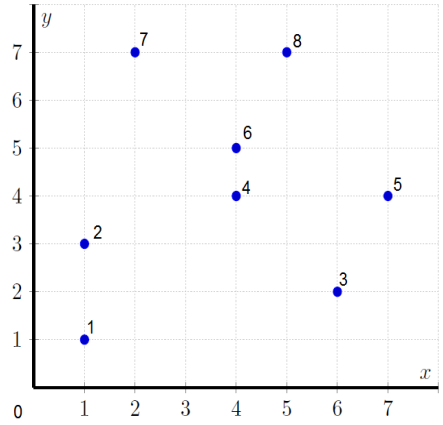


Figure 5.2: Sample spatial dataset

nodes according to the indexes in the first step. As shown in Figure 5.3, for $x = 0$ the binary string is "00000000" and from the Figure 5.4 we can see that $x = 0$ is associated with node n_7. Thus, we assign the bit string "00000000" to the node n_7 as highlighted in Figure 5.5 and we use the same technique for the rest of leaf nodes in the tree. For a parent node the assigned bit string is the addition of the bit strings of its child nodes. For example, in Figure 5.6, which represents the bit strings assigned to the nodes for y-axis, the bit string for a parent node such as n_1 is "11100000" which is the addition of its child nodes' bit strings; "10000000" and "01100000" for n_3 and n_4, respectively [5].

4. Finally, the data owner encrypts the inverted index generated in the third step, using an additive homomorphic encryption and outsource it a server.

5.5.1.2 Search

Assume that the client wants to find the points within the dashed rectangle shown in Figure 5.7. He needs to perform range query of 2 to 6 over x-dimension and 3 to 6

[5] An alternative way is to let the server to generate the bit strings of the parent nodes in the encrypted format using additive homomorphic encryption.

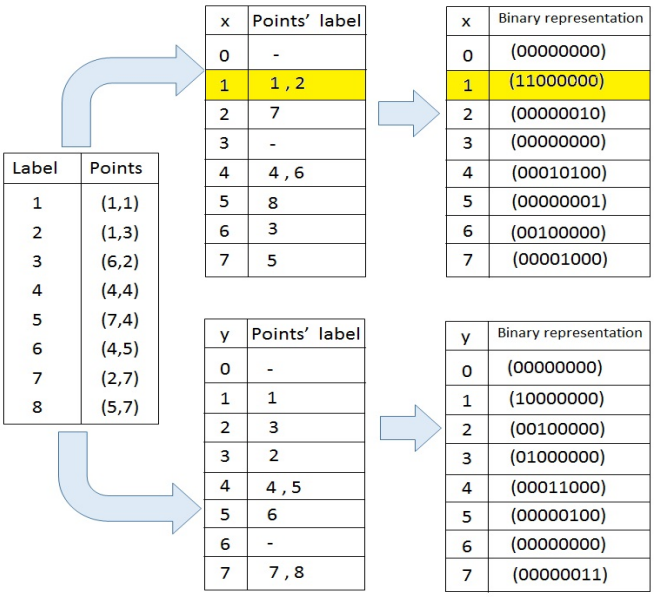


Figure 5.3: Converted dataset

over y-dimension. First, he needs to reconstruct the two binary trees (similar to the first step of setup phase) to find the minimum nodes which cover the range query. As shown in Figure 5.8, these nodes are n_4, n_5, and n_{13} for x-axis and n_{10}, n_5, and n_{13} for y-axis. Then, the data owner generates the search tokens for those nodes and sends them to the server. Upon receiving the results, the client first decrypts them and then computes the intersection of the x-axis and y-axis [6]. In the given example, the client first adds the bit strings of the highlighted nodes in the Figure 5.8, which results bit string (00110111) for x-axis and (01011100) y-axis, respectively. Then, to find the intersection, the client computes $(00110111) \wedge (01011100) = (00010100)$. That is, the points with the labels 4 and 6 are located within the issued range.

[6] An alternative solution is to allow the server to perform homomorphic operations and return the ciphertext of the result bit string to the user, however this solution will introduce additional leakage to the server. As these operations are lightweight we let the client to perform them locally.

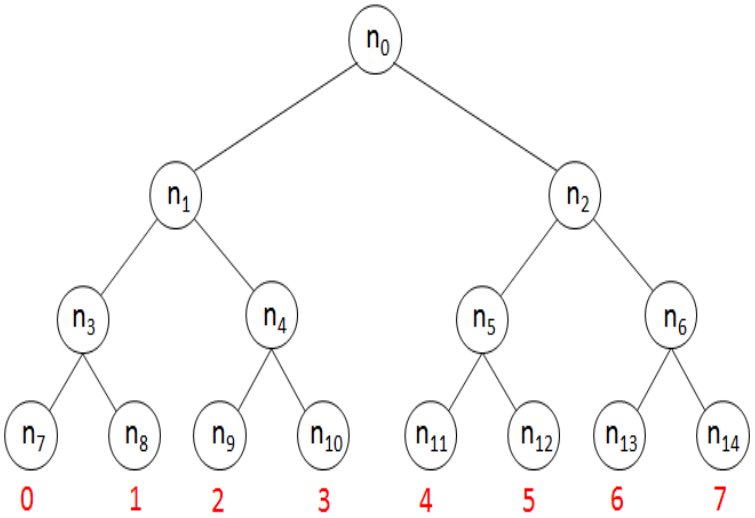


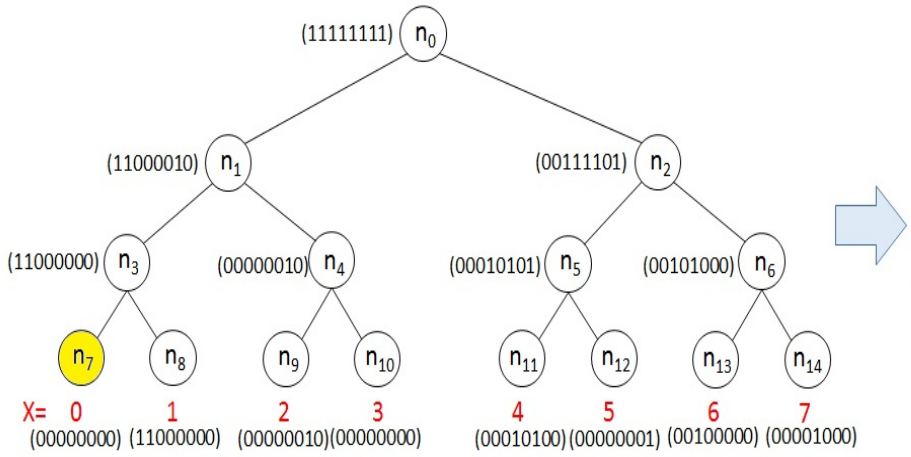
Figure 5.4: Sample indexed binary tree

5.5.1.3 Update

We proposed two constructions where the major difference comes from the update procedure. In the Construction-I, the update is mostly done on the client side and then transferred to the server for the replacement of the ciphertexts. More precisely, the client updates the path from leaf node to the root for the coordinates to be updated.

To update the encrypted dataset using the Construction-II, the following steps must be taken;

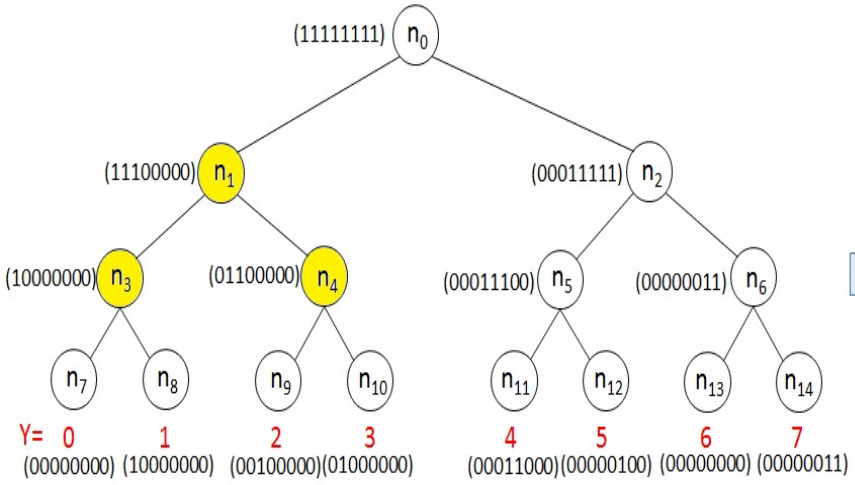
- Data owner finds the leaf nodes to be updated.
- Data owner generates the update bit string where the position of the point to be changed is equal to "1" and the rest are "0"s.
- Data owner encrypts the bit string and sends it to the server.
- Server uses homomorphic addition to update the ciphertext of the bit strings of the nodes to be updated.



Node Label	Bit string
n_0	11111111
n_1	11000010
n_2	00111101
n_3	11000000
n_4	00000010
n_5	00010101
n_6	00101000
n_7	00000000
n_8	11000000
n_9	00000010
n_{10}	00000000
n_{11}	00010100
n_{12}	00000001
n_{13}	00100000
n_{14}	00001000

Figure 5.5: Sample Node-Inverted index for x-axis

Let assume that the data owner wants to change the coordinate of the point with label "2" (P_2), from (1,3) to (2,6). Thus, the data owner first finds the leaf nodes to be updated which are n_8 and n_9 on x-dimension and n_{10} and n_{13} on the y-dimension, respectively as highlighted in Figure 5.9 and Figure 5.10. Then, he generates the update bit string where the second bit (according to the point label) is set to "1" and the rest are "0"s that is (01000000). As shown in Figure 5.9, homomorphic addition modulo 2 of the update bit string (blue) and the bit string of n_8 results in the red colour bit string that shows P_2 is deleted from n_8. To insert P_2 to n_9, the same update bit string (01000000) is added to the bit string of n_9 (00000010) using homomorphic addition modulo 2 that results in (01000010). The similar technique is used over the nodes n_{10} and n_{13} on y-axis as illustrated in Figure 5.10. To update the parent nodes the server uses homomorphic addition to add the encrypted bit



Node Label	Bit string
n_0	11111111
n_1	11100000
n_2	00011111
n_3	10000000
n_4	01100000
n_5	00011100
n_6	00000011
n_7	00000000
n_8	10000000
n_9	00100000
n_{10}	01000000
n_{11}	00011000
n_{12}	00000100
n_{13}	00000000
n_{14}	00000011

Figure 5.6: Sample Node-Inverted index for y-axis

strings of the nodes given by the data owner. This update leaks which points are updated, this leakage can be easily reduced by updating some random coordinates with the encryption of zeros.

5.5.2 The naive solution

A straightforward solution to support range search is to send all of the points to the client and let the client to find the matches. However, the communication overhead is much higher than the proposed solutions. More precisely, $(2 \times t \times N)$ bits whereas in our case it is $(2 \times N)$ bits where t indicates number of bits per coordinate and N is the number of dataset points. Thus, our scheme saves a factor of t in terms of bandwidth consumption. That is, for 32 bits t our scheme is 32 times more efficient than the trivial solution. Moreover, the Naive solution requires high post processing

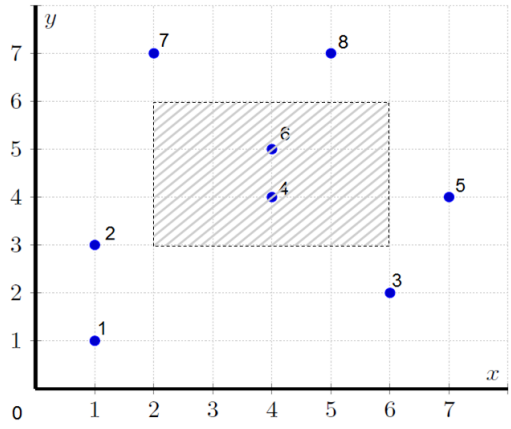


Figure 5.7: Sample Range search

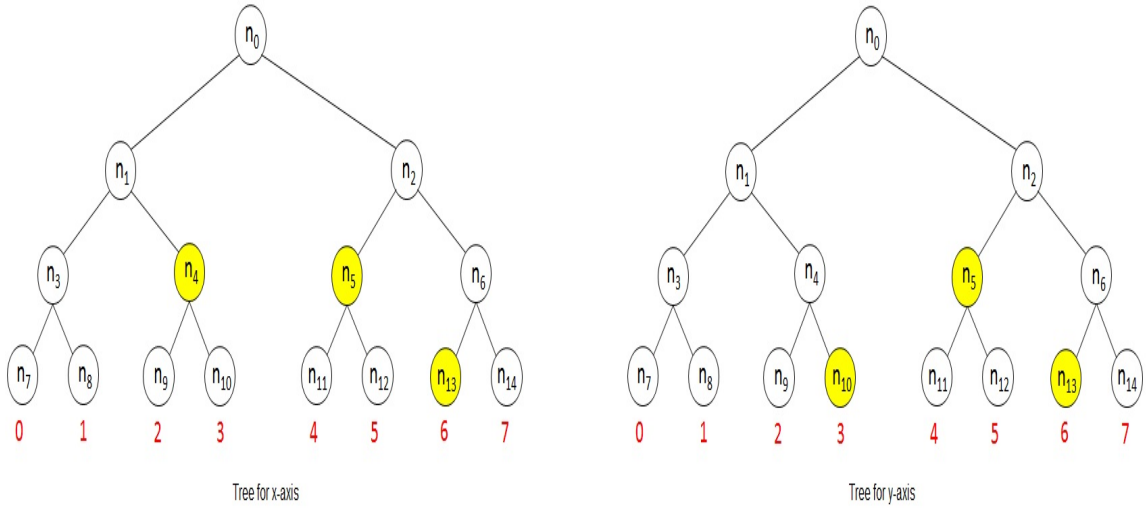


Figure 5.8: Binary tree for range search over x-axis and y-axis

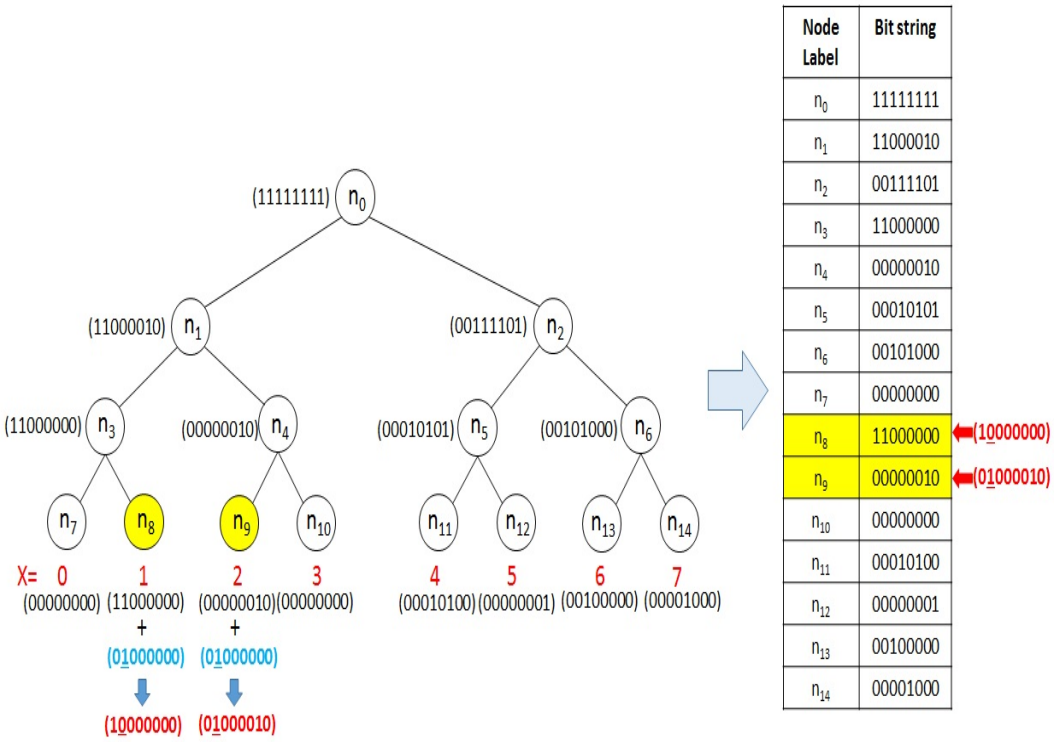


Figure 5.9: Example of update over x-axis

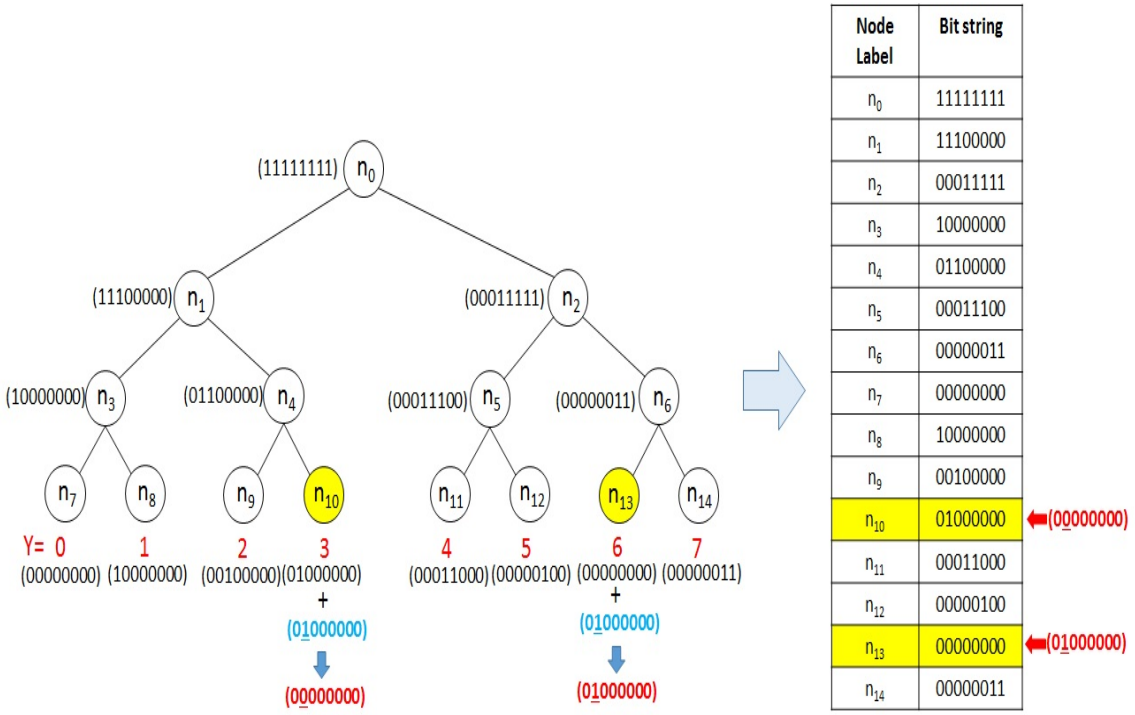


Figure 5.10: Example of update over y-axis

of the data by the client. In fact, the client has to decrypt the whole dataset and search for the points of interest for each geometric range queries.

5.5.3 Construction-I

In this section, we present the first construction for dynamic symmetric searchable encryption scheme which supports two dimensional geometric range queries. Two binary trees are required to achieve this goal; one for x-axis search and the other one for y-axis search. As mentioned in Section 5.4, our scheme consists of three main phases: Setup, Search, and Update which are explained in the following sections in details.

5.5.3.1 Setup

The first step is to generate two complete binary trees where the number of leaf nodes represents the size of the x-axis and y-axis of the environment. Then, the points which are going to be encrypted and outsourced are determined and assigned to the binary tree nodes. Finally, the encrypted node-point inverted index of the dataset would be stored on the server.

Build Binary Tree(t)

The data owner runs Algorithm 15, two times to generate two complete binary trees for x-axis and y-axis, BT_x and BT_y, respectively. In this algorithm the value t is used to determine the height of the tree and it is equal to tree level-1. In this algorithm, after defining the leaf nodes, the data owner assigns an index (j) to each of them which represents the values on the x/y-axis (similar to the red numbers in the given example in Section 5.5.1). Then, builds the parent nodes to the root, starting from the right most leaf node.

Algorithm 15 Build Binary Tree(t)

Input: t
Output: BT
1: $BT \leftarrow \{\}$
2: $j = 0$
3: **for** $i = 2^t - 1, ..., 2^{t+1} - 2$ **do**
4: Build (n_i) // Generating a leaf node with label n_i
5: Assign label $(n_i) \leftarrow i$ // Giving label to the leaf node
6: Assign index $(n_i) \leftarrow j$ // Assigning the coordinate to the node
7: Insert (n_i) to BT
8: $j \leftarrow j + 1$
9: **end for**
10: **for** $i = 2^t - 2, ..., 0$ **do**
11: Build (n_i) // Generating the intermediate nodes
12: Assign label $(n_i) \leftarrow i$
13: Assign Left-Child $(n_i) \leftarrow n_{2i+1}$
14: Assign Right-Child $(n_i) \leftarrow n_{2i+2}$
15: Insert (n_i) to BT
16: **end for**
17: **return** BT

Build Inverted Index(t, N, Δ)

Given the dataset Δ the data owner runs Algorithm 16 two times in order to generate binary inverted index ST_x and ST_y for x and y dimensions, respectively. In this algorithm it is assumed that there exists N points in the dataset[7]. For each value on the x-axis we define a N-bit string where bit i is set to "1" if the x-coordinate of the point i is equivalent to the considered value on the x-axis and "0" otherwise. Note that Algorithm 16 shows the procedure of generating the binary inverted index for the x-dimension. For the y-dimension in the line 5, $\Delta_x[j]$ should be change to $\Delta_y[j]$.

Build Node Inverted Index(ST, BT)

To build this inverted index, the data owner assigns the bit strings generated in the Algorithm 16 to the nodes in the binary tree. Algorithm 17 shows how these bit strings are first assigned to the leaf nodes and then the corresponding bit strings are generated for the parent nodes to the root. The data owner runs this algorithm

[7]To avoid ambiguity and redundancy in the algorithms we transformed the dataset $\Delta = (P_i, (x_i, y_i))_{i=1}^{N}$ to the three arrays $\Delta_P[i] = P_i$, $\Delta_x[i] = x_i$, $\Delta_y[i] = y_i$

Algorithm 16 Build Inverted Index(t, N, Δ)

Input: t, N, Δ
Output: ST // If we run this algorithm for x-axis it outputs ST_x and for y-axis it outputs ST_y
$ST \leftarrow \{\}$
for $i = 0, ..., 2^t - 1$ **do** // Indexes of a dimension
    $\sigma(i) \leftarrow \perp$ // Inverted index bit string for i value on a dimension
    for j=1,...,N **do** // Points
        if $\Delta_x[j] = i$ **then** // For y-dimension it uses $\Delta_y[j]$
            Assign string[i,j]$\leftarrow 1$
        else
            Assign string[i,j]$\leftarrow 0$
        end if
        $\sigma(i) \leftarrow \sigma(i) \parallel$ string[i,j]
    end for
    $ST \leftarrow \sigma(i)$
end for
return ST

twice in order to generate the inverted index S_x and S_y for the x-axis and y-axis, respectively.

Algorithm 17 Build Node Inverted Index(ST, BT)

Input: ST, BT
Output: S
$S \leftarrow \{\}$ // If we run this algorithm for x-axis it outputs S_x and for y-axis it outputs S_y
$j = 0$ // Indexes of a dimension
for $i = 2^t - 1, ..., 2^{t+1} - 2$ **do** // Leaf nodes
    retrieve n_i from BT
    Initialize $S(n_i) \leftarrow \emptyset$ // Initialize an empty bit string to associate it with the node n_i
    retrieve $\sigma(j)$ from ST
    Assign $S(n_i) \leftarrow \sigma(j)$
    Insert $S(n_i)$ to S
    $j \leftarrow j + 1$
end for
for $i = 2^t - 2, ..., 0$ **do** // Internal nodes
    Assign $S(n_i) \leftarrow S(n_{2i+1}) \vee S(n_{2i+2})$
    Insert $S(n_i)$ to S
end for
return S

EDS Setup$(\lambda, t, S_x, S_y, BT)$

The final stage of the system setup is to generate the encrypted dataset and outsource it to the server. To reach this, the data owner performs Algorithm 18. In this algorithm the data owner generates to lists U_x and U_y for x-dimension and y-dimension, respectively. These lists are indexed by the nodes tags (which help the

server in the search/update to retrieve the ciphertexts associated with a node) and beside each other form the encrypted dataset EDS. Note that for this construction the encryption (Enc_K which is introduced in Section 5.2.1.1) can be replaced by non-homomorphic scheme as the homomorphic properties are not used by this construction and only used in Construction-II.

Algorithm 18 Construction-I: **EDS Setup**$(\lambda, t, S_x, S_y, BT)$

Input: λ, S_x, S_y, t, BT
Output: EDS, K_s, K_x, K_y
Initialize $\mathbf{U_x} \leftarrow \emptyset$ and $\mathbf{U_y} \leftarrow \emptyset$ indexed by nodes' Tag
$K_x, K_y, K_s \xleftarrow{\$} \{0,1\}^\lambda$
$EDS \leftarrow \{\}$
for $i = 0, ..., 2^{t+1} - 2$ **do**
    retrieve n_i from BT
    $C \leftarrow 1$
    $C' \leftarrow 1$
    $K_i \leftarrow F(K_s, n_i)$ // F is a pseudo-random function (PRF)
    $TAGX_i \leftarrow F(K_x, n_i)$
    $TAGY_i \leftarrow F(K_y, n_i)$
    for $S_x(n_i) \in S_x$ **do**
        $e_i \leftarrow Enc_{K_i}(S_x(n_i), C)$
        $U_x(TAGX_i) \leftarrow e_i$
    end for
    for $S_y(n_i) \in S_y$ **do**
        $e'_i \leftarrow Enc_{K_i}(S_y(n_i), C')$
        $U_y(TAGY_i) \leftarrow e'_i$
    end for
end for
Set $EDS = (U_x, U_y)$
return EDS, K_s, K_x, K_y

5.5.3.2 Search

Algorithm 19 shows how the client generates the search token Tok for a range search which enables the server to search the encrypted dataset and return the matching results. This algorithm enables two-dimensional geometric search effectively. That is, given a range on each dimension, this algorithm first builds the binary tree of each dimension and finds the minimum nodes covering the issued ranges. Then, it generates the search token Tok which contains the sets $TAGX = \{TAGX_i\}$ and $TAGY = \{TAGY_i\}$ for the x-dimension and y-dimension, respectively. These tags enable the server to retrieve the encrypted bit strings assigned to the issued nodes

without getting any knowledge about the points that satisfy the range query. To recover the plaintext associated with each the node n_i covering the range, the client must perform the $Buildkey(n_i)$ function which outputs K as a set of keys $\{K_j = F(K_s, n_j)\}$ of the nodes n_js in the subtree rooted by n_i.

Algorithm 19 Construction-I: Search

Client Search (q, t, C, C')
Input: $q = ([a,b]_x, [a',b']_y), t, C, C'$
output: R

1: $BT_x \leftarrow$ **Build Binary Tree**(t)
2: $XR \leftarrow$ find minimum nodes covering $[a,b]_x$ in BT_x
3: $TAGX \leftarrow \{\}$
4: **for** $n_i \in XR$ **do**
5: $\quad TAGX_i \leftarrow F(K_x, n_i)$
6: $\quad TAGX \leftarrow \{TAGX_i\} \cup TAGX$
7: **end for**
8: $BT_y \leftarrow$ **Build Binary Tree**(t)
9: $YR \leftarrow$ find minimum nodes covering $[a',b']_y$ in BT_y
10: $TAGY \leftarrow \{\}$
11: **for** $n_i \in YR$ **do**
12: $\quad TAGY_i \leftarrow F(K_y, n_i)$
13: $\quad TAGY \leftarrow \{TAGY_i\} \cup TAGY$
14: **end for**
15: Sends $Tok = (TAGX, TAGY)$ to the server
16: $ER \leftarrow$ **Server Search** (EDS, Tok)
17: **for** $e_i \in ER$ **do**
18: $\quad K \leftarrow BuildKey(n_i)$
19: $\quad (S_x(n_i)) \leftarrow Dec_K(e_i, C)$
20: $\quad S_x \leftarrow S_x \vee S_x(n_i)$
21: **end for**
22: **for** $e'_i \in ER$ **do**
23: $\quad K \leftarrow BuildKey(n_i)$
24: $\quad (S_y(n_i)) \leftarrow Dec_K(e'_i, C')$
25: $\quad S_y \leftarrow S_y \vee S_y(n_i)$
26: **end for**
27: $R \leftarrow (S_x \wedge S_y)$
28: **return** R

Server Search (EDS, Tok)
Input: EDS, Tok
output: ER

1: $ER \leftarrow \{\}$
2: **for** $TAGX_i \in TAGX$ **do**
3: $\quad e_i \leftarrow U_x(TAGX_i)$
4: $\quad ER \leftarrow \{e_i\} \cup ER$
5: **end for**
6: **for** $TAGY_i \in TAGY$ **do**
7: $\quad e'_i \leftarrow U_y(TAGY_i)$
8: $\quad ER \leftarrow \{e'_i\} \cup ER$
9: **end for**
10: **return** ER

5.5.3.3 Update

Algorithm 20 shows the update procedure. That is, the client updates the leaf nodes and the corresponding intermediate nodes on the path to the root.

Note that, the **Update-y**$(K_s, K_y, t, P_i, P'_i, C')$ function is similar to **Update-x**$(K_s, K_x, t, P_i, P'_i, C)$ function, thus to avoid redundancy we did not present it in Algorithm 20. Given the outputs of **Client Update**$(K_s, K_x, K_y, t, P_i, P'_i, C, C')$, the server performs **Server Update**$(EDS = (U_x, U_y), LU_x, LU_y)$ in order to update the corresponding entries of the encrypted dataset.

Algorithm 20 Construction-I: **Update**

Client Update$(K_s, K_x, K_y, t, P_i, P'_i, C, C')$
Input: $t, P_i = (x_i, y_i), P'_i = (x_j, y_j), C, C'$
output: LU_x, LU_y

$LU_x \leftarrow \{\}$
$LU_y \leftarrow \{\}$
if $x_i \neq x_j$ **then**
  $C \leftarrow C + 1$
  $LU_x \leftarrow$ **Update-x**$(K_s, K_x, t, P_i, P'_i, C)$
  return LU_x
end if
if $y_i \neq y_j$ **then**
  $C' \leftarrow C' + 1$
  $LU_y \leftarrow$ **Update-y**$(K_s, K_y, t, P_i, P'_i, C')$
  return LU_y
end if

Update-x$(K_s, K_x, t, P_i, P'_i, C)$
Input: $K_s, K_x, t, P_i = (x_i, y_i), P'_i = (x_j, y_j), C, C'$
Output: LU_x

$BT_x \leftarrow$ Build Tree(t)
$LN_x \leftarrow$ Find leaf nodes to be updated in BT_x
$S_{ux} \leftarrow$ Update bit string
for $n_\alpha \in LN_x$ **do**
  $S_x(n_\alpha) \leftarrow S_{ux} \vee S_x(n_\alpha)$ // Update the bit string associated with the leaf node n_α
  $TAGX_\alpha \leftarrow F(K_x, n_\alpha)$
  $K_\alpha \leftarrow F(K_s, n_\alpha)$
  $e_u \leftarrow Enc_{K_\alpha}(S_x(n_\alpha), C)$ // Computing the ciphertext for the updated bit string
  $LU_x \leftarrow LU_x \cup \{(TAGX_\alpha, e_u)\}$
  if $\alpha \mod 2 = 0$ **then** // Checking the leaf node n_α is the right child or left child
    $\beta \leftarrow (\frac{\alpha}{2} - 1)$
    $S_x(n_\beta) \leftarrow S_x(n_\alpha) \vee S_x(n_{\alpha-1})$ // updating the parent node
  else
    $\beta \leftarrow (\frac{\alpha-1}{2})$
    $S_x(n_\beta) \leftarrow S_x(n_\alpha) \vee S_x(n_{\alpha+1})$ // updating the parent node
  end if
  $TAGX_\beta \leftarrow F(K_x, n_\beta)$
  $K_\beta \leftarrow F(K_s, n_\beta)$
  $e_u \leftarrow Enc_{K_\beta}(S_x(n_\beta), C)$ // Computing the ciphertext for the updated bit string
  $LU_x \leftarrow LU_x \cup \{(TAGX_\beta, e_u)\}$
  while $\beta \neq 0$ **do** // updating the nodes on the path to the root
    if $\beta \mod 2 = 0$ **then**
      $S_x(n_{\frac{\beta}{2}-1}) \leftarrow S_x(n_\beta) \vee S_x(n_{\beta-1})$
      $\beta \leftarrow \frac{\beta}{2} - 1$
    else
      $S_x(n_{\frac{\beta-1}{2}}) \leftarrow S_x(n_\beta) \vee S_x(n_{\beta+1})$
      $\beta \leftarrow (\frac{\beta-1}{2})$
    end if
    $TAGX_\beta \leftarrow F(K_x, n_\beta)$
    $K_\beta \leftarrow F(K_s, n_\beta)$
    $e_u \leftarrow Enc_{K_\beta}(S_x(n_\beta), C)$
    $LU_x \leftarrow LU_x \cup \{(TAGX_\beta, e_u)\}$
  end while
end for
return LU_x

Server Update$(EDS = (U_x, U_y), LU_x, LU_y)$
Input: $EDS = (U_x, U_y), LU_x, LU_y$
Output: Updated EDS

for $(TAGX_i, e_u) \in LU_x$ **do**
  $e_i \leftarrow U_x(TAGX_i)$
  replace e_i with e_u in the corresponding U_x entry
end for
for $(TAGY_i, e'_u) \in LU_y$ **do**
  $e'_i \leftarrow U_y(TAGY_i)$
  replace e'_i with e'_u in the corresponding U_x entry
end for

5.5.4 Construction-II

This section presents the second DSSE construction for geometric range search. The **Setup** phase of this construction is similar to **Setup** of Construction-I. That is, **Build Binary Tree**(t), **Build Inverted Index**(t, N, Δ), and **Build Node Inverted Index**(ST, BT) are exactly the same as Construction-I whereas **EDS**

Setup$(\lambda, t, S_x, S_y, BT)$ is different as it does not require TAG generation. **EDS Setup**$(\lambda, t, S_x, S_y, BT)$ is presented in Algorithm 21.

Algorithm 21 Construction-II: **EDS Setup**$(\lambda, t, S_x, S_y, BT)$

Input: λ, S_x, S_y, t, BT
Output: EDS, K_s, K_x, K_y
Initialize $\mathbf{U_x} \leftarrow \{\}$ and $\mathbf{U_y} \leftarrow \{\}$ indexed by nodes
$K_x, K_y, K_s \xleftarrow{\$} \{0,1\}^\lambda$
$EDS \leftarrow \{\}$
for $i = 0, ..., 2^{t+1} - 2$ **do**
    retrieve n_i from BT
    $C \leftarrow 1$
    $C' \leftarrow 1$
    $K_i \leftarrow F(K_s, n_i)$
    for $S_x(n_i) \in S_x$ **do**
        $e_i \leftarrow Enc_{K_i}(S_x(n_i), C)$
        $U_x \leftarrow \{(n_i, e_i)\}$
    end for
    for $S_y(n_i) \in S_y$ **do**
        $e' \leftarrow Enc_{K_i}(S_y(n_i), C')$
        $U_y \leftarrow \{(n_i, e'_i)\}$
    end for
end for
Set $EDS = (U_x, U_y)$
return EDS, K_s, K_x, K_y

Search. Algorithm 22 shows how the client generates the search token Tok for a range search which enables the server to search the encrypted dataset and return the matching results.

Update. To update the encrypted dataset we apply the homomorphic addition introduced in Section 5.2 (Equation 5.3). More precisely, if we want to update a point from $P_i = (x_i, y_i)$ to $P'_i = (x_j, y_j)$. First, we need to check which dimension(s) is changed and then generate an update bit sting where the position i is set to "1" and the rest to "0". By building the binary tree, the nodes to be updated will be determined. Then, using the homomorphic addition the server updates the bit string of the nodes according to Algorithm 23.

Algorithm 22 Construction-II: Search

Client Search (q, t, C, C')
Input: $q = ([a, b]_x, [a', b']_y), t, C, C'$
output: R

$XR \leftarrow \{\}$
$YR \leftarrow \{\}$
$BT_x \leftarrow$ **Build Binary Tree**(t)
$XR \leftarrow$ find minimum nodes covering $[a, b]_x$ in BT_x
$BT_y \leftarrow$ **Build Binary Tree**(t)
$YR \leftarrow$ find minimum nodes covering $[a', b']_y$ in BT_y
Sends $Tok = (XR, YR)$ to the server
$ER \leftarrow$ **Server Search** (EDS, Tok)
for $e_i \in ER$ **do**
    $K \leftarrow BuildKey(n_i)$
    $(S_x(n_i)) \leftarrow Dec_K(e_i, C)$
    $S_x \leftarrow S_x \vee S_x(n_i)$
end for
for $e'_i \in ER$ **do**
    $K \leftarrow BuildKey(n_i)$
    $(S_y(n_i)) \leftarrow Dec_K(e'_i, C')$
    $S_y \leftarrow S_y \vee S_y(n_i)$
end for
$R \leftarrow (S_x \wedge S_y)$
return R

Server Search (EDS, Tok)
Input: EDS, Tok
output: ER

$ER \leftarrow \{\}$
for $n_i \in XR$ **do**
    $e_i \leftarrow U_x(n_i)$
    $ER \leftarrow \{e_i\} \cup ER$
end for
for $n_i \in YR$ **do**
    $e'_i \leftarrow U_y(n_i)$
    $ER \leftarrow \{e'_i\} \cup ER$
end for
return ER

5.6 Comparison

Table 5.1 illustrates the summary of the security comparisons between our proposed constructions and the current related works. None of the current SSE scheme supporting geometric range search can avoid access pattern leakage. Hence, they can not meet the content privacy security requirement. Furthermore, only FastGeo [105] and EGRQ [106] support the dynamic setting but none of them considered forward and backward privacy. Moreover, EGRQ [106] also has OPE leakage which our constructions do not have. The Naive solution can support dynamic setting and achieve all of the security requirements if for each update it download, re-encrypt, and upload the whole dataset. However, this results in high communication overhead to the system and the computational overhead to the client.

Table 5.2 presents a summary of performance comparisons in terms of search overhead, update overhead, and storage size among the related works and the proposed constructions. To have a fair comparison, we compare the related works for

Algorithm 23 Construction-II: **Update**

Client Update$(K_s, K_x, K_y, t, P_i, P_i', C, C')$
Input: $t, P_i = (x_i, y_i), P_i' = (x_j, y_j), C, C'$
Output: LU_x, LU_y

$LU_x \leftarrow \{\}$
$LU_y \leftarrow \{\}$
if $x_i \neq x_j$ **then** // Checking the changes on x-dimension
  $C \leftarrow C + 1$
  $LU_x \leftarrow$ Update-x(P_i)
  return LU_x
end if
if $y_i \neq y_j$ **then** // Checking the changes on y-dimension
  $C' \leftarrow C' + 1$
  $LU_y \leftarrow$ Update-y(P_i)
  return LU_y
end if

Update-x:$(K_s, K_x, t, P_i, P_i', C)$
Input: $K_s, K_x, t, P_i = (x_i, y_i), P_i' = (x_j, y_j), C$
Output: LU_x

$BT_x \leftarrow$ Build Tree(t)
$LN_x \leftarrow$ Find leaf nodes to be updated in BT_x
$S_{ux} \leftarrow$ Update bit string
for $n_\alpha \in LN_x$ **do** // Update of the modified leaf nodes
  $K_\alpha \leftarrow F(K_s, n_\alpha)$
  $e_u \leftarrow Enc_{K_\alpha}(S_{ux}, C)$
  $LU_x \leftarrow LU_x \cup \{(n_\alpha, e_u)\}$
end for
for $n_\alpha \notin LN_x$ **do** // Update of the other leaf nodes
  $K_\alpha \leftarrow F(K_s, n_\alpha)$
  $e_u \leftarrow Enc_{K_\alpha}(0's, C)$
  $LU_x \leftarrow LU_x \cup \{(n_\alpha, e_u)\}$
end for
return LU_x

Update-y:$(K_s, K_y, t, P_i, P_i', C')$
Input: $K_s, K_y, t, P_i = (x_i, y_i), P_i' = (x_j, y_j), C'$
Output: LU_y

$BT_y \leftarrow$ Build Tree(t)
$LN_y \leftarrow$ Find leaf nodes to be updated in BT_y
$S_{uy} \leftarrow$ Update bit string
for $n_\beta \in LN_y$ **do** // Update of the modified leaf nodes
  $K_\beta \leftarrow F(K_s, n_\beta)$
  $e_u' \leftarrow Enc_{K_\beta}(S_{uy}, C')$
  $LU_y \leftarrow LU_y \cup \{(n_\beta, e_u')\}$
end for
for $n_\beta \in LN_y$ **do** // Update of the other leaf nodes
  $K \leftarrow F(K_s, n_\beta)$
  $e_u' \leftarrow Enc_K(0's, C')$
  $LU_y \leftarrow LU_y \cup \{(n_\beta, e_u')\}$
end for
return LU_y

Server Update:$(EDS = (U_x, U_y), LU_x, LU_y)$
Input: $EDS = (U_x, U_y), LU_x, LU_y$
Output: Updated EDS

$BT_x \leftarrow$ Build Tree(t)
$BT_y \leftarrow$ Build Tree(t)
for $(n_\alpha, e_u) \in LU_x$ **do**
  Retrieve e_α
  $e_\alpha \leftarrow e_u \oplus e_\alpha$
  replace e_α in the corresponding U_x entry
end for
for $\alpha = 2^t - 1, ..., 0$ **do**
  $e_\alpha \leftarrow e_{2\alpha+1} \oplus e_{2\alpha+2}$
  replace e_α in the corresponding U_x entry
end for
for $(n_\beta, e_u') \in LU_y$ **do**
  Retrieve e_β'
  $e_\beta' \leftarrow e_u \oplus e_\beta'$
  replace e_β' in the corresponding U_y entry
end for
for $\beta = 2^t - 1, ..., 0$ **do**
  $e_\beta' \leftarrow e_{2\beta+1}' \oplus e_{2\beta+2}'$
  replace e_β' in the corresponding U_y entry
end for

the same shape of the range search, circle with radius R. In both of our proposed constructions, if we want to perform the circle range search, we can find the square that contains the circle where the side length of the square is $2R$.

The search time of GRSE [50] is linear to the size of Bloom filter as well as the

Table 5.1: Security comparison

Scheme / Security requirement	Construction-I	Construction-II	Naive	[50]	[107]	[105]	[106]
Forward privacy	✗	✓	✓	NA	NA	?	✗
Backward privacy	Type-II	✗	Type-I	NA	NA	?	✗
Content privacy	✓	✓	✓	✗	✗	✗	✗
Dynamic	✓	✓	✓	✗	✗	✓	✓
Cryptographic primitive	SE/ASHE	ASHE	SE	PBPKE	PBPKE	PBPKE	OPE

SE: Symmetric Encryption; ASHE: Additive Symmetric Homomorphic Encryption; PBPKE: Pairing-based Public Key Encryption; OPE: Order Preserving Encryption ?: Unknown

number of data points. GRSE returns only the identifier of the matching results as the output of the search function. The authors suggested to use R-tree to enhance the search time, however as they noted this modification leads to higher false positives results. Similarly, R-tree is used in EGRQ [106] to achieve high performance. Both FastGeo and GRSE generate a significantly large vector for enumerating all possible search points which in turn results to larger computational overhead compared with EGRQ. The search complexity of EGRQ is directly related to N_{deg} which is the highest degree of a term in the utilized fitted polynomial (the increase in N_{deg} results to more accuracy in the results). However, it is not clear how N_{deg} should scale with R and t. Although our constructions and GRSE have linear search complexity regarding the number of data points, our constructions are much faster. That is, our constructions do not require any additional computations and only read bits whereas GRSE must perform a pairing operation for each database point due to the use of SSW[8]. The server search complexity of FastGeo [105] is linear to the resolution of the grid of coordinates (2^t) but sublinear to the number of data points (N). The server search complexity of our constructions is proportional to at most ($\log 2R$) which is equal to t, hence overall of $O(tN)$. Therefore, our constructions are suitable for smaller N but larger t.

Wang et al. [107] proposed two constructions for circular range search called CRSE-I and CRSE-II. The search complexity of CRSE-I is $O(\alpha^c)$ per data record where α represents the length of data (for two dimensional search $\alpha = 4$) and c is a finite number of concentric circles is sufficient to cover all the possible points. In CRSE-I the size of the ciphertext drastically increases with radius R whereas in

[8]SSW: Shen-Shi-Waters Encryption

CRSE-II it is independent of R. As mentioned by the authors this construction is impractical for circular range queries with large radiuse as c increases with $O(R^2)$. For CRSE-II, the search complexity is $O(\alpha c)$ per ciphertext. Therefor, CRSE-II has the overall search complexity of $O(R^2 N)$ for a dataset with N data points. In contrast, the search complexity of our constructions are scale only logarithmic in R.

In both FastGeo and our scheme the search token size is different for variant queries and it changes according to queried range. In FastGeo [105], client has to enumerate all of the possible points in the range and then the server checks whether a point is inside the considered range or not. Thus, the length of the utilized Bloom filter should be long enough to contain all of the possible grid points which is 2^{2t}. Moreover, for each of the possible grid points client must compute some exponentiations (T_{exp}) due to use of SSW (this computation is also required in [107] and [50]).

In comparison with Naive solution the client search complexity our construction is lighter at the ratio of $\frac{\log R}{t}$. That is, in Naive solution the client has to decrypt the whole dataset in order to find the matching data points whereas in our constructions the user only retrieves the nodes covering the range search ($O(\log R)$ covering nodes and each of them are N bits). In the worst case they would have equal client search complexity if the range search is as large as the size of the dimensions, however this is mostly not the case in the real range searches. For instance, a client usually looks for the points of interest which are close to him or have a specific distance from him and not all of the points in an environment (hence, $\log R \ll t$). On the other hand, our constructions require a bigger storage, however it is not an issue when using the cloud storage.

Although the update complexity of our Construction-II and FastGeo on the server side is higher than Naive solution, there is no overhead for the client and the communication overhead per update is lower. That is, for each update in Naive solution the whole dataset should be downloaded, re-encrypted, and uploaded to the server to achieve the security requirements given in Table 5.1. Whereas in our Construction-I

Table 5.2: Performance comparison

Performance \ Scheme	Construction-I	Construction-II	Naive	[50]	[107]	[105]	[106]
Search Comp. (Server)	$O(\log(2R)N)$	$O(\log(2R)N)$	$O(1)$	$O(mN)$	$O(R^2N)$	$O(2^t)$	$O(Nt^2N_{deg}^3)$
Search Comp. (Client)	$O((\log R)N)$	$O((\log R)N)$	$O(tN)$	$O(2^{2t}T_{exp})$	$O(RT_{exp})$	$O(R^2 2^t T_{exp})$	$O(N_{deg}^4 t^2)$
Length (Search result)	$O(\log(2R)N)$	$O(\log(2R)N)$	$O(tN)$	$O(n)$	$O(R^2N)$	$O(2^t)$	$O(n)$
Update Comp. (Server)	$O(1)$	$O(2^tN)$	$O(1)$	NA	NA	$O(2^tN)$	$O(1)$
Update Comp. (Client)	$O(ktN)$	$O(1)$	$O(tN)$	NA	NA	$O(1)$	$O(kt)$
Storage size (EDB)	$O(2^tN)$	$O(2^tN)$	$O(tN)$	$O(mN)$	$O(N)$	$O(2^tN)$	$O(N)$

R: Radius of the circle query; t: Bit length of coordinates (x and y); N: Number of the data points in the dataset; N_{deg}: highest degree of a term in the used fitted polynomial
m: size of Bloom filter; n: number of the matching result; k: number of update point; T_{exp} exponentiation time in token generation of SSW

only the new ciphertext of the affected nodes should be uploaded (hence no complexity for the server) and for our Construction-II there is no need to download any information and only the update bit strings are delivered to the server. EGRQ is very similar to the Naive solution, the as the updates are done on the client side.

For both of our proposed constructions the EDS storage size is proportional to the dimension size. Thus, for the N data points, the EDB storage size is $2 \times N \times (2^{t+1}-1)$ bits where $(2^{t+1}-1)$ indicates the number of the nodes in the binary tree for dimension of size 2^t. The storage size in all of the related works is linear to the number of data points.

5.7 Security analysis

5.7.1 Range search leakage functions

This section presents the leakage function of our work, denoted by $\mathcal{L}_\Sigma$. That is, the information which the server is allowed to learn about the dataset and the queries. This leakage function corresponds to the Setup, Search and Update of the dataset; $\mathcal{L}_\Sigma = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$. Our constructions have different leakage functions that use various combination of the following leakages (the theorem statement in Section 5.7 indicates the leakages for each of the proposed constructions).

- *Range Search pattern (sp(r))*: Similar to most of the existing searchable en-

cryption schemes, our scheme leaks the *search pattern*. In general, the information about whether any two queries are generated from the same range search or not. Each range query r is decomposed into a union of $n(r)$ covering ranges, $Decomp(r) = \bigcup_{i=1}^{n(r)} I_i$. The range search pattern $sp(r)$ reveals the intervals I_j's in $Decomp(r)$ which match I_j's in $Decomp(r')$ for the previous search queries on r'.

- *Update path ($Path(c_i)$)*: Is the path in the binary tree from the root to the modified leaf node.
- *Dimension size (ds)*: the upper bound of each dimension is known by the server as the server knows the size of the complete binary tree.
- *Dataset size (Δs)*: due to the use of the fixed length bit string for representing the dataset, the server learns the maximum possible number of points in the considered environment.
- *Number of updates (Nu)*: The server learns how many updates are performed on the dataset but he can not recognize the type of the update (insertion, deletion, modification) and also on what point is updated. Therefore, the update does not leak any information about the dataset and the search queries.

It is worth to note that unlike other DSSE schemes supporting the geometric range search, the design and also the application of our constructions do not require the retrieval of the matching documents at the end of the search protocol. Therefore, our scheme does not leak two common leakages from range queries: Access pattern and communication volume.

5.7.2 Construction-I

In this construction, each ciphertext includes all of the points labels as they were presented in the form of a bit string. For each update (all update operation are acting

the same), the old ciphertext is getting replaced by a new one. Therefore, a) it is not possible to distinguish which bit of the plaintext is updated, b) search queries do not leak matching entries after they have been deleted. Thus, this construction satisfies the content privacy and backward security as discussed below.

Theorem 9. *Let F be a pseudo random function and Enc be a secure additive homomorphic symmetric encryption (ASHE), then Construction-I is $\mathcal{L}$-adaptively-secure with the following leakage functions:*

$$\mathcal{L}^{Stp}(\Delta) = \Delta s$$

$$\mathcal{L}^{Srch}(r) = sp(r)$$

$$\mathcal{L}^{Updt}(op, (c_i, P_i)) = Path(c_i).$$

Proof. The proof proceeds using a hybrid argument, by game hopping, starting from the real-world game $REAL_{\mathcal{A}}(\lambda)$.

Game G_0: This game is exactly the same as the real world security game REAL. In this game, for the range query "q" (for the sake of simplicity we consider one dimension here as the procedure is the same) client first finds the intervals $(I_1, ..., I_c)$ covering the "q". Then, he maps each I_i to a node "n_i" on the binary tree and computes the corresponding search token $TAG_i = F(K, n_i)$. Finally, the token that is sent to the server is $Tok = (TAG_1, ..., TAG_c)$ that cover the range.

$$\mathbb{P}\left[REAL_{\mathcal{A}}(\lambda) = 1\right] = \mathbb{P}\left[G_0 = 1\right]$$

Game G_1: In this game, we pick random values instead of the output of the pseudo random function F for a given node in generating a search token and store it in a table "T" so it can be reused whenever the node is queried. The advantage of the adversary in distinguishing between G_0 and G_1 is exactly the same as advantage for the PRF F. Thus, we can build a reduction B_1 which is able to distinguish between

F and a truly random function.

$$\mathbb{P}\left[G_0 = 1\right] - \mathbb{P}\left[G_1 = 1\right] \leq \mathrm{Adv}^{\mathrm{prf}}_{F,B_1}(\lambda)$$

Game G_2: To delete(/insert) a point from the bit string associated to the corresponding leaf nodes on the binary tree, this game replaces all the ciphertexts on the path from the leaf node to the root with encrypting all 0's strings. For update token, it uses the leakage to learn the number of nodes to be updated. The adversary $\mathcal{A}$ can not distinguish the real ciphertext from the ciphertext of 0's. That is, the advantage of the adversary in distinguishing between G_1 and G_2 is exactly the same as the advantage for ASHE. We can build a reduction B_2 such that

$$\mathbb{P}\left[G_2 = 1\right] - \mathbb{P}\left[G_1 = 1\right] \leq \mathrm{Adv}^{ASHE}_{\Sigma,B_2}(\lambda)$$

Simulator. We can simulate the `IDEAL` the same as Game G_2. The $\mathcal{L}^{Stp}(\Delta) = \Delta s$ is required to determine the size of the ciphertext. $sp(r)$ is the only leakage that we need to simulate the search in G_2. For the update simulator $\mathcal{S}$ works the same as G_2 without knowing the content (point identifiers). The simulator only uses the $Path(c_i)$ in the update query and not the point identifier. Therefore, it can simulate the attacker's view using only $\mathcal{L}^{Updt} = Path(c_i)$.

$$\mathbb{P}\left[REAL_{\mathcal{A}}(\lambda) = 1\right] - \mathbb{P}\left[\mathcal{S}_{\mathcal{A}}(\lambda) = 1\right] \leq \mathbb{P}\left[G_0 = 1\right] - \mathbb{P}\left[G_2 = 1\right]$$
$$\leq \mathrm{Adv}^{\mathrm{prf}}_{F,B_1}(\lambda) + \mathrm{Adv}^{ASHE}_{\Sigma,B_2}(\lambda)$$

□

Construction-I is backward secure.

Construction-I is content private.

It is worth to note that both corollaries follows directly from Theorem 9 because the search leakage does not include $TimeDB(w)$ or $Updates(w)$.

5.7.3 Construction-II

The security of this construction relies on the security of the utilized additive homomorphic symmetric encryption (ASHE) scheme.

Theorem 10. *(forward privacy). Let F be a pseudo random function and ASHE be a secure additive homomorphic encryption, we define $\mathcal{L}_{FS} = (\mathcal{L}_{FS}^{Stp}, \mathcal{L}_{FS}^{Srch}, \mathcal{L}_{FS}^{Updt})$ as follows.*

$\mathcal{L}_{FS}^{Stp}(\Delta) = (\Delta s, ds)$

$\mathcal{L}_{FS}^{Srch}((q_1, ..., q_i), \Delta) = q_i$

$\mathcal{L}_{FS}^{Updt} = \perp$

Construction-II is $\mathcal{L}_{FS}$-adaptively-forward-secure.

Proof. The proof is conducted via a sequence of Games. The first game G_0 has the same distribution as the real game of Construction-II, $REAL_A^{FS}(\lambda)$. The last game G_3 is designed to be simulated by an efficient simulator $\mathcal{S}$. By showing that the distributions of these games (G_0 and G_1, G_1 ,G_2, and G_3) are indistinguishable to each other, we can see that the simulator $\mathcal{S}$ meets the requirements of the security definition, therefore completing the proof of the theorem.

Game G_0: This game is exactly the same as the real world security game $REAL_A^{FS}(\lambda)$.

$$\mathbb{P}\left[REAL_A^{FS}(\lambda) = 1\right] = \mathbb{P}\left[G_0 = 1\right]$$

Game G_1: In this game, we replace the dataset entries with the encryption of "0"s. The adversarial distinguishing advantage between Game G_0 and Game G_1 is the same as the adversarial advantage in breaking the $ASHE$. That is, if an adversary A can distinguish G_0 from G_1, we can build an adversary B_1 which can break ASHE.

$$\mathbb{P}\left[G_0 = 1\right] - \mathbb{P}\left[G_1 = 1\right] \leq \mathrm{Adv}_{\Sigma, B_1}^{\mathrm{ASHE}}(\lambda)$$

Game G_2: In this game, we pick random values instead of the output of the pseudo random function F in generating the encryption key used in encryption of the update bit string.

The advantage of the adversary in distinguishing between G_1 and G_2 is exactly the same as advantage for the PRF F. Thus, we can build a reduction B_2 which is able to distinguish between F and a truly random function.

$$\mathbb{P}\left[G_1 = 1\right] - \mathbb{P}\left[G_2 = 1\right] \leq \mathrm{Adv}^{\mathrm{prf}}_{F,B_2}(\lambda)$$

Game G_3: This game replaces the e_u part of the update token with ASHE of "0"s. That is, the advantage of the adversary in distinguishing between G_2 and G_3 is exactly the same as the advantage for $ASHE$. We can build a reduction B_3 such that

$$\mathbb{P}\left[G_3 = 1\right] - \mathbb{P}\left[G_2 = 1\right] \leq \mathrm{Adv}^{ASHE}_{\Sigma,B_3}(\lambda)$$

□

Simulator. We just require the size of the dataset Δs to learn the size of the ciphertext in order to replace it with the encryption of "0"s. Moreover, the search query "q" can be simulated directly from $\mathcal{L}^{Srch}_{FS}$.

$\mathbb{P}\left[REAL_{\mathcal{A}}(\lambda) = 1\right] - \mathbb{P}\left[\mathcal{S}_{\mathcal{A}}(\lambda) = 1\right] \leq \mathrm{Adv}^{\mathrm{ASHE}}_{\Sigma,B_1}(\lambda) + \mathrm{Adv}^{\mathrm{prf}}_{F,B_2}(\lambda) + \mathrm{Adv}^{ASHE}_{\Sigma,B_3}(\lambda)$

Construction-II is content private.

5.8 Summary

This chapter presented two dynamic symmetric searchable encryption schemes for geometric range search. Our constructions are the first to provide forward/backward

privacy in the context of SSE-based schemes supporting geometric range search. In addition, we defined a security notion called content privacy. This security notion captures the leakages that are critical in the context of geometric range search but not considered by forward/backward privacy. Content privacy eliminates the leakage on the updated points of the database during both search and update. Due to the inherent leakages associated with range queries, none of the existing related works can support content privacy whereas the design of our constructions avoids such leakages. When compared to the state-of-the-art schemes our constructions provide a higher level of security and practical efficiency supported by our experimental results.

Chapter 6

Conclusions and Future Work

This chapter summarizes the outcomes of this dissertation and presents recommendations for future work.

6.1 Summary

In this dissertation we addressed three problems of SSE. First, we gave an SSE scheme which supports multi-reader setting. Our scheme preserves the privacy of database content as well as user privacy (in terms of the search query). Moreover, two approaches for user revocation and a solution for fast and efficient re-encryption (update) of the database using the new key are provided. Second, we proposed a generic solution for multi-keyword ranked search over the encrypted cloud data. The most significant attribute of this solution is that it is not vulnerable against the common attacks that take advantage of OPE leakage while only a single cloud server is used. Lastly, we proposed two forward/backward secure dynamic SSE which support geometric range search. Moreover, we introduced a security notion called content privacy which captures the leakages that are critical in the context of geometric range search but not considered by forward/backward security.

6.2 Future Work

A further study could provide a multi-user SSE scheme which supports the dynamic setting. That is, despite of all of the nice features that the first scheme presented in this dissertation provides, it is designed in the static mode. Support of the dynamic setting can further improve the functionality and flexibility of the scheme.

For the second proposed scheme in this dissertation, further experimental investigations are needed to estimate the computation and communication costs in practice. Further study should be carried out on the possible improvement in terms of reducing the computational complexity.

Finally, further research could also be conducted to design a dynamic SSE scheme for spatial datasets that supports both forward and backward privacy at the same time.

References

[1] Tao Jiang, Xiaofeng Chen, Jin Li, Duncan S. Wong, Jianfeng Ma, and Joseph K. Liu. Towards secure and reliable cloud storage against data re-outsourcing. *Future Generation Comp. Syst.*, 52:86–94, 2015.

[2] Kaitai Liang, Willy Susilo, and Joseph K. Liu. Privacy-preserving ciphertext multi-sharing control for big data storage. *IEEE Trans. Information Forensics and Security*, 10(8):1578–1589, 2015.

[3] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO '13*, pages 353–373, 2013.

[4] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS '06*, pages 79–88, 2006.

[5] Dawn Xiaoding Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55, 2000.

[6] Melissa Chase and Seny Kamara. *Structured Encryption and Controlled Disclosure*, pages 577–594. Springer, 2010.

[7] Xu Yang, Ting-Ting Lee, Joseph K. Liu, and Xinyi Huang. Trust enhancement over range search for encrypted data. In *IEEE Trustcom*, pages 66–73, 2016.

[8] Kaitai Liang, Xinyi Huang, Fuchun Guo, and Joseph K. Liu. Privacy-preserving and regular language search over encrypted cloud data. *IEEE Trans. Information Forensics and Security*, 11(10):2365–2376, 2016.

[9] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51, August 2014.

[10] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. *Public Key Encryption with Keyword Search*, pages 506–522. Springer, 2004.

[11] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. On the integration of public key data encryption and public key encryption with keyword search. In *International Conference on Information Security*, pages 217–232. Springer, 2006.

[12] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In *International conference on Computational Science and Its Applications*, pages 1249–1259. Springer, 2008.

[13] Dalia Khader. Public key encryption with keyword search based on k-resilient ibe. In *International Conference on Computational Science and Its Applications*, pages 1086–1095. Springer, 2007.

[14] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Improved searchable public key encryption with designated tester. In *AsiaCCS*, pages 376–379, 2009.

[15] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. Private query on encrypted data in multi-user settings. In *ISPEC 2008*, pages 71–85, 2008.

[16] Eu-Jin Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[17] Yanjun Shen and Peng Zhang. Ranked searchable symmetric encryption supporting conjunctive queries. In *International Conference on Information Security Practice and Experience*, pages 350–360. Springer, 2017.

[18] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions on parallel and distributed systems*, 23(8):1467–1479, 2012.

[19] Xiuxiu Jiang, Jia Yu, Jingbo Yan, and Rong Hao. Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. *Information Sciences*, 403:22–41, 2017.

[20] Foteini Baldimtsi and Olga Ohrimenko. Sorting and searching behind the curtain. In *International Conference on Financial Cryptography and Data Security*, pages 127–146. Springer, 2015.

[21] Xianrui Meng, Haohan Zhu, and George Kollios. Top-k query processing on encrypted databases with strong security guarantees. *arXiv preprint arXiv:1510.05175*, 2015.

[22] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE Symposium on Security and Privacy (S&P) 2019*, 2019.

[23] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE, 2018.

[24] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340. ACM, 2016.

[25] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Data recovery on encrypted databases with k-nearest neighbor query leakage. In *Proceedings of IEEE Symposium on Security and Privacy*, 2018.

[26] Raphael Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1465–1482. ACM, 2017.

[27] Raphael Bost. $\sigma o \varphi o \varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154. ACM, 2016.

[28] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[29] Bellare, Namprempre, Pointcheval, and Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2003.

[30] David Chaum. *Blind Signature System*, pages 153–153. Springer US, 1984.

[31] Emmanuel Bresson, Jean Monnerat, and Damien Vergnaud. *Separation Results on the "One-More" Computational Problems*, pages 71–87. Springer, 2008.

[32] Jung Hee Cheon, Miran Kim, and Myungsun Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Transactions on Information Forensics and Security*, 11(1):188–199, 2016.

[33] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[34] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

[35] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. *Key Homomorphic PRFs and Their Applications*, pages 410–428. Springer, 2013.

[36] Emily Shen, Elaine Shi, and Brent Waters. *Predicate Privacy in Encryption Systems*, pages 457–473. Springer, 2009.

[37] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *International conference on the theory and applications of cryptographic techniques*, pages 506–522. Springer, 2004.

[38] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E Skeith. Public key encryption that allows pir queries. In *Annual International Cryptology Conference*, pages 50–67. Springer, 2007.

[39] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. *Order-Preserving Symmetric Encryption*, pages 224–241. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

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

[41] Michael T Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious ram simulation. In *International Colloquium on Automata, Languages, and Programming*, pages 576–587. Springer, 2011.

[42] Peter Williams, Radu Sion, and Bogdan Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 139–148. ACM, 2008.

[43] Emil Stefanov and Elaine Shi. Oblivistore: High performance oblivious distributed cloud data store. In *NDSS*, 2013.

[44] Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure oram with $\tilde{O}(\log^2 n)$ overhead. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 62–81. Springer, 2014.

[45] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel ram and applications. In *Theory of Cryptography Conference*, pages 175–204. Springer, 2016.

[46] Michael T Goodrich. Isogrammic-fusion oram: Improved statistically secure privacy-preserving cloud data access for thin clients. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 699–706. ACM, 2018.

[47] Michael T Goodrich. Bios oram: improved privacy-preserving data access for parameterized outsourced storage. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, pages 41–50. ACM, 2017.

[48] Benny Pinkas and Tzachy Reinman. Oblivious ram revisited. In *Annual Cryptology Conference*, pages 502–519. Springer, 2010.

[49] Shifeng Sun, Joseph K. Liu, Amin Sakzad, Ron Steinfeld, and Tsz Hon Yuen. An efficient non-interactive multi-client searchable encryption with support for boolean queries. In *ESORICS 2016, Part I*, pages 154–172, 2016.

[50] Boyang Wang, Ming Li, and Haitao Wang. Geometric range search on encrypted spatial data. *IEEE Transactions on Information Forensics and Security*, 11(4):704–719, 2016.

[51] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, volume 71, pages 72–75, 2014.

[52] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. Threshold privacy preserving keyword searches. *Lecture Notes in Computer Science*, 4910:646–658, 2008.

[53] Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Outsourced symmetric private information retrieval. In *ACM CCS'13*, pages 875–888. ACM, 2013.

[54] Christoph "Bösch, Qiang Tang, Pieter Hartel, and Willem" Jonker. *Selective Document Retrieval from Encrypted Database*, pages 224–241. Springer, 2012.

[55] Yan-Cheng Chang and Michael Mitzenmacher. *Privacy Preserving Keyword Searches on Remote Encrypted Data*, pages 442–455. Springer, 2005.

[56] Changyu Dong, Giovanni Russello, and Naranker Dulay. Shared and searchable encrypted data for untrusted servers. In *IFIP DBSEC*, pages 127–143, 2008.

[57] Philippe Golle, Jessica Staddon, and Brent Waters. *Secure Conjunctive Keyword Search over Encrypted Data*, pages 31–45. Springer, 2004.

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

[59] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.

[60] Florian Kerschbaum and Alessandro Sorniotti. *Searchable Encryption for Outsourced Data Analytics*, pages 61–76. Springer, 2011.

[61] M. Kuzu, M. S. Islam, and M. Kantarcioglu. Efficient similarity search over encrypted data. In *2012 IEEE Int. Conf. Data Engineering*, pages 1156–1167, 2012.

[62] M. Raykova, A. Cui, B. Vo, B. Liu, T. Malkin, S. M. Bellovin, and S. J. Stolfo. Usable, secure, private search. *IEEE Security Privacy*, 10(5):53–60, Sept 2012.

[63] Zhihua Xia, Xinhui Wang, Xingming Sun, and Qian Wang. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):340–352, 2016.

[64] Fangming Zhao, Takashi Nishide, and Kouichi Sakurai. *Multi-User Keyword Search Scheme for Secure Data Sharing with Fine-Grained Access Control*, pages 406–418. Springer, 2012.

[65] Yanjiang Yang, Haibing Lu, and Jian Weng. Multi-user private keyword search for cloud computing. In *CloudCom '11*, pages 264–271, 2011.

[66] Raluca A. Popa and Nickolai Zeldovich. Multi-key searchable encryption. *IACR Cryptology ePrint Archive*, 2013:508, 2013.

[67] Q. Tang. Nothing is for free: Security in searching shared and encrypted data. *IEEE Trans. on Information Forensics and Security*, 9(11):1943–1952, Nov 2014.

[68] Xiaoxin Wu, Lei Xu, and Xinwen Zhang. Poster: A certificateless proxy re-encryption scheme for cloud-based data sharing. In *ACMCCS*, pages 869–872. ACM, 2011.

[69] Yong Ho Hwang and Pil Joong Lee. *Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System*, pages 2–22. Springer, 2007.

[70] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. *Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions*, pages 205–222. Springer, 2005.

[71] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. *Deterministic and Efficiently Searchable Encryption*, pages 535–552. Springer, 2007.

[72] Dan Boneh and Brent Waters. *Conjunctive, Subset, and Range Queries on Encrypted Data*, pages 535–554. Springer, 2007.

[73] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *NDSS 2004*, 2004.

[74] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001*, pages 213–229. Springer, 2001.

[75] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS, 2014*.

[76] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374, 2014.

[77] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In *International Conference on Financial Cryptography*, pages 1–20. Springer, 2000.

[78] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[79] Ronald Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.

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

[81] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford University, 2009.

[82] Dan Boneh, David Mazieres, and Raluca Ada Popa. Remote oblivious storage: Making oblivious ram practical. 2011.

[83] Ivan Damgård, Sigurd Meldgaard, and Jesper Nielsen. Perfectly secure oblivious ram without random oracles. *Theory of Cryptography*, pages 144–163, 2011.

[84] Peter Williams and Radu Sion. Single round access privacy on outsourced storage. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 293–304. ACM, 2012.

[85] Jaideep Vaidya and Chris Clifton. Privacy-preserving top-k queries. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 545–546. IEEE, 2005.

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

[87] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[88] F Betül Durak, Thomas M DuBuisson, and David Cash. What else is revealed by order-revealing encryption? In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1155–1166. ACM, 2016.

[89] Ke Li, Weiming Zhang, Ce Yang, and Nenghai Yu. Security analysis on one-to-many order preserving encryption-based cloud data search. *IEEE Transactions on Information Forensics and Security*, 10(9):1918–1926, 2015.

[90] Ian H Witten, Alistair Moffat, and Timothy C Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[91] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[92] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Tfhe: Fast fully homomorphic encryption over the torus.

[93] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[94] Security estimates for the learning with errors problem, "https://bitbucket.org/malb/lwe-estimator".

[95] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[96] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[97] Florian Kerschbaum and Axel Schroepfer. Optimal average-complexity ideal-security order-preserving encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 275–286. ACM, 2014.

[98] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[99] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. Big data analytics over encrypted datasets with seabed. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 587–602, 2016.

[100] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 655–672. IEEE, 2017.

[101] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 707–720, 2016.

[102] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, volume 20.

[103] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679. ACM, 2015.

[104] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable

encryption in very-large databases: data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer, 2014.

[105] Boyang Wang, Ming Li, and Li Xiong. Fastgeo: Efficient geometric range queries on encrypted spatial data. *IEEE transactions on dependable and secure computing*, 2017.

[106] Guowen Xu, Hongwei Li, Yuanshun Dai, Kan Yang, and Xiaodong Lin. Enabling efficient and geometric range query with access control over encrypted spatial data. *IEEE Transactions on Information Forensics and Security*, 14(4):870–885, 2019.

[107] Boyang Wang, Ming Li, Haitao Wang, and Hui Li. Circular range search on encrypted spatial data. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 182–190. IEEE, 2015.

APPENDICES

Appendix A

Experimental Results

A.1 Multi-client symmetric searchable encryption

This section presents the implementation of our scheme and some results from using different number of keywords as well as different thresholds.

We implemented our scheme and OXT (there is no open source implementation of OXT) with multi-client to show the differences in token generation process. In multi-client OXT, a data owner provides search tokens to clients based on their queries whereas in our scheme a client (initiator) generates the search token with the assistance of a subset of clients (helping users). Therefore, we investigate the time consumed for computation and communication in the token generation process in both schemes.

The token generation process in our scheme consists of three main steps; keyword randomization, partial evaluation, and combine.

- keyword randomization: the initiator i_c starts the process by randomizing the

search keyword $z = H(w)^r$ and then sends it to other users i_j.

- partial evaluation: Each user performs partial evaluation over the received value using its share of the key $y_{i_j} = z^{k_{i_j}}$ and then returns the result to the initiator.

- combine: Upon receiving of $\theta - 1$ number of responses from the helping users, the initiator would be able to reach the full evaluation by combining them (the initiator also computes a partial evaluation of the search keyword) $y_r = (\mathcal{F}(k_{i_j}, z))_{j=1}^{\theta-1} = \mathcal{F}(k_0, z)$ and unrandomize it to gain the search token $y_r = y^{r^{-1}}$.

The first two steps are independent of the considered threshold θ, thus consume a fix amount of time.

Setup Environment. The system is deployed in Microsoft Azure. We use an E32s V3 instance (32 vCores, 256GB RAM) as the SSE index server, another D8s v3 instance (8 vCores, 32GB RAM) as the main client. We use 25 D2s v3 instances (2 vCores, 8GB RAM) to simulate the multi-client scenario. All servers are located in the same LAN, and equipped with 40 Gbps (5GB/s equivalent) NIC.

To imporve the runtime performance of our implementation (both for OXT and our protocol), we leverage an in-memory key-value storage Redis [**?**] to keep the generated TSet for querying purpose later. In addition, we deploy the Bloom filter from Alexandr Nikitin [**?**] as it is the fastest Bloom filter implementation for JVM. The false positive rate of the bloom filter is set to 10^{-6}, which enables our implementation to keep the XSet in a small fraction of RAM on the server.

We test our implementation on a dataset from Wikimedia Downloads [**?**] and the size of our dataset is 2.93GB[1] with $6.2 * 10^7$ (keyword, id) pairs.

We used the Java Pairing-Based Cryptography Library (JPBC) [**?**].

EDB Setup. We have tested 2.93GB dataset for Setup in OXT and our scheme. The time consumption for the generation of the encrypted database for our scheme

[1] enwiki-20161220-pages-articles22.xml

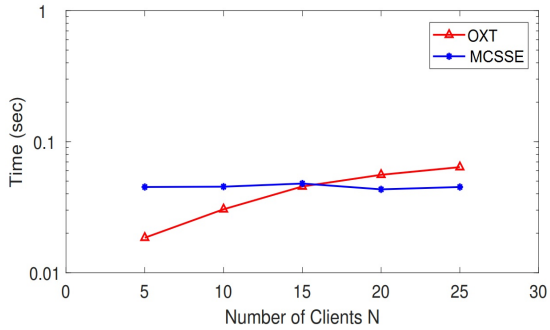
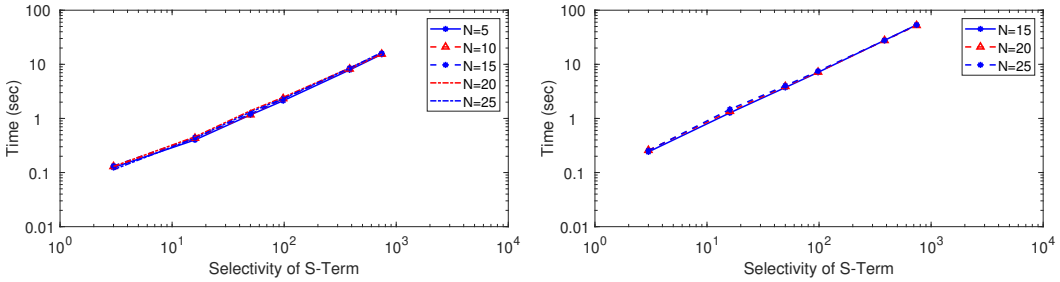


Figure A.1: Stag generation of the encrypted database with a fixed threshold



((a)) $\theta = 3$ ((b)) $\theta = 15$

Figure A.2: xtoken generation of the encrypted database with a fixed threshold

is 27.5 hours and for OXT is 5 hours. Note that in both schemes this phase is need to be performed once only.

Token Generation. Since the search token has the same components (Stag and Xtoken) in both MCSSE and OXT, the only difference is the way MCSSE generate these components. Therefore, we compare the generation time for Stag and Xtoken.

Fig. A.1 illustrates the stag generation time where the threshold is fixed ($\theta = 5$) and the number of client is increasing from 5 to 25. It is apparent from this figure that OXT is faster for the small number of clients. However, we observe that, with the increase of the number of clients, MCSSE maintains a constant time for generating the Stags where as the time consumption of OXT is linear to the number of users. In the given instantiation when the number of clients reaches 15 then both schemes behave almost the same. If there are more than 15 clients, MCSSE outperforms

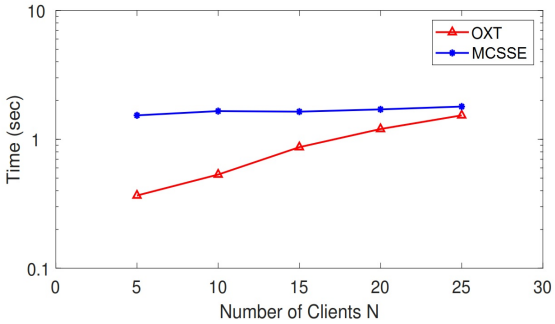


Figure A.3: xtoken selectivity

OXT in terms of stag generation. The reason for that is the client in MC-OXT can only rely on θ other clients to generate the stag, while the client of OXT should request stag from the data owner. As a result, if the data owner is busy, i.e., there are multiple stag requests, the client needs to wait longer for the response.

Later, we choose keyword with 3 to 742 matching documents (we call it the selectivity of the keyword) to evaluate the performance of xtoken generation. Fig. A.2(a) and Fig. A.2(b) demonstrate the xtoken generation time for two different thresholds $\theta = 3$ and $\theta = 15$. We can see that the xtoken generation time is a constant for a fixed threshold and selectivity. Therefore, MCSSE is highly scalable while its performance is not affected by the number of clients.

We also compare the xtoken generation time of MCSSE and OXT with a fixed threshold and selectivity. As shown in Figure A.3 for the fixed threshold 5 and selectivity 50, when the number of the client increases, the time for distribution of the xtoken in OXT increases whereas MC-OXT consumes a constant time. That is, to generate the xtoken in OXT, the client have to refer to the data owner and there might be a large amount of requests, which leads to a noticeable delay for xtoken generation.

EDB Update. We examine the runtime performance of proposed update strategy in our test dataset. In OXT, if we want to re-encrypt the EDB, we need to download the whole database and re-run the EDB setup phase, which is a time-

consuming procedure (5 hours). In MCSSE, this overhead is reduced to the half, that is, MCSSE only download TSet for updating, which takes only 3s. In addition, it requires 1.3 hour to update in a single-core machine on the data owner side. At the same time, XSet can be updated by within 40 minutes on the server side. In conclusion, compared to the traditional update strategy, MCSSE is able to update the EDB 3.8x faster.

Token Revocation. We evaluate the performance of the two revocation approaches of MCSSE with different number of users and different thresholds. For all the chosen thresholds (3 to 25) and numbers of clients (5 to 25 clients), the average time for user revocation in Approach 1 is 37.83 ms as the data owner requires to generate the new shares of the key and send it to the all of the valid clients. The Approach 2 offers a more efficient solution which only requires 200 μs for generation of the key material for the key update and publishes it in to the public repository, which avoids the network communication overhead.

RDPRF Performance Discussion. We give a brief discussion about the performance of RDPRF, as we use this new primitive to replace the PRF in OXT to achieve multi-client property. Table A.1 summarizes the performance of this cryptographic primitive for different number of clients and thresholds. To have realistic evaluation of the practicality of the proposed scheme, We considered two conditions for the location of the users; Intranet and Internet [**?**]. For the internet case, we considered that the main client is placed in US East but not hosted by Azure, and all assist clients are hosted by Azure US East. In both cases, we observe that the proposed RDPRF scheme has the property that if the threshold is fixed, it always runs with a constant delay. This is the key for MCSSE to support multi-client setting efficiently. While in supporting multi-client in the traditional way using OXT the network overhead becomes the bottleneck of the system.

Table A.1: RDPRF Performance

Number of clients	Threshold	Time (ms)	
		Intranet	Internet
5	3	41.2	118.2
	5	44.1	121.0
10	5	46.3	123.7
	10	60.2	137.1
15	10	60.5	136.5
	15	78.3	154.2
20	15	80.4	155.3
	20	94.5	171.5
25	20	97.0	173.9
	25	107.5	184.4

A.2 Geometric range search on encrypted data with forward/backward privacy

This section presents the experimental evaluation on the performance of the proposed constructions. Our scheme is implemented in Java (Nodejs v10.10.0, Typescript v3.4.3) on a 64-bit machine with 3.1GHz Intel®Core(i5) processor and 16 GB RAM inside. We implemented PRF evaluations with SHA-256.

Setup Cost. For a dataset with 20K data points where the dimension size of each coordinate is $D = 2^{15}$, the setup of both of our constructions takes about 23 seconds to be performed. For the mentioned dataset the setup phase generates 938.9 MB encrypted dataset.

Search Complexity. We examined the search running time in two different scenarios; when the dimension size grows from $D = 2^7$ to $D = 2^{15}$, and when the number of data points increases from 400 points to 20K points. From the graphs presented in Figure A.4 we can see that the server search is quite efficient for both Construction-I and Construction-II, at maximum of 0.91 ms and 0.45 ms, respectively. More precisely, the dimension size does not have a significant effect on the

server search time and the range size of a query results in variant search time.

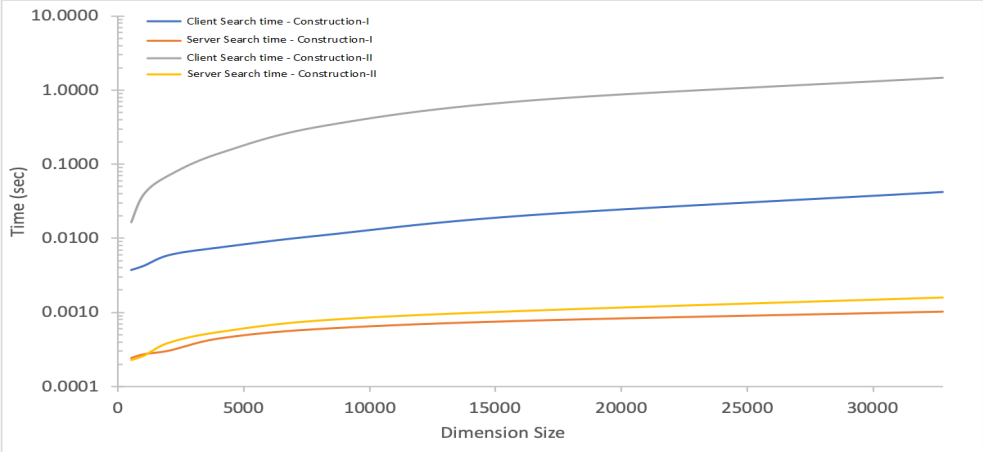


Figure A.4: Search time vs. dimension size (20K points)

From Figure A.5, it can be seen that although the search time is linear to the number of the data points in both of the proposed constructions for both server and client (as the number of the data points indicates the size of each ciphertext), both constructions are practical and quite fast at only about one second in the worst case. More precisely, for Construction-I in the worst case of 20K data points the server requires only 0.68 ms to find the matches and then client requires 0.1 ms to decrypt them and find the final results. Construction-II is slightly faster in the server side and a bit slower in the client side. That is, due to the use of homomorphic property of ASHE, client requires multiple keys for decryption of intermediate nodes in Construction-II. In the worst case of 20K data points, Construction-II requires 0.43 ms and 1.3 sec search time for the server and the client, respectively.

Update Complexity. We first evaluated the update complexity when the size of the environment expands while the number of data points and number of update points are fixed at 20K and 10, respectively. As shown in Figure A.6, the update complexity in Construction-I for the server is constant at less than 0.01 sec and for the client it grows slightly from 0.1 sec to 0.5 sec. However, in Construction-II the increase in the dimension size affects the update complexity in both client and server.

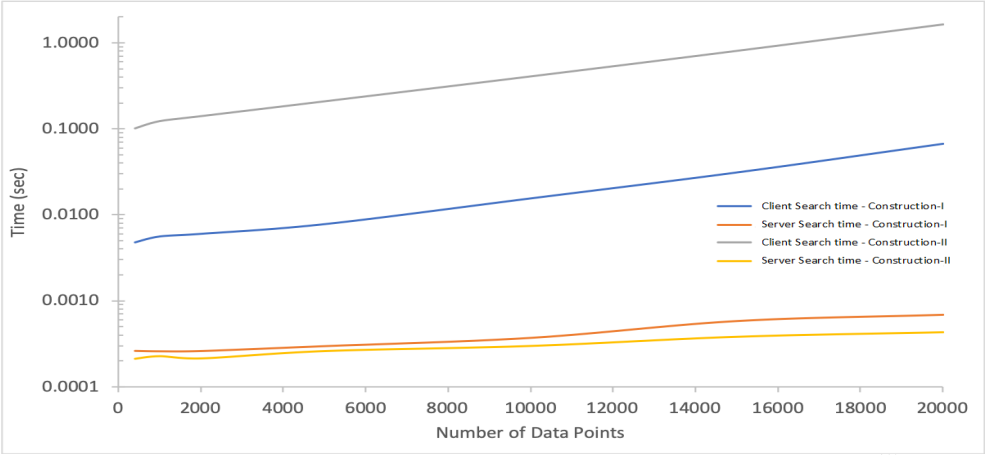


Figure A.5: Search time vs. number of data points ($D = 2^{15}$)

The main reason is that by expansion in the dimension size, the size of the binary tree also expands. Thus, the client has to generate the update tokens for all the leaf nodes which in turn results in higher communication overhead (as shown in Figure A.7). Moreover, the server also must update the whole binary tree, hence expansion in the dimension size will result more complexity for the server.

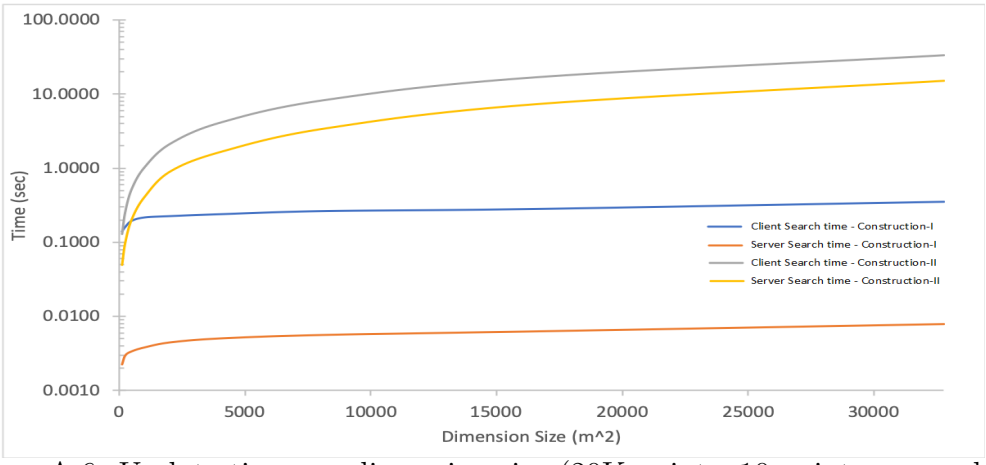


Figure A.6: Update time vs. dimension size (20K points, 10 points per update)

We extended our experiments to demonstrate the effect of the number of the update points on the update complexity. We chose the largest dimension size ($D = 2^t = 2^{15}$) from the previous experiment and a fixed number of data points at 20K. It

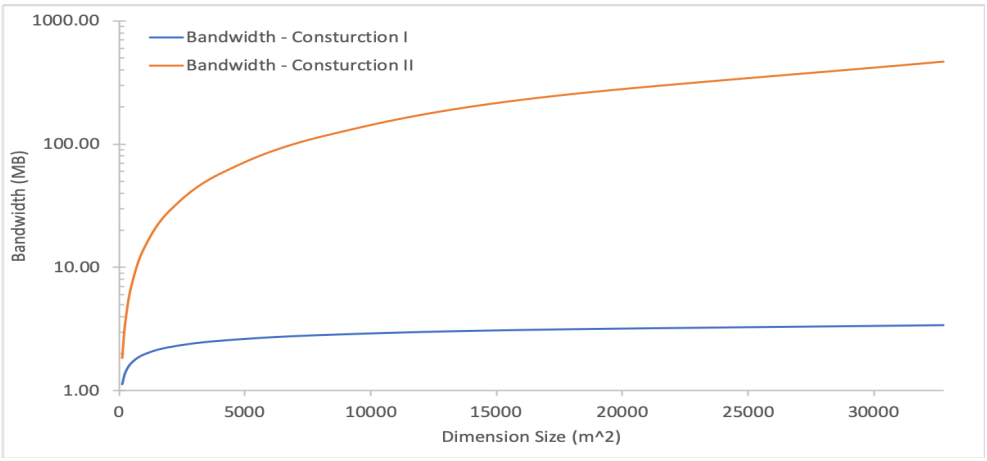


Figure A.7: Update communication vs. dimension size

is apparent from Figure A.8 that only the client update complexity in Construction-I grows linearly with the number of update points. However, as shown in Figure A.9 in terms of communication overhead, Construction-I is more efficient when the number of the update points are smaller.

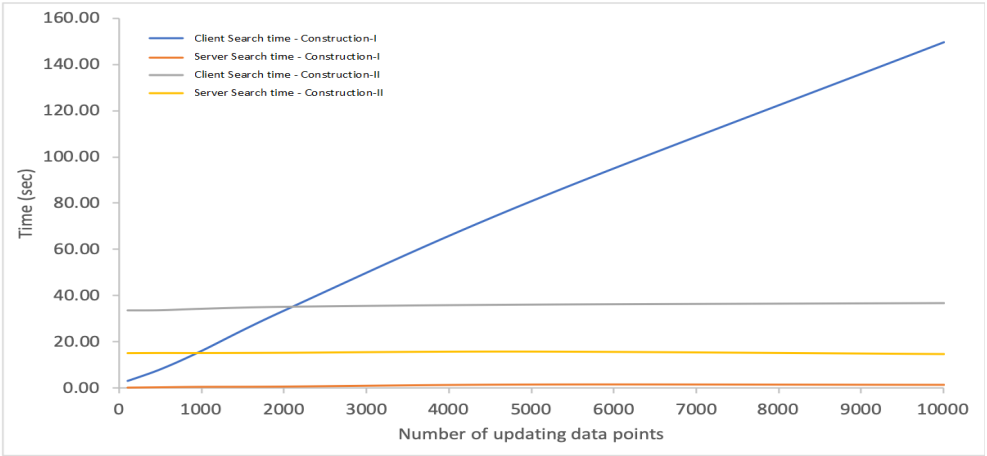


Figure A.8: Update time vs. number of updating points (20K points, $D = 2^{15}$)

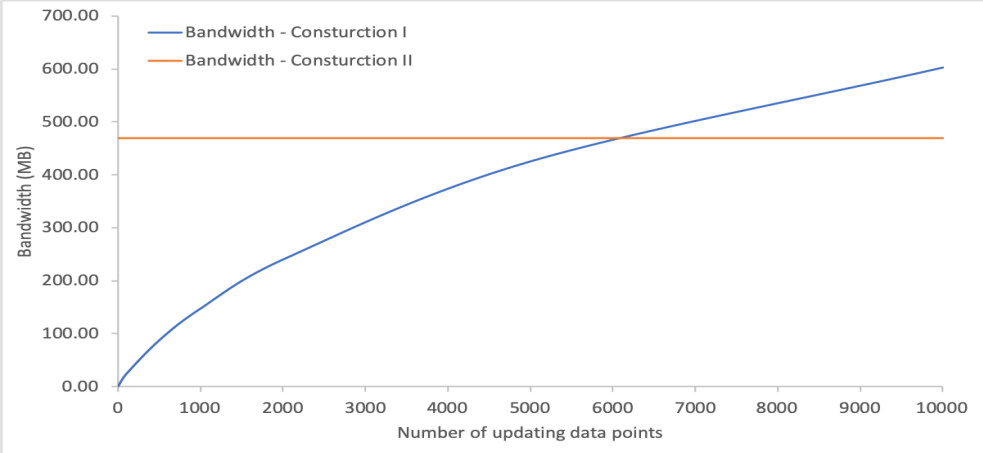


Figure A.9: Update communication vs. number of updating points