Oblivious keyword search

Yafu Ji

University of Wollongong
UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
Oblivious Keyword Search

A thesis submitted in fulfillment of the requirements for the award of the degree

Master of Computer Science by Research

from

UNIVERSITY OF WOLLONGONG

by

Yafu Ji

School of Computer Science and Software Engineering
January 2013
© Copyright 2013
by
Yafu Ji
All Rights Reserved
Dedicated to
My parents and my wife
Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

__________________________
Yafu Ji
January 12, 2013
The rapid development of information and communication technology has enabled more people to outsource and store their data on the networked databases. However, it brings challenges related to the data privacy and security.

Our research focuses on the privacy in which the party who requests for a service discloses the minimum amount of personal information, whereas the party that offers data is able to ensure that only the requested data are revealed. Particularly, we focus on the mechanism searching for keywords on encrypted data.

There are numbers of techniques that can be applied in the keyword search, such as public encryption with keyword search (PEKS), oblivious transfer (OT), private information retrieval (PIR), symmetric key encryption and homomorphic encryption. In this work, we consider PEKS, symmetric key encryptions and OT to perform oblivious keyword search. Our work can be summarized as follows.

- Keyword search in designated senders. Most of current keyword search schemes only consider the keyword stored in the email gateway or database service providers. Hence, an off-line keyword guessing attack was developed to perform brute force attacks on a limited keyword space. An efficient solution is providing another element, for example, the sender’s identity, in the ciphertext in order to improve the security of encrypted keywords. This allows us to achieve a new cryptography primitive namely keyword search with designated signers.

- Symmetric key encryption that provides a better security level than the other ones. The underlying principle of a symmetric key encryption is that the sender and the receiver must share the same secret key. We propose a construction based on the public key infrastructure that allows the sender and the receiver to exchange their public keys and construct a secret key. This scheme is based on a computable assumption that is more secure than the one based on
decisional assumption.

- Oblivious keyword search. The idea of oblivious keyword search is from the oblivious transfer (OT). The security of OT requires that both sender’s and receiver’s privacy are preserved. Therefore, from the privacy point of view, OT is suitable for enabling privacy of outsourced data.

- Public key encryption with oblivious keyword search. Combining the notions of PEKS and OT, we propose an oblivious keyword search scheme. The new scheme achieves the properties of both PEKS and OT. We note that the combination is done in a non-trivial manner in order to ensure the security of the system.
I would like to personally thank the following people, who supported me, with my sincere gratitude.

First and foremost, I would like to show my deepest appreciation to Professor Yi Mu and Professor Willy Susilo who are respectable and responsible scholars. Your valuable guidance always leads me forward and your help is accompanied with every stage of my research. Without your enlightening instruction, kindness and patience, this thesis would not be completed.

To all the staff and fellows, I would like to thank all the staff who has helped me. I also wish to acknowledge the help from all the Chinese postgraduates in the University of Wollongong, particularly, Jinguang Han, Fuchun Guo.

To my parents, I would like to thank you for supporting me, no matter when and where. I also would like to you for your endless love and selfless support. Without you, I would not have come so far in my education.

To my wife, I would like to thank you for understanding and encouragement. Without you, I would not overcome the difficult times.
## Contents

Abstract ................................. v
Acknowledgement ..................... vii

1 Introduction ........................ 1
   1.1 Crypto-Based Keyword Search ....... 2
   1.2 Goal of the Thesis .................. 4
   1.3 Organization of the Thesis ......... 6
   1.4 Notation ........................... 7

2 Background ........................... 8
   2.1 Preliminaries ...................... 8
      2.1.1 Cryptographic Hash Functions ... 8
      2.1.2 Random Oracle Model .......... 9
      2.1.3 Bilinear Pairing ............... 9
      2.1.4 Public Key Encryption .......... 9
      2.1.5 Diffie-Hellman Key Exchange ... 10
   2.2 Complexity Assumptions .......... 11
      2.2.1 Discrete Logarithm Problem ... 11
      2.2.2 Decisional Diffie-Hellman Assumption 11
      2.2.3 Divisible Computation Diffie-Hellman Assumption 11
      2.2.4 Gap Diffie-Hellman Problem ... 12
      2.2.5 Decision Linear Diffie-Hellman Assumption 12
      2.2.6 Bilinear Diffie-Hellman Assumption 13
   2.3 Related Works ...................... 13
      2.3.1 Oblivious Transfer .......... 13
      2.3.2 Private Information Retrieval 15
5.2.2 Security Model ........................................ 51
5.3 Implement of Oblivious Keyword Search .................. 52
  5.3.1 Construction ........................................ 53
  5.3.2 Security Analysis .................................... 54
5.4 Public Key Encryption with Oblivious Keyword Search .... 56
  5.4.1 Construction ........................................ 56
  5.4.2 Security Analysis .................................... 57
5.5 Summary .................................................. 60

6 Conclusion .................................................. 61
  6.1 Contributions .......................................... 61
  6.2 Future Work ............................................ 62

Bibliography .................................................. 64
List of Tables

1.1 Notations of the thesis ........................................... 7
4.1 System Parameter of SEKS ....................................... 43
5.1 System Parameters of Oblivious Keyword Search ............... 53
5.2 System Parameters of PEOKS ..................................... 56
List of Figures

1.1 Crypto-Based Keyword Search . . . . . . . . . . . . . . . . . . . . . . 3

2.1 Public key Encryption with Keyword Search . . . . . . . . . . . . . 16
Chapter 1

Introduction

With rapid development of information and communication technology, more and more people outsource or store their data on the networked databases. However, it brings some challenges related to privacy and security of the data. The information not only contains the users’ names and email addresses, but also includes some keywords related to contents or the preferences of users. Let us look at this scenario: assuming there is a perfect security scheme to protect your email or outsourced data, when the number of your emails or files grows large enough, you are not able to locate a single file in seconds by searching manually. To avoid this, you need to use keywords to search. Thus, the information of data will be revealed by the terms of you search if there is no protections. Certainly, data privacy is a hot topic in current cryptography and attracts many attentions in current cryptography [DVFJ+07, GSMB03, WLOB09, YSK09], and there is no doubt that the privacy on keyword search is another interesting issue.

At the beginning, information retrieval is a technology that provides people to collect the information easily. These tools such as google search engine have really improved people’s life. Moreover, the technology itself is improving everyday. Combining with the techniques of voice control, artificial intelligence and some other technologies, everyday life is much easier. The technology such as the Apple company’s new product “Siri”, we can just say “Hi Siri, Where is the best restaurant near us ?” and “How can we get there ?”. Then the name and the route to the restaurant will display on the phone. Unfortunately, this would be insecure in some specific scenarios. For example, the service provider may record and analyze your search results, or send advertisement or junk mail to your email address accordingly. In addition, when you store some confidential information such as employees’ salaries on the outsourced database, the opponent company may eavesdrop your communication with database server to acquire the information. Moreover, the
competitive company may corrupt the database server to obtain the information illegally. Therefore, it is worth to pay more attention to the security of communication by the service providers. It is always being concerned how to reveal the minimum secret to the service provider and gain more information.

Nevertheless, in some specific contexts, the users may try their best to obtain extra information from the service provider. Hence, not only the behavior of service providers, but also users’, need to be concerned.

1.1 Crypto-Based Keyword Search

To achieve privacy preserving keyword search, one of the most feasible solution is to apply cryptography on it. Regardless of emails or outsourced data, we should encrypt them prior to sending or storing them. Although encryption in a suitable form can improve the data confidentiality, it will cause more problems on data handling and maintenance. One of possible solutions is to decrypt all the ciphertexts, re-encrypt them after reading and store them back to the database server. However, this method is complicated and time-consuming, hence it is inefficient, especially in the case that the quantity of emails or outsourced data is extremely large. Thus, crypto-based keyword search is a viable solution.

The first encryption scheme for data transformation should be backward to oblivious transfer (OT) in 1981, which is proposed by Rabin [Rab81]. In this protocol, a sender may transfer a message to the receiver who has 1/2 probability of getting it, while the sender is not able to ensure whether the receiver is going to acquire the message. Then the concept has been extended to 1-out-of-n OT [BCR86, BCR87] and t-out-of-n OT [NP99b], which means that a receiver wants to obtain 1 or t of n messages from sender, the sender is not able to confirm which message is the receiver’s choice. Meanwhile the receiver can only obtain 1 or t message(s) from sender. OT not only improves the sender’s privacy, but also protects the receiver’s choice.

Following the concept of OT, Chor et al. proposed the notion of private information retrieve (PIR) in 1995 [CKGS98]. In their protocol, a user is able to retrieve an item from database server without revealing anything else. This seems to have addressed the problem that privacy of the keyword searcher should be protected. Unfortunately, some researchers pointed out that it was a weak version of 1-out-of-n
OT, and the confidentiality of service providers had not been considered. Nevertheless, after the improvement by Kushilevitz et al. [KO97], the PIR improves the communication efficiency.

With the introduction of the pairing and Identity Based Encryption (IBE) [BF03, Sha85], the public key encryption with keyword search (PEKS) scheme was proposed by Boneh et al. [BCOP04]. Their scheme enables users to create trapdoors that contain one or several keyword(s) to the email gateway. According to the user’s requirements, the gateway will route the email that contains certain keywords to be sent to the assigned digital devices. Other emails will be stored in the mail box to be delivered later. It is an extremely valuable crypto application since digital devices are ubiquitous. Moreover, it supplies keyword search with public key encryption, in such a case that people do not need to share the secret key among the data holders. It also provides a method to achieve a non-interactive keyword search. However, there are still some remaining problems, which attract attention, such as the ones mentioned in [ABC+05, BSNS05, BW06, CKRS09]. Additionally, no scheme is able to solve the problems thoroughly. Hence, our first work addresses the
1.2 Goal of the Thesis

We propose several schemes that solve the problem of revealing private information in these scenarios:

1. Suppose Alice is an email user. Due to the development of information technology, Alice is able to read her email on several devices: desktop, laptop, smart phone, iPad, etc. For convenience, she may set some keywords to her email gateway so that only the message which contains these keywords will be routed to her smart phone. Others will be sent to her desktop or laptop to be dealt with during her working hours. For instance, Bob wants to send two emails to Alice with the keywords “urgent” and “lunch” respectively and has encrypted using Alice’s public key. Alice has set the gateway that only emails containing the keyword “urgent” to be directed to her smart phone. Therefore, the former letter will be sent to smart phone and latter will be sent to desktop. In this scenario, Alice will send a trapdoor to gateway to perform the setting. Nevertheless, the trapdoor isn’t secure enough to prevent the malicious attack. For example, the one who acquires the trapdoor in some way may conduct off-line attacks to the trapdoor. Roughly speaking, the attacker could use brute force to guess the trapdoor, since the keywords space is limited. Therefore, we try to make a better keyword trapdoor resisting to the off-line keyword guessing attacks.

2. Similar to the scenario 1, Charlie and David are email users. Charlie wishes to set a keyword to email gateway that only allows the email encrypted under that
1.2. Goal of the Thesis

A keyword from David can be routed to his smart phone. However, he concerns about his keywords will be attacked by the other parties such as email gateway. Therefore, he wants to use a keyword encrypted by a secret key shared between him and David. However, if using a secret key, they should communicate to discuss a secret key before sending an email. Moreover, they always need to change the secret key to ensure the security is warranted. This change not only affects the efficiency, but also depends on the security of channel which transmits the secret key.

3. Frank is a user of a public database. He wants to search several keywords in a database. However, he wouldn’t wish the database provider to know what he searches. Despite the fact that the database is public, the services provider prefers Frank not to know the content but his search result. In addition, if the files in the database are updated by different people who wish to share the file to other parties. They might use public key to encrypt their files and then send to the database server.

The scenarios above outline the searchable encryption requirements: allowing the search in encrypted data and making leverage between safety and efficiency. Thus, the question is how we could search the encrypted data with the best possible efficiency and security. This thesis provides the solutions to these problems by searchable encryption.

Searchable encryption is a technology that provides capabilities to search encrypted data without decryption key. If the data can be viewed as messages, the special decryption key, which we call as a trapdoor, can only decrypt a particular message. Each message is combined with one or a set of keywords. The message will be encrypted by keywords which could be queried. A trapdoor is a set of keywords which process the functionalities of decryption key. The message can be decrypted, while the trapdoor could only be used for keyword matching.

In the first scenario, Alice wants to set a keyword so that the email gateway is able to redirect selected emails to her smart phone automatically. By using the public encryption with keyword search, this task could be easily performed. However, some attackers developed off-line keyword guessing attack which tries to perform brute force attack on limited keyword space. Therefore, we try to develop a new PEKS to prevent such an attack. We will add identity-based signcryption into keyword search, so that the message could be authorized from the certificated
sender. Moreover, the trapdoor is more secure than before.

In the second scenario, Charlie and David desire to use symmetric key to perform keyword search, since symmetric key encryption can provide a better security level than the other ones. The intuition of symmetric key encryption, Charlie and David should share the same secret key. We proposed a construction based on public key infrastructure. Both Charlie and David will have a private key and public key. Before they encrypted their keyword or establish a trapdoor, they might use Diffie-Hellman key exchange to derive the secret key. Therefore, this can be more confidential and secure than using secret key only. Moreover, it can be extended to multi-users conveniently.

In the third scenario, Frank would like to retrieval information on a database. Both Frank’s and database service provider’s privacy need to be guarantied. The properties are much similar to those of Oblivious Transfer. Therefore, we employ the OT to perform keyword search. Meanwhile, we improve it with PEKS to achieve public encryption with oblivious keyword search.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows.

- Chapter 2 presents the cryptographic background knowledge for the thesis. Firstly, we introduce the cryptographic primitives, including cryptographic hash function, public-key cryptography etc. Secondly, we define the complexity assumptions. We briefly describe some knowledge on security proof. Finally, we review related keyword search protocols.

- Chapter 3 presents a scheme of public key encryption with keyword search in a designated sender.

- Chapter 4 proposes a scheme of symmetric key encryption with keyword search.

- Chapter 5 proposes two schemes developed from oblivious transfer and improved to public key encryption with oblivious keyword search.

- Chapter 6 concludes the thesis by highlighting the contributions of this study, and points out the future work.
1.4 Notation

In this section, we briefly provide the general notations used in our thesis.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mathbb{Z})</td>
<td>the set of integers</td>
</tr>
<tr>
<td>(\mathbb{Z}_q)</td>
<td>an integer group of order (q)</td>
</tr>
<tr>
<td>(\mathbb{Z}_q^*)</td>
<td>an integer group of order (q) without zero</td>
</tr>
<tr>
<td>(r \in R \mathbb{Z}_q^*)</td>
<td>uniformly pick a element (r) from group (\mathbb{Z}) at random</td>
</tr>
<tr>
<td>(G)</td>
<td>a multiplicative cyclic group</td>
</tr>
<tr>
<td>(G_q)</td>
<td>a multiplicative cyclic group of order (q)</td>
</tr>
<tr>
<td>(g \in G)</td>
<td>a generator of a multiplicative cyclic group</td>
</tr>
<tr>
<td>(g \leftarrow G^*)</td>
<td>(g) is a random generate of group (G)</td>
</tr>
<tr>
<td>(</td>
<td>S</td>
</tr>
<tr>
<td>(\mathcal{A})</td>
<td>an algorithm</td>
</tr>
<tr>
<td>(\mathcal{A}(\cdot))</td>
<td>an algorithm has one input</td>
</tr>
<tr>
<td>(\mathcal{A}(\cdot, \cdot))</td>
<td>an algorithm has two input</td>
</tr>
<tr>
<td>(\hat{e})</td>
<td>a bilinear map</td>
</tr>
<tr>
<td>(r_i)</td>
<td>the index (i) of element (r)</td>
</tr>
<tr>
<td>(a, \ldots, b)</td>
<td>a set of integers from (a) to (b)</td>
</tr>
<tr>
<td>(\hat{e} : G \times G \rightarrow G_T)</td>
<td>a bilinear map from (\hat{e}(g, g)) to (G_T)</td>
</tr>
<tr>
<td>(H)</td>
<td>crypto-hash function</td>
</tr>
<tr>
<td>(\oplus)</td>
<td>exclusive or</td>
</tr>
<tr>
<td>(\Pr[a])</td>
<td>probability of (a)</td>
</tr>
<tr>
<td>(\epsilon)</td>
<td>a negligible probability</td>
</tr>
<tr>
<td>(y \leftarrow \mathcal{A}(\cdot))</td>
<td>(y) is the output of algorithm (\mathcal{A}) on one input</td>
</tr>
<tr>
<td>(y \leftarrow S)</td>
<td>(y) is chosen from set (S)</td>
</tr>
<tr>
<td>PPT</td>
<td>probabilistic polynomial time</td>
</tr>
<tr>
<td>TTP</td>
<td>trusted third party</td>
</tr>
<tr>
<td>(e)</td>
<td>the base of nature logarithm</td>
</tr>
</tbody>
</table>

| \(\mathbb{Z}_q\) | an integer group of order \(q\) |

Table 1.1: Notations of the thesis

In Table 1.1, we briefly introduce the notations used most in the thesis.
Chapter 2

Background

In this chapter we introduce a concise elucidation of the fundamental knowledge that will be used throughout this thesis. Firstly, we describe basic concepts of cryptographic hash function, random oracle and bilinear map. After that we provide a brief summary about the encryption system and in particular to the public key encryption. Then we discuss about complexity assumptions. Finally we review some related work of our research and present some problems.

2.1 Preliminaries

2.1.1 Cryptographic Hash Functions

A hash function is a mathematic function which maps a string with any length to a string with fix length. The hash function as a efficient cryptographic tool has been used to check the integrity of data. The formal definition of hash function is as follows:

**Definition 2.1 Hash Function** A hash function $H$ is a function: $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ with the following features [Mao03]:

1. **Determination.** For the same input $a$, it outputs the same value $H(a)$;
2. **Efficiency.** Given a $a \in \{0, 1\}^*$, $H(a)$ can be computed in polynomial time;
3. **Collusion resistance.** It is computationally impossible to find $a, b \in \{0, 1\}^*$ such that $H(a) = H(b)$;
4. **Pre-image resistance.** Given a hash value $h$, it is computationally impossible to find a $x \in \{0, 1\}^*$ such that $h = H(x)$.
2.1.2 Random Oracle Model

Random oracle model was first proposed by Bellare and Rogaway [BR93] in 1993. In this model, a hash function is used, which works as a completely random function and called random oracle. The random oracle is a deterministic function and has uniform output. It always outputs the same value for the same input. In this model, a simulator maintains a table and simulates all random oracles. For one query, the simulator checks the table and responds with the value recorded in the list.

Random oracle model is an important tool used to prove the security of cryptographic protocols. Generally speaking, protocols that can be proven to be secure in random oracle model are more efficient than those in standard model.

2.1.3 Bilinear Pairing

Let \( G \) and \( G_T \) be two multiplicative cyclic groups of order \( p \) for some large prime \( p \). There exists a bilinear map \( \hat{e} : G \times G \rightarrow G_T \). Let \( g, h \) be two random generators of group \( G \). It possesses the properties as follow:

1. **Computable**: It is efficient to compute \( \hat{e}(g, h) \in G_T \).

2. **Bilinearity**: For all \( g, h \in G \) and random \( a, b \in \mathbb{Z} \), the equation \( \hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab} \) is true.

3. **Non-Degeneracy**: For all \( g \in G \), if \( g \) is a generator of \( G \), \( \hat{e}(g, g) \) is a generator of \( G_T \). In another word, if \( g \neq 0 \), then \( \hat{e}(g, g) \neq 1 \).

We can find \( G \) and \( G_T \) which should possess the properties above from Weil pairing or Tate pairing.

2.1.4 Public Key Encryption

Diffie and Hellman [DH76] first proposed the definition of public key encryption in 1976. In a public key encryption scheme, there are a pair of keys. One is named as secret key and the other is named as public key. Notably, given the public key, it is impossible to compute the secret key. The distinctive feature of public key encryption is that, when a sender wants to send a message to a receiver, he can encrypt it under the receiver’s public key directly without any previous interaction.
to share a secret key. After receiving the ciphertext from the sender, the receiver can use his secret key to decrypt it and obtain the original message.

The formal definition for public key encryption is as follows:

**Definition 2.2** A public key encryption scheme comprise of three algorithms:

- **SysPara**$(k) \rightarrow (sk, pk)$. Taking as input a security parameter $k$, this algorithm outputs a pair of keys $(sk, pk)$, where $sk$ is the secret key and $pk$ is the corresponding public key.

- **Enc**$(pk, m) \rightarrow C$. Taking as input the public key $pk$ and a message $m$, this algorithm outputs a ciphertext $C$.

- **Dec**$(sk, C) \rightarrow m$. Taking as input the secret key and the ciphertext, this algorithm outputs the message $m$.

We say that a public key encryption scheme is correct, if

$$\Pr \left[ \begin{array}{c}
\text{De}(sk, C) \rightarrow m \\
\text{SysGen}(k) \rightarrow (sk, pk) \\
\text{Enc}(pk, m) \rightarrow C
\end{array} \right] = 1$$

There are some famous public key encryption schemes: RSA encryption [RSA78], ElGmagal encryption [ElG85] and Cramer-Shoup encryption [CS98].

### 2.1.5 Diffie-Hellman Key Exchange

Diffie and Hellman [DH76] Proposed a key exchange scheme which can be used by two entities to set a secure channel. The scheme is described as follows:

- **SysPara**$(k)$. Two entities Alice and Bob generate their key pair $(x_A, Y_A)$ and $(x_B, Y_B)$, respectively, where $Y_A = g^{x_A}$ and $Y_B = g^{x_B}$.

- Interaction. Alice sends her public key $Y_A$ to Bob. Bob sends $Y_B$ to Alice.

- **KeyGen**.

  1. Alice uses her secret key $x_A$ to compute $K = Y_B^{x_A}$;
  2. Bob uses his secret key $x_B$ to compute $K = Y_A^{x_B}$.

The key for Alice and Bob is $K = Y_B^{x_A} = Y_A^{x_B} = g^{x_Ax_B}$. 
2.2 Complexity Assumptions

2.2.1 Discrete Logarithm Problem

Discrete logarithm problem [COS86] is one of the basic assumptions in cryptography.

**Definition 2.3** Discrete Logarithm Problem. *Suppose that $\mathbb{G}$ is a cyclic group and $g$ is a generator in $\mathbb{G}$. Given $y \in \mathbb{G}$, the problem is to compute a $x$ such that $y = g^x$.***

**Definition 2.4** Discrete Logarithm Assumption. *Suppose that $\mathbb{G}$ is a cyclic group and $g$ is a generator in $\mathbb{G}$. Given a value $y \in \mathbb{G}$, if no PPT algorithm can compute a $x$ such that $y = g^x$ with advantage

$$\Pr[y = g^x : x \leftarrow A(\mathbb{G}, g, y)] \geq \epsilon.$$*

2.2.2 Decisional Diffie-Hellman Assumption

**Definition 2.5** Let $\mathbb{G}$ be multiplicative cyclic group of prime order $p$. $g, u$ are random generators in $\mathbb{G}$, $a, b$ are random elements in $\mathbb{Z}_p^*$. Given $g, g^a, g^b, u$ as input, there is little help to output whether $g^{ab} = u$ or not.

We define an algorithm $A$ that can output $g^{ab} = u$ with advantage $\epsilon$ in solving DDH if

$$\Pr[g \overset{R}{\leftarrow} \mathbb{G}, a, b \overset{R}{\leftarrow} \mathbb{Z}_p^* : A(g, g^a, g^b, g^{ab}) = 1] \geq \epsilon
- \Pr[z \overset{R}{\leftarrow} \mathbb{G} : A(g, g^a, g^b, u) = 1] \geq \epsilon$$

We say that an algorithm $A(t, \epsilon)$ has probability $\epsilon$ to solve DDH in $\mathbb{G}$ if the running time at most $t$.

2.2.3 Divisible Computation Diffie-Hellman Assumption

Divisible Computation Diffie-Hellman (DCDH) assumption which is mentioned in [BDZ03].

**Definition 2.6** (DCDH) *Let $\mathbb{G}$ be multiplicative group of prime order $p$. $g$ is a random generator in $\mathbb{G}$, $a, b$ are random elements in $\mathbb{Z}_p^*$. Given $g, g^a, g^b$ as input, there is little help to output $g^{ab}$.***
2.2. Complexity Assumptions

We define an algorithm $A$ that outputs $g^z$ has advantage $\epsilon$ in solving DCDH if

$$\Pr[g \overset{R}{\leftarrow} G, a, b \overset{R}{\leftarrow} \mathbb{Z}_p^*: g^z \leftarrow A(g, g^a, g^b)] \geq \epsilon$$

We say that an algorithm $A(t, \epsilon)$ has probability $\epsilon$ to solve DCDH in $G$ if the running time at most $t$.

2.2.4 Gap Diffie-Hellman Problem

Gap Diffie-Hellman (GDH) problem which is introduced by Okamoto and Pointcheval [OP01].

**Definition 2.7 (GDH)** Let $G$ be multiplicative group of prime order $p$. $g$ is a random generator in $G$, $a, b, c$ are random elements in $\mathbb{Z}_p^*$. The Gap Diffie-Hellman problem is stated as follows:

1. Computation Diffie-Hellman Problem (CDHP). Given $(g, g^a, g^b)$, compute $g^{ab}$.
2. Decisional Diffie-Hellman Problem (DDHP). Given $(g, g^a, g^b, g^c)$, decide whether $c = ab$ in $\mathbb{Z}_p$.

We call $G$ is a Gap Diffie-Hellman Group where DDHP can be solved in polynomial time and there is no probabilistic algorithm can solve CDHP in polynomial time.

2.2.5 Decision Linear Diffie-Hellman Assumption

The security of our system is based on complexity assumptions called Decision Linear Diffie-Hellman (DLDH) assumption which is introduced by Boneh, Boyen and Shacham [BBS04].

**Definition 2.8 (DLDH)** Let $G$ be multiplicative group of prime order $p$. $u_1, u_2, u_3$ are random generators in $G$, $a, b, c$ are random elements in $\mathbb{Z}_p^*$. Given $u_1, u_2, u_3, u_1^a, u_2^b, u_3^c \in G$ as input, there is little help to output whether $a + b = c$ or not.

One can easily show that an algorithm that solves DLDH problem in $G$ can also solve DDH problem. However, the converse is believed to be false. We define an algorithm $A$ that outputs $c = a + b$ has advantage $\epsilon$ in solving DLDH if

$$\left| \Pr[u_1, u_2, u_3 \overset{R}{\leftarrow} G, a, b \overset{R}{\leftarrow} \mathbb{Z}_p^*: A(u_1, u_2, u_3, u_1^a, u_2^b, u_3^{a+b}) = 1] - \Pr[u_1, u_2, u_3, \eta \overset{R}{\leftarrow} G, a, b \overset{R}{\leftarrow} \mathbb{Z}_p^*: A(u_1, u_2, u_3, u_1^a, u_2^b, \eta) = 1] \right| \geq \epsilon$$

We say that an algorithm $A(t, \epsilon)$ has probability $\epsilon$ to solve DLDH in $G$ if the running time at most $t$. 

2.2.6 Bilinear Diffie-Hellman Assumption

**Definition 2.9** Let $G$ be multiplicative group of prime order $p$. $g$ is a random generator in $G$, $a, b, c$ are random elements in $\mathbb{Z}_p^*$. $\hat{e}$ is a bilinear map from $G$ to $G_T$. Given $g, g^a, g^b, g^c$ as input, there is little help to output $\hat{e}(g, g)^{abc}$.

We define an algorithm $A$ that outputs $\hat{e}(g, g)^{abc} = \hat{e}(g, g)^z$ has advantage $\epsilon$ in solving BDH if

$$\left| \Pr[g \leftarrow G, a, b, c \leftarrow \mathbb{Z}_p^*: \hat{e}(g, g)^{abc} \leftarrow A(g, g^a, g^b, g^c)] \right| \geq \epsilon$$

We say that an algorithm $A(t, \epsilon)$ has probability $\epsilon$ to solve BDH in $G$ if the running time at most $t$.

**Definition 2.10** Let $G$ be multiplicative cyclic group of prime order $p$. $g$ is a random generator in $G$, $V$ is a random element in $G_T$, $a, b, c$ are random elements in $\mathbb{Z}_p^*$. $\hat{e}$ is a bilinear map from $G$ to $G_T$. Given $g, g^a, g^b, g^c, V$ as input, there is little help to output whether $\hat{e}(g, g)^{abc} = V$ or not.

We define an algorithm $A$ that can output $\hat{e}(g, g)^{abc} = \hat{e}(g, g)^z$ with advantage $\epsilon$ in solving DBDH if

$$\left| \Pr[g \leftarrow G, a, b, c \leftarrow \mathbb{Z}_p^*: \hat{e}(g, g)^{abc} \leftarrow A(g, g^a, g^b, g^c, V)] = 1 \right| - \Pr[g \leftarrow G, a, b, c \leftarrow \mathbb{Z}_p^*, V \leftarrow G_T: A(g, g^a, g^b, g^c, V) = 1] \geq \epsilon$$

We say that an algorithm $A(t, \epsilon)$ has probability $\epsilon$ to solve DBDH in $G$ if the running time at most $t$.

### 2.3 Related Works

#### 2.3.1 Oblivious Transfer

Before the Public Key Encryption with Keyword Search, Oblivious Transfer (OT) was always being considered as the best way to construct a security communication channel between sender and receiver. Since the property of OT is very suitable for conducting keyword search, sender knows nothing about the receiver’s choices, and receiver can not obtain extra information except his choices. Therefore, Ogata and Kurosawa proposed an oblivious keyword search scheme based on OT [OK04]. Prior
to introducing Ogata and Kusosawa’s scheme, we will explain another OT scheme, in order to allow readers to have a preliminary knowledge about OT.

Chu and Tzeng proposed two efficient oblivious transfer schemes with adaptive and non-adaptive queries [CT05]. In their scheme, the sender commits all data to receiver firstly, which are random numbers from the receivers’ perspective. Then, the receiver makes queries to sender and sender responds by the corresponding secret keys of his choices. We should notice that these keys can only decrypt the files the receiver has chosen. For the other files, the result of receiver’s calculation is just like random numbers. Furthermore, one of their schemes can be extended to adaptive OT. Although the receiver can send the queries one by one adaptively, the scheme supplies the same security as the non-adaptive one. Therefore, this property extremely suits keyword search. Additionally, it also suits searching in a public database. For that reason, we have thought that using accumulator to integrate the search item, and it fulfills the property of pay-as-you-use in cloud computing. The author also gives some comparisons between Mu’s [MZV02], Naor’s [NP99b], Ogata’s [OK04] schemes and themselves’, and their schemes have more advantages than the others on efficiency and confidentiality. Thus, our second application is based on it.

Back to Ogata and Kurosawa’s scheme, they firstly proposed the concept of oblivious transfer. Compared with traditional OT, their proposal not only uses keyword instead of the index to conduct keyword search, but also utilizes blind signature to process keyword extract, which illuminated the following researchers to perform keyword extract in the similar way [CKRS09].

Oblivious Polynomial Evaluation (OPE) is first proposed in 1999 [NP99a, NP06], which is a variant of OT. Instead of committing all the data to the receiver, the sender inputs a polynomial function such as $P(\alpha)$. Receiver will input a value such as $\alpha$. When $\alpha$ is the root of $P(\alpha)$, the function $P(\alpha)$ will return the result the receiver want to obtain. Otherwise, the receiver gets nothing from sender since the result will be like a random number. There are many applications for OPE, such as mutually authenticated key exchange, private comparison of data. Similarly, OPE can be extended to keyword search. Freedman has proposed a keyword search scheme with oblivious pseudorandom functions based on OPE [FIPR05]. They use a pseudorandom function to perform OPE, then employ it to compute $P(\alpha) \oplus M$ where $M$ is the message and the root is keyword which will make $P(\alpha)$ equals to zero. Since the pseudorandom function is published, the user can execute keyword search
by himself. On the other hand, the authors haven’t supply such a pseudorandom function, though it inherit the $OT$’s property of protect both sender’s and receiver’s privacy. Therefore the method is worth to continue thinking about. Similarly, [GSW04, ZB07] utilized OPE to perform oblivious transfer as well.

### 2.3.2 Private Information Retrieval

As we have mentioned previously, Private Information Retrieval (PIR) is a protocol to protect the privacy of receiver rather than sender in 1995 [CKGS98]. It seems that the security is weaker than oblivious transfer(OT), but the fact is just the opposite. PIR provides the security between one receiver and multiple senders. In another word, a receiver can search on $k$ ($k > 2$) databases without revealing any information under PIR protocol. Then they continued to extend to information retrieval by keywords [CGN97]. The definition is much more approached to the concept of searching in the cloud. However, to achieve this, they should store replicated copies on different database. In addition, Ostrovsky [OS97] extended it private information storage, in which people can read and write on public database confidentially. Moreover, Naor [NP00] proposed another OT scheme to achieve similar results.

In spite of PIR allowing users to search information privately, it is still inefficient since the database server might send the user all data where only some of them are results. To avoid this, by using of Bloom filters [Blo70], Boneh [BKOJ07] brought public key encryption into PIR and achieved that the total communication linear to $n$. However, PIR doesn’t protect the sender’s privacy.

### 2.3.3 Public Encryption with Keyword Search

The Public Encryption with Keyword Search (PEKS) was proposed in 2004 [BCOP04]. The scheme is based on bilinear pairing and its security is based on Bilinear Diffie-Hellman assumption. Generally speaking, the scheme encrypts the email body first. Then keywords should be encrypted by using receiver’s public key. At the same time, the receiver will create a trapdoor utilizing his private key and send the trapdoor to the email gateway. The gateway will use the trapdoor to test the ciphertext whether it contains a certain keyword and then route it to corresponding destination. Figure 2.1 shows how the PEKS works.

Public key encryption with conjunctive filed keyword search was proposed by Park et al. [PKL05]. In their filed keyword definition, keywords should be classified
2.3. Related Works

Figure 2.1: Public Key Encryption with Keyword Search

in different areas in order to avoid one keyword located in the different fields. For example, the content of email from “Bob” to “Alice” is related to “lunch”. The three keywords should be arranged to the field “From”, “To” and “Subject”. Therefore, their scheme can perform conjunctive keyword search accurately. Hwang [HL07] and Zhang [ZZ11] also gave other ways to construct public encryption with conjunctive keyword search separately. As the reason that there is no specific scheme to perform the encryption of email body, Baek [BSNS06] proposed public key encryption (Elgamal) with PEKS to refine the original scheme. In addition, they extend their scheme to multi-user setting. Gu [GZP08] has proposed an efficient scheme that provides no pairing operations involved in encryption.

However, the argument on PEKS never stops since it was proposed. Beak [BSNS05] pointed out that there must be a secure channel such as Secure Socket Layer (SSL) between receiver and email gateway, otherwise it cannot be secure. In addition, there is no secure channel in some special situation such as GPRS network. Therefore they proposed a secure channel free PEKS scheme by constructing a crypto secure channel between sender and email gateway. Byun et al. and Yau et al. proposed off-line keyword guessing attack separately [BRPL06, YHG08]. In their views, the keywords are chosen from smaller space than the password. Therefore the scheme is more vulnerable to resist from keyword guessing attack. Additionally, Jeong [JKHL09] gave the proof that constructing PEKS scheme under offline keyword guessing attack is impossible.

Abdalla [ABC+05] first suggested that the PEKS is a weak consistent scheme,
since the hash value of keyword is not statistical consistent resistant. Therefore, in order to preserve consistency, they add session key in PEKS and verify all four elements of ciphertext. Although it provides the consistency, it sacrifices the computing efficiency and increases the length of ciphertext. They also advised that any anonymous Identity Based Encryption (IBE) scheme could be transformed to a consistent PEKS scheme. Compared with the previous contribution, the author proposed public key encryption with temporary keyword search by allowing the testing of a keyword across multiple time periods using a single temporary trapdoor for one of the intervals. This is a significant improvement that solves the problem that the trapdoor can be used maliciously all the time. Furthermore, they suggest combining IBE with PEKS to perform identity based encryption with keyword search.

Similarly with Abdalla, Camenisch [CKRS09] suggested using anonymous IBE to achieve public key encryption with oblivious keyword search. From their opinions, PEKS is just like IBE scheme. The receiver should apply trapdoors from a third trusted party (TTP) who keeps the master key. Thus, they designed a blind keyword extraction protocol to protect the receiver’s privacy. Their scheme is suitable for search on encrypted data.

2.3.4 Other Searchable Encryption

Besides oblivious transfer, private information retrieval and public encryption with keyword search (PEKS), there are still some other techniques can perform keyword search. Freedman [FNP04] proposed a scheme to achieve keyword search by using homomorphic encryption. Considered the keywords from sender and receiver as two sets, the search progress can be presented in private match through a homomorphic algorithm without reveal anything to opponent. Bellare et al. [BBO06] presented an efficient searchable encryption that allowed a group member to submit data for one or all members in the group. The motivation looks a bit like what we would like to achieve, though the techniques are thoroughly different. They combine public key encryption with hash function together, making the ciphertext confidential to database server, but allows the database server to mark on it. When the data is being searched, the server can only identify the index of the encrypted data without knowing anything else. Nevertheless, we think it can not be consistent resistant because of the injective method they employed.

In another area, searchable symmetric encryption also fascinates researchers
2.3. Related Works

[CGKO06, SWP00]. Actually, they inherit the form of PEKS. The progress contains keywords encryption, trapdoors creation and tests. Different from PEKS, symmetric encryption adopted the method like what Bellare’s scheme [BBO06]. In that case, database server can reorder and conduct searching operation on encrypted data. That’s the reason why we don’t employ the Bellare’s scheme.

2.3.5 Broadcast Encryption

Broadcast encryption was first proposed by Fiat [FN94], that provides a method for the qualified people to decrypt message that has been sent to all. The application includes TV subscription services, and DVD content protection etc. In Fiat’s scheme, binary tree is utilized to present difference between privileged and revoked member. It guarantees the security against a collusion of t users. Later, an efficient full collusion resistant scheme was proposed by Naor [NNL01], but only can be performed with a small set of revoked member. Further, Dodis [DF03] extended the conception to a large set, and ensured the security to revoke any numbers users smaller than n. However, the running time increased dramatically.

Boneh, Gentry and Waters [BGW05] showed two collusion resistant schemes that are secure against any colluded revoked user. Constant and shorter ciphertext is another advantage of their schemes. Instead of managing the users’ identity by binary tree, they employ adding or reducing group elements to perform authority and revocation. A part of cipher text consist of the product of users’ public keys. Then, only the users whose public keys are in the ciphertext can decrypt message by their private keys. Broadcast encryption aims to restrict qualification to decrypt the message. Conversely, the message only from the qualified people can be searched. It can prevent the trapdoor to be malicious tested efficiently.

2.3.6 Accumulator and Batch Verification

Although some searchable scheme can supply keyword search by single keywords, there is still no any scheme offering a search scheme that can integrate searchable encryptions or test progresses in one element or one time.

The concept of crypto-accumulator was firstly proposed by Benaloh [BdM93] in 1993, which combines two or more values into one element with the constant size. They supposed to use one way hash functions to perform accumulator, which satisfies quasi-communicative property. Their scheme is based on RSA assumption.
and the accumulator only can be computed by central authority. Later, Camenisch [CL02] presented the conception with the dynamic accumulator that allows someone to add and delete a value dynamically. Nguyen [Ngu05] developed a new scheme based on bilinear pairing, though it has been proved that their scheme was not collision resistant [TZL+08]. Camenisch [CKS09] gave an efficient and suitable one. Nevertheless, all the keywords have been encrypted before storing on the database. It is hard to evaluate an encrypted data whether in the accumulator. It is indeed an interesting open problem.

Batch verification was firstly presented by Fiat [Fia89] in 1989. At that time, he presented this concept to reduce the individual private operations of RSA scheme. In 1995, Yen brought the batch verification in digital signature [YL95]. Batch verification attracts people’s eye all the time, and supplies that different signer on different message can be verified very quickly[CHP07]. Moreover, it is suitable for the aim of keyword search.
Chapter 3

Public key encryption with keyword search in designated sender

The public key encryption with keyword search (PEKS) scheme was proposed by Boneh, Di Crescenzo, Ostrovsky and Persiano[BCOP04], which enables one to search the encrypted keywords without revealing the keywords and content. Since the PEKS was proposed, it attracts numerous researchers’ attention. PEKS was originally applied in the e-mail system which can enable gateway test whether an email contains a certain keyword. However, some researcher has proposed perform offline keyword attack to guess the keyword due to that the keyword space is limited. Our schemes allow one to search the encrypted keywords with an encrypted identity and keyword. One of the obvious answers is that the sender can combine his signature with the ciphertext. Nevertheless, the scheme we proposed combines the signature and encryption together and reduces the size of ciphertext. Therefore, though the server and other outside attacker can guess the keyword the receiver wants to search, they still could not forge the encrypted message or trapdoor to perform further attacks.

3.1 Introduction

The public key encryption with keyword search (PEKS) scheme was proposed by Boneh et al. [BCOP04]. They realize the following scenario: suppose Alice is an email user. Due to the development of information technology, Alice can read her email on several devices: desktop, laptop, smart phone, iPad, etc. For convenience, she can set some keywords to her email gateway, so that only the message contain these keywords can be directed to her smart phone. Others will be sent to her desktop or laptop later. For instance, Bob want to send two emails to Alice with the keywords “urgent” and “lunch” respectively and has encrypted using Alice’s public
key. Alice has set the gateway that only emails containing the keyword “urgent” will be sent to her smart phone. Therefore, the former letter will be sent to smart phone and latter will be sent to desktop to be read later. In this scenario, Alice will send a trapdoor to gateway to perform the setting. Nevertheless, the trapdoor isn’t secure enough to prevent the malicious attack. For example, the one who acquires the trapdoor in some way can conduct offline attack to the trapdoor [YHG08]. Roughly speaking, the attack could use brute force to guess the trapdoor, since the keywords space is limited. Once he breaks down the trapdoor, he can send junk mail to Alice with the keywords which are acquired from the cracked trapdoor. Moreover, if the gateway acquires the cracked trapdoor, the message sent in an encrypted form is meaningless for Alice.

In short, the PEKS proposed a method that enables the gateway server to search the senders’ email keywords by using the receivers’ trapdoor. We extend the original scheme with identity based signature scheme to confirm the email’s sender. Our scheme not only filters the keyword and sender, but also requires senders’ authentication.

Along with the development of information technology, the keyword search is always an interesting issue that attracts people’s eyes. Before the public encryption with keyword search, there was private information retrieval (PIR) [CKGS98], oblivious transfer (OT) [Rab81] and searchable symmetric encryption applied on the database data. In the literature, the PIR was introduced by Chor, Goldreich, Kushilevitz and Sudan in 1995, which allows a person search on a database without revealing the items he has searched. However, the PIR doesn’t protect the database’s privacy. In 2002, Ogata and Kurosawa proposed an oblivious keyword search scheme that uses OT into keyword search. The aim of the scheme is to protect both searcher and database’s privacy [OK04]. The searcher doesn’t reveal any information to database server, and the database server doesn’t reveal any information except the user’s search item. Nevertheless, it costs the communication efficiency. Mike Freedman et al. proposed a keyword search scheme based on a pseudo-random function, it improves from both PIR and OT and achieves more efficient on communication [FIPR05]. It achieves search numbers of keyword in a single trapdoor. However the PEKS performs public key encryption and reduces extra communications between sender and keyword search performer for particular keyword, there are still some parts vulnerable in the scheme. The offline attack is a significant problem to PEKS. Therefore, many researchers have focused on this issue. Abdalla,
Bellare, Catalano and Kiltz proposed a public encryption with temporary keyword search to address this problem [ABC+05]. They use the technique of hierarchical identity based encryption to perform the temporary keyword search. In their opinion, they distribute trapdoors in different time interval. Hence, even though one of the trapdoors has been broken down, Other trapdoors can be still used safely. Baek, Safavi-Naini and Susilo has proposed several schemes [BSNS05][BSNS06]. One of them enables receiver sends trapdoor without a security channel, others extends PEKS against chosen ciphertext attack and also extends PEKS to multi-receiver. Moreover, Park et al. [PKL05] and Hwang et al. [HL07] extends PEKS to search conjunctive keywords respectively.

3.1.1 Contributions of This Chapter

We concern the problem of the fact that PEKS is vulnerable by the offline attack [YHG08]. Although some people have proposed solutions, we address the problem in another way, but no less security. The target of our scheme is that receiver can search the encrypted keyword in a list which only includes the people he trusts and he wants. Therefore, the offline attack can not be conducted on our scheme. At the same time, in our scheme, the senders will not reveal any information about their identities and keywords. The receivers will not reveal their targets and searched keywords. Furthermore, we think our scheme can be used in cloud computing to perform the keyword search.

This chapter is structured as follows. We describe the keyword search in designated sender scheme and security notion in Section 2. Then, in Section 3, we will give the detail of the scheme and the security proof. Some applications will be showed in section 4 and we will give conclusion in Section 5.

3.2 General Structure

We begin by formally defining public key encryption with keyword search in designate senders. There are three parties in our public key encryption in designated sender (PEKSDS) scheme: sender, receiver and server. The purpose of PEKSDS is constructing a secure searchable system which enables receiver only search among the senders he trusts. Both outside sender and server have no capabilities to forge
an encryption that receiver trusts and the encryption is secure against indistinguishable chosen keyword attack.

Next, we will precisely define the work procedure. Firstly, the sender will send the encrypted files with keywords $W_1, ..., W_k$, and his identity in the ciphertext. The message will be sent as follows:

$$[E_{A_{pub}}[msg], \text{PEKSDS}(A_{pub}, d_{ID}, W_1), ..., \text{PEKSDS}(A_{pub}, d_{ID}, W_k)]$$

where $A_{pub}$ is receiver’s public key, $d_{ID}$ is sender’s secret key, $msg$ is the message body, PEKSDS is the scheme we will discuss below. Then, the receiver will send a secret trapdoor $T_W$ to server which includes the keywords and sender’s identity. Finally, the server will locate the message that contains the certain keywords from the designate sender and send the encrypted message back to the receiver.

At the end of this section, we will state the complexity assumption needed for our proof of security.

### 3.2.1 Definition of PEKSDS

**Definition 3.1 (PEKSDS)** A public encryption with keyword search in designated senders is made of six randomized algorithms:

- **SysPara($k$):** Taking a security parameter $k \in \mathbb{N}$, it generates a system parameters $sp$.
- **KeyGenReceiver($sp$):** Taking system parameters $sp$ as input, it generates receiver’s key pair $A_{priv}/A_{pub}$.
- **KeyGenPKG($sp$):** Taking system parameters $sp$ as input, it generates a master secret key $Msk$ and public key $P_{pub}$ for all senders.
- **KeyGenSender($sp$):** Taking system parameters $sp$ and master key $Msk$ as input, it generates sender’s secret key $d_{ID}$ according to the sender’s ID.
- **Encryption($A_{pub}, d_{ID}, W$):** Taking $A_{pub}$ and sender’s secret key $d_{ID}$, it produces a searchable encryption of $W$.
- **Trapdoor($A_{priv}, W, ID$):** Taking receiver’s private key, a sender’s identity $ID$, and keyword $W$ as input, receiver produces a trapdoor $T_W$.
- **Test($T_W, P_{pub}, Enc$);** Given the searchable encryption $Enc(A_{pub}, d_{ID}, W)$, trapdoor $T_W(A_{priv}, W', ID)$ and $P_{pub}$. Output ‘yes’ while both $W = W'$ and sender is designated, or ‘no’ otherwise.
3.2.2 Security Model

We define PEKSDS is semantically secure against a static adversary. The encryption possess security we called “indistinguishability against chosen keyword and identity attack” (IND-CIA-CKA), “unforgeability against chosen identity attack” (UF-CIA) and “unforgeability against chosen Trapdoor attack” (UF-CTA). Let algorithm $A_i$ ($i = 1, 2, 3$) be an attacker who wants to attack PEKSDS system bounded by polynomial time $t$. To illustrate security for PEKSDS, we define three games between $A_i$ and a challenger as follows:

Game 1: $A_1$ aims to distinguish PEKSDS by chosen keyword and identity attack.

**Setup & KeyGen:** The challenger runs $\text{SysPara}(k)$ to obtain system parameters and generate key pairs $A_{pub}/A_{priv}$ and $P_{pub}/Msk$ separately. Then sends $A_{pub}$ and $P_{pub}$ to $A_1$.

**Trapdoor query phase 1:** $A_1$ adaptively issues queries $q_1, \ldots, q_m$ for the trapdoor $T_W$ for keywords $W_i \in \{0,1\}^*$ and $ID_i \in \{0,1\}^*$, where $1 \leq i \leq m$. The challenger responds with $T_W(A_{priv}, W_i, ID_i)$ to $A_1$.

**Challenge:** $A_1$ sends two pairs $(W_0, ID_0)$ and $(W_1, ID_1)$ to challenger on which it wishes to be challenged on. Next, the challenger picks a random $\beta \in \{0,1\}$. It sets $\text{Enc}_\beta = \text{Enc}(A_{pub}, d_{ID_\beta}, W_\beta)$ and gives it $A_1$. The only requirement is that $A_1$ hasn’t queried pair $(W_0, ID_0)$ and $(W_1, ID_1)$ for trapdoor.

**Trapdoor query phase 2:** $A_1$ continues to adaptively issue pairs $(W_i, ID_i)$ for trapdoor. The only constraint is that $W_i \neq W_0, W_1$ and $ID_i \neq ID_0, ID_1$. The challenger responds the same as in phase 1.

**Guess:** Algorithm $A_1$ outputs its guess $\beta' \in \{0,1\}$ for $\beta'$ and wins the game if $\beta = \beta'$.

We refer to such an adversary $A_1$ as IND-CIA-CKA adversary. We defined the advantage of $A_1$ is attacking the scheme as

$$Adv_{A_1} = \left| \Pr[\beta = \beta'] - \frac{1}{2} \right| .$$

Game 2: $A_2$ aims to forge a PEKSDS to act as a sender who receiver trusts. Let $A_2$ be an attacker who can obtain the keyword $W$ which receiver wants to search.
Setup & KeyGen: The challenger runs \( \text{SysPara}(k) \) to obtain system parameters and generate key pairs \( A_{pub}/A_{priv} \) and \( P_{pub}/\text{senders' } master \) key separately. Then sends \( A_{pub} \) and \( P_{pub} \) to \( A_2 \).

Encryption Query: To forge the sender who receiver trusts, \( A_2 \) issues encryption queries \( q_1, ..., q_m \) of keyword \( W_i \) and \( ID_i \), where \( 1 \leq i \leq m \). The challenger responds with \( \text{Enc}(A_{pub}, d_{ID_i}, W_i) \).

Forgery: \( A_2 \) announces an identity he wants to forgery, and outputs an \( \text{Enc}(A_{pub}, d_{ID}, W) \) where \( W \) is the keyword it intends to forge. \( A_2 \) wins the game if \( \text{Enc}(A_{pub}, d_{ID}, W) \) is valid encryption.

We refer to such an adversary \( A_2 \) as UF-CIA adversary. We defined the advantage of \( A_2 \) is attacking the scheme as

\[
\text{Adv}_{A_2} = \left| \Pr[\text{Verify}(\text{PEKSDS})] = \text{valid} \right|
\]

Game 3: \( A_3 \) aims to forge a trapdoor.

Setup & KeyGen: The challenger runs \( \text{SysPara}(k) \) to obtain system parameters and generate key pairs \( A_{pub}/A_{priv} \). Then sends \( A_{pub} \) to \( A_3 \).

Trapdoor Query: \( A_3 \) adaptively issues queries \( q_1, ..., q_m \) for keywords \( W_i \) and \( ID_i \), where \( 1 \leq i \leq m \). The challenger responds with the \( T_W(A_{priv}, W_i, ID_i) \) to \( A_3 \).

Forgery: \( A_3 \) outputs a forged trapdoor \( T_W(A_{priv}, W, ID) \). \( A_3 \) wins the game if \( (A_{priv}, W, ID) \) is valid trapdoor.

We refer to such an adversary \( A_3 \) as UF-CTA adversary. We defined the advantage of \( A_3 \) is attacking the scheme as

\[
\text{Adv}_{A_3} = \left| \Pr[\text{Verify}(\text{Trapdoor})] = \text{valid} \right|
\]

Definition 3.2 Our PEKS scheme is \((t, q, \epsilon)\)-semantically secure, if there exists no adversary \( A_i \), \((i \in 1, 2, 3)\), who runs in at most \( t \) time and queries at most \( q \) times, the probability at least \( \epsilon \).

3.3 Construction

3.3.1 PEKSDS System

The PEKSDS scheme consists of the following algorithm:
SysPara: The algorithm chooses two multiplicative cyclic groups $\mathbb{G}, \mathbb{G}_T$ of prime order $p$ for some large prime number to construct a bilinear map $\hat{e}: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Then, it picks a random generator $g \in \mathbb{G}$. We will need two hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{G}$. The system parameters are $(\mathbb{G}, \mathbb{G}_T, p, \hat{e}, g, H_1, H_2)$.

KeyGen$_{Receiver}$: The algorithm picks a random $\alpha \in \mathbb{Z}_p^*$, it computes $g^\alpha = h \in \mathbb{G}$. $(g,h)$ is published as receiver’s public key. $\alpha$ is kept by receiver as private key.

KeyGen$_{PKG}$: The algorithm picks a random $s \in \mathbb{Z}_p^*$, $s$ is set as master secret key $Msk$. It computes $g^s \in \mathbb{G}$, $g^s$ is published as $P_{pub}$.

KeyGen$_{Sender}$: The secret key $d_{ID}$ for individual sender is set as $H_2(ID)^s$.

Encryption: Sender picks a random number $r \in \mathbb{Z}_p^*$, uses his secret key $H_2(ID)^s$, receiver’s public key $h$, and keyword $W$, to compute

$$(C_1, C_2) = (H_2(ID)^s \cdot H_1(W)^r, h^r) \in \mathbb{G}^2$$

and transfers $(C_1, C_2)$ to the server.

Trapdoor: Receiver picks a random $t \in \mathbb{Z}_p^*$, uses his private key $\alpha$ and sender’s identity $ID$ to compute:

$$(T_1, T_2, T_3) = (H_1(W^t)^{t/\alpha}, H_2(ID)^t, g^t) \in \mathbb{G}^3$$

and sends $(T_1, T_2, T_3)$ to the server.

Test: Then, the server will search the keyword user by user according to $P_{pub}$ ($(g^s)$), and test:

$$\hat{e}(C_1, T_3) = \hat{e}(T_2, C_3) \cdot \hat{e}(T_1, C_2)$$

If the result is true, return the content corresponding to the keyword. Else, nothing.

Correctness: We first verify that the system is correct, namely that the test algorithm works correctly. Server computes as:

$$\hat{e}(C_1, T_3) = \hat{e}(T_2, C_3) \cdot \hat{e}(T_1, C_2)$$

$$= \hat{e}(H_2(ID)^s \cdot H_1(W)^r, h^r)$$

$$= \hat{e}(H_2(ID)^s, g^t) \cdot \hat{e}(H_1(W)^r, g^t)$$

$$= \hat{e}(H_2(ID)^t, g^s) \cdot \hat{e}(H_1(W)^{t/\alpha}, h^r)$$

$$= \hat{e}(T_2, C_3) \cdot \hat{e}(T_1, C_2)$$
3.3.2 Security Analysis

We now present the security proof of our PEKSDS scheme.

**Theorem 3.1** Let $\mathcal{G}$ be a multiplicative bilinear group of prime order $p$. Our system is $(t, \epsilon, q_{T}, q_{H_2})$ semantically secure against indistinguishable chosen keyword and identity attack (IND-CIA-CKA) assuming the $(t, \epsilon, q_{T}, q_{H_2})$-DLDH assumption holds in $\mathcal{G}$.

**Proof:** Let algorithm $\mathcal{A}_1$ be an attacker bounded in a polynomial time $t$, who has advantage $\epsilon$ in breaking PEKSDS. $\mathcal{A}_1$ can perform at most $q_{T}$ trapdoor queries, $q_{H_1}$, $q_{H_2}$ hash function queries to $H_1$ and $H_2$ separately. Let $\mathcal{B}$ be an algorithm who aims to solve DLDH problem with the probability at least $\epsilon' = \frac{\epsilon}{e^{q_{T}q_{H_2}}}$, and its running time is similar to $\mathcal{A}_1$. Thus, if DLDH problem holds in $\mathcal{G}$, then $\epsilon'$ is a negligible function. Consequently, $\epsilon$ is a negligible probability for $\mathcal{A}_1$ to break the PEKSDS scheme.

Let $u_1, u_2, u_3$ be generators of $\mathcal{G}$. Algorithm $\mathcal{B}$ is given $u_1, u_2, u_3, u_1^a, u_2^b, \eta \in \mathcal{G}$ as input, to determine whether $\eta$ equals to $u_3^{a+b}$ or a random element in $\mathcal{G}$. $\mathcal{B}$ proceeds as follows:

**Setup & KeyGen:** Algorithm $\mathcal{B}$ starts by setting $g = u_1$ to be the system parameter. To simulate receiver, algorithm $\mathcal{B}$ sets $h = u_2$. In another word, $(g, h)$ is receiver’s public key, and $\alpha$ where $h = g^\alpha$ is set to be receiver’s secret key. On the other hand, $\mathcal{B}$ sets $s = a$ to be master secret key $Msk$. Therefore, $P_{pub}$ is $g^s$ where $g^s = u_1^a$ and sender’s secret key $d_{ID}$ is $H_2(ID)^a$. Finally, $\mathcal{B}$ gives $(u_1, u_2, u_1^a)$ to the adversary $\mathcal{A}_1$.

**$H_1, H_2$-queries:** At any time, $\mathcal{A}_1$ can issue at most $q_{H_1}$ and $q_{H_2}$ queries to random oracles $H_1$ and $H_2$ separately. $\mathcal{B}$ simulates the responds. Two lists $\langle W_i, h_i, x_i, c_1 \rangle$ and $\langle ID_i, f_i, y_i, c_2 \rangle$ are maintained by $\mathcal{B}$ as the random oracle queries. The former one is called $H_1$ list and the later one is $H_2$ list. They are empty at the beginning. The random oracle query procedure is as follows:

- **$H_1$ queries:**
  - When receive a query for keyword $W_i$, $\mathcal{B}$ responds as $H_1(W_i) = h_i \in \mathcal{G}$ if $W_i$ is already in $H_1$ list.
• Otherwise, $B$ will flip a coin $c_1 \in \{0, 1\}$ with $\Pr[c_1 = 0] = \delta_1$.

\[
\begin{cases}
  \text{If } c_1 = 0, & h_i = u_3^{x_i} \in \mathbb{G}, \quad x_i \in R \mathbb{Z}_p^* \\
  \text{If } c_1 = 1, & h_i = u_2^{x_i} \in \mathbb{G}, \quad x_i \in R \mathbb{Z}_p^*
\end{cases}
\]

• At last, $B$ adds $\langle ID_i, h_i, x_i, c_1 \rangle$ to $H_1$ list, then sends the value $h_i$ to adversary $A_1$ as the result of random oracle query.

$H_2$ queries:

• When receive a query for identity $ID_i$, $B$ responds as $H_2(ID) = f_i \in \mathbb{G}$, if $ID_i$ is already in $H_2$ list.

• Otherwise, $B$ will flip a coin $c_2 \in \{0, 1\}$ with $\Pr[c_2 = 0] = \delta_2$.

\[
\begin{cases}
  \text{If } c_2 = 0, & f_i = u_3^{y_i} \in \mathbb{G}, \quad y_i \in R \mathbb{Z}_p^* \\
  \text{If } c_2 = 1, & f_i = u_1^{y_i} \in \mathbb{G}, \quad y_i \in R \mathbb{Z}_p^*
\end{cases}
\]

• At last, $B$ adds $\langle ID_i, f_i, y_i, c_2 \rangle$ to $H_2$ list, then sends the value $f_i$ to adversary $A_1$ as the result of random oracle query.

**Trapdoor query phase 1:** $A_1$ can choose a keyword $W_i$ and an identity $ID_i$ to $B$ to perform trapdoor queries. $B$ will responds as follows:

1. For every trapdoor query, the algorithm above will be run by $B$ to derive $H_1$ and $H_2$ list such as $\langle W_i, h_i, x_i, c_1 \rangle$ and $\langle ID_i, f_i, y_i, c_2 \rangle$. If $c_1 = 0$ or $c_2 = 0$, then $B$ claims failure and terminates.

2. Otherwise, $B$ will obtain $h_i = u_2^{x_i}$ and $f_i = u_1^{y_i}$. In addition, it randomly picks $t \in \mathbb{Z}_p^*$. Then, we can simulate the trapdoor as:

\[
(T_{i1}, T_{i2}, T_{i3}) = (u_1^{x_{ti}}, u_1^{y_{ti}}, u_1^{t})
\]

Since $h = g^\alpha$, we have:

\[
\begin{align*}
  u_1^{x_{ti}} &= g^{x_{ti}} = h^{x_{ti}/\alpha} = u_2^{x_{ti}/\alpha} = h_i^{t/\alpha} = H_1(W_i)^{t/\alpha} \\
  u_1^{y_{ti}} &= f_i^t = H_2(ID_i)^t \\
  u_1^t &= g^t
\end{align*}
\]
Challenge: Eventually algorithm $A_1$ produces two pairs of keywords and identities $\langle W_0, ID_0 \rangle$ and $\langle W_1, ID_1 \rangle$ that it wishes to be challenged on. The only requirement is that pairs $\langle W_0, ID_0 \rangle$, $\langle W_1, ID_1 \rangle$ haven’t been queried before. Algorithm $B$ generates the challenge PEKSDS as follows:

1. Algorithm $B$ runs the above algorithm for responding to $H_1$-queries and $H_2$-queries twice. If neither of pair $(c_{1_0}, c_{2_0})$ and $(c_{1_1}, c_{2_1})$ both equal 0, then $B$ claims failure and terminates.

2. Otherwise, at least one of the pair $(c_{1_0}, c_{2_0})$ and $(c_{1_1}, c_{2_1})$ equal to 0. $B$ picks $i \in \{0, 1\}$ randomly, and sets pair $(W_i, ID_i) = (W_i, ID_i)$, where both $(c_{1_i}, c_{2_i})$ equal to 0. Note, if only one pair $(c_{1_i}, c_{2_i})$ equals to 0, there is no randomized. Let $\langle W_i, h_i, x_i, c_i \rangle$ and $\langle ID_i, f_i, y_i, c_i \rangle$ be the corresponding tuples on the $H_1$-list and $H_2$-list separately.

3. Algorithm $B$ computes $z = \frac{y_i}{x_i}$. In another form, $x_i = \frac{y_i}{z}$. Then $B$ sets $r = bz$ and responds with the challenge PEKS as:

   $$(C_1, C_2) = (\eta^{y_i}, u_2^{bz})$$

If $\eta = u_3^{a+b}$, the following equation is true:

   $$\eta^{y_i} = u_3^{(a+b)y_i} = u_3^{ay_i} \cdot u_3^{by_i} = (u_3^{y_i})^a \cdot (u_3^{x_i})^b = (u_3^{y_i})^a \cdot (u_3^{x_i})^b$$
   
   $$= f_1^a \cdot h_i^{bz} = H_2(ID)^s \cdot H_1(W)^r$$
   
   $$u_2^{bz} = h^{bz} = h^r$$

If $\eta \neq u_3^{a+b}$, $C_1 = \eta^{y_i}$ is only a random element in $G$.

Trapdoor query phase2: $A_1$ can continue to issue trapdoor queries for keywords $W_i$ where $W_i \neq W_0, W_1$ and $ID_i \neq ID_0, ID_1$. $B$ will response as before.

Guess: $A_1$ will guess $\beta' \in \{0, 1\}$ for $\beta$. If $\beta = \beta'$, $B$ will output $\eta = u_3^{a+b}$. Otherwise, it means $\eta$ is a random element in $G$.

Next, we will explain the probability that $B$ can output the correct answer. The procedure should not abort during trapdoor queries and challenge phase. As the result of [BCOP04], we let the probability of coin equals 0 be $Pr[c_i = 0] = \delta$. It should satisfy that the function $(1 - \delta)^q \cdot \delta$ would have the largest value. To optimize the value, $\delta_{opt} = \frac{1}{q_r + 1}$. Therefore $\delta_2 = \frac{1}{q_r^2 + 1}$, $\delta_2 = \frac{1}{q_r^2 + 1}$. Therefore, the probability that $B$ doesn’t abort during the trapdoor queries phase is $(1 - \frac{1}{q_r^2 + 1})^{q}$.\]
(1 - \frac{1}{q_{H_2}+1})^{q_{H_2}} \geq \frac{1}{e} \text{ where } e \text{ is the base of nature logarithm. Subsequently, when perform oracle queries for pairs } (W_0, ID_0), (W_1, ID_1), \text{ both } (c_{i_0}, c_{i_0}) \text{ and } (c_{i_1}, c_{i_1}) \text{ equal to 1. Algorithm } B \text{ will abort during challenge phase. The probability is } (1 - \frac{1}{q_{H_1}+1})^2 \cdot (1 - \frac{1}{q_{H_2}+1})^2 \leq (1 - \frac{1}{q_{H_1}+1})(1 - \frac{1}{q_{H_2}+1}). \text{ Moreover, the probability that } B \text{ doesn’t abort until challenge phase is at least } \frac{1}{e^{q_{H_1}q_{H_2}}}. \text{ Hence, the overall of } B \text{ succeeding is } \frac{e}{e^{q_{H_1}q_{H_2}}} \text{ as required. Then proof of theorem is completed.}

**Theorem 3.2** Let \( G \) be a multiplicative bilinear group of prime order p. Our system is \((t, \epsilon, q_{H_1}, q_{H_2})\) unforgeable secure against chosen identity attack (UF-CIA) assuming the \((t, \epsilon, q_{H_1}, q_{H_2})\)-CDH assumption holds in \( G \).

**Proof:** Let algorithm \( A_2 \) be an attacker bounded in a polynomial time \( t \), who has advantage \( \epsilon \) in breaking PEKSDS. \( A_2 \) can perform at most \( q_T \) trapdoor queries, \( q_{H_1}, q_{H_2} \) hash function queries to \( H_1 \) and \( H_2 \) separately. Let \( B \) be an algorithm who aims to solve CDH problem with the probability at most \( \epsilon' \), and its running time is similar to \( A_2 \). Thus, if CDH problem holds in \( G \), then \( \epsilon' \) is a negligible function. Consequently, \( \epsilon \) is a negligible probability for \( A_2 \) to break the PEKSDS scheme.

Let \( g \) be a generator of \( G \), \( a, b \) be random numbers in \( \mathbb{Z}_p^* \). Algorithm \( B \) is given \( g, g^a, g^b \in G \) as input, and aims to compute \( g^{ab} \). \( B \) proceeds as follows:

**Setup & KeyGen:** Algorithm \( B \) starts by setting \( g \) to be the system parameter. To simulate sender, algorithm \( B \) sets \( g^s = g^a \). In another word, \( g^s \) is \( P_{\text{pub}} \), \( a \) is set to be senders’ master key and sender’s secret key \( d_{ID} \) is \( H_2(ID)^a \). To simulate receiver, \( B \) picks a generator \( h \in G \) randomly, where \( h = g^\alpha \). \( (g, h) \) is set to be receiver’s public key. \( \alpha \) to be set as receiver’s private key. Finally, \( B \) gives \((g, h, g^a)\) to the adversary \( A_2 \).

**H_1, H_2-queries:** At any time, \( A_2 \) can issue at most \( q_{H_1} \) and \( q_{H_2} \) queries to random oracles \( H_1 \) and \( H_2 \) separately. \( B \) simulates the responds. Two lists \((W_i, h_i, x_i)\) and \((ID_i, f_i, y_i, c_i)\) is maintained by \( B \) as the random oracle queries. The former one is called \( H_1 \) list and the later one is \( H_2 \) list. They are empty at the beginning. The random oracle query procedures are as follows:

**H_1 queries:** When receive a query for keyword \( W_i \), algorithm \( B \) responds as \( H_1(W) = h_i \in \mathbb{G}_1 \) if \( W_i \) is already in \( H_1 \) list. Otherwise, randomly pick \( x_i \in \mathbb{Z}_p^* \), and set \( h_i = h^{x_i} \). Then \( B \) responds \( H_1 \) query as \( h_i \), and adds \((W_i, h_i, x_i)\) to the \( H_1 \) list.
3.3. Construction

$H_2$ queries:

- When receive a query for identity $ID_i$, $B$ responds as $H_2(ID) = f_i \in \mathbb{G}$, if $ID_i$ is already in $H_2$ list.

- Otherwise, $B$ will flip a coin $c_i$, where $c_i \in \{0, 1\}$ with $\Pr[c_i = 0] = \delta$.

\[
\begin{align*}
\text{If } & c_i = 0, \quad f_i = g^b \cdot g^{y_i} \in \mathbb{G}, \quad y_i \in R \mathbb{Z}_p^* \\
\text{If } & c_i = 1, \quad f_i = g^{y_i} \in \mathbb{G}, \quad y_i \in R \mathbb{Z}_p^*
\end{align*}
\]

- At last, $B$ adds $(ID_i, f_i, y_i, c_i)$ to $H_2$ list, then sends the value $f_i$ to adversary $A_2$ as the result of random oracle query.

Encryption Query: $A_2$ can choose a keyword $W_i^*$ and an identity $ID_i$ to $B$ to perform encryption queries. Where the identity is belonged to the sender who receiver trusts. $B$ will responds as follows:

1. For every encryption query, the algorithm above will be run by $B$. If $c_i = 0$, then $B$ claims failure and terminates.

2. Otherwise, $B$ will obtain $h_i = h^{x_i}$ and $f_i = g^{y_i}$. It randomly picks $r \in \mathbb{Z}_p^*$, and sets:

\[
(C_{i_1}, C_{i_2}) = (g^{x_i r} \cdot g^{y_i a}, h_i^r)
\]

and we have:

\[
g^{x_i r} \cdot g^{y_i a} = f_i^a \cdot h_i^r = H_2(ID)^s \cdot H_1(W)^r
\]

Forgery: Eventually algorithm $A_2$ announces an identity $ID^*$ and a keyword $W^*$ and output the forgery $(C_1, C_2)$. We assume that $ID^*$ and $W^*$ have been queried before. Then, if $H_1(W^*) = h_i = h^{x_i}$ and $H_2(ID^*) = f_i = g^b \cdot g^{y_i}$, $A_2$ forge the encryption as:

\[
(C_1, C_2) = (h^{x_i r} \cdot g^{ab} \cdot g^{ay_i}, h_i^r)
\]

Moreover, $B$ can output $g^{ab} = \frac{C_1}{(g^a)^{x_i} \cdot (h_i)^r} = \frac{h^{x_i r} \cdot g^{ab} \cdot g^{ay_i}}{(g^a)^{x_i} \cdot (h_i)^r}$

Next, we will explain the probability that $B$ can output the correct answer. The procedure should not abort during the queries phase. As we already known, $\delta = \frac{1}{q_{H_2} + 1}$. Therefore, the probability that $B$ doesn’t abort during the queries phase is 

\[
(1 - \frac{1}{q_{H_2} + 1})q_{H_2} \geq \frac{1}{\epsilon}
\]

where $\epsilon$ is the base of nature logarithm. Subsequently, in the forgery phase, if we want to calculate $g^{ab}$, we need to find $H_1(W) = h^{x_i}, H_2(ID^*) = h^{x_i}$.
3.3. Construction

g^b \cdot g^{u_i} in the \( H_1 \) and \( H_2 \) lists separately. As the result, the probability will be \( 2/(q_{H_1} + q_{H_2}) \). Hence, the overall of \( \mathcal{B} \) succeeding is \( \frac{2\varepsilon}{\epsilon(q_{H_1} + q_{H_2})} \) as required. Then proof of theorem is completed.

**Theorem 3.3** The trapdoor of our scheme is semantically secure assuming CDH and DCDH assumption holds in \( \mathbb{G} \).

*Proof:* Assume that \( \mathcal{A}_3 \) is the algorithm who can forge the trapdoor of our PEKSDS scheme. We describe the reduction by falling into two types of attacks:

1. Given the keyword part of trapdoor, \( \mathcal{A}_3 \) can forge a valid trapdoor which includes other identity.

2. Given the identity part of trapdoor, \( \mathcal{A}_3 \) can forge a valid trapdoor which includes other keyword.

In the lemma 1 and lemma 2, we reduce the attacks to solve the CDH assumption and DCDH assumption respectively. This concludes the security proof of trapdoor of our PEKSDS scheme.

**Lemma 3.4** The trapdoor of our scheme is \( (t, \varepsilon, q_{H_2}) \) semantically secure assuming \( (t, \varepsilon, q_{H_2}) \)-CDH assumption holds in \( \mathbb{G}_1 \).

*Proof:* Let algorithm \( \mathcal{A}_{3_1} \) be an attacker bounded in a polynomial time \( t \), who has advantage \( \varepsilon \) in breaking our PEKS. \( \mathcal{A}_{3_1} \) can perform at most \( q_T \) trapdoor queries, \( q_{H_1}, q_{H_2} \) hash function queries to \( H_1 \) and \( H_2 \) separately. Let \( \mathcal{B} \) be an algorithm who aims to solve CDH problem with the probability at most \( \varepsilon' \), and its running time is similar to \( \mathcal{A}_{3_1} \). Thus, if CDH problem holds in \( \mathbb{G} \), then \( \varepsilon' \) is a negligible function. Consequently, \( \varepsilon \) is a negligible probability for \( \mathcal{A}_{3_1} \) to break the new PEKS scheme.

Let \( g \) be a generator of \( \mathbb{G} \), \( a, b \) be random numbers in \( \mathbb{Z}_p^* \). Algorithm \( \mathcal{B} \) is given \( g, g^a, g^b \in \mathbb{G} \) as input, and aims to compute \( g^{ab} \). \( \mathcal{B} \) proceeds as follows:

**Setup & KeyGen:** Algorithm \( \mathcal{B} \) starts by setting \( g \) to be the system parameter. To simulate receiver, \( \mathcal{B} \) picks a generator \( h \in \mathbb{G} \) randomly, where \( h = g^\alpha \). \( (g, h) \) is set to be receiver’s public key. \( \alpha \) to be set as receiver’s private key. Finally, \( \mathcal{B} \) gives \( (g, h) \) to the adversary \( \mathcal{A}_{3_1} \).

**\( H_1, H_2 \)-queries:** At any time, \( \mathcal{A}_{3_1} \) can issue at most \( q_{H_1} \) and \( q_{H_2} \) queries to random oracles \( H_1 \) and \( H_2 \) separately. \( \mathcal{B} \) simulates the responds. Two lists \( \langle W_i, h_i, x_i \rangle \) and
3.3. Construction

\( \{ID_i, f_i, y_i, c_i\} \) is maintained by \( B \) as the random oracle queries. The former one is called \( H_1 \) list and the later one is \( H_2 \) list. They are empty at the beginning. The random oracle query procedures are as follows:

**\( H_1 \) queries:** When receive a query for keyword \( W_i \), algorithm \( B \) responds as \( H_1(W_i) = h_i \in \mathbb{G}_1 \) if \( W_i \) is already in \( H_1 \) list. Otherwise, it randomly picks \( x_i \in \mathbb{Z}_p^* \), and set \( h_i = h_i^{x_i} \). Then \( B \) responds \( H_1 \) query as \( h_i \), and adds \( \langle W_i, h_i, x_i \rangle \) to the \( H_1 \) list.

**\( H_2 \) queries:**

- When receive a query for identity \( ID_i \), \( B \) responds as \( H_2(ID_i) = f_i \in \mathbb{G}_1 \) if \( ID_i \) is already in \( H_2 \) list.

- Otherwise, \( B \) will flip a coin \( c_i \), where \( c_i \in \{0, 1\} \) with \( \Pr[c_i = 0] = \delta \).

\[
\begin{cases}
\text{If } c_i = 0, & f_i = g^b \cdot g^{y_i} \in \mathbb{G}, \quad y_i \in \mathbb{R} \mathbb{Z}_p^* \\
\text{If } c_i = 1, & f_i = g^{y_i} \in \mathbb{G}, \quad y_i \in \mathbb{R} \mathbb{Z}_p^*
\end{cases}
\]

- At last, \( B \) adds \( \langle ID_i, f_i, y_i, c_i \rangle \) to \( H_2 \) list, then sends the value \( f_i \) to adversary \( A_3 \) as the result of random oracle query.

**Trapdoor Query:** \( A_3 \) can choose a keyword \( W_i \) and an identity \( ID_i \) to \( B \) to perform trapdoor queries. \( B \) will responds as follows:

1. For every trapdoor query, the algorithm above will be run by \( B \). If \( c_i = 0 \), then \( B \) claims failure and terminates.

2. Otherwise, \( B \) will obtain \( h_i = h_i^{x_i} \) and \( f_i = g^{y_i} \). It randomly picks \( t \in \mathbb{Z}_p^* \). It simulates the trapdoor as:

\[
(T_{11}, T_{12}, T_{13}) = (g^{x_i t}, g^{y_i t}, g^t)
\]

Since \( h = g^a \), we have:

\[
\begin{align*}
g^{x_i t} &= h_i^{x_i t / \alpha} = h_i^{t / \alpha} = H_1(W)^{t / \alpha} \\
g^{y_i t} &= H_2(ID)^t
\end{align*}
\]

**Forgery:** Eventually algorithm \( A_3 \) acquires a trapdoor from \( B \), where \( B \) sets \( t = a \). Note: \( g^{x_i a} = h_i^{x_i t / \alpha} = H_1(W)^{t / \alpha} \), \( g^a = g^t \).

\[
(T^{*}_{1}, T^{*}_{3}) = (g^{x_i a}, g^a)
\]
Then, $A_{3\infty}$ announces an identity $ID^*$ that will be the target to forged. We assume that $ID^*$ has been queried before. Then, if $H_2(ID) = f_i = g^b \cdot g^{yi}, A_3$, forge the trapdoor identity part as 

$$T_2' = f_i^a = g^{ab} \cdot g^{ay_i}$$

Moreover, $B$ can output $g^{ab} = T_2 = g^{a} \cdot g^{ay_i}$. 

Next, we will explain the probability that $B$ can output the correct answer. The procedure should not abort during the queries phase. As we already known, $\delta = \frac{1}{q_{H_2} + 1}$. Therefore, the probability that $B$ doesn’t abort during the queries phase is $(1 - \frac{1}{q_{H_2} + 1})^{q_{H_2}} \geq \frac{1}{e}$ where $e$ is the base of nature logarithm. Subsequently, in the forgery phase, if we want to calculate $g^{ab}$, we need to find $H_2(ID) = g^b \cdot g^{yi}$ in $H_2$ list. As the result, the probability will be $1/q_{H_2}$. Hence, the overall of $B$ succeeding is $\frac{1}{q_{H_2}}$ as required. Then proof of theorem is completed.

**Lemma 3.5** The trapdoor of our scheme is $(t, \epsilon, q_{H_1})$ semantically secure assuming $(t, \epsilon, q_{H_1})$-DCDH assumption holds in $G$.

*Proof:* Let algorithm $A_{3_2}$ be an attacker bounded in a polynomial time $t$, who has advantage $\epsilon$ in breaking PEKSDS. $A_{3_2}$ can perform at most $q_T$ trapdoor queries, $q_{H_1}$, $q_{H_2}$ hash function queries to $H_1$ and $H_2$ separately. Let $B$ be an algorithm who aims to solve DCDH problem with the probability at most $\epsilon'$, and its running time is similar to $A_{3_2}$. Thus, if DCDH problem holds in $G_1$, then $\epsilon'$ is a negligible function. Consequently, $\epsilon$ is a negligible probability for $A_{3_2}$ to break the PEKSDS scheme.

Let $g$ be a generator of $G_1$, $a, b$ be random numbers in $\mathbb{Z}_p^{\ast}$. Algorithm $B$ is given $g, g^a, g^b \in G$ as input, and aims to compute $g^{zb}$. $B$ proceeds as follows:

**Setup & KeyGen:** Algorithm $B$ starts by setting $g$ to be the system parameter. To simulate receiver, $B$ picks a generator $g \in G_1$ randomly, where $h = g^a$. $(g, h)$ is set to be receiver’s public key. $a (= \alpha)$ is set as receiver’s private key. Finally, $B$ gives $(g, g^a, h)$ to the adversary $A_{3_2}$.

**$H_1, H_2$-queries:** At any time, $A_{3_2}$ can issue at most $q_{H_1}$ and $q_{H_2}$ queries to random oracles $H_1$ and $H_2$ separately. $B$ simulates the responds. Two lists $\langle W_i, h_i, x_i, c_i \rangle$ and $\langle ID_i, f_i, y_i \rangle$ is maintained by $B$ as the random oracle queries. The former one is called $H_1$ list and the later one is $H_2$ list. They are empty at the beginning. The random oracle query procedures are as follows:

$H_1$ queries:
3.3. Construction

- When receive a query for keyword $W_i$, $B$ responds as $H_1(W_i) = h_i \in \mathbb{G}_1$ if $W_i$ is already in $H_1$ list.

- Otherwise, $B$ will flip a coin $c_i$, where $c_i \in \{0, 1\}$ with $\Pr[c_i = 0] = \delta$.

\[
\begin{cases} 
    \text{If } c_i = 0, & h_i = g^{x_i} \in \mathbb{G} \quad x_i \in_R \mathbb{Z}_p^* \\
    \text{If } c_i = 1, & h_i = g^{ax_i} \in \mathbb{G} \quad x_i \in_R \mathbb{Z}_p^*
\end{cases}
\]

- At last, $B$ adds $\langle ID_i, h_i, x_i, c_i \rangle$ to $H_1$ list, then sends the value $h_i$ to adversary $\mathcal{A}_{32}$ as the result of random oracle query.

$H_2$ queries:

When receive a query for keyword $ID_i$, algorithm $B$ responds as $H_2(ID_i) = f_i \in \mathbb{G}$ if $W_i$ is already in $H_2$ list. Otherwise, randomly pick $y_i \in \mathbb{Z}_p^*$, and set $f_i = g^{y_i}$. Then $B$ responds $H_2$ query as $f_i$, and adds $\langle ID_i, f_i, y_i \rangle$ to the $H_2$ list.

**Trapdoor Query**: $\mathcal{A}_{32}$ can choose a keyword $W_i$ and an identity $ID_i$ to $B$ to perform trapdoor queries. $B$ will responds as follows:

1. For every trapdoor query, the algorithm above will be run by $B$. If $c_i = 0$, then $B$ claims failure and terminates.

2. Otherwise, $B$ will obtain $h_i = g^{ax_i}$ and $f_i = g^{y_i}$. It randomly picks $t \in \mathbb{Z}_p^*$. It simulates the trapdoor as:

\[ (T_{i1}, T_{i2}, T_{i3}) = (g^{x_it}, g^{y_it}, g^t) \]

We have:

\[ g^{x_it} = g^{ax_it/a} = h_i^{t/a} = H_1(W_i)^{t/a} \]
\[ g^{y_it} = f_i^t = H_2(ID_i)^t \]

**Forgery**: Eventually algorithm $\mathcal{A}_{32}$ acquires a trapdoor from $B$, where $B$ sets $t = b$. Note: $g^{yib} = H_2(ID)^t$.

\[ (T_2^*, T_3^*) = (g^{yib}, T_3^* = g^b) \]

given by $B$ and announces an identity $ID^*$ which will be forged. Note, $B$ sets $t = b$ We assume that $ID^*$ have been queried before. Then, if $H_1(W) = h_i = g^{x_i}$, $\mathcal{A}_{32}$ forge the trapdoor keyword part as

\[ T_{i1}^* = H_1(W)^{t/a} = g^{x_i b} \]
Moreover, $B$ can output $g_b^\frac{b}{a} = T_i^\tau = g^{x_i}$. 

Next, we will explain the probability that $B$ can output the correct answer. The procedure should not abort during the queries phase. As we already known, $\delta = \frac{1}{q_{H_1} + 1}$. Therefore, the probability that $B$ doesn’t abort during the queries phase is $(1 - \frac{1}{q_{H_1} + 1})^{q_{H_1}} \geq \frac{1}{e}$ where $e$ is the base of nature logarithm. Subsequently, in the forgery phase, if we want to calculate $g^{ab}$, we need to find $H_1(W) = g^{x_i}$ in $H_1$ list. As the result, the probability will be $1/q_{H_1}$. Hence, the overall of $B$ succeeding is $\frac{e}{eq_{H_1}}$ as required. Then proof of theorem is completed.

### 3.4 Application

#### 3.4.1 Multi-Receiver

In this section, we extend our PEKSDS scheme to a multi-user PEKSDS scheme. In this part, we will introduce multi-receiver first. Assume sender Bob desires to send emails to Alice and Carlos with the same keyword. In traditional PEKS, Bob needs to compute keyword with receiver’s public key and moreover, maybe perform some paring computing. Furthermore, if the computing capacity is insufficient, the situation will be turned to thorny. In our scheme, this problem can be solved straightforwardly. The only need is utilizing receiver’s public key, powering the key with the same exponent as before, for example:

$$(C_1, C_2, C_2, ..., C_2, ...) = (H_2(ID)^s \cdot H_1(W)^r, h_1^r, h_2^r, ..., h_n^r)$$

where $h_1, h_2, ..., h_n$ is the public key of the receiver 1, 2, ...n. Creating trapdoor and testing process is the same as before. The proof can refer to [HL07]. This approach reduces the communication and computation overhead and processes no less security.

#### 3.4.2 Search Multi-Sender

To search multi-sender, we first introduce a simple version and show a general version. Suppose receiver Alice desires to search a keyword in the email sent by Bob and David. Compared to create the trapdoor twice, we only need to add an element in the trapdoor. The example will show receiver searches $n$ senders’ together.

$$(T_1, T_2, T_3, T_3, ..., T_3) = (H_1(W)^{t/\alpha}, g^t, H_2(ID_1)^t, H_2(ID_2)^t, ..., H_2(ID_n)^t)$$
where \(ID_1, ID_2, ..., ID_n\) are the senders’ identities. Encryption and test phase is the same as before. In addition, there is still no paring operation in our system.

### 3.4.3 A Modification of Our Scheme

We get this construct from [BGW05], using broadcast encryption inversely. Broadcast encryption aims to enable only qualified user to decryption. We change the target that only qualified sender can be searched. The scheme is described as follows.

**KeyGen:** Let \(G, G_T\) be two groups of prime order \(p\), and \(\hat{e}\) be a bilinear map: \(G \times G \rightarrow G_T\). The algorithm first picks a random generator \(g \in G\) and a random \(\alpha \in \mathbb{Z}_p^*\), it computes \(g^\alpha = h \in G\). Then, it picks a random \(\beta \in \mathbb{Z}_p\), it computes \(g_i = g^{(\beta_i)} \in G\) for \(i = 1, 2, ..., n, n+2, ..., 2n\). Next, it picks a random \(\gamma \in \mathbb{Z}_p\) and sets \(v = g^\gamma \in G\). We will need two hash functions \(H_1: \{0,1\}^* \rightarrow G\) and \(H_2: \{0,1\}^* \rightarrow \mathbb{Z}_n\). The public parameters are:

\[
(g, h, g_1, ..., g_n, g_{n+2}, ..., g_{2n}, v) \in G^{2n+2}
\]

We also compute decryption identities of senders in \(\mathbb{Z}_n\) is set as: \(d_i = g_i^\gamma \in G\). Note that \(d_i = v^{(\beta_i)}\). The decryption identities will also be kept by TTP. The maximum number of senders is \(n\).

The master key kept by the TTP is \(\alpha, g_{n+1}\).

\(S\) is the set of senders in which the receiver wants to search.

The system will compute a hash value by using the sender’s identity and send back a permanent identity (public key) to the sender as follows: \(H_2(ID) = i \in \{1, ..., n\}\) and \(g_i\) is the permanent identity for the sender.

**Encryption:** Sender picks a random number \(r \in \mathbb{Z}_p\), and uses his identity \(g_i\) and keyword \(W\), computes

\[
(C_1, C_2, C_3, C_4) = (H_1(W)^r, g_i^r, g^r, h^r)
\]

and transfers \((C_1, C_2, C_3, C_4)\) with the encrypted file to the server.

**Trapdoor:** Receiver(searcher) picks a random number \(t \in \mathbb{Z}_p\), computes

\[
(T_1, T_2) = (g^t, H_1(W^t)^t)
\]

and name set \(S\) to TTP.
3.5 Summary

TTP will according to the set $S$ to find the senders’ identities $g_i$ where $i \in S$, and computes

$$(T_3, T_4, \rho_i) = (v \cdot \prod_{j \in S} g_{n+1-j})^\alpha, \quad T_2, \quad d_i \cdot \prod_{j \in S, j \neq i} g_{n+1-j+i}$$

Then, TTP sends $(T_1, T_3, T_4, \rho_1, ..., \rho_s)$ where $|S| = s$, to the server.

Test: Then, the server will search the keyword user by user. Test:

$$\hat{e}(T_4, C_3) \cdot \hat{e}(\rho_i, C_4) = \hat{e}(C_1, T_1) \cdot \hat{e}(T_3, C_2)$$

If the result is true, return the content corresponding to the keyword. Else, nothing.

3.5 Summary

In this chapter, we discuss the disadvantage of the public encryption with keyword search (PEKS) and from this we construct a provably secure keyword search in trusted sender scheme by introducing sender’s identity into scheme. We have proved the cyphertext is secure against IND-CIA-CKA, UF-CIA, and trapdoor is secure against UF-CTA. In addition, it possesses the properties of signcryption and improve the efficiency of combining encryption and signature. Our scheme can be straightforward to be applied in cloud computing, since we don’t require paring operation in both sender and receiver. In addition, it can be extended to multi-sender and multi-receiver much more efficiency. What’s more is that we confirm the identity of both sender and receiver. Therefore, our scheme extremely suits for cloud computing.

As we have already known, the OT possesses the properties that protect both the privacy of sender and receiver. Therefore, another interesting problem is that we can build a scheme that performs combining OT and PEKS. The sender will send the encrypted message and keyword by using the receivers’ or the server’s public key, and the receiver conduct oblivious transfer with the database server, so that performs both privacy. In the addition, the scenario could be performed in multi-receiver and multi-sender environment.
Chapter 4

Symmetric key encryption with Keyword Search

4.1 Introduction

Since regular private-key encryption prevents one from searching over encrypted data, clients also lose the ability to selectively retrieve segments of their data. To address this problem, searchable symmetric encryption has been proposed [CGKO06, SWP00]. Actually, they inherit the form of PEKS, which contains keywords encryption, trapdoors creation and tests. Unlike PEKS, symmetric encryption adopted the method like what Bellare’s done [BBO06]. In that case, database servers can reorder and conduct searching operation on encrypted data. That’s the reason why we don’t employ the Bellare’s scheme.

Secure index [Goh03] provides symmetric encryption with search capabilities, in which an index is a file that stores document collections while supporting efficient keyword search. In other words, given a keyword, the index returns the corresponding contents that contain it. We say an index is secure if the search for a keyword $W$ can only be performed by users who set a trapdoor for $W$ with a secret key, and the index leaks no information about its contents without the knowledge of trapdoors. Gho [Goh03] proposed a symmetric searchable encryption scheme from a secure index, in which the user indexes and encrypts its document collection and sends secure index together with the encrypted data to the server. When searching for a keyword $w$, the user should generate and send a trapdoor for $w$ such that the email gateway can run the search algorithm to find the appropriate documents.

Ostrovsky and Goldreich [Ost90, GO96] designed a symmetric searchable encryption in full generality and optimal security, with which any type of search request can be achieved (including conjunctions and disjunctions of keywords) without leaking any information to the email gateway. However, to guarantee strong privacy,
it needs $n$ rounds of interaction for each read and write where $n$ is the number of documents. Although a two-round solution is given, it needs large square-root overhead.

However, the previously mentioned work focused only on the single user environment. If they are applied directly in the multi-user setting, the computation overhead might be considerably large.

### 4.1.1 Contributions of This Chapter

We propose a symmetric key encryption with keyword search which has perfect security for both sender and receiver. In addition our scheme is based on a computable assumption that is more secure than the one based on decisional assumption.

This chapter is structured as follows. We describe the symmetric keyword search and security notion in Section 2. Then, in Section 3, we will give the detail of the scheme and the security proof. Then, we will give conclusion in Section 4.

### 4.2 General Structure

#### 4.2.1 Definition of Symmetric Key Encryption with Keyword Search

**Definition 4.1 (SEKS)** A symmetric key encryption with keyword search is made of four randomized algorithms:

- **SysPara**($k$): Taking a security parameter $k \in \mathbb{N}$, it generates a system parameters $sp$.

- **KeyGen**(sp) : Taking system parameters $sp$ as input, it generates key pairs $A_{priv}/A_{pub}$, and $A'_{priv}/A'_{pub}$ for the receiver and sender respectively.

- **Encryption**($A'_{pub}, A_{priv}, W$): Taking receiver’s $A_{pub}$ and sender’s secret key $A'_{priv}$, it generates the symmetric key and produces a searchable encryption $Enc_W$ of keyword $W$.

- ** Trapdoor**(A_{priv}, A'_{pub}, W’): Taking sender’s $A'_{pub}$ and receiver’s secret key $A_{priv}$, it generates the symmetric key and produces the trapdoor $T'_W$ of keyword $W'$. 

4.2. General Structure

Test \( (T_W, \text{Enc}_W) \): Given the searchable encryption \( \text{Enc}(A_{\text{pub}}, A'_{\text{priv}}, W) \), trapdoor \( T_W(A_{\text{priv}}, A'_{\text{pub}}, W') \) and \( A_{\text{pub}} \). Output ‘yes’ while both \( W = W' \) and sender is designated, or ‘no’ otherwise.

4.2.2 Security Model

We define SEKS is semantically secure against a static adversary. The encryption and trapdoor possess security we called “indistinguishability against chosen keyword attack” (IND-CKA). Let algorithm \( \mathcal{A}_i \) \((i = 1, 2)\) be an attacker who wants to attack SKKS system bounded by polynomial time \( t \). To illustrate security for SKKS, we define two games between \( \mathcal{A}_i \) and a challenger as follows:

Game1: \( \mathcal{A}_1 \) aims to distinguish the encryption by chosen keyword attack.

Setup & KeyGen: The challenger runs \( \text{SysPara}(\kappa) \) to obtain system parameters and generate key pairs \( A_{\text{priv}}/A_{\text{pub}} \) and \( A'_{\text{priv}}/A'_{\text{pub}} \) separately. Then sends \( A_{\text{pub}} \) and \( A'_{\text{pub}} \) to \( \mathcal{A}_1 \).

Query phase 1: \( \mathcal{A}_1 \) adaptively issues queries \( q_1, \ldots, q_m \) for keywords \( W_i \), where \( 1 \leq i \leq m \). The challenger responds with \( T_W(A_{\text{priv}}, A'_{\text{pub}}, W_i) \) to \( \mathcal{A}_1 \).

Challenge: \( \mathcal{A}_1 \) sends a pair \( W_0 \) and \( W_1 \) to challenger on which it wishes to be challenged on. Next, the challenger picks a random \( \beta \in \{0, 1\} \). It sets \( \text{Enc}_\beta = \text{Enc}(A_{\text{pub}}, A'_{\text{priv}}, W_\beta) \) and gives it \( \mathcal{A}_1 \). The only requirement is that \( \mathcal{A}_1 \) hasn’t queried pair \( W_0 \) and \( W_1 \) for trapdoor.

Query phase 2: \( \mathcal{A}_1 \) continues to adaptively issue pairs \( W_i \) for trapdoor. The only constraint is that \( W_i \neq W_0, W_1 \). The challenger responds the same as in phase 1.

Guess: Algorithm \( \mathcal{A}_1 \) outputs its guess \( \beta' \in \{0, 1\} \) for \( b \) and wins the game if \( \beta = \beta' \).

We refer to such an adversary \( \mathcal{A}_1 \) as encryption IND-CKA adversary. We defined the advantage of \( \mathcal{A}_1 \) is attacking the scheme as

\[
\text{Adv}_{\mathcal{A}_1} = \left| \Pr[\beta = \beta'] - \frac{1}{2} \right|
\]

Game2: \( \mathcal{A}_2 \) aims to distinguish the trapdoor by chosen keyword attack.
4.3. Implement of a Symmetric Key Encryption with Keyword Search

Setup & KeyGen: The challenger runs $\text{SysPara}(k)$ to obtain system parameters and generate key pairs $A_{\text{priv}}/A_{\text{pub}}$ and $A'_{\text{priv}}/A'_{\text{pub}}$ separately. Then sends $A_{\text{pub}}$ and $A'_{\text{pub}}$ to $A_2$.

Query phase 1: $A_2$ adaptively issues queries $q_1, \ldots, q_m$ for keywords $W_i$, where $1 \leq i \leq m$. The challenger responds with $T_{W}(A_{\text{priv}}, A'_{\text{pub}}, W_i)$ to $A_2$.

Challenge: $A_2$ sends a pair $W_0$ and $W_1$ to challenger on which it wishes to be challenged on. Next, the challenger picks a random $\beta \in \{0, 1\}$. It sets $T_{W_{\beta}} = \text{Trapdoor}(A_{\text{priv}}, A'_{\text{pub}}, W_{\beta})$ and gives it $A_2$. The only requirement is that $A_2$ hasn't queried pair $W_0$ and $W_1$ for trapdoor.

Query phase 2: $A_2$ continues to adaptively issue pairs $W_i$ for trapdoor. The only constraint is that $W_i \neq W_0, W_1$. The challenger responds the same as in phase 1.

Guess: Algorithm $A_2$ outputs its guess $\beta' \in \{0, 1\}$ for $\beta$ and wins the game if $\beta = \beta'$.

We refer to such an adversary $A_2$ as trapdoor IND-CKA adversary. We defined the advantage of $A_2$ is attacking the scheme as

$$\text{Adv}_{A_2} = \left| \Pr[\beta = \beta'] - \frac{1}{2} \right|$$

Definition 4.2: Our SEKS scheme is $(t, q, \epsilon)$-semantically secure, if there exists no adversary $A_i$ ($i \in 1, 2$), who runs in at most $t$ time and queries at most $q$ times, the probability at least $\epsilon$.

4.3 Implement of a Symmetric Key Encryption with Keyword Search

4.3.1 Construction

$\text{SysParas}(q, g_1, g_2, H_1, H_2, G, G_T, \hat{e})$
4.3. Implement of a Symmetric Key Encryption with Keyword Search

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>the prime order of group $G$</td>
</tr>
<tr>
<td>$g_1 \leftarrow G$</td>
<td>a random generator of $G$</td>
</tr>
<tr>
<td>$g_2 \leftarrow G$</td>
<td>a random generator of $G$</td>
</tr>
<tr>
<td>$H_1$</td>
<td>a crypto-hash function maps ${0, 1}^* \rightarrow \mathbb{G}_q^*$</td>
</tr>
<tr>
<td>$H_2$</td>
<td>a crypto-hash function maps $\mathbb{G}_T \rightarrow \mathbb{Z}_q^*$</td>
</tr>
<tr>
<td>$G$</td>
<td>a multiplicative cyclic group</td>
</tr>
<tr>
<td>$\mathbb{G}_T$</td>
<td>a multiplicative cyclic group</td>
</tr>
<tr>
<td>$\hat{e}$</td>
<td>a bilinear map from $G$ to $\mathbb{G}_T$</td>
</tr>
</tbody>
</table>

Table 4.1: System Parameter of SEKS

KeyGen:
Since anyone can be sender or receiver, everyone picks a random number $x_i \in \mathbb{Z}$.

The public key is $g_1^{x_i}$ the private key is $x_i$

Encrypt:
If $user_0$ will send message to $user_1$, they compute

$$y = H_2(\hat{e}(g_1^{x_1}, g_2^{x_0})) = H_2(\hat{e}(g_1^{x_0}, g_2^{x_1}))$$

Then, encrypt the keyword as

$$H_1(W)^{\frac{t}{y}} \in G$$

Trapdoor:

$$(T_1 = g_1^{yt}, T_2 = \hat{e}(g_1, H_1(W))^t) \in \mathbb{G}^2, \ t \in R \mathbb{Z}$$

Correctness:

$$\hat{e}(g_1^{yt}, H_1(W)^{1/y}) = \hat{e}(g_1, H_1(W))^t$$

4.3.2 Security Analysis

Theorem 4.1 Let $G$ be a multiplicative bilinear group of prime order $q$. Our encryption is $(t, e, q_T)$ semantically secure against indistinguishable chosen keyword attack (IND-CKA) assuming the $(t, e, q_T)$-DCDH assumption holds in $G$. 
4.3. Implement of a Symmetric Key Encryption with Keyword Search

Proof: Let algorithm $A_1$ be an attacker bounded in a polynomial time $t$, who has advantage $\epsilon$ in breaking the encryption. $A_1$ can perform at most $q_T$ trapdoor queries, $q_H$ hash function queries to $H_1$. Let $B$ be an algorithm who aims to solve DDH problem with the probability at most $\epsilon = \frac{\epsilon'}{2e^{q_T}}$, and its running time is similar to $A_1$. Thus, if DDH problem holds in $G$, then $\epsilon'$ is a negligible function. Consequently, $\epsilon$ is a negligible probability for $A_1$ to break the SEKS encryption.

Let $g$ be a generator of $G$. Algorithm $B$ is given $g, g^a, g^b, u \in G$ as input, to determine whether $g^a = g^b$ equals to $u$ or a random element in $G$. $B$ proceeds as follows:

Setup & KeyGen: Algorithm $B$ starts by setting $g_1 = g$ to be the system parameter. To simulate users, algorithm $B$ picks random numbers $x_1, x_2 \in Z_p^*$, sets $x_1, x_2$ are receiver’s and sender’s private keys separately and $g^{x_1}, g^{x_2}$ are receiver’s and sender’s public keys respectively. In addition, $B$ picks a random element $g_2 \in G$. Finally, $B$ gives $(g, g_2, g^{x_1}, g^{x_2})$ to the adversary $A_1$.

$H_1$-queries: At any time, $A_1$ can issue at most $q_{H_1}$ queries to random oracles $H_1$. $B$ simulates the responses. $H_1$ list $\langle W_i, h_i, z_i, c_i \rangle$ is maintained by $B$ as the random oracle queries. It is empty at the beginning. The random oracle query procedure is as follows:

- When receive a query for keyword $W_i$, $B$ responds as $H_1(W_i) = h_i \in G$ if $W_i$ is already in $H_1$ list.

- Otherwise, $B$ will flip a coin $c_i \in \{0, 1\}$ with $Pr[c_i = 0] = \delta$.

\[
\begin{align*}
&\quad \text{If } c_i = 0, \quad h_i = g^{z_i} \in G, \quad z_i \in R Z_p^* \\
&\quad \text{If } c_i = 1, \quad h_i = g^{a z_i} \in G, \quad z_i \in R Z_p^*
\end{align*}
\]

- At last, $B$ adds $\langle W_i, h_i, z_i, c_i \rangle$ to $H_1$ list, then sends the value $h_i$ to adversary $A_1$ as the result of random oracle query.

$H_2$-query: For $H_2$ query, $A_1$ only need to query one time, and $B$ will response $g_2^{y} = g^y = g^{\frac{1}{2}}$.

Query Phase: $A_1$ can choose a keyword $W_i$ to $B$ to perform trapdoor queries. $B$ will responds as follows:

1. For every trapdoor query, the algorithm above will be run by $B$ to derive $H_1$ list such as $\langle W_i, h_i, z_i, c_i \rangle$. If $c_i = 0$, then $B$ claims failure and terminates.
2. Otherwise, $\mathcal{B}$ will obtain $h_i = g^{az_i}$. In addition, it picks $t \in \mathbb{Z}_p^*$. Then, we can simulate the trapdoor as:

$$(T_1, T_2) = ((g^{\frac{1}{g}})^t, \hat{e}(g^t, g^{az_i}))$$

Since, We have:

$$g^y = g^{\frac{1}{g}}$$

$$(g^{\frac{1}{g}})^t = g^{yt}$$

$$\hat{e}(g^t, g^{az}) = \hat{e}(g^t, H_1(W))$$

**Challenge:** Eventually algorithm $\mathcal{A}_1$ produces two keywords $W_1, W_0$ that it wishes to be challenged on. The only requirement is that $W_1, W_0$ haven’t been queried before. Algorithm $\mathcal{B}$ generates the challenge encryption as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to $H_1$-queries twice. If neither of $(c_0, c_1)$ equals 0, then $\mathcal{B}$ claims failure and terminates.

2. Otherwise, at least one of $c_0, c_1$ equals to 0. $\mathcal{B}$ picks $i \in \{0, 1\}$ randomly, and sets $W_\beta = W_i$, where $c_i$ equal to 0. Note, if only one of $c_0, c_1$ equals to 0, there is no randomized. Let $(W_i, h_i, z_i, c_i)$ be the corresponding tuples of the $H_1$-list.

3. then algorithm $\mathcal{B}$ responds the challenge encryption as:

$$C = z^{x_\beta}, \quad z \in \mathbb{G}$$

if $z = g^{\frac{a}{g}}$ the following is true:

$$z^{x_\beta} = g^{ax_\beta}$$

**Query Phase2:** $\mathcal{A}_1$ can continue to issue decryption queries for keywords $W_i$ where $W_i \neq W_0, W_1$. $\mathcal{B}$ will response as before.

**Guess:** $\mathcal{A}_1$ will guess $\beta' \in \{0, 1\}$ for $\beta$. If $\beta = \beta'$, $\mathcal{B}$ will output $g^{\frac{a}{g}} = u$. Otherwise, output nothing.

Next, we will explain the probability that $\mathcal{B}$ can output the correct answer. The procedure should not abort during trapdoor queries and challenge phase. Therefore,
4.3. Implement of a Symmetric Key Encryption with Keyword Search

from the knowledge of previous chapter, we know the probability is \( \frac{1}{e^{qt}} \), where \( e \) is the base of nature logarithm. Moreover, If \( A_1 \) can output the correct answer, \( B \) has \( \frac{1}{2} \) probability to solve the complexity assumption. Hence, the overall of \( B \) succeeding is \( \frac{\epsilon'}{2e^{qt}} \) as required. Then proof of theorem is completed.

**Theorem 4.2** Let \( G \) be a multiplicative bilinear group of prime order \( p \). The trapdoor of our system is \( (t, \epsilon, qt) \) semantically secure against indistinguishable chosen keyword attack (IND-CKA) assuming the \( (t, \epsilon, qt) \)-DBDH assumption holds in \( G \).

*Proof:* Let algorithm \( A_2 \) be an attacker bounded in a polynomial time \( t \), who has advantage \( \epsilon \) in breaking the encryption. \( A_2 \) can perform at most \( q_T \) trapdoor queries. Let \( B \) be an algorithm who aims to solve DBDH problem with the probability at most \( \epsilon' = \frac{\epsilon}{2e^{qt}} \), and its running time is similar to \( A_2 \). Thus, if DBDH problem holds in \( G \), then \( \epsilon' \) is a negligible function. Consequently, \( \epsilon \) is a negligible probability for \( A_2 \) to break the SEKS trapdoor.

Let \( g \) be a generator of \( G \). Algorithm \( B \) is given \( g, g^a, g^b, g^\frac{1}{z} \in G \) and \( V \in G_T \) as input, to determine whether \( \hat{e}(g, g)^{abz} \) equals to \( V \) or a random element in \( G_T \). \( B \) proceeds as follows:

**Setup & KeyGen:** Algorithm \( B \) starts by setting \( g_1 = g \) to be the system parameter. To simulate users, algorithm \( B \) picks a random number \( x_1, x_2 \in \mathbb{Z}_p^* \), sets \( x_1, x_2 \) is receiver’s and sender’s private key separately and \( g^{x_1}, g^{x_2} \) is receiver’s and sender’s public key separately. In addition, \( B \) picks a random element \( g_2 \in G \). Finally, \( B \) gives \( (g, g_2, g^{x_1}, g^{x_2}) \) to the adversary \( A_2 \).

**H_1-queries:** At any time, \( A_2 \) can issue at most \( q_{H_1} \) queries to random oracles \( H_1 \). \( B \) simulates the responds. \( H_1 \) list \( (W_i, h_i, z_i, c_i) \) is maintained by \( B \) as the random oracle queries. It is empty at the beginning. The random oracle query procedure is as follows:

- When receive a query for keyword \( W_i \), \( B \) responds as \( H_1(W_i) = h_i \in G \) if \( W_i \) is already in \( H_1 \) list.
- Otherwise, \( B \) will flip a coin \( c_i \in \{0, 1\} \) with \( \Pr[c_i = 0] = \delta \).

\[
\begin{cases}
\text{If } c_i = 0, & h_i = g^{z_i} \in G, \quad z_i \in R \mathbb{Z}_p^* \\
\text{If } c_i = 1, & h_i = g^{az_i} \in G, \quad z_i \in R \mathbb{Z}_p^* 
\end{cases}
\]
4.3. Implement of a Symmetric Key Encryption with Keyword Search

- At last, $B$ adds $\langle W_i, h_i, z_i, c_i \rangle$ to $H_1$ list, then sends the value $h_i$ to adversary $A_1$ as the result of random oracle query.

$H_2$-query: For $H_2$ query, $A_2$ only needs to query one time, and $B$ will response as $g^{H_2} = g^y = g^c$.

Query Phase: $A_2$ can choose a keyword $W_i$ to $B$ to perform trapdoor queries. $B$ will responds as follows:

1. For every trapdoor query, the algorithm above will be run by $B$ to derive $H_1$ list such as $\langle W_i, h_i, z_i, c_i \rangle$. If $c_i = 0$, then $B$ claims failure and terminates.

2. Otherwise, $B$ will obtain $h_i = g^{az_i}$. In addition, it picks $t = \frac{b}{c}$. Then, we can simulate the trapdoor as:

$$(T_1, T_2) = (g^b, \hat{\epsilon}(g^b, g^{az_i}))$$

Since, We have:

$$g^y = g^c$$
$$g^b = g^{cz} = g^{bt}$$
$$\hat{\epsilon}(g^b, g^{az_i}) = \hat{\epsilon}(g^z, g^{az_i}) = \hat{\epsilon}(g^t, H_1(W))$$

Challenge: Further, algorithm $A_2$ produces two keywords $W_1, W_0$ that it wishes to be challenged on. The only requirement is that $W_1, W_0$ haven’t been queried before. Algorithm $B$ generates the challenge encryption as follows:

1. Algorithm $B$ runs the above algorithm for responding to $H_1$-queries twice. If neither of $(c_0, c_1)$ equals 0, then $B$ claims failure and terminates.

2. Otherwise, at least one of $c_0, c_1$ equals to 0. $B$ picks $i \in \{0, 1\}$ randomly, and sets $W_\beta = W_i$, where $c_i$ equal to 0. Note, if only one of $c_0, c_1$ equals to 0, there is no randomized. Let $\langle W_i, h_i, z_i, c_i \rangle$ be the corresponding tuples of the $H_1$-list.

3. Then algorithm $B$ responds the challenge encryption as:

$$(T_1, T_2) = (g^b, V^{z\beta}), \ V \in \mathbb{G}_T$$

if $V = \hat{\epsilon}(g, g)^{ab}$, the following is true:

$$V^{z\beta} = \hat{\epsilon}(g, g)^{abz\beta} = \hat{\epsilon}(g^z, g^{az\beta}) = \hat{\epsilon}(g^t, H_1(W_\beta))$$
4.4 Summary

Query Phase 2: $A_2$ can continue to issue decryption queries for keywords $W_i$ where $W_i \neq W_0, W_1$. $B$ will respond as before.

Guess: $A_2$ will guess $\beta' \in \{0,1\}$ for $\beta$. If $\beta = \beta'$, $B$ will output $V = \hat{e}(g, g)_{\beta'}$. Otherwise, output nothing.

Next, we will explain the probability that $B$ can output the correct answer. Since our scheme employs symmetric key encryption techniques, hence $A_2$ and $B$ have the same probability to break the scheme and complexity assumption separately as the previous theorem. Then proof of theorem is completed.

4.4 Summary

In order to solve the problem that users with limited resources or expertise, can selectively retrieve segments of their data at low cost, symmetric encryption with search capabilities is proposed. In this chapter, we proposed a symmetric key encryption with keyword search scheme. Regarding the security reduction, we define a security notion called “indistinguishability against chosen keyword attack” (IND-CKA) between a static adversary and a challenger. Compared to the existing symmetric encryption with keyword search schemes, our new scheme can be applied among multiple users. Moreover, our construction is based on a computable assumption that is more secure than the one based on decisional assumption. In addition, our construction can be easily extended to multi-user setting.
Chapter 5

Oblivious keyword search

5.1 Introduction

Before the Public Key Encryption with Keyword Search has been proposed, Oblivious Transfer (OT) was always being considered as the best way to construct a security communication channel between sender and receiver. Since the property of OT is very suitable for conducting keyword search, sender knows nothing about the receiver’s choices, and receiver can not obtain extra information except his choices. Therefore, Ogata and Kurosawa proposed an oblivious keyword search scheme based on OT [OK04]. Before introducing Ogata’s scheme, we will bring in another OT scheme, in order that readers can have a preliminary imagination about OT.

Chu and Tzeng proposed two efficient oblivious transfer schemes with adaptive and non-adaptive queries [CT05]. In their scheme, sender commits all data to receiver firstly, which are random numbers from the receivers’ perspective. Then, the receiver makes queries to the sender who will responds by supplying corresponding secret keys of his choices. As we must mention here, these keys can only decrypt the files the receiver has chosen. For the other files, the result of receiver’s calculation is just like random numbers. Furthermore, one of their schemes can be extended to adaptive OT. Although the receiver can send the queries one by one adaptively, the scheme provides the same security as the non-adaptive one. Therefore, this property extremely suits keyword search. Additionally, it is also suitable for searching in a public database. For this reason, we have thought that using accumulator to integrate the search item, and it fulfills the property of pay-as-you-use in cloud computing. The author also gives some comparisons between Mu’s [MZV02], Naor’s [NP99b], Ogata’s [OK04] schemes and themselves’, and their schemes have more advantages than most of theirs. Thus, our second application is based on it.

Back to Ogata and Kurosawa’s scheme, they firstly proposed the concept of
oblitative transfer. Compared with traditional $OT$, their proposal not only uses keyword instead of the index to conduct keyword search, but also utilizes blind signature to process keyword extract, which illuminates the following researchers to perform keyword extract in the similar way [CKRS09].

Oblivious Polynomial Evaluation (OPE) was first proposed in 1999 [NP99a, NP06], which is a variant of $OT$. Instead of committing all the data to receiver, sender inputs a polynomial function such as $P()$. Receiver will input a value such as $\alpha$. When $\alpha$ is the root of $P()$, the function $P(\alpha)$ will return the result the receiver want to obtain. Otherwise, the receiver gets nothing from sender since the result will be like a random number. There are many applications for OPE, such as mutually authenticated key exchange, private comparison of data. Similarly, OPE can be extended to keyword search. Freedman has proposed a keyword search scheme with oblivious pseudorandom functions based on OPE [FIPR05]. They use a pseudorandom function to perform OPE, then employ it to compute $P() \oplus M$ where $M$ is the message and the root is keyword which will make $P()$ equals to zero. Since the pseudorandom function is published, the user can execute keyword search by him. On the other hand, the authors haven’t supply such a pseudorandom function, though it inherit the $OT$’s property of protect both sender’s and receiver’s privacy. Therefore the method is worth to continue thinking about. Similarly, [GSW04, ZB07] utilized OPE to perform oblivious transfer as well.

5.1.1 Contributions of This Chapter

We propose oblivious keyword search scheme and public encryption with oblivious keyword search scheme in this chapter. Since Oblivious Transfer (OT) guarantees both sender’s and receiver’s privacy, it can be easily transformed to Oblivious Keyword (OKS) Search. The first scheme combines OT and PEKS directly. In our second OKS scheme, we assume the database can be updated by every user. Thus, Public Encryption with Oblivious Keyword Search (PEOKS) enables that anyone can be sender encrypted data by the same public key. The scheme improved the convenience of receiver.

This chapter is structured as follows. We describe the oblivious keyword search and security notion in Section 5.2. Then, Section 5.3, 5.4 will give the detail of the scheme and the security proof. Then, we will summarize this chapter in Section 5.5.
5.2 General Structure

5.2.1 Definition of Oblivious Keyword Search

The oblivious keyword search (OKS) involves two parties: Server (S) and Receiver (R). Let \( W \) be the set of keywords, and \( M \) be the set of messages.

**Definition 5.1 (OKS)** An oblivious keyword search scheme is made of follow phases:

1. R takes random numbers to mask the keyword and sends masked keyword to S.
2. S using masked keyword to encrypt the message and send them back with his public key.
3. R can decrypt the message if and only if the keywords are matched.

**Definition 5.2 (PEOKS)** A public key encryption with keyword search is made of four randomized algorithms:

- **SysPara**: Taking a security parameter \( k \in \mathbb{N} \), it generates a system parameters \( sp \).
- **KeyGen**: Taking system parameters \( sp \) as input, it generates Servers’ key pair \( A_{\text{priv}}/A_{\text{pub}} \).
- **Commitment**: Taking random numbers and keywords as input, it generates masked keywords \( W \).
- **Encryption**: Taking Server’s \( A_{\text{pub}} \), keywords \( W \) and message as input, it generates the encryption of keyword \( W \). Meanwhile, taking Server’s \( A_{\text{priv}} \) and masked keywords as input, it generates the decryption key.
- **Decryption**: Taking the decryption key and ciphertext as input, it decrypts the message if and only if the keywords matched.

5.2.2 Security Model

Let us assume that Server holds \( n \) messages \( m_1, m_2, ..., m_n \) and Receiver’s \( k \) keywords are \( W_1, W_2, ..., W_k \). We need to know that two sets \( \mathcal{Y} \) and \( \mathcal{Y}' \) are different if there is \( x \) in \( \mathcal{Y} \), but not in \( \mathcal{Y}' \), or vice versa. For the scheme oblivious keyword search, the following security requirements will be listed below:
1. Receiver’s privacy - indistinguishability: for any two different sets of choices \( \mathcal{Y} = W_1, W_2, ..., W_k \) and \( \mathcal{Y}' = W'_1, W'_2, ..., W'_k \), the transcripts, corresponding to \( \mathcal{Y} \) and \( \mathcal{Y}' \), received by the sender are indistinguishable. The choices of Receiver are proven to be unconditionally secure, if it is found that the received messages of Server for \( \mathcal{Y} \) and \( \mathcal{Y}' \) are distributed identically.

2. Server’s privacy - indistinguishability: for any choice set \( C = W_1, W_2, ..., W_k \), the unchosen messages should be indistinguishable from the random ones.

If the ciphertexts of unchosen messages are uniformly distributed for R, the security of S is unconditional.

Public encryption with oblivious keyword search scheme should meet the following security requirements:

1. Receiver’s privacy - indistinguishability: the same as the case of the oblivious keyword search receiver.

2. Server’s privacy - compared with the Ideal model: in the Ideal model, the sender sends all messages and the receiver sends his choices to the trusted third party (TTP). TTP then sends the chosen messages to the receiver. This is the securest way to implement the public encryption with oblivious keyword search scheme. The receiver R cannot obtain extra information from the server in the Ideal model. We say that the servers privacy is achieved if for any receiver R in the real public encryption with oblivious keyword search scheme, there is another simulator R’ in the Ideal model such that the outputs of R and R’ are indistinguishable.

## 5.3 Implement of Oblivious Keyword Search

Our first oblivious keyword search scheme is based on a non-adaptive oblivious transfer scheme. The scheme inherit the nice property of universal parameters from Chu’s Scheme[CT05]. Compared with others’ schemes, our scheme gives a faster way to perform test. Instead of using every trapdoor to test keyword one by one, we only need to test one time to achieve keyword searching. Moreover, our scheme doesn’t need extra variable to prevent the privacy against malicious server. The construction shows as follow:
5.3. Implement of Oblivious Keyword Search

5.3.1 Construction

System parameters: \((q, g_1, g_2, g_3, H_1, H_2, G, G_T, \hat{e})\)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q)</td>
<td>the prime order of group (G)</td>
</tr>
<tr>
<td>(g_1)</td>
<td>a random generator of (G)</td>
</tr>
<tr>
<td>(g_2)</td>
<td>a random generator of (G)</td>
</tr>
<tr>
<td>(g_3)</td>
<td>a random generator of (G)</td>
</tr>
<tr>
<td>(H_1)</td>
<td>a crypto-hash function maps ({0, 1}^* \rightarrow \mathbb{Z}_q^*)</td>
</tr>
<tr>
<td>(H_2)</td>
<td>a crypto-hash function maps (G_T \rightarrow {0, 1}^{\log q})</td>
</tr>
<tr>
<td>(G)</td>
<td>a multiplicative cyclic group</td>
</tr>
<tr>
<td>(G_T)</td>
<td>a multiplicative cyclic group</td>
</tr>
<tr>
<td>(\hat{e})</td>
<td>a bilinear map: (G \times G \rightarrow G_T)</td>
</tr>
</tbody>
</table>

Table 5.1: System Parameters of Oblivious Keyword Search

Server(S) has message \(M_1, M_2, ..., M_n\) with the corresponding keyword \(W_1, W_2, ..., W_n\).

S selects \(\alpha \in_R \mathbb{Z}\) and sets \(\alpha\) as his private key, and \((h = g_1^\alpha)\) as his public key.

Receiver(R) has the keywords he want to search \(W'_1, W'_2, ..., W'_{k}\).

Step1: R chooses three polynomials

\[
\begin{align*}
f_1(x) &= a_0 + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} + x^k \pmod{q} \in \mathbb{Z}_q \\
f_2(x) &= b_0 + b_1 x + b_2 x^2 + ... + b_{k-1} x^{k-1} + x^k \pmod{q} \in \mathbb{Z}_q \\
f_3(x) &= c_0 + c_1 x + c_2 x^2 + ... + c_{k-1} x^{k-1} + x^k \pmod{q} \in \mathbb{Z}_q \\
&\equiv (x - H_1(W'_1))(x - H_1(W'_2))...(x - H_1(W'_k))
\end{align*}
\]

Step2: R computes:

\[
g_1^{a_0}, g_1^{a_1}, g_1^{a_2}, ..., g_1^{a_{k-1}} \\
g_2^{b_0}, g_2^{b_1}, g_2^{b_2}, ..., g_2^{b_{k-1}} \\
g_3^{c_0}, g_3^{c_1}, g_3^{c_2}, ..., g_3^{c_{k-1}},
\]

and sends them to S.
Step3: $S$ computes:
\[
\begin{align*}
g_1^{f_1(i)} &= g_1^{a_0} g_1^{a_1 H_1(W_i)} g_2 H_1(W_i)^2 \cdots g_1^{g_{k-1} H_1(W_i)^{k-1}} H_1(W_i)^k \\
g_2^{f_2(i)} f_3(i) &= g_2 b_0 H_1(W_i) g_2 c_1 H_1(W_i) b_2 H_1(W_i)^2 c_2 H_1(W_i)^2 \\
&\quad\cdots g_2^{b_{k-1} H_1(W_i)^{k-1}} g_2^{c_{k-1} H_1(W_i)^{k-1}} H_1(W_i)^k H_1(W_i)^k \\
\end{align*}
\]
and computes:
\[
[C_1, C_2] = [h^{r_i}, M_i \oplus H_2(\hat{e}(g_1^{f_1(i)}, (g_2^{f_2(i)} g_3^{f_3(i)})^{r_i}))] \quad r_i \in_r \mathbb{Z}
\]
then sends to $R$.

Step4: $R$ computes
\[
M_i = C_2 \oplus H_2(\hat{e}(C_1, g_2^{f_2(i)} f_1(i)))
\]

Correctness: If $W_i = W'_i$, the value of function $f_3(i)$ equals to zero, receiver can obtain the message. Else return $\perp$.

5.3.2 Security Analysis

R’s privacy:

**Theorem 5.1** For our scheme, R’s choices are unconditionally secure.

**Proof.** For every tuple $(c'_0, c'_1, c'_2, \ldots, c'_{k-1})$, there is a tuple $(b'_0, b_1, b'_2, \ldots, b'_{k-1})$ satisfies
\[
g_2^{b_i} g_3^{c'_i} = g_2^{b'_i} g_3^{c'_i} \quad \text{for } i = 0, 1, \ldots, k - 1.
\]

S’s Privacy:

**Theorem 5.2** Let $\mathbb{G}$ be a multiplicative bilinear group of prime order $p$. Our system is $(t, \epsilon, q_T, q_H)$ semantically secure against indistinguishable chosen keyword attack (IND-CKA) assuming the $(t, \epsilon, q_T, q_H)$-DBDH assumption holds in $\mathbb{G}$.

**Proof.** We show that for all $i \notin \{W_1, W_2, \ldots, W_k\}$, $M_i$’s look random if the DBDH assumption holds. Assume that there is a polynomial-time distinguisher $D = (D_1, D_2)$, where $D_1$ takes $k$ keywords as inputs and outputs $f_1^*(x), f_2^*(x), f_3^*(x)$, and $D_2$ distinguishes the following two distributions:
5.3. Implement of Oblivious Keyword Search

- $X$: $\hat{e}(h f_1(i), g f_2(i))^r_i,$
  where $h, g_2 \in \mathbb{G}$, $r_i \in_R \mathbb{Z}_q$.
- $E$: $R f_1(i) f_2(i),$ 
  where $R \in \mathbb{G}_T$.

Then we can construct another PPTM $\mathcal{D}'$, which takes $\mathcal{D}$ as subroutine to distinguish two distributions:

- $\mathcal{Y}_1$: $g, g^a, g^b, g^c, \hat{e}(g, g)^{abc},$
  where $g \in \mathbb{G}$, $a, b, c \in_R \mathbb{Z}_q$
- $\mathcal{Y}_2$: $g, g^a, g^b, g^c, z.$
  where $g \in \mathbb{G}$, $a, b, c \in_R \mathbb{Z}_q$, $z \in_r \mathbb{G}_T$

The difference between $(\mathcal{Y}_1, \mathcal{Y}_2)$ and $(\mathcal{Y}_1, \mathcal{Y}_2)$ is that $g$ can’t be 1 in $\mathcal{Y}_1$ and $\mathcal{Y}_2$.

Machine $\mathcal{D}'$:
Input $(g, g^a, g^b, g^c)$ (either from $\mathcal{Y}_1$ or $\mathcal{Y}_2$.)

1. Let $g = g_1$, $g^a = h = g^a$, $g^b = g_2$ and randomly select $g_3 \in \mathbb{G}$.

2. Randomly pick $r_1, r_2, ..., r_n \in_R \mathbb{Z}_q$.
Let $g^{r_1} = g^{c_1}, g^{r_2} = g^{c_2}, ..., g^{r_n} = g^{c_n}$.

3. Compute the value of $H_1(W_1), H_1(W_2), ..., H_1(W_k)$,
then perform $\mathcal{D}_1(H_1(W_1), H_1(W_2), ..., H_1(W_k))$ as $(f_1(i), f_2(i), f_3(i)).$

4. Randomly select $i \in H_1(W_i)$.

5. Output $\mathcal{D}_2(g^*, h^*, f_1^*(x), f_2^*(x), f_3^*(x))$ for all $i \in H_1(W_i)$, where

$$
(C_1, C_2) \begin{cases} h^{c_i}, \hat{e}(g^{a f_1(i)}, g^{b f_2(i)})^{c_i} & \text{if } i \in i_1, ..., i_l \\
 h^{c_i}, z^{f_1(i) f_2(i)} & \text{if } i \in i_{l+1}, ..., i_{n-k} \end{cases}
$$

If $\mathcal{D}$ have a non-negligible advantage $\epsilon$ to distinguish $X$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$ can be distinguished by $\mathcal{D}'$ in the DBDH problem with at least non-negligible advantage $\epsilon - 2/q$. This finished the proof.
5.4 Public Key Encryption with Oblivious Keyword Search

5.4.1 Construction

Our Second oblivious keyword search scheme is based on an adaptive scheme which means receiver can send his query one by one, and modify his choice whenever he wants. Additionally, it provides the same security as the non-adaptive version. In our scheme, we introduce the public key to the scheme. If we look at the scheme as public key encryption with keyword search, the scheme is secure against offline guessing attack. Thus we choose blind extraction to provide trapdoor security. The construction shows as follows:

- Server(S) has message $M_1, M_2, ..., M_n$ with the corresponding keyword $W_1, W_2, ..., W_n$.
- Receiver(R) has the keywords he want to search $W'_1, W'_2, ..., W'_k$.

SysParas: $(q, g, H_1, H_2, G, G_T, \hat{e})$

<table>
<thead>
<tr>
<th>Notation</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>the prime order of group $G$</td>
</tr>
<tr>
<td>$g \overset{$}{\leftarrow} G$</td>
<td>a random generator of $G$</td>
</tr>
<tr>
<td>$H_1$</td>
<td>a crypto-hash function maps ${0, 1}^* \rightarrow G$</td>
</tr>
<tr>
<td>$H_2$</td>
<td>a crypto-hash function maps $G_T \rightarrow {0, 1}^{\log q}$</td>
</tr>
<tr>
<td>$G$</td>
<td>a multiplicative cyclic group</td>
</tr>
<tr>
<td>$G_T$</td>
<td>a multiplicative cyclic group</td>
</tr>
<tr>
<td>$\hat{e}$</td>
<td>a bilinear map: $G \times G \rightarrow G_T$</td>
</tr>
</tbody>
</table>

Table 5.2: System Parameters of PEOKS

KeyGen:

Server(S) randomly selects $\alpha_1, ..., \alpha_n \in_R \mathbb{Z}$ and sets $\alpha_i$ where $i \in n$ as his private key, and $(h_i = g_i^{\alpha_i})$ as his public key.

Commitment:
5.4. Public Key Encryption with Oblivious Keyword Search

Receiver(R) computes \( A_j = H_1(W_j)g^{t_j} \), where randomly picks \( t_j \in \mathbb{Z}_q^* \) and sends \( A_1, A_2, A_3, ..., A_k \) to Server(S).

Encryption:

Server(S) computes: \( D_j = (A_j)^{a_i}, C_i = M_i \oplus H_2(e(H(W_i), h_i^{r_i})) \), where randomly picks \( r_i \in \mathbb{Z}_q^* \), then, sends

\[
(g^{r_1}, C_1), (g^{r_2}, C_2), ..., (g^{r_n}, C_n), D_1, D_2, ..., D_k
\]

to R.

Decryption:

R computes \( K_j = D_j / h_i^{t_j}, M_j = C_i \oplus H_2(e(K_j, g^{r_i})) \) for \( j = 1, ..., k \)

Correctness:

\[
M = C_i \oplus H_2(D_j / h_i^{t_j}, g^{r_i})
\]
\[
= M_i \oplus H_2(e(H_1(W_i), h_i^{r_i})) \oplus H_2(e(H_1(W_j)^{a_i} \cdot g^{a_i t_j} / h_i^{t_j}, g^{r_i}))
\]
\[
= M_i \oplus H_2(e(H_1(W_i), h_i^{r_i})) \oplus H_2(e(H_1(W_j)^{a_i}, g^{r_i}))
\]
\[
= M_i
\]

In this scheme, we can add another role sender. Server will hold the private key \( \alpha \). Sender can use public key \( h \) to encrypt the keyword as \( (C_1, C_2) = (H_2(e(H_1(W_i), h^{r_i})), g^{r_i}) \). Receiver will search the keyword described as in the scheme above.

5.4.2 Security Analysis

R’s Privacy:

**Theorem 5.3** For our scheme, R’s choices are unconditionally secure

For any \( A_j = H_1(W_j) \cdot g^{a_j} \), and \( W_l \), where \( l \neq j \), there is an \( a'_l \) that satisfies \( A_j = w_l g^{a'_l} \). For S, \( A_j \) can be a masked value of any index. Thus, the receiver’s privacy is unconditional secure.

S’s Privacy:

**Theorem 5.4** Let \( G \) be a multiplicative bilinear group of prime order \( p \). Our system is \((t, e, q_T, q_{H_2})\) semantically secure against indistinguishable chosen keyword attack (IND-CKA) assuming the \((t, e, q_T, q_{H_2})\)-DBDH assumption holds in \( G \).
5.4. Public Key Encryption with Oblivious Keyword Search

Proof: Let algorithm $A$ be an attacker bounded in a polynomial time $t$, who has advantage $\epsilon$ in breaking our scheme. $A$ can perform at most $q_T$ trapdoor queries, $q_{H_1}$, $q_{H_2}$ hash function queries to $H_1$ and $H_2$ separately. Let $B$ be an algorithm who aims to solve DBDH problem with the probability at most $\epsilon' = \frac{\epsilon}{\epsilon_{t_{\text{pol}}}}$, and its running time is similar to $A$. Thus, if DBDH problem holds in $G$, then $\epsilon'$ is a negligible function. Consequently, $\epsilon$ is a negligible probability for $A$ to break our scheme.

Let $g$ be a generator of $G$. Algorithm $B$ is given $u_1 = g^a$, $u_2 = g^b$, $u_3 = g^c \in G$ and $\hat{e}(g, g)^z \in G_T$ as input, to determine whether $\hat{e}(g, g)^{abc}$ equals to $\hat{e}(g, g)^z$ or a random element in $G_T$. Algorithm $B$ simulate the challenger to interact with $A$. $B$ proceeds as follows:

Setup & KeyGen: Algorithm $B$ starts by setting $g = g$ to be the system parameter. Then, algorithm $B$ sets $h = g^{\alpha_i}$ where $i \in n$. In another word, $(g, h_1, h_2, ..., h_n)$ is public key, and $\alpha_i$ where $h_i = g^{\alpha_i}$ is set to be secret key. Therefore, $P_{\text{pub}}$ is $u_1$, where $h_i = u_1^{\alpha_i}$ and secret key is $a_i$. Finally, $B$ gives $(g, u_1)$ where $i \in n$ to the adversary $A$.

$H_1, H_2$-queries: At any time, $A$ can issue at most $q_{H_1}$ and $q_{H_2}$ queries to random oracles $H_1$ and $H_2$ separately. To simulate the responds, $B$ maintains $(W_i, f_i, x_i, c_i)$ called $H_1$ list as the random oracle query. It is empty at the beginning. The random oracle query procedure is as follows:

- When receive a query for keyword $W_i$, $B$ responds as $H_1(W_i) = f_i \in G$ if $W_i$ is already in $H_1$ list.

- Otherwise, $B$ will flip a coin $c_i \in \{0, 1\}$ with $\Pr[c_i = 0] = \delta$.

\[
\begin{cases}
\text{If } c_i = 0, & f_i = u_2 \cdot g^{x_i} \in G, \quad x_i \in R \mathbb{Z}_p^* \\
\text{If } c_i = 1, & f_i = g^{x_i} \in G, \quad x_i \in R \mathbb{Z}_p^*
\end{cases}
\]

- At last, $B$ adds $(W_i, f_i, x_i, c_i)$ to $H_1$ list, then sends the value $f_i$ to adversary $A$ as the result of random oracle query.

To simulate the $H_2$ queries, algorithm $B$ maintains a $H_2$ list $(y, V)$ as the random oracle query. For every query $H_2(y)$, $B$ picks a new random value $V \in \{0, 1\}^{\log q}$ as the value of $H_2(y)$ for each new $y$. Moreover, $(y, V)$ will be added to $H_2$ list. $H_2$ list is also empty at the start.
5.4. Public Key Encryption with Oblivious Keyword Search

Query Phase 1: \( A \) can choose a keyword \( W_i \) to \( B \) to perform encryption queries. \( B \) will responds as follows:

1. For every encryption query, the algorithm above will be run by \( B \) to derive \( H_1 \) list such as \( \langle W_i, f_i, x_i, c_i \rangle \). If \( c_i = 0 \), then \( B \) claims failure and terminates.
2. Otherwise, \( B \) will obtain \( f_i = g^{x_i} \).
3. To simulate \( R \), \( B \) randomly picks \( t_1, t_2, \ldots, t_k \) and outputs \( A_1, A_2, \ldots, A_k \) as
   \[
g^{x_1} \cdot g^{f_1}, g^{x_2} \cdot g^{f_2}, \ldots, g^{x_k} \cdot g^{f_k}
   \]
4. To simulate \( S \), \( B \) use \( t_1, t_2, \ldots, t_k \) above. Then, simulates the decryption key \( D_i \) as:
   \[
u_1^{x_i^*} \cdot u_1^{t_i^*}
   \]
   Since \( h_i = g^{a_i} \), we have:
   \[
u_1^{x_i^*} \cdot u_1^{t_i^*} = g^{x_i^*+a_i^*} \cdot g^{t_i^*+a_i^*} = H_1(W_i) \cdot h_i^{t_i^*} = D_i^*
   \]

Challenge: Eventually algorithm \( A \) produces two keywords \( W_0, W_1 \) that it wishes to be challenged on. The only requirement is that \( W_0, W_1 \) haven’t been queried before. Algorithm \( B \) generates the challenge encryption as follows:

1. Algorithm \( B \) runs the above algorithm for responding to \( H_1 \)-queries and \( H_2 \)-queries twice. If neither of \( (c_0, c_1) \) equals 0, then \( B \) claims failure and terminates.
2. Otherwise, at least one of \( c_0, c_1 \) equals to 0. \( B \) picks \( i \in \{0, 1\} \) randomly, and sets \( W_\beta = W_i \), where \( c_i \) equal to 0. Note, if only one of \( c_0, c_1 \) equals to 0, there is no randomized. Let \( \langle W_i, f_i, x_i, c_i \rangle \) be the corresponding tuples of the \( H_1 \)-list.
3. Let \( r_i = c_i \), and then algorithm \( B \) responds the challenge encryption as:
   \[
   [g^{r_i}, C_\beta] = [u_{3i}, V_i], \quad V_i \in \{0, 1\}^{\log q}
   \]
   Since, we have
   \[
   V_i = \hat{H}_2(\hat{e}(H_1(W_i), u_1^{c_i})) = \hat{H}_2(\hat{e}(u_2 \cdot g^{r_i}, g^{c_i})) = \hat{H}_2(\hat{e}(g, g)^{a_i c_i (b+x_i)})
   \]
   Note: \( C_i = M_i \oplus H_2(\hat{e}(H_1(W_i), h_i^{t_i^*})) \) where \( M_i \) will not affect on encryption, we omit \( M_i \) here. Therefore \( C_i \) is simplified for \( H_2(\hat{e}(H_1(W_i), h_i^{t_i^*})) \).
Query Phase 2: $A$ can continue to issue decryption queries for keywords $W_i$ where $W_i \neq W_0, W_1$. $B$ will response as before.

Guess: $A$ will guess $\beta' \in \{0, 1\}$ for $\beta$. If $\beta = \beta'$, $B$ will output $\hat{e}(g, g)^{a_i bc_i}$ according to $t/\hat{e}(u_{1i}, u_{3i})$, since $t$ must be queried in $H_2$ list. Otherwise, output nothing.

Next, we will explain the probability that $B$ can output the correct answer. The procedure should not abort during trapdoor queries and challenge phase. Therefore, from the knowledge of previous chapter, we know the probability is $\frac{1}{eqT}$, where $e$ is the base of nature logarithm. Moreover, If $A_1$ can output the correct answer, $B$ should find the $\hat{e}(g, g)^{a_i bc}$ in $H_2$ list. Hence, the overall of $B$ succeeding is $\frac{e}{eqTqH}$ as required. Then proof of theorem is completed.

## 5.5 Summary

We have presented two very efficient oblivious keyword search schemes with perfect security of either receiver or server. Since Oblivious Transfer (OT) ensures both sender’s and receiver’s privacy is warranted, it can be easily transformed to Oblivious Keyword Search (OKS). Therefore, in the first scheme, we employ the OT to perform keyword search. Meanwhile, we improve it with PEKS to achieve public encryption with oblivious keyword search. In our second OKS scheme, the database is assumed to be updated by every user. Thus, Public Key Encryption with Oblivious Keyword Search (PEOKS) enables that everyone can be the sender and encrypt data by the same public key. The scheme improved the convenience for receiver. As a result, their privacy and security is governed.
Chapter 6

Conclusion

In this thesis, we have demonstrated the keyword search on encrypted data. Our research focuses on the privacy in which the party who requests a service discloses the minimum personal information, where the party that offers the data is guaranteed that only the requested data is revealed. In this chapter, we will outline the contribution of our research and future work of searching on encrypted data.

6.1 Contributions

At the beginning of the thesis, we introduced three scenarios. Following on, our thesis explored these scenarios, reviewed the related work and proposed four schemes. We summarize the major contributions as follows:

- In the first scenario, we require that the search not only matches the keywords, but also should be in designated users. For example, in email system, this addresses the problem that the user who has known the keyword in the trapdoor may send junk mails to the receiver. As a result, we have defined the concept of Public Key Encryption with Keyword Search in Designated Sender (PEKSDS) in Chapter 3. The intuition of the scheme is based on Public Encryption with Keyword Search scheme (PEKS) and signcryption. Moreover, we have extended the scheme to multi-user setting.

- In the second scenario, our research focuses on improving security level from the schemes based on decisional assumptions. The best possible solution is to employ symmetric key encryption. However, the symmetric searchable encryption is usually used in single-user setting. By the inspiration of utilizing symmetric searchable encryption in interactive scenario, we propose our second scheme which possesses the properties of symmetric encryption and based
on the Diffie-Hellman key exchange in Chapter 4. In our scheme, the symmetric key using in trapdoor creation is produced by Diffie-Hellman key exchange. In addition, our scheme can be easily extended to multi-user setting.

- In the last scenario, since Oblivious Transfer (OT) ensures both sender’s and receiver’s privacy are warranted, it can be easily transformed to Oblivious Keyword Search (OKS). Hence, we proposed two schemes in Chapter 5. The first of our OKS schemes combines OT and PEKS directly. In our second OKS scheme, we assume the database can be updated by every user. Thus, Public Key Encryption with Oblivious Keyword Search (PEOKS) enables that everyone can be the sender and encrypt data by the same public key. The scheme improved the convenience for receiver.

### 6.2 Future Work

Although crypto-based keyword search has attracted enough eyeballs, improvement is still needed. The issues need to be considered in future are outlined as below:

- Public key encryption with keyword search under standard model. The scheme can be proved under random oracle model might not get the proof under standard model. Thus, PEKS under standard model is an interesting topic for the researchers. The first PEKS scheme evolved from the Identity-based Encryption (IBE) scheme. Currently, some IBE schemes can be proved under standard model. Hence, the construction of a PEKS under standard model will become a hot topic in the future.

- The computational time for keyword search requires more attention. Due to rapid development of cloud computing, people are more and more familiar to the “pay as you go” model. As a result, how to reduce the consumption of time on searching encrypted data will turn to a valuable subject.

- Homomorphic encryption with keyword search. The lattice based encryption is predicted to be next generation of encryption that resists against quantum computing. Since Gentry[Mit09] first proposed fully homomorphic encryption using ideal lattice, the homomorphic encryption has already attracted researchers’ attention. Moreover, the property of homomorphic encryption suits
for searching on encrypted data. Therefore, the homomorphic encryption with keyword search will lead the way of searchable encryption research.


