J Supercomput (2018) 74:1001–1023
https://doi.org/10.1007/s11227-015-1515-8

# A shareable keyword search over encrypted data in cloud computing

**Li Xu[1] · Chi-Yao Weng[2] · Lun-Pin Yuan[2] · Mu-En Wu[3] · Raylin Tso[4] · Hung-Min Sun[2]**

Published online: 12 September 2015

**Abstract** Cloud storage is one of the most important applications in our daily lives. User can store their own data into cloud storage and remotely access the saved data. Owing to the social media develops, users can share the digital files to other users, leading to the amount of data growing rapidly and searching abilities necessarily. In the some cases, servers cannot avoid data leakage even if the server provides complete access control. The encrypted data is a best way to resolve this problem but it may eliminate original structure and searching may become impossible. Applying searchable encryption for each receiver may produce messy duplication and occupy the quota of cloud storage from each receiver. User requires keeping their shared documents belonging up to date which are compared with the latest version. To this

✉ Hung-Min Sun
hmsun@cs.nthu.edu.tw

Li Xu
xuli@fjnu.edu.cn

Chi-Yao Weng
cyweng@is.cs.nthu.edu.tw

Lun-Pin Yuan
lunpin@is.cs.nthu.edu.tw

Mu-En Wu
mn@scu.edu.tw

Raylin Tso
raylin@cs.nccu.edu.tw

1 College of Mathematics and Computer Science, Fujian Normal University, Fuzhou, China

2 Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, ROC

3 Department of Mathematics, Soochow University, Taipei, Taiwan, ROC

4 Department of Computer Science, National Chengchi University, Taipei, Taiwan, ROC

aim, we thus propose a sharable ID-based encryption with keyword search in cloud computing environment, which enables users to search in data owners' shared storage while preserving privacy of data. For the performance analysis, we demonstrate the compared resultant with others ID-based or ID-relative encryption. In addition to that, we show the formal proof to verify the security of our proposed.

**Keywords** Cloud storage · Searchable encryption · ID-based encryption · Keyword search

## 1 Introduction

Cloud computing has been one of the popular technologies in the past recent years. Tons of useful applications and services are developed and later provided to regular users. Take cloud storage service as an example; Dropbox, Google Drive, ASUS CLOUD, SkyDrive, etc., allow users to store their data in cloud storage and access them remotely later. Cloud storage service allows users to access their data whenever and wherever users want. Furthermore, enterprises can outsource their data in cloud storage instead of establishing private data center to reduce the costs of management and maintenance [1]. As a result, the number of accounts of cloud storage service increases rapidly, and so does the amount of data per user. Cloud storage has become essential in our daily lives. When the amount of data per user grows, it becomes much more difficult for a user to trace a file they needed at once. Users may forget where it is placed, or what it is titled as file. It may take a lot of time for a user to find that particular file. Therefore, the capability of searching keywords in file content is quite important to users [2]. Searching in plaintext content is not difficult, but it is not secure if doing so.

Cloud security has become an important issue to users and providers [3,4]. Users and enterprises may require to store sensitive and costly information in cloud storage. If the data is not encrypted, search operations can be done easily. But then, we cannot store data secretly even if service providers guarantee to provide complete access control management and guarantee not to peek at our data. There are always some novel methods that are meant to invade our privacy, discovered by attackers. The easiest way to protect our data is to encrypt them with our secret key [5], which is not revealed to cloud storage providers, so that unauthorized parties cannot obtain the original files. By doing so, however, it will significantly limit the searching capability because everything is transformed into unstructured ciphertext.

Another important issue is that, thanks to social applications, users are now able to share their data, which is stored in the cloud, with their friends. Allowing their friends to remotely access, as well as to find, data in data owners' cloud storage as the data owners can do. In outsourced storage of enterprises, employees are also required to share sensitive documents to other employees. However, friends and co-workers do not know data owners' secret keys and therefore they can neither operate decryption nor operate queries. Revealing data owners' private keys to them is not a good approach to resolve this particular problem because there may be some other data which are inappropriate to share. Simply re-encrypting the data and sending it to all friends or co-workers may cost a lot of computational efforts and consume great amounts of storage. Since most cloud storage providers restrict each user with a certain quota (volume limitation), users may not want their quota being filled up with the data that they do not necessarily need. Moreover, users may require keeping the shared documents up to date with the latest modification which is done by data owners, re-encrypting and sending the data again is not a good solution. In addition, data owners may require to set expiration dates on the same data to different users, which means friends or co-workers can obtain the data only within a certain duration [6]. This feature leads to the deletion of searching capability on certain files.

A lot of research on searchable encryption has been developed in the past recent years [7,8]. Cena et al. presented an efficient construction of searchable encryption (SE) based on symmetric algorithm. In this paper, they revisited the existing SE schemes and pointed out that the existing scheme has significant practical drawbacks. Authors developed an efficient searchable symmetric encryption to avoid these drawbacks [8]. The goal of searchable encryption is to encrypt data using a special approach, for instance, building secure indices, so that later queries can still be executed. A query will be transformed into a trapdoor and then be sent to the service provider. By executing the testing algorithm, the cloud storage provider is able to determine whether a file contains the given keywords that a user requires. Therefore, searching capabilities are preserved. Additionally, service providers and attackers can learn nothing from ciphertext or from trapdoors, thus privacy is also preserved. However, although researchers have shown us how to encrypt our data while preserving searching capabilities, none of them have addressed the need of searching in data owners' storage and the problem on how to share encrypted files in data owners' cloud storage while preserving searchability as well as the owner's privacy. Very few of them have addressed the requirement of deletion of searching capability.

ID-based encryption is a well-known approach for encrypted data [9–11]. This paper thus follows the advantages of ID-based approach and takes them into consideration. We use ID-based encryption in our approach for several reasons. First of all, when a user attempts to share data to friends, it is reasonable that the user has the identity (ID) of the friends, where the ID can be one of the following: an account, an email address, a phone number, and/or anything that represent each unique user. In ID-based encryption, the ID works as the public key of the receiver, thus complex key management, certificate management, and certificate verification are no longer required, which means that the sharing process is simplified in our approach. Second, it is good to have a trusted authority party which is responsible for authorization in enterprise point of view. For example, only registered and authorized users can use the system, or another example, only authorized company members are able to use company's outsourced storage service. ID-based cryptography requires a trusted third party to generate private keys for each user. A user cannot do anything if the user does not have a private key in the system. Third, ID-based encryption is a public key encryption, different secure indices and trapdoors can be generated even if the source keywords are identical.

Our goal is to design a reasonable approach of shareable and searchable encryption that allows data owners to share their data to their friends, so that their friends can both decrypt and search in owner's storage without occupying friends' storage quota. Our work has the following features: (1) privacy preserving; not only should data be protected, but also the owners' privacy. Our work prevents friends from peeking into the data that is not shared with them. (2) Efficiency; we compared the computational complexity between our work and other related works. (3) The data owners can specify different keyword sets to different set of receivers. For example, Bob can find a particular document by searching a keyword "technical", while Coral cannot; however, Coral may find the same file by searching the keyword "Bob" as the owner configured. (4) Data owners can enable and disable the searching ability on their data to their friends whenever they want.

In this paper, we focus on asymmetric searchable encryption schemes, more precisely, public key encryption with keyword search (PEKS) schemes and identity-based encryption with keyword search (IDEKS) schemes. However, unlike other researchers, we mainly focus on IDEKS in cloud storage, in which the public information of data owner is also involved in algorithms. This is because users are able to search in the owner's storage and we need to take care of the owner's privacy. We have following two basic assumptions. We assume that cloud storage service providers are semi-trusted, which means they are curious but honest to user data. Also, we assume that there is always a trusted authority party which is responsible for user management and authorization in the system.

The rest of this paper is organized as follows: Sect. 2 introduces several related works in searchable encryption that has been done in the past recent years. Section 3 presents mathematical preliminaries that is essential to our work. Section 4 describes our work in detail. Analysis and evaluation are given in Sect. 5. Finally, Sect. 6 describes future work and concludes this paper.

## 2 Related works

In 1985, Shamir [9] first introduced ID-based encryption and signature scheme to simplify key management and certificate management in certificate-based public key cryptosystems. In 2001, Boneh and Franklin [10] first proposed a practical ID-based encryption from Weil pairing. Later in 2004, Boneh and Boyen [11] proposed efficient and selective-ID secure ID-based encryption without random oracles. In the same year 2004, Boneh et al. [12] proposed public encryption with keyword search (PEKS), which enables users to search encrypted keywords without compromising the security of original data.

Many related research on searchable encryption have been done in the past recent years [7,8]. We focus on schemes of public key encryptions with keyword search (PEKS). Baek et al. [13] proposed a provably secure PEKS scheme that removes the secure channel assumption. Tian and Wang [14] proposed an ID-based encryption with keyword search (IDEKS) scheme which can be used in mail system to help classify mails by different keywords or subjects. Rhee et al. [15] proposed an improved searchable public key encryption with designated tester. Liu et al. [16] proposed an efficient privacy preserving keyword search scheme in cloud computing, which allows service providers to participate in partial decipherment to reduce computational overhead in decryption for users. Lai et al. [17] proposed a scheme of efficient and expressive search on encrypted data which supports arbitrary monotone boolean predicates.

There is also research on efficient and conjunctive keyword search in encrypted data. It allows users to generate trapdoors based on different set of keywords. Wang et al. [18] proposed a scheme of keyword-field-free conjunctive keyword search on encrypted data and apply it to dynamic group settings. Kerschbaum [19] proposed a searchable encryption scheme that supports conjunctive keyword search without the need to specify positions for keywords. Ding et al. [21] proposed an efficient public key encryption with conjunctive keyword search scheme, in which no pairing operation is involved in encryption and trapdoor phases.

Some research on PEKS allows data owners to delegate the searching abilities to multiple authorized users, so that authorized users can search and decrypt the encrypted data. Ibraimi et al. [23] proposed a public key encryption with delegated search, which allows private key holders to create master trapdoors and delegate it to other entities with the ability to search encrypted data. Li et al. [24] proposed authorized private keyword search over encrypted data in cloud computing, in which multiple data owners encrypt their records and allow searches by multiple users. Yang et al. [25] proposed a scheme of multi-user private keyword search for cloud computing, especially for enterprise-outsourcing-database-to-cloud settings, in which multiple entitled users can search and write to the database. Tang et al. [26] proposed an asymmetric searchable encryption scheme, in which data owner can keep data encrypted while still being able to recover the plaintext and authorize third-party servers to search.

Few related research on PEKS with delegated search mentioned that previous works have some disadvantages in some aspects. Some requires cloud servers being fully trusted because servers hold master trapdoors and leakage is possible if servers are compromised. Some requires data owners to share either everything or nothing to authorized users, in other words, authorized users can search either all data or none from owners because owners cannot choose what to share or not. And most of all, all of them require data owners to keep tracks of public keys of other users, and certificate management may be involved in procedures. Li et al. [27] proposed efficient and secure data sharing in cloud computing based on pairings, which is most similar to our work. However, in [27], when a new coming user asks for private keys, all private keys kept in users have to be updated. This approach is not good when serving large amount of users, especially in cloud applications. In our work, new user requests are much more efficient. Furthermore, different from [27] and all previous works, we focus on enabling cloud storage service users to search in data owners' storage to eliminate great volume of duplicate occupation on same data while still preserving data owners' privacy. Our work can be applied in volume-limited (quota-based) cloud storage. In our work, data owners can enable and disable searching capabilities of other users upon certain files whenever the owners want. All in all, we proposed a shareable and searchable ID-based encryption in cloud storage, so that process can be more flexible and more efficient.

## 3 Preliminaries

In this section, we briefly introduce preliminaries of our work, including related definitions and complexity assumptions. Also, we give definitions of operations used in ID-based encryption with keyword search (IDEKS).

### 3.1 Bilinear pairings

Let $(\mathbb{G}_1, +)$ and $(\mathbb{G}_2, \times)$ be two cyclic groups of some large prime order $q$. We view $(\mathbb{G}_1, +)$ as an additive group and $(\mathbb{G}_2, \times)$ as a multiplicative group.

**Definition 1** A bilinear pairing is defined as a function $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with following properties.

1. Bilinear: for all $P, Q \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}_q$, $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$.
2. Non-degenerate: if $P$ is a generator in $\mathbb{G}_1$, then $\hat{e}(P, P) \neq 1$ is a generator in $\mathbb{G}_2$, where 1 is the identity.
3. Computable: there exists a polynomial time algorithm to compute $\hat{e}(P, Q) \in \mathbb{G}_2$ for all $P, Q \in \mathbb{G}_1$.

Such bilinear pairings can be achieved using Weil or Tate pairings on supersingular elliptic curves. Next, we define complexity assumptions to prove security in later sections, especially in the security analysis section. Please note that the following complexity assumptions are not used in evaluating performance but used only in security proofs.

**Definition 2** The discrete logarithm (DL) problem is defined as follows: given a random element $X \in \mathbb{G}$, where $(\mathbb{G}, \star) = \langle \mathcal{G} \rangle$ is a finite cyclic group of prime order $q$ with a generator $\mathcal{G}$, find an integer $x \in \mathbb{Z}_q$ such that $X = power(\mathcal{G}, x)$. Note that, we use $x\mathcal{G}$ notations in $(\mathbb{G}_1, +)$ and $\mathcal{G}^x$ in $(\mathbb{G}_2, \times)$. We say that the DL assumption holds if no polynomial time algorithm has a least advantage in solving the DL problem.

**Definition 3** The bilinear Diffie–Hellman (BDH) problem is defined as follows: given $P, aP, bP, cP \in \mathbb{G}_1$ for some $a, b, c \in \mathbb{Z}_q$, compute $\hat{e}(P, P)^{abc} \in \mathbb{G}_2$. We say that the BDH assumption holds if no polynomial time algorithm has a least advantage in solving the BDH problem.

### 3.2 Outline of the IDEKS scheme

In ID-based encryption with keyword search (IDEKS) scheme, three parties are involved, including sender, receiver, and server. The sender is a party that creates and sends encrypted document along with IDEKS ciphertexts (or secure indices) to the server. The server is a trusted third party that generates secret keys for each user, it then sends the user's secret key to the user via secure channel; it also performs testing algorithm upon IDEKS ciphertext after receiving trapdoors from receivers. The receiver is a party that generates trapdoors and sends them to the server to specify the document that it requires. Unfortunately, since the server holds the master key, it has the ability to generate trapdoors as receivers in IDEKS by forging queries. Next, we give definitions of operations in IDEKS.

**Definition 4** As shown in Fig. 1, an ID-based encryption with keyword search (IDEKS) scheme consists of the following algorithms.

1. $Setup(k)$: given a security parameter $k$ as input, it outputs system parameters and a public/master key pair $(P_{pub}, s)$ for the server.

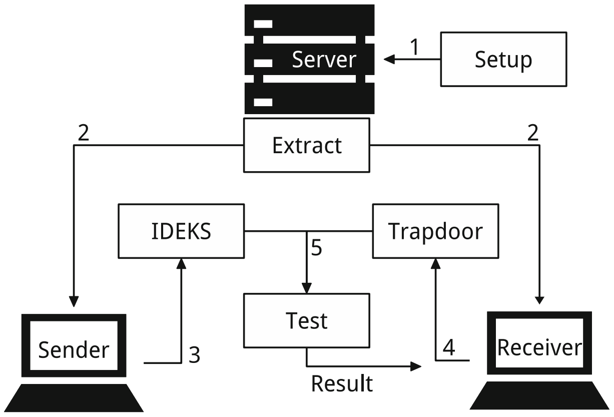

**Fig. 1** ID-based encryption with keyword search

2. $Extract(s, ID_i)$: given a master key $s$ and a unique identity $ID_i$ representing a user $u_i$, it computes $u_i$'s public/secret key pair $(Q_{ID}, S_{ID})$ and sends $S_{ID}$ to $u_i$ through a secure channel.
3. $IDEKS(Q_{ID}, P_{pub}, w)$: given receiver's public key $Q_{ID}$, server's public key $P_{pub}$, and a keyword $w$ as inputs, it outputs an IDEKS ciphertext.
4. $Trapdoor(S_{ID}, w')$: given receiver's secret key $S_{ID}$ and a keyword $w'$, it outputs a trapdoor $T_w$ associated to the keyword $w'$.
5. $Test(Q_{ID}, s, T_w)$: given receiver's public key $Q_{ID}$, server's master key $s$, and a trapdoor $T_w$ as inputs, it outputs 1 if $w = w'$ or 0 otherwise.

### 3.3 Assumptions

We assume that all users have adequate knowledge of all identities of their friends by any means. We assume that the cloud server is semi-trusted, which means the cloud server is under a curious-but-honest model assumption. The cloud server may try to decrypt and peek at the data users uploaded, but it will never modify the data. Therefore, encryptions on document and keywords are necessary to secure data from cloud server. On the other hand, an authority is assumed to be a trusted party who holds the master key and is responsible for user registration and secret key generation. The authority may want to investigate encrypted documents if necessary, however, request forgery cannot be done in our scheme. And finally, we assume keyword fields exists (title, author, date, etc.) as well as keyword fields indexing for conjunctive keywords search.

## 4 Proposed scheme

In this section, we describe our shareable ID-based encryption with keyword search scheme. We first give an overview of our scheme and related definitions, then we describe two proposed algorithms, SIDEKS and simplified SIDEKS. Our proposed schemes can be adapted to different data communication environments, e.g., the SIDEKS is used for the data communication without encrypted channel, the simplified SIDEKS is applied to the communication environment with encrypted channel. Later on, our proposed schemes will be introduced in details.

## 4.1 Overview

In shareable ID-based encryption with keyword search (SIDEKS) scheme, four parties are involved, including data owner, receiver, server, and authority. The authority is a trusted party that generates private key for each registered user, it then sends user's private key to user via a secure channel. The data owner is a party that creates and sends encrypted document along with SIDEKS ciphertexts (or secure indices) to the server; it can specify different keywords to different users. The server is a party that receives and stores encrypted document as well as SIDEKS ciphertexts; it also performs the testing algorithm after receiving trapdoors from receivers. The receiver is a party that generates trapdoors and sends them to the server to specify the document that it requires. Note that, the receiver does not actually receive the encrypted document which is sent by data owners; instead, the receiver searches and downloads the encrypted document from data owners' storage. After downloading from cloud storage, the receiver can further share the document as a data owner.

Here, we give a brief example of how it works. As soon as the service is established, the administrator runs $Setup$ to install master key and cloud's secret key to the authority and the cloud server respectively. Later, when a user, say Alice, is trying to register to this service, the authority runs $Extract$ as soon as Alice is authorized and permitted to use the service. After that, authority generates a public/secret key pair $(Q_{ID}, S_{ID})$ and secretly sends $S_{ID}$ to Alice. After receiving the $S_{ID}$, Alice finalizes it secret keys $sk_{ID}$ and reveals $P_{ID}$. Alice then runs *SIDEKS* and uploads an encrypted project which is shared with another user, say Bob, a project team member. Alice can run $AddReq$ to further enable another team member, say Charlie, to search for this encrypted project. Alice also can run $DelReq$ to disable Bob's searchability on this project. When Charlie is about to work, he runs $Trapdoor$ to obtain a reference link of the project. And after downloading it, he runs $Decrypt$.

Next, we give definitions of operations in SIDEKS.

**Definition 5** As shown in Fig. 2, a shareable ID-based encryption with keyword search (SIDEKS) scheme consists of following algorithms.

1. $Setup(k)$: given a security parameter $k$, it outputs system parameters $\mathcal{SP}$, the authority's master key $s$, and a shared secret $c$ which is shared with the cloud server.
2. $Extract(s, ID)$: run by the authority. Given the authority's master key $s$ and a unique ID representing a user $u_i$, it calculates $u_i$'s public/secret key pair $(Q_{ID}, S_{ID})$ and secretly sends $S_{ID}$ to $u_i$. By receiving $S_{ID}$, $u_i$ finalizes its secret key $sk_{ID}$ and reveals $P_{ID}$. Note that, all users can derive $Q_{ID}$ from $ID$, therefore, public key and certificate delivery are no longer required. We use notation $Q_{ID}$ as the public key instead of $ID$ in following algorithms because $Q_{ID}$ is automatically calculated.

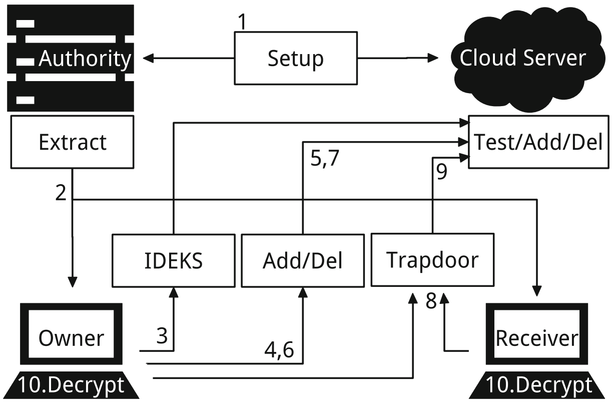

**Fig. 2** Shareable ID-based encryption with keyword search

3. *SIDEKS*$(sk_{ID}, \mathcal{D}, \mathcal{U}, \mathcal{W})$: run by data owners. Given the owner's secret key $sk_{ID}$, a document $\mathcal{D}$, a set of receiver $\mathcal{U}$, and a set of keywords $\mathcal{W}$, it outputs an encrypted document $\mathcal{D}'$ and an upload request $\mathcal{UPLOAD}$. As a result, for each receiver $u_i \in \mathcal{U}$ can find $\mathcal{D}$ by querying any keyword $w_j \in \mathcal{W}$. Note that the data owner itself should be included in $\mathcal{U}$.
4. $AddRequest(sk_{ID}, \mathcal{R}, \mathcal{U}, \mathcal{W})$: run by data owners. Given the owner's secret key $sk_{ID}$, a reference $\mathcal{R}$ which refers to an encrypted document, a set of receiver $\mathcal{U}$, and a set of keywords $\mathcal{W}$, it outputs an add request $\mathcal{ADD}$. Note that $\mathcal{R}$ can be a temporary data pointer (link, address, index, etc.) specifying an encrypted document; $\mathcal{R}$ can be obtained only after Test algorithm. To set different keyword set for different user set, the data owner has to run this algorithm after *SIDEKS* is done.
5. $AddUser(\mathcal{ADD}, \mathcal{D}')$: run by the cloud server. Given an add request $\mathcal{ADD}$ and an encrypted document $\mathcal{D}'$, it adds an entry into $\mathcal{D}'$.
6. $DelRequest(sk_{ID}, \mathcal{R}, \mathcal{U})$: Run by data owners. Given the owner's secret key $sk_{ID}$, a reference $\mathcal{R}$, and a set of receiver $\mathcal{U}$, it outputs a delete request $\mathcal{DEL}$ which will disable $\mathcal{U}$ to find $\mathcal{R}$.
7. $DelUser(c, P_{ID}, \mathcal{DEL}, \mathcal{D}')$: run by the cloud server. Given the shared secret $c$, a delete request $\mathcal{DEL}$ and an encrypted document $\mathcal{D}'$, it deletes an entry in $\mathcal{D}'$.
8. $Trapdoor(sk_{ID}, Q_{ID_0}, \mathcal{W})$: run by users. Given user's secret key $sk_{ID}$, data owner's public key $Q_{ID_0}$, and a set of keywords $\mathcal{W}$, it outputs a trapdoor $T_w$ and a trapdoor request $\mathcal{TREQ}$.
9. $Test(c, T_w, \mathcal{D}')$: run by the cloud server. Given the shared secret $c$, a trapdoor $T_w$, and an encrypted document $\mathcal{D}'$, it outputs $(1, \mathcal{R})$ if $\mathcal{D}'$ contains specified keywords, or $(0, null)$ otherwise.
10. $Decrypt(sk_{ID}, Q_{ID_0}, d')$: run by users. Given user's secret key $sk_{ID}$, data owner's public key $Q_{ID_0}$, and a customized encrypted document $d'$, it outputs the original document $\mathcal{D}$.
11. $Sign(sk_{ID}, \mathcal{REQ})$: run by users. Given user's secret key $sk_{ID}$ and a request $\mathcal{REQ}$, it signs the request and outputs a signed $\mathcal{REQ}'$.

12. $Verify(P_{ID}, \mathcal{REQ}')$: run by the cloud server. Given user's public verifying key $P_{ID}$ and a signed request $\mathcal{REQ}'$, it verifies $\mathcal{REQ}'$ and outputs either valid or invalid.

## 4.2 Security model

Here, we introduce our semantic security model. The overview of semantic security model is stated as follows. In game 1, an adversary $\mathcal{A}_1$, which is assumed to be the cloud server, cannot deduce document contents from trapdoors even if the adversary has the shared secret $c$. In game 2, an adversary $\mathcal{A}_2$, which is assumed to be a receiver, cannot deduce document contents without the shared secret $c$ even if the adversary has all trapdoors. The goal is to prove that trapdoors and SIDEKS ciphertexts do not reveal anything about the encrypted document.

**Definition 6** (*IND-CKA Game* 1) The indistinguishability under chosen keyword attack (IND-CKA) game 1 is defined as follows. $\mathcal{A}_1$ is assumed to be the cloud server. Given a keyword and two encrypted documents of equal size, determine which document contains the keyword.

– $Setup$: a challenger $\mathcal{B}$ runs the Setup($k$) algorithm and gives $\mathcal{A}_1$ system parameters $\mathcal{SP}$ and $\mathcal{A}_1$'s shared secret $c$.
– $Phase1$: $\mathcal{A}_1$ asks for a number of trapdoors $T_w$ for any $(id, w)$ of its choice from a trapdoor oracle. Note that $id$ represents a user and $w$ can be either a single keyword or a keyword list.
– $Challenge$: $\mathcal{A}_1$ outputs a target pair $((id_0^*, w_0^*), (id_1^*, w_1^*))$ with restriction that trapdoors for $(id_0^*, w_0^*)$ and $(id_1^*, w_1^*)$ are not asked in $Phase1$. $\mathcal{A}_1$ then sends $((id_0^*, w_0^*), (id_1^*, w_1^*))$ to a challenger $\mathcal{B}$. Upon receiving this, $\mathcal{B}$ randomly chooses $b \in_R \{0, 1\}$ and computes SIDEKS ciphertext $\mathcal{S}^* = SIDEKS(sk_{ID}, \mathcal{D}, u_{id}, w_b^*)$. $\mathcal{B}$ returns $S^*$ to $\mathcal{A}_1$.
– $Phase2$: $\mathcal{A}_1$ asks for a number of trapdoors $T_w$ for any $(id, w)$ of its choice from a trapdoor oracle with restriction that $(id, w) \neq (id_0^*, w_0^*), (id_1^*, w_1^*)$.
– $Guess$: $\mathcal{A}_1$ outputs its guess $b' \in \{0, 1\}$ and wins the game if $b = b'$.

We define $\mathcal{A}_1$'s advantage in breaking SIDEKS as

$$Adv_{\mathcal{A}_1}(k) = |Pr[b = b'] - 1/2|$$

**Definition 7** (*IND-CKA Game* 2) The indistinguishability under chosen keyword attack (IND-CKA) game 2 is defined as follows. $\mathcal{A}_2$ is assumed to be a receiver. Given a keyword and two encrypted documents of equal size, determine which document contains the keyword.

– $Setup$: by running Extract(ID) algorithm, $\mathcal{A}_2$ receives a secret $S_{ID}$ and finalizes its secret key $sk_{ID}$. Later, $\mathcal{A}_2$ reveals its verifying key $P_{ID}$ to challenger $\mathcal{B}$.
– $Phase1$: $\mathcal{A}_2$ asks for a number of test results for any given $\mathcal{S}$ and the trapdoor $T_w$ of its choice from a test oracle.

- *Challenge*: $\mathcal{A}_2$ outputs a target keyword pair $(w_0^*, w_1^*)$ with restriction that trapdoors for $w_0^*$ and $w_1^*$ are not asked in *Phase*1. $\mathcal{A}_2$ then sends $(w_0^*, w_1^*)$ to a challenger $\mathcal{B}$. Upon receiving this, $\mathcal{B}$ randomly chooses $b \in_R \{0, 1\}$ and computes SIDEKS ciphertext $\mathcal{S}^* = SIDEKS(sk_{ID}, \mathcal{D}, \mathcal{U}, w_b^*)$. $\mathcal{B}$ returns $S^*$ to $\mathcal{A}_2$.
- *Phase*2: $\mathcal{A}_2$ asks for a number of test results for any given $\mathcal{S}$ and the trapdoor $T_w$ of its choice from a test oracle.
- *Guess*: $\mathcal{A}_2$ outputs its guess $b' \in \{0, 1\}$ and wins the game if $b = b'$.

We define $\mathcal{A}_2$'s advantage in breaking SIDEKS as

$$Adv_{\mathcal{A}_2}(k) = |Pr[b = b'] - 1/2|$$

Friday, August 28, 2015 at 6:47 am

**Definition 8** We say that SIDEKS is semantically secure under chosen keyword attack if for all polynomial time attacker $\mathcal{A}_{1,2}$ has a negligible advantage in winning the game.

## 4.3 SIDEKS details

We construct SIDEKS as follows. Note that the encrypted file will occupy only the data owner's storage quota.

1. *Setup*$(k)$: given a sufficiently large security parameter $k \in \mathbb{Z}^+$, it generates two cyclic groups $(\mathbb{G}_1, +)$, $(\mathbb{G}_2, \times)$ of some large prime order $q$ and a bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Randomly, choose a generator $P \in_R \mathbb{G}_1$ and $s, c \in_R \mathbb{Z}_q$, then compute $P_{pub} = sP$ and $P_{cld} = cP$. Select collision-free one-way hash functions $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}_1$, $H_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$, $H_3 : \mathbb{G}_2 \rightarrow \mathbb{Z}_q$, and $H_4 : \mathbb{Z}_q \rightarrow \{0, 1\}^k$. For example, $H_4$ can be SHA256 and $H_1, H_2, H_3$ can be combination of SHA256 with element_from_hash function in PBC library. Select symmetric encryption/decryption $(Enc, Dec)$. Output authority's master key $s$, the shared secret $c$, and system parameters

$$\mathcal{SP} = (\hat{e}, \mathbb{G}_1, \mathbb{G}_2, q, P, P_{pub}, P_{cld}, H_1, H_2, H_3, H_4, Enc, Dec)$$

2. *Extract*$(s, ID)$: compute user $u_i$'s public key $Q_{ID} = H_1(ID)$ and a secret $S_{ID} = sQ_{ID}$. Send $S_{ID}$ to $u_i$ through a secure channel such as ssh. By receiving $S_{ID}$, $u_i$ randomly chose $s_{ID} \in_R \mathbb{Z}_q$ and finalize its secret key $sk_{ID} = (S_{ID}, s_{ID})$. Later, $u_i$ reveals $P_{ID} = s_{ID}P$ to the cloud server as a part of the cloud server's system parameters. Note that, since $H_1$ is a part of system parameters $\mathcal{SP}$, all users can derive the public key $Q_{ID}$ from $ID$. Therefore, public key and certificate delivery are no longer required; the only information of each receiver that the sender has to know is their identity. We use notation $Q_{ID}$ as the public key instead of $ID$ in following algorithms because $Q_{ID}$ is automatically calculated by each user.
3. *SIDEKS*$(sk_{ID}, \mathcal{D}, \mathcal{U}, \mathcal{W})$: randomly choose an encryption key $ek$ and a random number $x \in_R \mathbb{Z}_q$. For each receiver $u_i$ in $\mathcal{U}$, compute $Q_{IDi} = H_1(ID_i)$, $\mu_i = H_3(\hat{e}(S_{ID}, Q_{IDi}))$, $U_i = x\mu_i Q_{IDi}$, $V_i = x\mu_i P_{cld}$, and $K_i = Enc(H_4(\mu_i), ek)$,

then set receiver search entry $E_i = (U_i, V_i, K_i)$. For each keyword $w_j$ in keyword field $j$, compute SIDEKS ciphertext $C_j = xH_2(w_j)P_{pub}$. For the document, compute the ciphertext $\mathcal{C} = Enc(ek, \mathcal{D})$, digest $h = H_2(\mathcal{C} \| ek)$, and document signature $\sigma_{\mathcal{D}} = (\sigma_{\mathcal{D}_1}, \sigma_{\mathcal{D}_2}) = (xP, xQ_{ID} + hS_{ID})$. Finally, output encrypted document $\mathcal{D}' = (\mathcal{C}, \{E_i\}, \{C_j\}, \sigma_{\mathcal{D}})$, and the upload request $\mathcal{UPLOAD} = (ID, \mathcal{D}')$, sign $\mathcal{UPLOAD}$ and send it to the cloud server.

4. $AddRequest(sk_{ID}, \mathcal{R}, \mathcal{U}, \mathcal{W})$: obtain $ek$ by decrypting any $K_i$ generated previously; randomly choose a number $x \in_R \mathbb{Z}_q$. Compute search entries $\{E_i\}$ and SIDEKS ciphertexts $\{C_j\}$ as in previous algorithm. Output an add request $\mathcal{ADD} = (ID, \mathcal{R}, \{E_i\}, \{C_j\})$, sign it and send it to the cloud server.
5. $AddUser(\mathcal{ADD}, \mathcal{D}')$: after verification on $\mathcal{ADD}$, the cloud server inserts $\{E_i\}$ and $\{C_j\}$ to specified document.
6. $DelRequest(sk_{ID}, \mathcal{R}, \mathcal{U})$: for each receiver $u_i$, compute $F_i = s_{ID}Q_{IDi}$. Output a delete request $\mathcal{DEL} = (ID, \mathcal{R}, \{F_i\})$, sign it and send it to the cloud server.
7. $DelUser(c, P_{ID}, \mathcal{DEL}, \mathcal{D}')$: after verification on $\mathcal{DEL}$, the cloud server operates following steps. For each entry $E_i = (U_i, V_i, K_i)$ in $\mathcal{D}'$, and each $F_j$ in $\mathcal{DEL}$, the cloud server tests whether the following equation holds; if holds, delete $E_i$ in $\mathcal{D}'$.

$$H_3(\hat{e}(U_i, P_{ID})) = H_3(\hat{e}(F_j, V_i)^{1/c})$$

8. $Trapdoor(sk_{ID}, Q_{ID_0}, \mathcal{W})$: randomly choose a number $y \in_R \mathbb{Z}_q$. Compute $\mu = H_3(\hat{e}(Q_{ID_0}, S_{ID}))$ and the trapdoor $T_w = (\mathcal{I}, T_{w1}, T_{w2}, T_{w3})$, where $\mathcal{I}$ is a set of specified keyword fields, $T_{w1} = \hat{e}(s_{ID}Q_{ID} + S_{ID}, y\mu P_{cld})$, $T_{w2} = y\mu Q_{ID}$, and $T_{w3} = \sum yH_2(w_i)\ S_{ID}$ for keyword $w_i$ in keyword field $i$. Output a trapdoor request $\mathcal{TREQ} = (ID, T_w)$, sign it and send it to the cloud server.
9. $Test(c, T_w, \mathcal{D}')$: after verification and taking $T_w = (\mathcal{I}, T_{w1}, T_{w2}, T_{w3})$ and $\mathcal{D}' = (C, \{U_i, V_i, K_i\}, \{C_j\}, \sigma_{\mathcal{D}'})$ as inputs, the cloud server first extracts specified keyword fields from $\mathcal{I}$ and computes $C_{sum} = \sum C_j$ for SIDEKS ciphertexts $C_j$ in keyword field $j$. Determine the result by testing whether the following equation holds. If holds, output 1 and generate a temporary entry reference $\mathcal{R}$ which specified an entry

$$H_3(T_{w1} \cdot \hat{e}(T_{w3}, V_i)) = H_3(\hat{e}(P_{ID} + P_{pub} + C_{sum}, cT_{w2}))$$

10. $Decrypt(sk_{ID}, Q_{ID_0}, d')$: by receiving $d' = (C, \{U, V, K\}, \sigma_D)$, the user computes $\mu = H_3\ (\hat{e}(Q_{ID_0}, S_{ID}))$ and obtains encryption key $ek = Dec(H_4(\mu), K)$. Verify it by testing whether the following equation holds. If holds, obtain original document by $\mathcal{D} = Dec(ek, \mathcal{C})$.

$$H_3(\hat{e}(Q_{ID_0}, \sigma_{\mathcal{D}_1} + H_2(\mathcal{C} \| ek)P_{pub})) = H_3(\hat{e}(P, \sigma_{\mathcal{D}_2}))$$

11. $Sign(sk_{ID}, \mathcal{REQ})$: generate a timestamp $t$, compute signature $\sigma = s_{ID}H_1(\mathcal{REQ} \| t)$, and output signed request $\mathcal{REQ}' = (\mathcal{REQ}, t, \sigma)$.

12. $Verify(P_{ID}, \mathcal{REQ}')$: verify it by testing whether the following equation holds. Output valid if it holds, invalid otherwise.

$$H_3(\hat{e}(\sigma, P)) = H_3(\hat{e}(H_1(\mathcal{REQ}\|t), P_{ID})$$

### 4.4 Simplified SIDEKS

The SIDEKS has a simplified form which is constructed similarly except for Trapdoor and Test. The rest of the operations are the same as mentioned above. We describe how we construct simplified SIDEKS (sSIDEKS) in this section. In the original thought, the reason of $T_{w1}$ is to give the user a chance to verify cloud server and to give cloud server a chance to verify whether or not the user is authorized when short signature scheme is not applied because of preference or performance. Further protocols will be put in future works. We can observe that $T_{w1}$ equals to $\hat{e}(s_{ID}Q_{ID} + S_{ID}, y\mu P_{cld})$ and equals to $\hat{e}(P_{ID} + P_{pub}, T_{w2})^c$ in later consistency section. We suggest that service providers should always use non-simplified proposed scheme with short signature verification for double safety.

– $Trapdoor(sk_{ID}, Q_{ID_0}, \mathcal{W})$: randomly choose a number $y \in_R \mathbb{Z}_q$. Compute $\mu = H_3(\hat{e}(Q_{ID_0}, S_{ID}))$ and the trapdoor $T_w = (\mathcal{I}, T_{w2}, T_{w3})$, where $\mathcal{I}$ is a set of specified keyword fields, $T_{w2} = y\mu Q_{ID}$, and $T_{w3} = \sum yH_2(w_i)S_{ID}$ for keyword $w_i$ in keyword field $i$. Output a trapdoor request $\mathcal{TREQ} = (ID, T_w)$, sign it and send it to the cloud server.
– $Test(c, T_w, \mathcal{D}')$: after verification and taking $T_w = (\mathcal{I}, T_{w1}, T_{w2}, T_{w3})$ and $\mathcal{D}' = (C, \{U_i, V_i, K_i\}, \{C_j\}, \sigma_{\mathcal{D}'})$ as inputs, the cloud server first extracts specified keyword fields from $\mathcal{I}$ and computes $C_{sum} = \sum C_j$ for SIDEKS ciphertexts $C_j$ in keyword field $j$. Determine the result by testing whether the following equation holds. If holds, output 1 and generate a temporary entry reference $\mathcal{R}$ which specified an entry

$$H_3(\hat{e}(T_{w3}, V_i)) = H_3(\hat{e}(C_{sum}, cT_{w2}))$$

### 4.5 Consistency

Here, we show the consistency of our scheme, including request signature verification, document signature verification, user deletion, and trapdoor testing.

1. Request signature verification: further details can be found in short signature from the Weil pairing [20].

$$\begin{aligned}\hat{e}(\sigma, P) &= \hat{e}(s_{ID}H_1(\mathcal{REQ}), P)\\ &= \hat{e}(H_1(\mathcal{REQ}), s_{ID}P)\\ &= \hat{e}(H_1(\mathcal{REQ})\|t), P_{ID})\end{aligned}$$

2. Document signature: further details can be found in batch verification with id-based signature [30].

$$\begin{aligned}\hat{e}(P, \sigma_{\mathcal{D}_2}) &= \hat{e}(P, xQ_{ID} + hS_{ID})\\ &= \hat{e}(P, xQ_{ID} + hsQ_{ID})\\ &= \hat{e}(Q_{ID}, xP + H_2(\mathcal{C}\|ek)sP)\\ &= \hat{e}(Q_{ID}, \sigma_{\mathcal{D}_1} + H_2(\mathcal{C}\|ek)P_{pub})\end{aligned}$$

3. User deletion:

$$\begin{aligned}\hat{e}(U, P_{ID}) &= \hat{e}(x\mu Q_{ID}, s_{ID}P)^{c/c}\\ &= \hat{e}(s_{ID}Q_{ID}, x\mu P_{cld})^{1/c}\\ &= \hat{e}(F, V)^{1/c}\end{aligned}$$

4. SIDEKS trapdoor test:

$$\begin{aligned}T_{w1}\cdot\hat{e}(T_{w3}, V) &= \hat{e}(s_{ID}Q_{ID} + S_{ID}, y\mu P_{cld})\cdot\hat{e}\left(\sum_i yH_2(w_i)S_{ID}, x\mu P_{cld}\right)\\ &= \hat{e}(s_{ID}Q_{ID} + sQ_{ID}, y\mu cP)\cdot\hat{e}\left(\sum_i yH_2(w_i)sQ_{ID}, x\mu cP\right)\\ &= \hat{e}(s_{ID}P + sP, y\mu Q_{ID})^c\cdot\hat{e}\left(\sum_i xH_2(w_i))sP, y\mu Q_{ID}\right)^c\\ &= \hat{e}\left(P_{ID} + P_{pub} + \sum_i xH_2(w_i)P_{pub}, y\mu Q_{ID}\right)^c\\ &= \hat{e}\left(P_{ID} + P_{pub} + C_{sum}, T_{w2}\right)^c\end{aligned}$$

5. sSIDEKS Trapdoor Test:

$$\begin{aligned}\hat{e}(T_{w3}, V) &= \hat{e}\left(\sum_i yH_2(w_i)S_{ID}, x\mu P_{cld}\right)\\ &= \hat{e}\left(\sum_i yH_2(w_i)sQ_{ID}, x\mu cP\right)\\ &= \hat{e}\left(\sum_i xH_2(w_i))sP, y\mu Q_{ID}\right)^c\\ &= \hat{e}\left(\sum_i xH_2(w_i)P_{pub}, y\mu Q_{ID}\right)^c\\ &= \hat{e}\left(C_{sum}, T_{w2}\right)^c\end{aligned}$$

## 5 Evaluation and analysis

Two analyses, including performance analysis and security analysis, and evaluation environment are provided in this section.

### 5.1 Performance analysis

We compare our scheme only with ID-based or ID-relative encryption with keyword search. The features provided is shown in Table 1. Scheme [14] is the original IDEKS described in the preliminaries section. It resolves the searchability upon encrypted emails for each receivers, but it does not addressed the shareability. Scheme [27] proposed an IDEKS in share storage using based on secret sharing. However, it does not mention revocation/deletion of searchability of the receivers. In this paper, our approach allows data owners to dynamically enable/disable searchability of the receivers. As far as we know, our approach is the first scheme which allows receivers to conduct search in owner's storage.

We denote $\hat{e}$ by a bilinear pairing operation, $m$ by a scalar operation in $\mathbb{G}_1$, $P$ by a power operation in $\mathbb{G}_2$, $M$ by a multiplication in $\mathbb{G}_2$, $p$ by a $H_1$ MapToPoint operation, and $u$ by the number of users, which is an extremely large number. Note that the comparison of theoretical time consumption of each PBC operation is:

$$p > m > \hat{e} > M > P$$

The evaluation of operation using C-written PBC library executed with workstation equipped with Intel(R) Xeon(R) CPU W3530 (2.80 GHz) is shown in Table 2.

In the server's point of view, the Test algorithm is the most important operation because it is frequently operated while Extract is not; SIDEKS/IDEKS and Trapdoor algorithm are operated in user side so they are not the bottleneck. In Table 3, we can see that our simplified approach provides the best performance in the test phase among these four. In other phase, our proposed scheme has a little bit higher overall computational cost than scheme [14], and slightly less than scheme [27].

Scheme [27] has an extremely high computational cost in Extract algorithm when $u$ is large as it supposed to be in cloud environment. Our approach, therefore, is better than scheme [27].

**Table 1** Related work

| | Proposed | Scheme [27] | Scheme [14] |
|---|---|---|---|
| Extract | ✓ | ✓ | ✓ |
| IDEKS | ✓ | ✓ | ✓ |
| Trapdoor | ✓ | ✓ | ✓ |
| Test | ✓ | ✓ | ✓ |
| Encrypt | ✓ | ✓ | |
| Decrypt | ✓ | ✓ | |
| AddUser | ✓ | | |
| DelUser | ✓ | | |

**Table 2** Notation Table

| Notation | Description | Average time (s) |
|---|---|---|
| $u$ | The number of cloud users | |
| $p$ | MapToPoint (hash) operation | 0.0046 |
| $m$ | Scaler operation over $\mathbb{G}_1$ | 0.0020 |
| $\hat{e}$ | Pairing operation | 0.0016 |
| $M$ | Multiplication operation over $\mathbb{G}_2$ | 0.0006 |
| $P$ | Power operation over $\mathbb{G}_2$ | 0.0002 |

**Table 3** SIDEKS/IDEKS performance

| Scheme | Extract | SIDEKS/IDEKS | Trapdoor | Test |
|---|---|---|---|---|
| Scheme [14] | $p + m$ | $3m$ | $m$ | $m + 2\hat{e}$ |
| Scheme [27] | $4um$ | $p + 5m + 4\hat{e} + M + 3P$ | $p + m$ | $m + \hat{e}$ |
| Proposed | $p + 2m$ | $p + 3m + \hat{e}$ | $p + 4m + 2\hat{e}$ | $2\hat{e} + M + P$ |
| Simplified | $p + 2m$ | $p + 3m + \hat{e}$ | $p + 2m + \hat{e}$ | $2\hat{e} + P$ |

**Table 4** SIDEKS performance

| Scheme | DelReq | DelUser | Decrypt |
|---|---|---|---|
| Scheme [27] | – | – | $2m + 4\hat{e} + 5M + 2P$ |
| Proposed | $p + m$ | $2\hat{e} + P$ | $p + m + 3\hat{e}$ |

As shown in Table 4, we can see that our proposed scheme has fewer computational cost in Decrypt algorithm. Furthermore, we have $DelReq$ and $DelUser$ in our scheme. The performance of $AddReq$ is the same with SIDEKS, and $AddUser$ is always executed in constant time.

## 5.2 Security analysis

In our scheme, we can easily see that each receiver can generate different trapdoors even though the keywords are the same. It is because $y$ is picked randomly and thus if $y \neq y'$ then $T_w \neq T'_w$. Also, signatures are involved to avoid reply attacks. On the other hand, the cloud server does not know whom the document is shared with, in other words, SIDEKS is an anonymous multireceiver encryption with keyword search. What is more, friends cannot decrypt the documents that are not being shared.

Next, we prove the security of our scheme under assumptions described in the preliminaries section. We use the simplified SIDEKS equation in our proof because it will be easier.

**Corollary 1** *The cloud server cannot determine the shared secret* $\mu = H_3(\hat{e}(Q_{ID}, S_{ID_0})) = H_3(\hat{e}(S_{ID}, Q_{ID_0})) = H_3(\hat{e}(Q_{ID}, Q_{ID_0})^s)$ *between the receiver* $u_{ID}$ *and the owner* $u_0$ *under BDH assumption.*

*Proof* Given $P_{pub} = sP = \alpha P$, $Q_{ID} = \beta P$ and, $Q_{ID_0} = \gamma P$ for some secret $\alpha, \beta, \gamma \in \mathbb{Z}_q$, compute $\hat{e}(Q_{ID}, Q_{ID_0})^s = \hat{e}(P, P)^{\alpha\beta\gamma}$ is equivalent to the BDH problem, which no polynomial time algorithm has a least advantage to solve. □

**Corollary 2** *The cloud server cannot determine the keyword $w_i$ and $s$ from given SIDEKS ciphertext $c_i$ under DL assumption.*

*Proof* Given $c_i = \alpha P_{pub}$, $P_{cld} = cP$, and cloud server's secret $c$, where $\alpha \in \mathbb{Z}_q$ is secretly kept from the cloud server, compute $\alpha$ from $c_i = \alpha P_{pub}$ is equivalent to the DL problem. □

**Theorem 1** (IND-CKA Game 1) *Our scheme is semantically secure against chosen keyword attack in Game* 1 *under the random oracle model assuming BDH problem is intractable.*

*Proof* Suppose that $\mathcal{A}_1$ is a malicious cloud server which has advantage $\epsilon$ in winning the game. We build an adversary $\mathcal{B}$ that uses $\mathcal{A}_1$ and thus $\mathcal{B}$ has advantage $\epsilon'$ in solving the BDH problem. $\mathcal{B}$ is given $P$, $P_\alpha = \alpha P$, $P_\beta = \beta P$, $P_\gamma = \gamma P \in \mathbb{G}_1$. $\mathcal{B}$'s goal is to compute $\hat{e}(P, P)^{\alpha\beta\gamma} \in \mathbb{G}_2$.

– *Setup*: $\mathcal{A}_1$ is given its secret key $c$, and system parameters $\mathcal{SP} = (\hat{e}, \mathbb{G}_1, \mathbb{G}_2, q, P, P_{pub}, P_{cld}, H_1, H_2, H_3, H_4)$, where $H_1$, $H_2$, $H_3$, and $H_4$ are controlled by $\mathcal{B}$. $\mathcal{B}$ randomly picks $y, \mu \in_R \mathbb{Z}_q$ and responds to oracle queries as follows.
  To respond to $H_1$ queries, $\mathcal{B}$ maintains $H_1$-list (initially empty) with entries $(id_i, h_i, r_i, c_i)$.
  1. If query $id_i$ already exists in $H_1$-list, then $\mathcal{B}$ responds with $H_1(id_i) = h_i \in \mathbb{G}_1$. Otherwise, $\mathcal{B}$ flips a coin $c_i \in \{0, 1\}$ so that $Pr[c_i = 0] = \delta$.
  2. $\mathcal{B}$ randomly chooses $r_i \in \mathbb{Z}_q$. If $c_i = 0$, $\mathcal{B}$ computes $h_i = r_i P \in \mathbb{G}_1$. If $c_i = 1$, $h_i = (\frac{r_i}{s})P_\alpha$.
  3. $\mathcal{B}$ adds $(id_i, h_i, r_i, c_i)$ to $H_1$-list and responds $H_1(id_i) = h_i$.

  To respond to $H_2$ queries, $\mathcal{B}$ maintains $H_2$-list (initially empty) with entries $(w_i, q_i)$. If query $w_i$ does not exist in $H_2$-list, then $\mathcal{B}$ randomly chooses $q_i \in \mathbb{Z}_q$ and adds $(w_i, q_i)$ to $H_2$-list. $\mathcal{B}$ responds $H_2(w_i) = q_i$. Similar to responding $H_2$, $\mathcal{B}$ maintains $H_3$-list in similar way using $(t_i, r_{ti})$ entries.
– *Phase*1: when $\mathcal{A}_1$ asks for trapdoor of $(id, w)$, $\mathcal{B}$ obtains $(id_i, h_i, r_i, c_i)$ from $H_1$-list and $(w_j, q_j)$ from $H_2$-list where $id_i = id$ and $w_j = w$. If $c_i = 0$, then $\mathcal{B}$ aborts and reports failure. Otherwise, $\mathcal{B}$ computes $T_{w2} = y\mu Q_{ID} = \frac{r_i}{s} y\mu P_\alpha$, where $y, \mu$ are picked in Setup phase. $\mathcal{B}$ gives $T_w = (\mathcal{I}, T_{w2}, T_{w3})$ to $\mathcal{A}_1$.
– *Challenge*: $\mathcal{A}_1$ sends $((id_0^*, w_0^*), (id_1^*, w_1^*))$ to $\mathcal{B}$. $\mathcal{B}$ obtains $(id_i, h_i, r_i, c_i)$ from $H_1$-list and $(w_j, q_j)$ from $H_2$-list where $id_i = id_0, id_1$ and $w_j = w_0, w_1$. $\mathcal{B}$ flips a coin $b \in_R \{0, 1\}$ and responds SIDEKS ciphertext $\mathcal{S}^* = (U^*, V^*, C^*)$ where $C^* = xH_2(w_b)P_{pub} = \gamma q_b P_\beta$ is defined.
– *Phase*2: Identical to *Phase*1, except that $(id, w) \neq (id_0^*, w_0^*), (id_1^*, w_1^*)$.
– *Guess*: $\mathcal{A}_1$ outputs $b' \in \{0, 1\}$. $\mathcal{B}$ picks $t$ in random oracle $H_3$ and returns $t^{\frac{s}{r_i c y q \mu}}$ as its guess for $\hat{e}(P, P)^{\alpha\beta\gamma}$. □

**Theorem 2** (IND-CKA Game 2) *Our scheme is semantically secure against chosen keyword attack in Game* 2 *under the random oracle model assuming BDH problem is intractable.*

*Proof* Suppose that $\mathcal{A}_2$ is a malicious user which has advantage $\epsilon$ in winning the game. We build an adversary $\mathcal{B}$ uses $\mathcal{A}_2$ and thus $\mathcal{B}$ has advantage $\epsilon'$ in solving the BDH problem. $\mathcal{B}$ is given $P, P_\alpha = \alpha P, P_\beta = \beta P, P_\gamma = \gamma P \in \mathbb{G}_1$. $\mathcal{B}$'s goal is to compute $\hat{e}(P, P)^{\alpha\beta\gamma} \in \mathbb{G}_2$.

- $Setup$: $\mathcal{A}_2$ is given its secret key $S_{ID} = P_\alpha$ and system parameters $\mathcal{SP} = (\hat{e}, \mathbb{G}_1, \mathbb{G}_2, q, P, P_{pub}, P_{cld}, H_1, H_2, H_3, H_4)$ where $P_{cld} = P_\beta = \beta P$. $H_1, H_2, H_3$, and $H_4$ are controlled by $\mathcal{B}$ and $\mathcal{B}$ maintains $H_1$-list, $H_2$-list, and $H_3$-list as described in previous theorem. $\mathcal{B}$ randomly picks $y, \mu \in_R \mathbb{Z}_q$ and gives them to $\mathcal{A}_2$.
- $Phase1$: when $\mathcal{A}_2$ asks for test result for trapdoor $T_w = (\mathcal{I}, T_{w2}, T_{w3})$ of $(id, w)$ and SIDEKS ciphertext $\mathcal{S} = (U, V, C)$, $\mathcal{B}$ returns its test result by testing whether $H_3(\hat{e}(T_{w3}, V)) = H_3(\hat{e}(C_{sum}, T_{w2}))$. $\mathcal{B}$ obtains $(id_i, h_i, r_i, c_i)$ from $H_1$-list and $(w_j, q_j)$ from $H_2$-list where $id_i = id$ and $w_j = w$. If $c_i = 0$, then $\mathcal{B}$ aborts and reports failure. Otherwise, $\mathcal{B}$ computes $T_{w3} = yH_2(w)S_{id} = yq_jr_1P_\alpha$ and returns its test result.
- $Challenge$: upon receiving $((id_0^*, w_0^*), (id_1^*, w_1^*))$ from $\mathcal{A}_2$, $\mathcal{B}$ obtains $(id_i, h_i, r_i, c_i)$ from $H_1$-list and $(w_j, q_j)$ from $H_2$-list where $id_i = id_0, id_1$ and $w_j = w_0, w_1$. $\mathcal{B}$ flips a coin $b \in_R \{0, 1\}$ and responds SIDEKS ciphertext $\mathcal{S}^* = (U^*, V^*, C^*)$ where $V^* = \mu\gamma P_\beta$ is defined.
- $Phase2$: identical to $Phase1$.
- $Guess$: $\mathcal{A}_2$ outputs $b' \in \{0, 1\}$. $\mathcal{B}$ picks $t$ in random oracle $H_3$ and returns $t^{1/(r_i\mu yq)}$ as its guess for $\hat{e}(P, P)^{\alpha\beta\gamma}$.

□

## 5.3 Evaluation environment

We evaluated the proposed encryption on a workstation equipped with Intel(R) Xeon(R) CPU W3530 (2.80 GHz). The implementation code is written by C language with pairing-based cryptography (PBC) library. The goal of implementation code is used to evaluate the proposed scheme by evaluating the most commonly called functions such as sSIDEKS, Trapdoor, and Test algorithms. We thus focus on these following algorithms: Setup, Extract, sSIDEKS, Trapdoor, and Test. Here, the algorithms are shown below.

- $Setup(k)$

```
pairing_from_file(pairing, "a.param");
element_random (P);
element_random (s);
element_random (c);
element_mul_zn (Ppub, P, s);
element_mul_zn (Pcld, P, c);
```

- $S_{ID} = Extract(s, ID)$

```
element_from_hash (Qid, ID,
                        strlen(ID));
element_mul_zn (Sid, Qid, s);
return Sid;
```

- $\mathcal{D}' = sSIDEKS(sk_{ID}, \mathcal{D}, \mathcal{U}, \mathcal{W})$

```
element_random (x);
for_each_user (u, U) {
    get_mu (mu, Sid, u->Qid);
    element_mul_zn (y, x, mu);
    element_mul_zn (Ui, Qid, y);
    element_mul_zn (Vi, Pcld, y);
    Ulist.add (Ui);
    Vlist.add (Vi);
}
for_each_keyword (w, W) {
    element_from_hash (y, w,
                        strlen(w));
    element_mul_zn (y, y, x);
    element_mul_zn (Ci, Ppub, y);
    Clist.add (Ci);
}
Dprime.addUlist (Ulist);
Dprime.addVlist (Vlist);
Dprime.addClist (Clist);
return Dprime;
```

- $T_w = sTrapdoor(sk_{ID}, Q_{ID_0}, w)$

```
element_random (y);
get_mu (mu, Qid0, Sid);
element_from_hash (x, w, strlen(w));
element_mul_zn (mu, mu, y);
element_mul_zn (Tw2, Qid, mu);
element_mul_zn (x, x, y);
element_mul_zn (Tw3, Sid, x);
return makeSimplifiedTrapdoor (Tw2,
                              Tw3);
```

– $result = sTest(c, T_w, \mathcal{D}')$

```
int i, j;
for_each_V (Vi, Dprime.Vlist) {
    element_pairing(leftHandSide,
                    Tw3, Vi);
    for_each_C (Cj, Dprime.Clist) {
        element_pairing(rightHandSide,
                        Cj, Tw2);
        element_pow_zn(rightHandSide,
                       rightHandSide,
                       c);
        if (element_cmp(leftHandSide,
                        rightHandSide)
                        == 0)
            return 1;
    }
}
return 0;
```

– $\mu = \hat{e}(e1, e2)$

```
char *byte;
int len;
element_pairing(mubh, e1, e2);
len = element_length_in_bytes(mubh);
byte = malloc(len);
element_to_bytes(byte, mubh);
element_from_hash(mu, byte, len);
```

In addition to that, we applied the process of the code optimization to reduce the program complexity. The pairing parameter we used is type A parameters “a.param”. Note that, we also omitted the initialization and the deletion of declared elements here, but they are still required in the code. To simplify our evaluation, the process of whole document encryption and decryption is omitted.

## 5.4 Evaluation result

We compiled the code with O2-optimization and then executed the implementation using single thread. It turns out that applying test algorithm upon a trapdoor and a SIDEKS ciphertext takes only 0.0033 s in average. The practical computation time 0.0033 s is very close to the estimated time 0.0034 s$(= 2 \times 0.0016 + 0.0002)$. Obviously, the performance is extremely great, not to mention applying the implementation

**Table 5** Time consumption in seconds

| | Extract | SIDEKS | Trapdoor | Test |
|---|---|---|---|---|
| Average | 0.0090 | 0.0083 | 0.0060 | 0.0033 |

to parallel processing and distributed computing such as cloud environment. However, the performance is bounded by the PBC library.

The evaluation of proposed algorithms is shown in Table 5. We run the algorithms thousands of time. The numbers presented here are average time consumption in second. Note that, there is a little difference between arithmetic time consumption and actual time consumption.

This code is well optimized. Therefore, the time consumption of algorithms is much less than arithmetic summation of time consumption of sequential PBC operations. Note that, bad coding style and bad sequence of functions may result in lack of efficiency. Take $Q = xyP$ as an example where $Q, P \in \mathbb{G}_1$ and $x, y \in Z_q^*$, first compute $a = xy$ then compute $Q = aP$ is always faster than compute $Q = xP$ then $Q = yQ$. The reason is that, $a = xy$ is a multiplication over integers, while $Q = xP$ is a scaler operation over $\mathbb{G}_1$. And scaler over $\mathbb{G}_1$ always takes more time than multiplication over integers. Another example is $\mathcal{G} = \hat{e}(P, P)^a$ will be always faster than $\mathcal{G} = \hat{e}(P, aP)$, where $\mathcal{G} \in \mathbb{G}_T$. The reason is that the power operation over $\mathbb{G}_T$ is always faster than scaler operation over $\mathbb{G}_1$.

## 6 Conclusion

A lot of research on searchable encryption has been done in recent years. None of them has addressed the need of searching in data owners' storage. For instance, friends or co-workers may want their documents kept up to date with the latest version from data owners, and they do not want their cloud storage quota being filled with unnecessary files. Traditional searchable encryptions simply re-encrypt and send the encrypted documents to all receivers, however, this may cause complex computations and messy duplications in limited quota.

In this paper, we proposed an efficient and shareable ID-based encryption with keyword search which has the following features: (1) data owners can share their document with their friends, and friends can search in data owner's storage while preserving the owner's privacy; therefore, the encrypted files will only occupy the owners' storage quota. Furthermore, friends are not able to obtain other encrypted documents that are not being shared with; the cloud server has no clue of whom the documents are shared with unless a test is succeeded. Since each receiver is searching in owner's storage, the receiver has to specify the owner. (2) Provides better efficiency compared with other related works. The performance analysis is given in Sect. 5. (3) Different keyword set for different user set can be done using $AddReq$ algorithm iteratively. (4) Data owner can dynamically enable and disable the searching capability of their friends using $AddReq$ and $DelReq$ algorithms. In addition to that, we have proved that our scheme is secure against chosen keyword attack (IND-CKA) under bilinear Diffie–Hellman assumption.

**Acknowledgements** This work was supported in part by the Ministry of Science and Technology, Taiwan, under Contract MOST 103-2221-E-007-073-MY3, MOST 104-2811-E-007-005, MOST 103-2218-E-031-001, and MOST 103-2221-E-004-009. The work of Prof. Li Xu was supported in part by the National Natural Science Foundation of China (U1405255).

## References

1. Wang C, Ren K, Lou WJ, Li J (2010) Toward publicly auditable secure cloud data storage services. IEEE Netw 24:19–24
2. Cao Q, Jujita S (2014) Cost-effective replication schemes for query load balancing in DHT-based peer-to-peer file searches. J Inf Process Syst 10:628–645
3. Subashini S, Kavitha V (2011) A survey on seucrity issues in service delivery models of cloud computing. J Netw Comput Appl 34:1–11
4. Lee SH, Lee IY (2013) A secure index management scheme for providing data sharing in cloud storage. J Inf Process Syst 9:287–300
5. Naruse T, Mohri M, Shiraish Y (2015) Provably secure attribute-based encryption with attribute revocation and grant function using proxy re-encryption and attribute key for updaing. Hum-Cent Comput Inf Sci 5:8:1–8:13
6. Bringer J, Chabane H (2012) Embedding edit distance to enable private keyword search. Hum-cent Comput Inf Sci 2:2:1–2:12
7. Bellare M, Boldyreva A, O'Neill A (2007) Deterministic and efficiently searchable encryption. In: Proceedings of advances in cryptology—CRYPT, vol 2007, pp 535–552
8. Reza C, Juan G, Seny K, Rafail O (2006) Searchable symmetric encryption: improved definitions and efficient constructions, In: Proceedings of 13th ACM conference on computer and communications security (CCS), pp 79–88
9. Shamir A (1985) Identity-based cryptosystems and signature schemes. In: Proceedings of advances in cryptology—CRYPTO, vol 85, pp 47–53
10. Boneh D, Franklin M (2001) Identity-based encryption from the weil pairing. In: Proceedings of advances in cryptology—CRYPTO, vol 2001, pp 213–229
11. Boneh D, Boyen X (2004) Efficient selective-ID secure identity-based encryption without random oracles. In: Proceedings of advances in cryptology—EUROCRYPT, vol 2004, pp 223–238
12. Boneh D, Crescenzo GD, Ostrovsky R, Persiano G (2004) Public key encryption with keyword search. In: Proceedings of advances in cryptology—EUROCRYPT, vol 2004, pp 506–522
13. Baek J, Safavi-Naini R, Susilo W (2008) Public key encryption with keyword search revisited. In: Proceedings of computational science and its applications ICCSA, vol 2008, pp 1249–1259
14. Tian X, Wang Y (2008) ID-Based encryption with keyword search scheme from bilinear pairings. In: Proceedings of 4th international conference on wireless communications, networking and mobile computing (WiCOM), pp 1–4
15. Rhee HS, Park JH, Susilo W, Lee JH (2009) Improved searchable public key encryption with designated tester. In: proceedings of 4th international symposium on information, computer, and communications security (ASIA CCS), pp 376–379
16. Liu Q, Wang G, Wu J (2012) Secure and privacy preserving keyword searching for cloud storage services. J Netw Comput Appl 35:927–933
17. Lai J, Zhou X, Deng R, Li Y (2013) Expressive search on encrypted data. In: Proceedings of 8th ACM SIGSAC symposium on information, computer and communications security, pp 243–252
18. Wang P, Wang H, Pieprzyk J (2008) Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups. Cryptol Netw Secur 5339:178–195
19. Kerschbaum F (2011) Secure conjunctive keyword searches for unstructured text. In: Proceedings of 5th international conference on network and system security (NSS), pp 285–289
20. Boneh D, Lynn B, Shacham H (2004) Short signatures from the weil pairing. J Cryptol 14:297–319
21. Ding M, Gao F, Jin Z, Zhang H (2012) An efficient public key encryption with conjunctive keyword search scheme based on pairings. In: Proceedings of 3rd IEEE international conference on network infrastructure and digital content (IC-NIDC), pp 526–530
22. Gupta S, Satapathy SR, Mehta P, Tripathy A (2013) A secure and searchable data stroage in cloud computing. In: Proceedings of 3rd international conference on advance computing conference (IACC), pp 106–109

23. Ibraimi J, Nikova S, Hartel P, Jonker W (2011) Public-key encryption with delegated search. Appl Cryptograph Netw Secur 6715:532–549
24. Li M, Yu S, Cao N, Lou W (2011) Authorized private keyword search over encrypted data in cloud computing. In: Proceedings of 31th international conference on distributed computing systems (ICDCS), pp 383–392
25. Yang Y, Lu H, Weng J (2011) Multi-user private keyword search for cloud computing. In: Proceedings of 3th international conference on cloud computing technology and science (CloudCom), pp 264–271
26. Tang Q, Chen X (2013) Towards asymmetric searchable encryption with message recovery and flexible search authorization. In: Proceedings of 8th international symposium on information, computer, and communications security (ASIA CCS), pp 253–264
27. Li J, Li J, Liu Z, Jia C (2013) Enabling efficient and secure data sharing in cloud ccomputing. Concurr Comput Pract Exp 26:1052–1066
28. Lin MP, Hong WCh, Chen CH, Cheng CM (2013) Design and implementation of multi-user secure indices for encrypted cloud storage. In: Proceedings of 11th annual international conference on privacy, security and trust (PST), pp 177–184
29. Liu Q, Wang G, Wu J (2009) An efficient privacy preserving keyword search scheme in cloud computing. In: Proceedings of international conference on computational science and engineering (CSE), pp 725–720
30. Yoon HJ, Cheon JH, Kim YD (2005) Batch verifications with ID-based signatures. In: Proceedings of information security and cryptology ICISC, vol 2005, pp 233–248