IEEE TRANSACTIONS ON COMPUTERS VOL: PP NO: 99 YEAR 2015

Key-Aggregate Searchable Encryption (KASE) for Group Data Sharing via Cloud Storage

Baojiang Cui, Zheli Liu* and Lingyu Wang

Abstract—The capability of selectively sharing encrypted data with different users via public cloud storage may greatly ease security concerns over inadvertent data leaks in the cloud. A key challenge to designing such encryption schemes lies in the efficient management of encryption keys. The desired flexibility of sharing any group of selected documents with any group of users demands different encryption keys to be used for different documents. However, this also implies the necessity of securely distributing to users a large number of keys for both encryption and search, and those users will have to securely store the received keys, and submit an equally large number of keyword trapdoors to the cloud in order to perform search over the shared data. The implied need for secure communication, storage, and complexity clearly renders the approach impractical. In this paper, we address this practical problem, which is largely neglected in the literature, by proposing the novel concept of *key-aggregate searchable encryption (KASE)* and instantiating the concept through a concrete KASE scheme, in which a data owner only needs to distribute a single key to a user for sharing a large number of documents, and the user only needs to submit a single trapdoor to the cloud for querying the shared documents. The security analysis and performance evaluation both confirm that our proposed schemes are provably secure and practically efficient.

Index Terms—Searchable encryption, data sharing, cloud storage, data privacy

1 Introduction

Cloud storage has emerged as a promising solution for providing ubiquitous, convenient, and on-demand accesses to large amounts of data shared over the Internet. Today, millions of users are sharing personal data, such as photos and videos, with their friends through social network applications based on cloud storage on a daily basis. Business users are also being attracted by cloud storage due to its numerous benefits, including lower cost, greater agility, and better resource utilization.

However, while enjoying the convenience of sharing data via cloud storage, users are also increasingly concerned about inadvertent data leaks in the cloud. Such data leaks, caused by a malicious adversary or a misbehaving cloud operator, can usually lead to serious breaches of personal privacy or business secrets (e.g., the recent high profile incident of celebrity photos being leaked in iCloud). To address users' concerns over potential data leaks in cloud storage, a common approach is for the data owner to encrypt all the data before uploading them to the cloud, such that later the encrypted data may be retrieved and decrypted by those who have the decryption keys. Such a cloud storage is often called the cryptographic cloud storage [6]. However, the encryption of data makes it challenging for users to search and then selectively retrieve only the data containing given keywords. A common solution is to employ a searchable encryption (SE) scheme in which the data owner is required to encrypt potential keywords and upload them to the cloud together with encrypted data, such that, for retrieving data matching a keyword, the user will send the corresponding keyword *trapdoor* to the cloud for performing search over the encrypted data.

Although combining a searchable encryption scheme with cryptographic cloud storage can achieve the basic security requirements of a cloud storage, implementing such a system for large scale applications involving millions of users and billions of files may still be hindered by practical issues involving the efficient management of encryption keys, which, to the best of our knowledge, are largely ignored in the literature. First of all, the need for selectively sharing encrypted data with different users (e.g., sharing a photo with certain friends in a social network application, or sharing a business document with certain colleagues on a cloud drive) usually demands different encryption keys to be used for different files. However, this implies the number of keys that need to be distributed to users, both for them to search over the encrypted files and to decrypt the files, will be proportional to the number of such files. Such a large number of keys

- *B. Cui is with School of Computer Science, Beijing University of Posts and Telecommunications, National Engineering Laboratory for Mobile Network Security, China.*
  Email:cuibj@bupt.edu.cn.
  Z. Liu is with the College of Computer and Control Engineering, Nankai University, China, 300071.
  Email: liuzheli@nankai.edu.cn.
  L. Wang is with Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada.
  Email:wang@ciise.concordia.ca.
- *Z. Liu and B. Cui contribute to this work equally.*
  Corresponding authors: liuzheli@nankai.edu.cn

must not only be distributed to users via secure channels, but also be securely stored and managed by the users in their devices. In addition, a large number of trapdoors must be generated by users and submitted to the cloud in order to perform a keyword search over many files. The implied need for secure communication, storage, and computational complexity may render such a system inefficient and impractical.

In this paper, we address this challenge by proposing the novel concept of *key-aggregate searchable encryption (KASE)*, and instantiating the concept through a concrete KASE scheme. The proposed KASE scheme applies to any cloud storage that supports the *searchable group data sharing* functionality, which means any user may selectively share a *group* of selected files with a *group* of selected users, while allowing the latter to perform keyword search over the former. To support searchable group data sharing the main requirements for efficient key management are twofold. *First, a data owner only needs to distribute a single aggregate key (instead of a group of keys) to a user for sharing any number of files. Second, the user only needs to submit a single aggregate trapdoor (instead of a group of trapdoors) to the cloud for performing keyword search over any number of shared files*. To the best of our knowledge, the KASE scheme proposed in this paper is the first known scheme that can satisfy both requirements (the key-aggregate cryptosystem [4], which has inspired our work, can satisfy the first requirement but not the second).

Contributions. More specifically, our main contributions are as follows.

1) We first define a general framework of key-aggregate searchable encryption (KASE) composed of seven polynomial algorithms for security parameter setup, key generation, encryption, key extraction, trapdoor generation, trapdoor adjustment, and trapdoor testing. We then describe both functional and security requirements for designing a valid KASE scheme.
2) We then instantiate the KASE framework by designing a concrete KASE scheme. After providing detailed constructions for the seven algorithms, we analyze the efficiency of the scheme, and establish its security through detailed analysis.
3) We discuss various practical issues in building an actual group data sharing system based on the proposed KASE scheme, and evaluate its performance. The evaluation confirms our system can meet the performance requirements of practical applications.

The rest of the paper is organized as follows. First, we review some background knowlege in Section 2. We then define the general KASE framework in Section 3. We describe related work in Section 4. We design a concrete KASE scheme and analyze its efficiency and security in Section 5. We implement and evaluate a KASE-based group data sharing system in Section 6. Finally, we conclude the paper in Section 7.

2 Preliminaries

In this section, we review some basic assumptions and cryptology concepts which will be needed later in this paper. In the rest of our discussions, let $\mathcal{G}$ and $\mathcal{G}_1$ be two cyclic groups of prime order p, and g be a generator of $\mathcal{G}$. Moreover, let doc be the document to be encrypted, k the searchable encryption key, and Tr the trapdoor for keyword search.

2.1 Complexity Assumption

2.1.1 Bilinear Map

A bilinear map is a map $e : \mathcal{G} \times \mathcal{G} \to \mathcal{G}_1$ with the following properties:

1. Bilinearity: for all $u, v \in \mathcal{G}$ and $a, b \in \mathcal{Z}_p^*$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $e(g, g) \neq 1$.
3. Computability: there is an efficient algorithm to compute $e(u, v)$ for any $u, v \in \mathcal{G}$.

2.1.2 Bilinear Diffie-Hellman Exponent Assumption

The bilinear Diffie-Hellman exponent (BDHE) assumption has been widely used to prove the security of some broadcast encryption (BE) schemes (e.g., [27]).

The l-BDHE problem in $\mathcal{G}$ is stated as follows. Given a vector of $2l + 1$ elements $(h, g, g^{\alpha}, g^{(\alpha^2)}, \cdot\cdot\cdot, g^{(\alpha^l)}, g^{(\alpha^{l+2})}, \cdot\cdot\cdot, g^{(\alpha^{2l})}) \in (\mathcal{G}^*)^{2l+1}$ as input, output $e(g, h)^{(\alpha^{l+1})} \in \mathcal{G}_1$.

For convenience, we use g_i to denote $g_i = g^{(\alpha^i)} \in \mathcal{G}$. Note that the input vector is missing the term g_{l+1} (i.e., $g^{(\alpha^{l+1})}$) such that the bilinear map seems to be of little help in computing the required $e(g, h)^{(\alpha^{l+1})}$.

An algorithm $\mathcal{A}$ has advantage ε in solving l-BDHE in $\mathcal{G}$ if $\mathbf{Pr}[\mathcal{A}(h, g, g_1, \cdots, g_l, g_{l+2}, \cdots, g_{2l}) = e(g_{l+1}, h)] \geq \varepsilon$, where the probability is over the random choice of generators g, h in G, the random choice of α in $\mathcal{Z}_p$, and the random bits used by $\mathcal{A}$.

Definition 1. *The $(l, \varepsilon)-$BDHE assumption holds in $\mathcal{G}$ if no algorithm has advantage more than ε in solving the $l-$BDHE problem in $\mathcal{G}$.*

2.2 Broadcast Encryption

In a broadcast encryption (BE) scheme, a broadcaster encrypts a message for some subset S of users who are listening on a broadcast channel. Any user in S can use his/her private key to decrypt the broadcast. A BE scheme can be described as a tuple of three polynomial-time algorithms *BE = (Setup, Encrypt, Decrypt)* as follows:

- **Setup**(1^λ, n): this algorithm is run by the system to set up the scheme. It takes as input a security

parameter 1^λ and the number of receivers n, outputs n private keys $d_1, \cdots, d_n$ and a public key pk.
- **Encrypt**(pk, S): this algorithm is run by the broadcaster to encrypt a message for a subset of users. It takes as input a public key pk and a subset of users $S \subseteq \{1, \cdots, n\}$, outputs a pair ($Hdr$,$K$), where Hdr is called the header and K is a message encryption key which is encapsulated in Hdr. We will often refer to Hdr as the broadcast ciphertext. For a concrete message, it will be encrypted by K and broadcasted to the users in S.
- **Decrypt**(pk, S, i, d_i, Hdr): this algorithm is run by the user to decrypt the received messages. It takes as input a public key pk, a subset of users $S \subseteq \{1, \cdots, n\}$, a user id $i \in \{1, \cdots, n\}$, the private key d_i for user i and a header Hdr, outputs the message encryption key K or the failure symbol $\perp$. The K will be used to decrypt the received messages.

To ensure the system to be correct, it is required that, for all $S \subseteq \{1, \cdots, n\}$ and all $i \in S$, if (pk, $(d_1, \cdots, d_n)$$\xleftarrow{R}$**Setup**($1^\lambda$, n) and (Hdr, K)$\xleftarrow{R}$**Encrypt**(pk, S)), then **Decrypt**(pk, S, i, d_i, Hdr)=K.

2.3 Searchable Encryption

Generally speaking, searchable encryption schemes fall into two categories, i.e., searchable symmetric encryption (SSE) and public key encryption with keyword search (PEKS). Both SSE and PEKS can be described as the tuple *SE= (Setup, Encrypt, Trapdoor, Test)*:

- **Setup**(1^λ): this algorithm is run by the owner to set up the scheme. It takes as input a security parameter 1^λ, and outputs the necessary keys.
- **Encrypt**(k, m): this algorithm is run by the owner to encrypt the data and generate its keyword ciphertexts. It takes as input the data m, owner's necessary keys including searchable encryption key k and data encryption key, outputs data ciphertext and keyword ciphertexts C_m.
- **Trpdr**(k, w): this algorithm is run by a user to generate a trapdoor Tr for a keyword w using key k.
- **Test**(Tr, C_m): this algorithm is run by the cloud server to perform a keyword search over encrypted data. It takes as input trapdoor Tr and the keyword ciphertexts C_m, outputs whether C_m contains the specified keyword.

For correctness, it is required that, for a message m containing keyword w and a searchable encryption key k, if (C_m←**Encrypt**(k, m) and Tr←**Trpdr**(k, w)), then **Test**(Tr, C_m)=*true*.

3 The Key-Aggregate Searchable Encryption (KASE) Framework

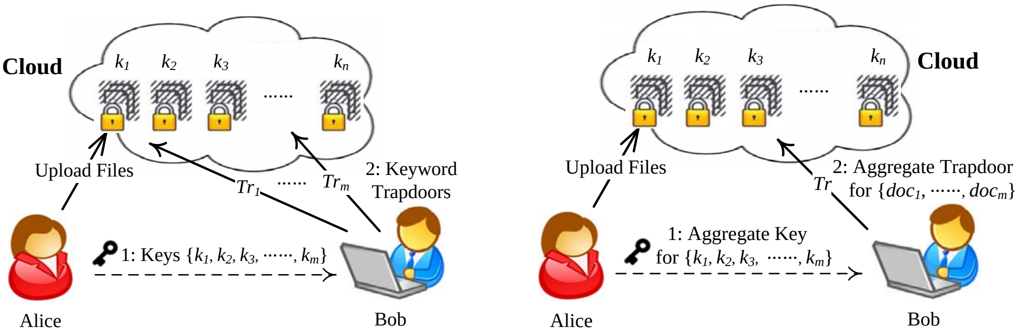
In this section, we first describe the general problem, and then define a generic framework for key-aggregate searchable encryption (KASE) and provide requirements for designing a valid KASE scheme.

3.1 Problem Statement

Consider a scenario where two employees of a company would like to share some confidential business data using a public cloud storage service (e.g., dropbox or syncplicity). For instance, Alice wants to upload a large collection of financial documents to the cloud storage, which are meant for the directors of different departments to review. Suppose those documents contain highly sensitive information that should only be accessed by authorised users, and Bob is one of the directors and is thus authorized to view documents related to his department. Due to concerns about potential data leakage in the cloud, Alice encrypts these documents with different keys, and generates keyword ciphertexts based on department names, before uploading to the cloud storage. Alice then uploads and shares those documents with the directors using the sharing functionality of the cloud storage. In order for Bob to view the documents related to his department, Alice must delegate to Bob the rights both for keyword search over those documents, and for decryption of documents related to Bob's department.

With a traditional approach, Alice must securely send all the searchable encryption keys to Bob. After receiving these keys, Bob must store them securely, and then he must generate all the keyword trapdoors using these keys in order to perform a keyword search. As shown in Fig.1(a), Alice is assumed to have a private document set $\{doc_i\}_{i=1}^{n}$, and for each document doc_i, a searchable encryption key k_i is used. Without loss of generality, we suppose Alice wants to share m documents $\{doc_i\}_{i=1}^{m}$ with Bob. In this case, Alice must send all the searchable encryption keys $\{k_i\}_{i=1}^{m}$ to Bob. Then, when Bob wants to retrieve documents containing a keyword w, he must generate keyword trapdoor Tr_i for each document doc_i with key k_i and submit all the trapdoors $\{Tr_i\}_{i=1}^{m}$ to the cloud server. When m is sufficiently large, the key distribution and storage as well as the trapdoor generation may become too expensive for Bob's client-side device, which basically defies the purpose of using cloud storage.

In this paper, we propose the novel approach of *key-aggregate searchable encryption* (KASE) as a better solution, as depicted in Fig.1(b). , in KASE, Alice only needs to distribute a single aggregate key, instead of $\{k_i\}_{i=1}^{m}$ for sharing m documents with Bob, and Bob only needs to submit a single aggregate trapdoor, instead of $\{Tr_i\}_{i=1}^{m}$, to the cloud server. The cloud


Cloud
k_1 k_2 k_3 ...... k_n
Upload Files
Tr_1 ...... Tr_m
2: Keyword Trapdoors
1: Keys $\{k_1, k_2, k_3, \cdots\cdots, k_m\}$
Alice
Bob
k_1 k_2 k_3 ...... k_n
Cloud
Upload Files
Tr
2: Aggregate Trapdoor for $\{doc_1, \cdots\cdots, doc_m\}$
1: Aggregate Key for $\{k_1, k_2, k_3, \cdots\cdots, k_m\}$
Alice
Bob

(a) Traditional approach

(b) Key-Aggregate Searchable Encryption

Fig. 1. keyword search in group data sharing system.

server can use this aggregate trapdoor and some public information to perform keyword search and return the result to Bob. Therefore, in KASE, the delegation of keyword search right can be achieved by sharing the single aggregate key. We note that the delegation of decryption rights can be achieved using the *key-aggregate encryption* approach recently proposed in [4], but it remains an open problem to delegate the keyword search rights together with the decryption rights, which is the subject topic of this paper. To summarize, the problem of constructing a KASE scheme can be stated as:

"To design a key-aggregate searchable encryption scheme under which any subset of the keyword ciphertexts (produced by the SE.Encrypt algorithm to be introduced in Section 5) from any set of documents is searchable (performed by the SE.Test algorithm) with a constant-size trapdoor (produced by SE.Trpdr algorithm) generated by a constant-size aggregate key."

3.2 The KASE Framework

The KASE framework is composed of seven algorithms. Specifically, to set up the scheme, the cloud server would generate public parameters of the system through the **Setup** algorithm, and these public parameters can be reused by different data owners to share their files. For each data owner, he/she should produce a public/master-secret key pair through the **Keygen** algorithm. Keywords of each document can be encrypted via the **Encrypt** algorithm with the unique searchable encryption key. Then, the data owner can use the master-secret key to generate an aggregate searchable encryption key for a group of selected documents via the **Extract** algorithm. The aggregate key can be distributed securely (e.g., via secure e-mails or secure devices) to authorized users who need to access those documents. After that, as shown in Fig.2, an authorized user can produce a keyword trapdoor via the **Trapdoor** algorithm using this aggregate key, and submit the trapdoor to the cloud. After receiving the trapdoor, to perform the keyword search over the specified set of documents, the cloud server will run the **Adjust** algorithm to generate the right trapdoor for each document, and then run the **Test** algorithm to test whether the document contains the keyword. This framework is summarized in the following.

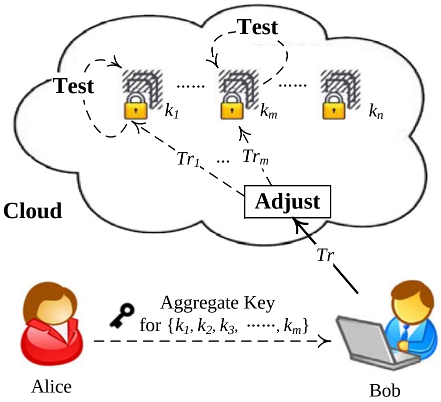


Fig. 2. Framework of key-aggregate searchable encryption.

- **Setup**(1^λ, n): this algorithm is run by the *cloud service provider* to set up the scheme. On input of a security parameter 1^λ and the maximum possible number n of documents which belongs to a data owner, it outputs the public system parameter $params$.
- **Keygen**: this algorithm is run by the data owner to generate a random key pair (pk,msk).
- **Encrypt**(pk, i): this algorithm is run by the data owner to encrypt the i-th document and generate its keywords' ciphertexts. For each document, this algorithm will create a delta Δ_i for its searchable encryption key k_i. On input of the owner's public key pk and the file index i, this algorithm outputs data ciphertext and keyword ciphertexts C_i.
- **Extract**(msk, S): this algorithm is run by the data owner to generate an aggregate searchable encryption key for delegating the keyword search right for a certain set of documents to other users. It takes as input the owner's master-secret key

msk and a set S which contains the indices of documents, then outputs the aggregate key k_{agg}.

- **Trapdoor**(k_{agg}, w): this algorithm is run by the user who has the aggregate key to perform a search. It takes as input the aggregate searchable encryption key k_{agg} and a keyword w, then outputs only one trapdoor Tr.
- **Adjust**($params$, i, S, Tr): this algorithm is run by cloud server to adjust the aggregate trapdoor to generate the right trapdoor for each different document. It takes as input the system public parameters $params$, the set S of documents' indices, the index i of target document and the aggregate trapdoor Tr, then outputs each trapdoor Tr_i for the i-th target document in S.
- **Test**(Tr_i, i): this algorithm is run by the cloud server to perform keyword search over an encrypted document. It takes as input the trapdoor Tr_i and the document index i, then outputs *true* or *false* to denote whether the document doc_i contains the keyword w.

3.3 Requirements for Designing KASE Schemes

The KASE framework introduced in the previous section provides general guidance to designing a KASE scheme. However, a valid KASE scheme must also satisfy several functional and security requirements, as stated in the following.

A KASE scheme should satisfy three functional requirements as follows.

- **Compactness**. This requirement demands a KASE scheme to ensure the size of the aggregate key to be independent of the number of files to be shared. Formally, for a set of keys $\{k_i\}_{i \in S}$, it requires that $k_{agg} \leftarrow$**Extract**(msk, S). *How to aggregate the set of keys into a single key without invalidating later steps is a key challenge in designing KASE schemes.*
- **Searchability**. This requirement is central to all KASE schemes since it enables users to generate desired trapdoors for any given keyword for searching encrypted documents. In another word, reducing the number of keys should preserve the search capability. Formally, for each document containing the keyword w with index $i \in S$, the searchability requires that if (Tr=**Trapdoor**(k_{agg}, w) and $Tr_i \leftarrow$**Adjust**($params$, i, S, Tr)), then **Test**(Tr_i, i)=*true*.
- **Delegation**. The main goal of KASE is to delegate the keyword search right to a user through an aggregate key. *To ensure any user with the delegated key can perform keyword search, this requirement requires that the inputs of the adjustment algorithm must not be public, i.e., these inputs should not rely on any user's private information. This is the second key challenge in designing KASE schemes.*

In addition, any KASE scheme should also satisfy two security requirements as follows.

- **Controlled searching**. Meaning that the attackers cannot search for an arbitrary word without the data owner's authorization. That is, the attacker cannot perform keyword search over the documents which are not relevant to the known aggregate key, and he/she cannot generate new aggregate searchable encryption keys for other set of documents from the known keys.
- **Query privacy**. Meaning that the attackers cannot determine the keyword used in a query, apart from the information that can be acquired via observation and the information derived from it. That is, the user may ask an untrusted cloud server to search for a sensitive word without revealing the word to the server.

4 Related Work

Before we introduce our KASE scheme, this section first reviews several categories of existing solutions and explain their relationships to our work.

4.1 Multi-user Searchable Encryption

There is a rich literature on searchable encryption, including SSE schemes [5]–[8] and PEKS schemes [9]–[15]. In contrast to those existing work, in the context of cloud storage, keyword search under the multi-tenancy setting is a more common scenario. In such a scenario, the data owner would like to share a document with a group of authorized users, and each user who has the access right can provide a trapdoor to perform the keyword search over the shared document, namely, the "multi-user searchable encryption" (MUSE) scenario.

Some recent work [6], [13]–[15], [19] focus to such a MUSE scenario, although they all adopt single-key combined with access control to achieve the goal. In [6], [19], MUSE schemes are constructed by sharing the document's searchable encryption key with all users who can access it, and broadcast encryption is used to achieve coarse-grained access control. In [13]–[18], attribute based encryption (ABE) is applied to achieve fine-grained access control aware keyword search. As a result, in MUSE, the main problem is how to control which users can access which documents, whereas how to reduce the number of shared keys and trapdoors is not considered. Key aggregate searchable encryption can provide the solution for the latter, and it can make MUSE more efficient and practical.

4.2 Multi-Key Searchable Encryption

In the case of a multi-user application, considering that the number of trapdoors is proportional to the number of documents to search over (if the user

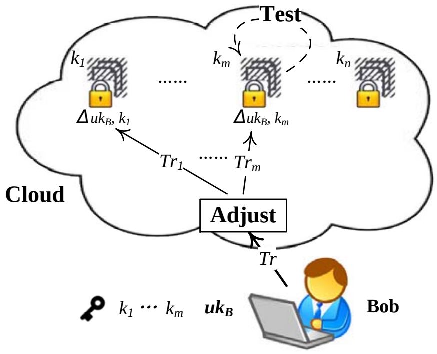


Fig. 3. Multi-Key Searchable Encryption.

provides to the server a keyword trapdoor under each key with which a matching document might be encrypted), Popa [28] firstly introduces the concept of multi-key searchable encryption (MKSE) and puts forward the first feasible scheme in 2013. MKSE allows a user to provide a single keyword trapdoor to the server, but still allows the server to search for that trapdoor's keyword in documents encrypted with different keys. This might sound very similar to the goal of KASE, but these are in fact two completely different concepts. The goal of KASE is to delegate the keyword search right to any user by distributing the aggregate key to him/her in a group data sharing system, whereas the goal of MKSE is to ensure the cloud server can perform keyword search with one trapdoor over different documents owing to a user.

More specifically, denote by uk_i the key of user i. Suppose a user, say Bob (with key uk_B), has m encrypted documents on the cloud server, and each is encrypted under a key k_j for $j = \{1, \cdots, m\}$. To allow the cloud server to adjust the trapdoor for each document with index j, Bob stores on the cloud server a public information called *delta* (denoted as Δ_{uk_B,k_j}) which is relevant to both uk_B and k_j. As shown in Fig. 3, when Bob wants to search for a word w over all the documents, he will use uk_B to compute a trapdoor for the word w and submit it to the cloud server. The cloud server can use Δ_{uk_B,k_j} to convert a keyword trapdoor under key uk_B to a keyword trapdoor under k_j; this process is called *adjust*. In such a way, the cloud server can obtain trapdoors for word w under $k_1, \cdots, k_m$ while only receiving one trapdoor from Bob, and then perform a traditional single-key search with the new trapdoors.

This approach of MKSE inspires us to focus on the problem of keyword search over a group of shared documents from the same user in the multi-user applications, and the *adjust* process in MKSE also provides a general approach to perform keyword search over a group of documents with only one trapdoor. However, the *adjust* process of MKSE needs a *delta* generated from both user's key and SE key of the document, so it does not directly apply to the design of a concrete KASE scheme.

4.3 Key-aggregate Encryption for Data Sharing

Data sharing systems based on cloud storage have attracted much attention recently [1]–[4]. In particular, Chu et al. [4] consider how to reduce the number of distributed data encryption keys. To share several documents with different encryption keys with the same user, the data owner will need to distribute all such keys to him/her in a traditional approach which is usually impractical. Aiming at this challenge, a key-aggregate Encryption (KAE) scheme for data sharing is proposed to generate an aggregate key for the user to decrypt all the documents.

To allow a set of documents encrypted by different keys to be decrypted with a single aggregate key, user could encrypt a message not only under a public-key, but also under the identifier of each document. The construction is inspired by the broadcast encryption scheme [27]. In this construction, the data owner can be regarded as the broadcaster, who has public key pk and master-secret key msk; each document with identifier i can be regarded as a receiver listening to the broadcast channel, and a public information used in decryption is designed to be relevant to both the owner's msk and the encryption key; the message encryption process is similar to data encryption using symmetric encryption in BE, but the key aggregation and data decryption can be simply regarded as the further mathematical transformation of BE.**Encrypt** algorithm and BE.**Decrypt** algorithm respectively.

The scheme [4] allows efficiently delegating the decryption rights to other users, and is the main inspiration of our study, but it does not support any search over the encrypted data. In the cloud environment, to achieve the goal of privacy-preserving data sharing, keyword search is a necessary requirement. Fortunately, the KAE provides insights to the design of a KASE scheme, although our scheme will require a more complex mathematical transformation to support keyword ciphertext encryption, trapdoor generation and keyword matching.

5 The Proposed Scheme

5.1 Overview

The design of our KASE scheme draws its insights from both the multi-key searchable encryption scheme [28] and the key-aggregate data sharing scheme [4]. Specifically, in order to create an aggregate searchable encryption key instead of many independent keys, we adapt the idea presented in [4]. Each searchable encryption key is associated with a particular index of document, and the aggregate key is created by embedding the owner's master-secret key into the product of public keys associated with the documents. In order to implement keyword search over different documents using the aggregate trapdoor, we employ a similar process as in [28]. The cloud server can use this process to produce an adjusted trapdoor for every document.

5.2 Description of the Scheme

Based on the framework described in section 3.2, we propose a concrete KASE scheme as follows.

1) **Setup(1^λ, n)**: the *cloud server* will use this algorithm to initialize system parameters as follows:
   - Generate a bilinear map group system $\mathcal{B}$=(p, $\mathcal{G}$, $\mathcal{G}_1$, $e(\cdot,\cdot)$), where p is the order of $\mathcal{G}$ and $2^\lambda \leq p \leq 2^{\lambda+1}$.
   - Set n as the maximum possible number of documents which belongs to a data owner.
   - Pick a random generator $g \in \mathcal{G}$ and a random $\alpha \in Z_p$, and computes $g_i = g^{(\alpha^i)} \in \mathcal{G}$ for $i = \{1, 2, \cdots, n, n+2, \cdots, 2n\}$.
   - Select a one-way hash function H: $\{0,1\}^* \rightarrow \mathcal{G}$.

   Finally, *cloud server* publishes the system parameters $params$ = ($\mathcal{B}$, $PubK$, H), where $PubK = (g, g_1, \cdots, g_n, g_{n+2}, ..., g_{2n}) \in \mathcal{G}^{2n+1}$.
2) **Keygen**: data owner uses this algorithm to generate his/her key pair. It picks a random $\gamma \in \mathbb{Z}_p$, and outputs:
$$pk = v = g^\gamma, msk = \gamma.$$
3) **Encrypt**(pk, i): data owner uses this algorithm to encrypt data and generate its keyword ciphertexts when uploading the i-th document. To generate the keyword ciphertexts, this algorithm takes as input the file index $i \in \{1, ..., n\}$, and:
   - randomly picks a $t \in \mathbb{Z}_p$ as the searchable encryption key k_i of this document.
   - generates a delta Δ_i for k_i by computing:
$$c_1 = g^t, c_2 = (v \cdot g_i)^t$$
   - for a keyword w, outputs its ciphertext c_w as:
$$c_w = e(g, H(w))^t / e(g_1, g_n)^t.$$

   Note that c_1, c_2 are public and can be stored in the cloud server.
4) **Extract**(msk, S): data owner uses this algorithm to generate an aggregate searchable encryption key. For any subset $S \subseteq \{1, \cdots, n\}$ which contains the indices of documents, this algorithm takes as input the owner's master-secret key msk and outputs the aggregate key k_{agg} by computing:
$$k_{agg} = \Pi_{j \in s} g_{n+1-j}^\gamma.$$
   To delegate the keyword search right to a user, data owner will send k_{agg} and the set S to the user.
5) **Trapdoor**(k_{agg}, w): the user uses this algorithm to generate the trapdoor to perform keyword search. For all documents which are relevant to the aggregate key k_{agg}, this algorithm generates the only one trapdoor Tr for the keyword w by computing:
$$Tr = k_{agg} \cdot H(w)$$
   Then, the user sends (Tr, S) to the cloud server.
6) **Adjust**($params$, i, S, Tr): the cloud server uses this algorithm to produce the right trapdoor. For each document in the set S, this algorithm takes as input the system public parameters $params$, the document index $i \in S$ and the aggregate trapdoor Tr, outputs the right trapdoor Tr_i by computing:
$$Tr_i = Tr \cdot \Pi_{j \in s, j \neq i} g_{n+1-j+i}$$
   Then, the cloud server will use **Test** algorithm to finish the keyword search.
7) **Test**(Tr_i, i): the cloud server uses this algorithm to perform keyword search over the i-th document. For the i-th document, this algorithm takes as input the adjusted trapdoor Tr_i, the Δ_i=(c_1, c_2) relevant to its searchable encryption k_i and the subset S, outputs *true* or *false* by judging:
$$c_w \overset{?}{==} e(Tr_i, c_1)/e(pub, c_2)$$
   where $pub = \Pi_{j \in s} g_{n+1-j}$. Note that for efficiency consideration, the pub for the set S can be computed only once.

Remark. If there is only one element in the subset S, the above scheme will be a concrete public key encryption with keyword search scheme, in which the **Adjust** algorithm will not work.

5.3 Efficiency

In terms of efficiency, our scheme clearly achieves constant-size keyword ciphertext, trapdoor and aggregate keys. In addition, we should point out that:

1) the set S, which includes the indices of shared documents, has a linear size in the number of documents associated with the aggregate key. However, this does not affect the usefulness of the data sharing system, because the content of S can be safely stored in the cloud server (more details will be provided in section 5.5), such that there is no need to submit them to the cloud server when submitting the trapdoor.
2) the public system parameters $PubK$ is $O(n)$ in size, which is linear in the maximum possible number of documents belonging to a data owner, but not dependent on the number of documents stored in the cloud server, and hence this will not affect the system's practicality.

5.4 Security Analysis

To analyze the security of our scheme, and in particular show that the scheme satisfies the security requirements given in Section 3.3, we assume that the public cloud is "honest-but-curious". That is, the cloud server will only provide legitimate services according to pre-defined schemes, although it may try to recover secret information based on its knowledge.

We also assume that the authorized users may try to access data either within or out of the scopes of their privileges. Moreover, communication channels involving the public cloud are assumed to be insecure.

Based on the above considerations, we will prove the security of our scheme in terms of controlled searching and query privacy.

Theorem 1. *The proposed scheme supports controlled searching.*

Proof: Theorem 1 requires that any user with the aggregate key can perform a keyword search over documents in the set S, but he cannot do it over the documents outside this set. He also cannot generate other aggregate searchable encryption keys for a new set S' from the known one.Theorem 1 can be deduced from the following lemmas: □

Lemma 1. *Each user who has the aggregate key can perform a successful keyword search.*

Proof: Lemma 1 is equivalent to the correctness of the proposed scheme. After receiving the submitted single trapdoor Tr, the cloud server can adjust Tr to generate a desired trapdoor Tr_i for the i-th document in the S, and then execute KASE.$Test$ algorithm to perform keyword search. For correctness, we can see that:

$$\begin{aligned}
& e(Tr_i, c_1)/e(pub, c_2) \\
= \; & \frac{e(k_{agg} \cdot \Pi_{j \in s, j \neq i} g_{n+1-j+i} \cdot H(w), g^t)}{e(\Pi_{j \in s} g_{n+1-j}, (v \cdot g_i)^t)} \\
= \; & \frac{e(k_{agg}, g^t) \cdot e(\Pi_{j \in s, j \neq i} g_{n+1-j+i} \cdot H(w), g^t)}{e(\Pi_{j \in s} g_{n+1-j}, (v \cdot g_i)^t)} \\
= \; & \frac{e(k_{agg}, g^t) \cdot e(\Pi_{j \in s, j \neq i} g_{n+1-j+i} \cdot H(w), g^t)}{e(\Pi_{j \in s} g_{n+1-j}, g^{\gamma^t}) \cdot e(\Pi_{j \in s} g_{n+1-j}, (g_i)^t)} \\
= \; & \frac{e(\Pi_{j \in s, j \neq i} g_{n+1-j+i} \cdot H(w), g^t)}{e(\Pi_{j \in s} g_{n+1-j}, (g_i)^t)} \\
= \; & \frac{e(\Pi_{j \in s, j \neq i} g_{n+1-j+i}, g^t) \cdot e(H(w), g^t)}{e(\Pi_{j \in s} g_{n+1-j}, (g_i)^t)} \\
= \; & \frac{e(\Pi_{j \in s, j \neq i} g_{n+1-j+i}, g^t) \cdot e(H(w), g^t)}{e(\Pi_{j \in s} g_{n+1-j+i}, g^t)} \\
= \; & \frac{e(H(w), g^t) \cdot e(\Pi_{j \in s, j \neq i} g_{n+1-j+i}, g^t)}{e(\Pi_{j \in s} g_{n+1-j+i}, g^t)} \\
= \; & \frac{e(H(w), g^t) \cdot \frac{e(\Pi_{j \in s} g_{n+1-j+i}, g^t)}{e(g_{n+1}, g^t)}}{e(\Pi_{j \in s} g_{n+1-j+i}, g^t)} \\
= \; & \frac{e(H(w), g^t)}{e(g_{n+1}, g^t)} = \frac{e(H(w), g)^t}{e(g_1, g_n)^t} \\
= \; & c_w \qquad (1)
\end{aligned}$$

So, the user with the aggregate key can perform a successful keyword search. □

Lemma 2. *Even when the cloud server colludes with a malicious authorized user, they are unable to perform a keyword search over any document not in the scope of the user's aggregate key.*

Proof: In the case of collusion, an attacker $\mathcal{A}$ may have the knowledge of both a curious cloud server and a malicious authorized user. This kind of attacker may try to perform keyword search over a document not in the scope of his/her aggregate key. From the Equation 1, we can see that if pub is generated by a wrong set S', the expression $e(k_{agg}, g^t)$ will be equal to the expression $e(pub, v^t)$ (that is $e(\Pi_{j \in s} g_{n+1-j}, g^{\gamma^t})$), and they cannot be canceled out of the equation. So, the pub must be computed by the same set S of the aggregate key. Based on the above-mentioned fact, after receiving the single trapdoor Tr, the attacker may take a target set S' as the input of KASE.**Adjust** algorithm to generate the adjusted trapdoor, but for the reason that pub must be computed by the set S, such that the KASE.**Test** algorithm will output *false* for any document with index $i \notin S$. □

Lemma 3. *An attacker is unable to produce the new aggregate key for any new set of documents from the known aggregate key.*

Proof: A malicious authorized user $\mathcal{A}$ who owns an aggregate key k_{agg} of a set of documents S from a data owner, may try to generate a new aggregate key for a new set S' of the same data owner. To generate the new key, $\mathcal{A}$ should know the value of g^{γ}_{n+1-j} for any $j \in S'$. Although $\mathcal{A}$ has known the $k_{agg} = \Pi_{j \in s} g^{\gamma}_{n+1-j}$, he cannot get each multiplier from the product, and each multiplier is protected by the owner's master-secret key γ, so that $\mathcal{A}$ is unable to achieve his/her goal. □

Theorem 2. *The proposed scheme can achieve the goal of query privacy.*

Proof: The attackers may obtain some information to launch an attack. For example, the curious cloud server can obtain the stored keyword ciphertexts, the Δ_i associated with the SE key of i-th document, the submitted trapdoor, and so on. The malicious authorized user, say Tom, can have an aggregate key k_{agg} and the ability to perform keyword search over a set of documents of Alice. However, even if they can obtain these information, our scheme can be proved to achieve the goal of query privacy. The result of Theorem 2 can be derived from following lemmas: □

Lemma 4. *An attacker is unable to determine a keyword in a query from the submitted trapdoor.*

Proof: Assume attacker $\mathcal{A}$ wants to determine a keyword in a query after getting the submitted trapdoor T_r=$k_{agg} \cdot H(w)$, he will succeed only when he can guess the aggregate key k_{agg}. In this case, $\mathcal{A}$ can know 1) the system parameters $params$=($\mathcal{B}$, $PubK$), where $PubK = (g, g_1, \cdots, g_n, g_{n+2}, \cdots, g_{2n}) \in \mathcal{G}^{2n+1}$; 2) the document set S. However, to compute the k_{agg}, for each $j \in S$, the attacker $\mathcal{A}$ must compute the g^{γ}_{n+1-j}. Because γ is the data owner's master-secret key, which must be kept secret, $\mathcal{A}$ can only have a negligible probability to get it. So, the attacker cannot launch a successful attack in this case. □

Lemma 5. *An attacker is unable to determine a keyword in a document from the stored keyword ciphertexts and the related public information.*

Proof: The curious cloud server $\mathcal{A}$ may try to learn something from the stored encrypted data. With the knowledge of $PubK = (g, g_1, \cdots, g_n, g_{n+2}, \cdots, g_{2n}) \in \mathcal{G}^{2n+1}$, $c_1 = g^t$, $c_2 = (g^\gamma \cdot g_i)^t$ and $c_w = e(g, H(w))^t / e(g_1, g_n)^t$, $\mathcal{A}$ may try to launch two kinds of attacks as follows:

1) Retrieve the value of t from the known c_1 or c_2. However, the discrete logarithm problem means $\mathcal{A}$ cannot compute the value of t in this case.
2) Compute the value of $e(g_1, g_n)^t$. Notice that $\mathcal{A}$ can get the value of $e(g, H(w))^t$ by computing $e(c_1, H(w))$, so when he gets the value of $e(g_1, g_n)^t$, he will determine whether keyword w is in the c_w of the traget document. To obtain $e(g_1, g_n)^t$, $\mathcal{A}$ will compute $e(c_1, g_{n+1})$. However, because $PubK$ is missing the term $g_{n+1} = g^{a^{n+1}}$, the attacker $\mathcal{A}$ cannot finish this computation. In fact, this result is ensured by the assumption of the intractability of BDHE problem.

As a result, an attacker cannot learn the content from the stored information. □

Lemma 6. *An attacker is unable to determine a keyword in a query from the* ***Adjust*** *algorithm.*

Proof: For a submitted single trapdoor T_r=$k_{agg} \cdot H(w)$, the KASE.**Adjust** algorithm running in cloud server only involves a product of some public information, i.e., $Tr_i = Tr \cdot \Pi_{j \in s, j \neq i} g_{n+1-j+i}$. Because multipliers are all public, this algorithm provides no help for an attacker to determine a keyword in this trapdoor. So, a successful attack cannot be launched in this case. □

5.5 A Concrete Group Data Sharing System

When constructing a practical group data sharing system, it is important to reduce the number of keys belonging to a user. In this subsection, we will introduce how to build such a system based on the KASE and KAE schemes with the same public parameters.

We consider a group data sharing system without using any private cloud, but instead based on widely available public cloud services, such as Dropbox or citrix. Based on such a consideration, we assume a group manager (e.g., the HR director of an organization) with an authorized account to act in the role of "manager" who will be responsible for management of the system including maintaining the public system parameters stored in the cloud.

5.5.1 Table Definitions

We assume the cloud uses databases to manage the necessary information and four database (traditional or NOSQL databases) tables are thus defined as follows:

- Table **group**<*groupID, groupName, parameters*> is to store the system parameters.
- Table **member**<*memberID, membeName, password, publicKey*> is to store members' information including their public key.
- Table **docs**<*docID, docName, OwnerID, EncKey, SEKey, filePath*> is to store the uploaded document of an owner with identity *ownerID*.
- Table **sharedDocs**<*SID, memberID, OwnerID, docIDSet*> is to store the documents of a member with identity *memberID* shared by the owner with identity *OwnerID*. Field *docIDSet* is for all the indices of documents.

Note that the owner can encrypt the encryption key and SE key by his/her private key and store the ciphertexts in the fields *EncKey* and *SEKey*. In this way, he can reduce the cost of key management and only focus on how to safely store his/her private key.

5.5.2 Work Flows

To further describe this system in details, we describe its main work flows in this section.

System setup.When an organization submits a request, the cloud will create a database containing above four tables, assign a *groupID* for this organization and insert a record into table **company**. Moreover, it assigns an administrator account for the *manager*. Then, the group data sharing system will work under the control of *manager*. To generate the system parameters $params$, *manager* runs the algorithm KASE.**Setup** and updates the field *parameters* in table **company**.

User registration. When adding a new member, the *manager* assigns *memberID*, *membeName*, *password* and a key pair generated by any public key encryption (PKE) scheme for him, then stores the necessary information into the table **member**. A user's private key should be distributed through a secure channel.

User login. Like most popular data sharing products (e.g., Dropbox and citrix), our system relies on password verification for authenticating users. To further improve the security, multi-factor authentication or digital signatures may be used when available.

Data uploading. To upload a document, the owner runs KAE.**Encrypt** to encrypt the data and KASE.**Encrypt** to encrypt the keyword ciphertexts, then uploads them to the cloud. The cloud assigns a *docID* for this document and stores the encrypted data in the path *filePath*, then inserts a record into the table **docs**. In addition, the owner can encrypt the keys using his/her private key and store them into the table **docs**.

Data sharing. To share a group of documents with a target member, the owner runs KAE.**Extract** and

KASE.**Extract** to generate the aggregate keys, and distributes them to this member, then inserts/updates a record in table **sharedDocs**. If the shared documents for this member are changed, the owner must re-extract the keys and update the field *docIDSet* in table **sharedDocs**.

Keyword Search. To retrieve the documents containing an expected keyword, a member runs KASE.**Trapdoor** to generate the keyword trapdoor for documents shared by each owner, then submits each trapdoor and the related owner's identity *OwnerID* to the cloud. After receiving the request, for each trapdoor, the cloud will run KASE.**Adjust** the trapdoor for each document in the *docIDSet* and run KASE.**Test** to perform keyword search. Then, the cloud will return the encrypted documents which contains the expected keyword to the member.

Data retrieving. After receiving the encrypted document, the member will run KAE.**Decrypt** to decrypt the document using the aggregate key distributed by the document's owner.

5.5.3 Analysis

From the work flows above, we can see that the number of keys of a member is linear in the number of users who share documents with him, and the number of trapdoors in a keyword search is the same. Compared to traditional data sharing solutions, this system has better efficiency.

6 Performance Evaluation

Considering that: 1) in a practical data sharing system based on cloud storage, the user can retrieve data by any possible device and the mobile devices are widely used now; 2) the performance is highly dependent on the basic cryptographic operations especially in the pairing computation, we study whether the cryptographic operations based on pairing computation can be efficiently executed using both computers and mobile devices.

6.1 Implementation Details

In our implementation, two source libraries about pairing computation are used: 1) *jpbc* library is used to implement cryptographic operations running in mobile smartphones; 2) *pbc* library is used to implement cryptographic operations running in computers. Because the aggregate key and trapdoor contains one $\mathcal{G}$ element, and the keyword ciphertexts only contain two $\mathcal{G}_\infty$ elements, we can select the Type-A pairing. Type-A pairing is the fastest (symmetric) pairing among all types of curves, which is constructed on the curve $Y^2 = X^3 + X$ over the field F_p for some prime p=3 mod 4.

Each cryptographic operation involved in our system will be evaluated in two different platforms: one is in Java on *Samsung G3502U* phone with Android OS 4.2, the other is in C++ on Computer of Intel(R) Core(TM)i5-3337U CPU @ 1.80GHZ with Windows7 OS.

6.2 Pairing Computation

About pairing computation, some experiment results have been published. In mobile devices, Oliveira et al. [25] show that it needs 5 *second* in 2007, but now it will be faster; In sensor nodes and personal digital assistant (PDA), Li et al. [26] shows that it only needs 1.5 *second* and 0.5 *second* respectively in 2010.

TABLE 1
Execution times of type A pairing computation (ms)

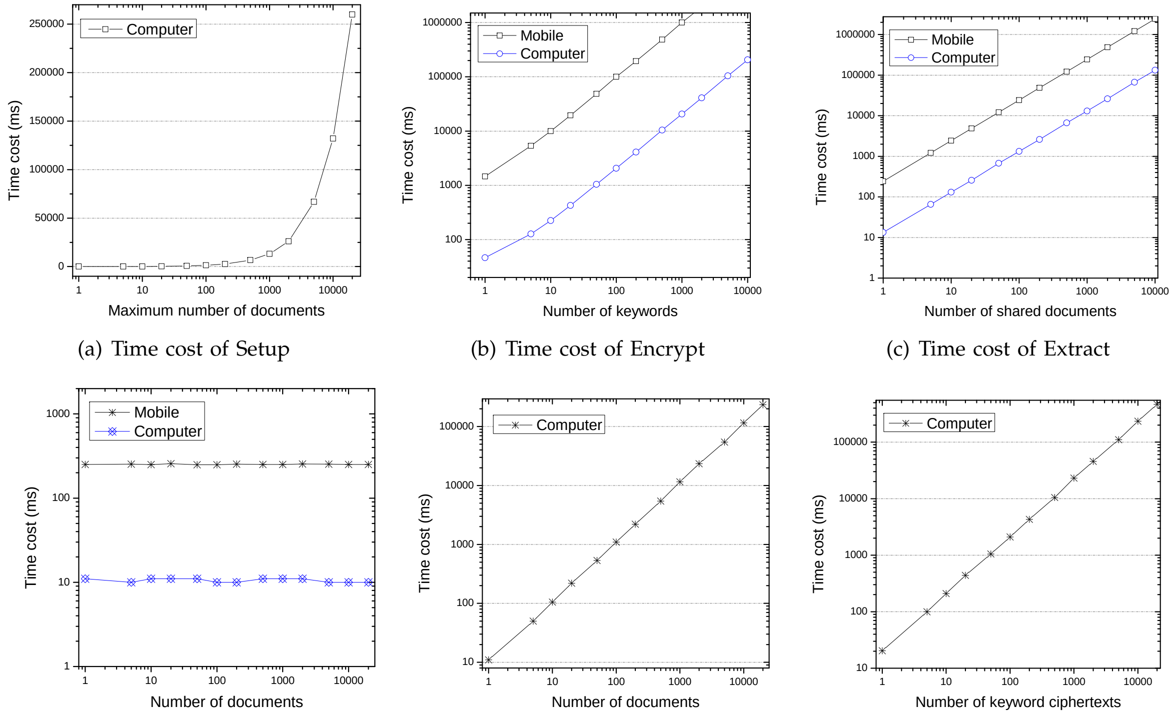
	Pairing	pow(in $\mathcal{G}$)	pow(in $\mathcal{G}_1$)	pow(in Z_p)
Mobile Devices	485	243	74	0.8
Computer	10.2	13.3	1.7	0.05

Table.1 shows the average time of pairing computation (two different elements in $\mathcal{G}$ as input, for example, $e(g,h)$) is 487*ms* in mobile device, which has the same result with the experiments in PDA in [26]. It also shows the results of execution time of element *pow* computation in different groups. In computers, the average times of pairing and *pow* computation are much faster than in mobile devices.

6.3 Evaluation of KASE Algorithms

Considering that the algorithms including KASE.**Setup**, KASE.**Adjust** and KASE.**Test** are only run in the cloud server, only the execution times in computer are tested. As shown in Fig.4, we can see that:

1) The execution time of KASE.**Setup** is linear in the maximum number of documents belonging to one owner, and when the maximum number grows up to 20000, it is reasonable that KASE.**Setup** algorithm only needs 259 *second*.
2) The execution time of KASE.**Encrypt** is linear in the number of keywords, and when the number grows up to 10000, KASE.**Encrypt** algorithm only needs 206 *second* in computers, but 10018 *second* in mobile devices. Therefore, we can draw two conclusions; one is that it is not feasible to upload document with lots of keywords using a mobile phone; the other is that the keyword search with pairing computation can be executed quickly in computers now.
3) The execution time of KASE.**Extract** is linear in the number of shared documents, and when the number grows up to 10000, KASE.**Extract** algorithm only needs 132 *second* in computer, but 2430 *second* in mobile devices. Because the KASE.**Extract** always runs along with the KASE.**Encrypt**, it is not suggested to be executed in the mobile devices.



(a) Time cost of Setup

(b) Time cost of Encrypt

(c) Time cost of Extract

(d) Time cost of Trapdoor

(e) Time cost of Adjust

(f) Time cost of Test

Fig. 4. Time cost of KASE algorithms.

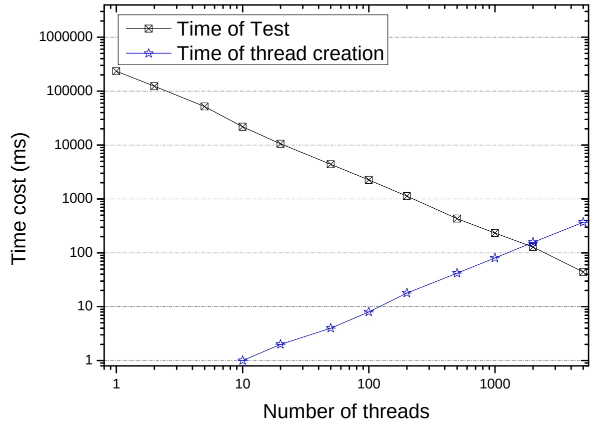


Fig. 5. Time cost of keyword search.

4) The execution time of KASE.**Trapdoor** is a constant, i.e., 0.01 *second* in computer and 0.25 *second* in mobile devices. In fact, the mathematical operation in KASE.**Trapdoor** is the once multiplication in $\mathcal{G}$, so that the keyword search can be performed efficiently in both mobile devices and computer. Compared with other schemes, there is a significant improvement in our scheme.
5) The execution time of KASE.**Adjust** is linear in the number of documents. In fact, it can be improved in the practical application, and the details are shown in section 6.4.
6) The execution time of KASE.**Test** is linear in the number of keyword ciphertexts. In fact, the mathematical operation in KASE.**Test** is twice as much as the pairing computations. When the number grows up to 20000, it will take 467 *second*.

6.4 Evaluation of the Group Data Sharing System

Considering that the system's performances most critically depend on the KASE algorithms, we consider employing caching techniques in the group data sharing system to further improve the efficiency of the **keyword search** procedure. After receiving an aggregate trapdoor, the cloud server will run KASE.**Adjust** and KASE.**Test** to finish the keyword search.

Fig.4(e) shows that the execution time of KASE.**Adjust** is linear in the number of documents. In fact, to avoid the repeated calculation and improve the performance, the cloud server can cache the computation result of (S, i, P), where P is the product $\Pi_{j\in s,j\neq i}g_{n+1-j+i}$. Because the input and calculation process are the same for all users, this will greatly save the calculation time. When the user queries for the second time, or the same S has appeared in the past queries, KASE.**Adjust** can run quickly by using the pre-computed result.

Fig.4(f) shows that the execution time of KASE.**Test** is linear in the number of keyword ciphertexts. To improve the efficiency, some parallel computing and distributed computing techniques maybe applied, such as multi-thread, hadoop, etc. The multi-thread technique is adopted in our experiment. To test the performance, we set the number of keyword ciphertexts as 10000. As shown in Fig.5, we can see that the execution time of KASE.**Test** will be reduced when we increase the number of threads. When the number grows up to 200, it only needs 1 *second* to finish the keyword search over 10000 keyword ciphertexts. We also see that when the number of threads is large, it would take more time to create these threads. When the number grows up to 1000, the time of thread creation will become 80 *millisecond*. So, the multi-thread technique can provide the help for improving performance, but the number of threads should be selected carefully in the practical applications.

7 Conclusion

Considering the practical problem of privacy-preserving data sharing system based on public cloud storage which requires a data owner to distribute a large number of keys to users to enable them to access his/her documents, we for the first time propose the concept of key-aggregate searchable encryption (KASE) and construct a concrete KASE scheme. Both analysis and evaluation results confirm that our work can provide an effective solution to building practical data sharing system based on public cloud storage.

In a KASE scheme, the owner only needs to distribute a single key to a user when sharing lots of documents with the user, and the user only needs to submit a single trapdoor when he queries over all documents shared by the same owner. However, if a user wants to query over documents shared by multiple owners, he must generate multiple trapdoors to the cloud. How to reduce the number of trapdoors under multi-owners setting is a future work. Moreover, federated clouds have attracted a lot of attention nowadays, but our KASE cannot be applied in this case directly. It is also a future work to provide the solution for KASE in the case of federated clouds.

Acknowledgments

This work is supported by the National Key Basic Research Program of China (No. 2013CB834204), National Natural Science Foundation of China (Nos. 61272423, 61300241, 61170268, 61100047 and 61272493), National Natural Science Foundation of Tianjin (Nos 13JCQNJC00300 and 14JCYBJC15300), Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20120031120036).

References

[1] S. Yu, C. Wang, K. Ren, and W. Lou, “Achieving Secure, Scalable, and Fine-Grained Data Access Control in Cloud Computing”, *Proc. IEEE INFOCOM*, pp. 534-542, 2010.

[2] R. Lu, X. Lin, X. Liang, and X. Shen, “Secure Provenance: The Essential of Bread and Butter of Data Forensics in Cloud Computing”, *Proc. ACM Symp. Information, Computer and Comm. Security*, pp. 282-292, 2010.

[3] X. Liu, Y. Zhang, B. Wang, and J. Yan. “Mona: secure multi-owner data sharing for dynamic groups in the cloud”, *IEEE Transactions on Parallel and Distributed Systems*, 2013, 24(6): 1182-1191.

[4] C. Chu, S. Chow, W. Tzeng, et al. “Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage”, *IEEE Transactions on Parallel and Distributed Systems*, 2014, 25(2): 468-477.

[5] X. Song, D. Wagner, A. Perrig. “Practical techniques for searches on encrypted data”, *IEEE Symposium on Security and Privacy*, IEEE Press, pp. 44C55, 2000.

[6] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky. “Searchable symmetric encryption: improved definitions and efficient constructions”, *In: Proceedings of the 13th ACM conference on Computer and Communications Security*, ACM Press, pp. 79-88, 2006.

[7] P. Van,S. Sedghi, JM. Doumen. “Computationally efficient searchable symmetric encryption”, *Secure Data Management*, pp. 87-100, 2010.

[8] S. Kamara, C. Papamanthou, T. Roeder. “Dynamic searchable symmetric encryption”, *Proceedings of the 2012 ACM conference on Computer and communications security (CCS)*, ACM, pp. 965-976, 2012.

[9] D. Boneh, C. G, R. Ostrovsky, G. Persiano. “Public Key Encryption with Keyword Search”, *EUROCRYPT 2004*, pp. 506C522, 2004.

[10] Y. Hwang, P. Lee. “Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System”, *In: Pairing-Based Cryptography C Pairing 2007*, LNCS, pp. 2-22, 2007.

[11] J. Li, Q. Wang, C. Wang. “Fuzzy keyword search over encrypted data in cloud computing”, *Proc. IEEE INFOCOM*, pp. 1-5, 2010.

[12] C. Bosch, R. Brinkma, P. Hartel. “Conjunctive wildcard search over encrypted data”, *Secure Data Management*. LNCS, pp. 114-127, 2011.

[13] C. Dong, G. Russello, N. Dulay. “Shared and searchable encrypted data for untrusted servers”, *Journal of Computer Security*, pp. 367-397, 2011.

[14] F. Zhao, T. Nishide, K. Sakurai. Multi-User Keyword Search Scheme for Secure Data Sharing with Fine-Grained Access Control. *Information Security and Cryptology*, LNCS, pp. 406-418, 2012.

[15] J. W. Li, J. Li, X. F. Chen, et al. “Efficient Keyword Search over Encrypted Data with Fine-Grained Access Control in Hybrid Cloud”, *In: Network and System Security 2012*, LNCS, pp. 490-502, 2012.

[16] J. Li, K. Kim. “Hidden attribute-based signatures without anonymity revocation”, *Information Sciences*, 180(9): 1681-1689, Elsevier, 2010.

[17] X.F. Chen, J. Li, X.Y. Huang, J.W. Li, Y. Xiang. “Secure Outsourced Attribute-based Signatures”, *IEEE Trans. on Parallel and Distributed Systems*, DOI.ieeecomputersociety.org/10.1109/TPDS.2013.180, 2013.

[18] J.Li, X.F. Chen, M.Q. Li, J.W. Li, P. Lee, Wenjing Lou. “Secure Deduplication with Efficient and Reliable Convergent Key Management”, *IEEE Transactions on Parallel and Distributed Systems*, 25(6): 1615-1625, 2014.

[19] Z. Liu, Z. Wang, X. Cheng, et al. “Multi-user Searchable Encryption with Coarser-Grained Access Control in Hybrid Cloud”, *Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT)*, IEEE, pp. 249-255, 2013.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou, “Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing”, *Proc. IEEE INFOCOM*, pp. 525-533, 2010.

[21] B. Wang, B. Li, and H. Li, “Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud”, *Proc. 10th Int'l Conf. Applied Cryptography and Network Security*, pp. 507-525, 2012.

[22] D. Boneh, C. Gentry, B. Waters.” Collusion resistant broadcast encryption with short ciphertexts and private keys”, *Advances in CryptologyCCRYPTO 2005*, pp. 258-275, 2005.

[23] D. H. Phan, D. Pointcheval, S. F. Shahandashti, et al. “Adaptive CCA broadcast encryption with constant-size secret keys and ciphertexts”, *International journal of information security*, 12(4): 251-265, 2013.

[24] D. Boneh, B. Lynn, H. Shacham. “Short signatures from the Weil pairing”, *Advances in Cryptology ASIACRYPT 2001*, pp. 514-532, 2001.

[25] L. B. Oliveira, D. F. Aranha, E. Morais, et al. "Tinytate: Computing the tate pairing in resource-constrained sensor nodes", *IEEE Sixth IEEE International Symposium on Network Computing and Applications*, pp. 318-323, 2007.

[26] M. Li, W. Lou, K. Ren. "Data security and privacy in wireless body area networks", *Wireless Communications*, IEEE, 17(1): 51-58, 2010.

[27] D. Boneh, C. Gentry and B. Waters. "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys", *CRYPTO'05*, pp. 258C275, 2005.

[28] R. A. Popa ,N. Zeldovich. "Multi-key searchable encryption". *Cryptology ePrint Archive*, Report 2013/508, 2013.

Baojiang Cui recieved B.S. in Hebei University of Technology, China, in 1994, M.S. in Harbin Institute of Technology, China, in 1998 and Ph.D. in Control Theory and Control Engineering in Naikai University, China in 2014. He is an Associate Professor in the School of Computer Science at Beijing University of Posts and Telecommunications, China. His main research area are detection of software, cloud computing and the Internet of Things.

Zheli Liu received the BSc and MSc degrees in computer science from Jilin University, China, in 2002 and 2005, respectively. He received the PhD degree in computer application from Jilin University in 2009. After a postdoctoral fellowship in Nankai University, he joined the College of Computer and Control Engineering of Nankai University in 2011. His current research interests include applied cryptography and data privacy protection.

Lingyu Wang is an associate professor in the Concordia Institute for Information Systems Engineering (CIISE) at Concordia University, Montreal, Quebec, Canada. He received his Ph.D. degree in Information Technology in 2006 from George Mason University. He holds a M.E. from Shanghai Jiao Tong University and a B.E. from Shenyang Aerospace University in China. His research interests include data privacy, network security, security metrics, cloud computing security, and software security. He has co-authored over 90 peer-reviewed scientific publications on security and privacy. He has served as the program co-chair of six international conferences, the panelist of National Science Foundation, and the technical program committee member of over 80 international conferences.