Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

MULKASE: a novel approach for key-aggregate searchable encryption for multi-owner data

Mukti PADHYA[‡1], Devesh C. JINWALA[2]

[1]*Department of Computer Engineering, Sardar Vallabhbhai National Institute of Technology, Surat 394000, India*

[2]*Department of Computer Science and Engineering, Indian Institute of Technology, Jammu 180001, India*

E-mail: mukti.padhya@yahoo.in; dcjinwala@gmail.com

Received Mar. 28, 2018; Revision accepted Aug. 5, 2018; Crosschecked Aug. 12, 2019; Published online Oct. 29, 2019

Abstract: Recent attempts at key-aggregate searchable encryption (KASE) combine the advantages of searching encrypted data with support for data owners to share an aggregate searchable key with a user delegating search rights to a set of data. A user, in turn, is required to submit only one single aggregate trapdoor to the cloud to perform a keyword search across the shared set of data. However, the existing KASE methods do not support searching through data that are shared by multiple owners using a single aggregate trapdoor. Therefore, we propose a MULKASE method that allows a user to search across different data records owned by multiple users using a single trapdoor. In MULKASE, the size of the aggregate key is independent of the number of documents held by a data owner. The size of an aggregate key remains constant even though the number of outsourced ciphertexts goes beyond the predefined limit. Security analysis proves that MULKASE is secure against chosen message attacks and chosen keyword attacks. In addition, the security analysis confirms that MULKASE is secure against cross-pairing attacks and provides query privacy. Theoretical and empirical analyses show that MULKASE performs better than the existing KASE methods. We also illustrate how MULKASE can carry out federated searches.

Key words: Searchable encryption; Cloud storage; Key-aggregate encryption; Data sharing
https://doi.org/10.1631/FITEE.1800192 **CLC number:** TP309

1 Introduction

Cloud storage servers provide storage and computational resources to users that in turn reduce the storage overhead, processing cost, and management cost of data on the user side. However, while an open communication channel is being considered between a user and the cloud server, an adversary can access this communication channel. The confidentiality of data is lost if any information leakage takes place from the user's personal data to the cloud service provider or to an unauthorized user. Therefore, data owners must store their data in an encrypted form on the cloud. Because the data are required to be stored in an encrypted form, a keyword-based search technique is required to retrieve specific data from the set of encrypted data, which exploits searchable encryption (SE) (Song et al., 2000). SE is a cryptographic technique that allows an authorized user to search across encrypted data containing specified query keywords without leaking any information about the data. However, existing SE methods (Song et al., 2000; Goh, 2003; Chang and Mitzenmacher, 2005; Curtmola et al., 2011; Wang et al., 2013; Sun et al., 2014) satisfy only the primary requirement of secure data retrieval from cloud storage. The existing SE methods do not address the challenge of efficient delegation of search and access rights for a set of files. Specifically, to delegate search and access rights for m files ($|S| = m$ meaning that there are

‡ Corresponding author
ORCID: Mukti PADHYA, http://orcid.org/0000-0002-0498-4188
© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2019

m elements in set S) encrypted under different keys, a data owner is required to share $O(|S|)$ keys. This limitation of the existing SE methods introduces issues in terms of storage as well as communication overhead of $O(|S|)$.

Therefore, data owners are required to store, manage, and distribute a large number of keys, proportional to the number of shared files. In addition, an authorized user is required to generate and submit multiple trapdoors to search through all of the shared data. Therefore, in existing SE methods, ensuring secure communication for key distribution at lower computational and storage requirement levels is an issue yet to be addressed.

Key-aggregate searchable encryption (KASE) (Cui et al., 2016) was proposed as a way to reduce the number of keys required for sharing when delegating search rights across a set of data. KASE allows a data owner to delegate keyword search rights on a set S of files. KASE inherits the property of key-aggregate encryption (KAE) (Chu et al., 2014); i.e., to delegate search rights on m files, the data owner is required to share a single aggregate key with a user. In addition, a user is required to submit a single aggregate trapdoor (instead of a group of trapdoors) to the cloud to perform a keyword search across m shared files from the same data owner. However, KASE does not support searching across data owned by different owners using a single searchable secret key. Formally, to search across multi-owner data for U data owners, the user is required to submit $O(U)$ trapdoors (generated using U different aggregate keys) to the cloud server. This results in a system with $O(U)$ communication overhead on the user side and $O(U)$ storage as well as computational overhead on the server side. Therefore, Li et al. (2016) proposed KASE to support multi-owner search using a single trapdoor and also to provide verification of the search results using an aggregate key. However, security analysis against chosen message attack and chosen keyword attack was not discussed in the KASE methods (Cui et al., 2016; Li et al., 2016). Moreover, KASE is insecure against cross-pairing or keyword guessing attacks as proved in Kiayias et al. (2016). The KASE method from Li et al. (2016) follows the same construction as the KASE method from Cui et al. (2016). Therefore, the cross-pairing attack is possible in the method proposed by Li et al. (2016) as well. The cross-pairing attack is possible in the KASE methods, which do not provide data and trapdoor privacy as discussed in Kiayias et al. (2016). The KAE and KASE methods use the security assumption of the bilinear Diffie-Hellman exponent (BDHE), which is a variant of the strong Diffie-Hellman (SDH) problem (i.e., finding $g^{\alpha+1}$ given $\{g, g^{\alpha}, ..., g^{\alpha^l}\}$). However, SDH or its variants such as BDHE do not guarantee security for specific parameters, as discussed in Cheon (2006). Security analysis of the SDH problem shows that if g, g^{α}, and g^{α^d} are given for a positive divisor d of $p-1$, then secret $\alpha \in Z_p$ can be computed in $O(\log p \times (\sqrt{p/d} + \sqrt{d}))$ group operations using $O(\max(\sqrt{p/d}, \sqrt{d}))$ memory. Furthermore, in the KASE methods (Cui et al., 2016; Li et al., 2016), when a user submits an aggregate trapdoor Tr to the cloud server for searching across m ($|S| = m$) files, the server first requires generating trapdoor Tr_i for each index $i \in S$ using the submitted trapdoor Tr. Here, S is a set of the ciphertext's indices across which search rights are delegated by the data owner. The process of generating an individual trapdoor from the aggregate trapdoor at the time of keyword searching adds an extra computational overhead of $O(|S|)$ on the server side. The predefined bound on the maximum number of possible ciphertext classes at the time of system setup makes the KAE and KASE methods impractical for real-time use. If the number of outsourced ciphertexts goes beyond the predefined limit n, then public parameters and keys (as they rely on n) are required to be updated in the KAE and KASE methods. Specifically, the KAE and KASE methods are not flexible in terms of complex classifications of ciphertext classes, i.e., when the data owner wants to classify the ciphertexts under more than n (predefined limit of documents held by a data owner) different classes, then the number of aggregate keys required to be shared is not constant (a detailed discussion on the same limitation is presented in Section 2). Therefore, Guo et al. (2017) proposed a KAE method in which n is variable and an aggregate key is independent of the number of maximum possible ciphertext classes. This KAE (Guo et al., 2017) approach also allows the cloud server to check the authenticity of the users who request access to the outsourced ciphertext. However, this KAE does not allow the user to perform a search for a keyword across the shared data. Additionally, in their method the size

of an aggregate key is two group elements instead of one group element as in the existing methods (Chu et al., 2014; Cui et al., 2016; Li et al., 2016). It is also important to note the growing use of federated clouds, which have attracted considerable attention in recent years (Cui et al., 2016). However, one cannot directly apply existing KASE methods to searches across data scattered across multiple clouds. Therefore, in this study, we discuss how the proposed MULKASE method can be used to carry out a federated search using a single trapdoor. We address the following issues: (1) Is it possible to allow searching across different data records owned by multiple users using a single aggregate trapdoor? (2) How can flexibility in the delegation of access as well as search rights be achieved when the classifications of ciphertext classes are complex? (3) How can the query performance of the existing KASE methods be improved? (4) How can the storage as well as the computational overhead of transforming a submitted trapdoor Tr to generate an individual trapdoor Tr_i before keyword searching on the server side be overcome? (5) How can federated searches using KASE be implemented?

We propose a collision-resistant privacy-preserving MULKASE to overcome the limitations of the existing methods (Chu et al., 2014; Cui et al., 2016; Li et al., 2016; Guo et al., 2017). The proposed MULKASE allows a user to generate and submit a single trapdoor to search across a set of data shared by multiple owners.

We emphasize that the proposed MULKASE is a better choice compared to the existing methods in the following ways:

1. Search across multi-owner data

MULKASE supports searching across multi-owner encrypted data using a single trapdoor (generated using an aggregate key having a size the same as that of a single secret key), which is one of the open problems in KASE (Cui et al., 2016). An authorized user can access and search across any ciphertext in the range of a given aggregate key k_{agg}. However, the user is unable to obtain any information related to the ciphertext for which access and search rights are not permitted. In addition, the aggregate key is of a constant size (i.e., it consists of a single group of elements irrespective of the number of data owners and documents) that can be stored in mobile devices or Internet of Things (IoT) sensors having limited storage resources.

2. Flexible classification

If the number of outsourced ciphertexts goes beyond the predefined limit of n, then MULKASE allows the data owner to classify ciphertexts under more than n different classes. In addition, by sharing a single aggregate key, the data owner can delegate search and access rights for the set of ciphertexts classified under different classes. This makes the MULKASE method flexible in terms of complex classifications of ciphertexts. In MULKASE, the size of an aggregate key is independent of the number of documents held by a data owner. Therefore, if the number of outsourced ciphertexts goes beyond the predefined limit, then the size of the aggregate key remains constant.

3. No trapdoor transformation required

In MULKASE, we overcome the computational overhead of transforming the submitted trapdoor Tr to generate the individual trapdoor Tr_i on the server side by allowing the cloud server to search across encrypted data using the submitted trapdoor only. Therefore, we improve query performance in comparison to KASE methods (Cui et al., 2016; Li et al., 2016). The detailed analysis is done in Sections 5.3 and 6.

4. Security and theoretical analyses

We analyze the efficiency of our method and establish its security through the proofs of Theorems 1–5. The security analysis shows that the proposed method is secure against adaptive chosen keyword attacks and also satisfies the indistinguishability of messages. The hardness of our method is based on the decision bilinear Diffie-Hellman (DBDH) assumption. We also show that the proposed MULKASE method is secure against cross-pairing or keyword guessing attack. In addition, the proposed approach uses a fixed number of pairing operations during decryption as well as keyword search irrespective of the number of data owners and documents.

5. Empirical analysis

We analyze and compare the performance of the proposed method with those of the existing methods (Cui et al., 2016; Li et al., 2016) for different parameters (with a different number of keywords and users). The performance analysis concurs with the theoretical analysis and justifies the effectiveness of the proposed MULKASE as compared to the

existing methods.

6. Federated search

We define a system model to perform a search across a set of data federated across multiple clouds using MULKASE.

2 Related work

In this section, we review several categories of existing solutions and explain their relationships to the proposed MULKASE method.

Data sharing is one of the significant functionalities of cloud storage as discussed in Chu et al. (2014). Cloud service providers allow a data owner to share data with other users by providing storage resources on the cloud server. However, data sharing with different users via public cloud storage gives rise to the issue of data security and privacy. Therefore, a cryptographic solution is required to share data with a selected group of users using a minimum number of encryption keys. Here, we focus on public key based encryption methods for data sharing as they do not require the secure exchange of one (or more) symmetric key(s) between the communicating parties. However, in the traditional public-key cryptographic (PKC) methods for sharing encrypted data with a group of users, the sender is required to perform multiple encryptions using the public key of each user. This results in an overhead in multiple key managements and a greater number of encryptions on the sender side. Specifically, the traditional PKC methods suffer from certain challenges, i.e., concurrently achieving fine-grained access control and system scalability, managing user accountability, and efficient management of the large number of keys (Rivest et al., 1978). In the literature, the following solutions have been proposed to reduce the cost of key management.

2.1 Hierarchical key management methods based on a tree structure

The hierarchical key management methods based on a tree structure are proposed to reduce the number of keys a user required to share and to overcome the key management issues (Akl and Taylor, 1983; Ateniese et al., 2006; Atallah et al., 2009; Ragab-Hassen, 2010). However, key compression and key management are effective only under a scenario where the tree structure is predefined. For example, in Ateniese et al. (2006), if a data owner wants to share files categorized under different branches, then he/she is required to share a number of keys equaling the number of classes.

2.2 Broadcast encryption

Fiat and Naor (1993) proposed a broadcast encryption (BE) method to overcome the key management issues while sharing data in a multicast scenario. BE allows the sender to broadcast a ciphertext to a selected group of users, and the members of the group can decrypt it using their private key. However, no one outside the selected group can access the message. The method proposed in Fiat and Naor (1993) is not fully collision-resistant. The proposed BE method chooses a value for k, i.e., the number of colliding revoked users, before the initial setup of the algorithms. Therefore, the method is secure against k colliding users only. Moreover, in BE, the user's keys must be updated after every transmission of a message. However, there are solutions proposed in the literature to overcome the issues in a BE method (Kurosawa et al., 2000; Dodis and Fazio, 2003; Yao et al., 2004; Park and Lee, 2008).

2.3 Attribute-based encryption

Sahai and Waters (2005) proposed a public-key cryptosystem called attribute-based encryption (ABE), which is widely used for secure data sharing. ABE was further classified into two types: key-policy attribute-based encryption (KP-ABE) introduced by Goyal et al. (2006) and ciphertext-policy attribute-based encryption (CP-ABE) put forth by Bethencourt et al. (2007). However, the existing ABE methods (Sahai and Waters, 2005; Goyal et al., 2006; Bethencourt et al., 2007; Cheung and Newport, 2007; Daza et al., 2010; Pirretti et al., 2010; Waters, 2011) suffer from one (or more) of the following issues: (1) The number of secret key components grows linearly with the number of user attributes; (2) The size of the ciphertext is proportional to the number of attributes that are in the access policy; (3) The number of exponentiations and pairing computations required during encryption and decryption grows linearly with the number of attributes attached to ciphertext and user's key; (4) The access policy is sent along with the ciphertext (in CP-ABE) (i.e., the method does not provide recipient anonymity);

(5) The leakage resiliency is against the leakage only from the decryptor and not from the data owner, i.e., the encryptor side.

Wang et al. (2017) proposed a security model for leakage-resilient ABE with an improved auxiliary input that provides leakage resilience against both sides: encryptor and decryptor. However, as mentioned in the security issues above, in ABE the sizes of the secret key and ciphertext often increase linearly with the number of attributes encompassed. In IoT applications, we cannot expect large storage for secret keys and more computation time to encrypt the data in resource-constrained end-devices such as smartphones, smart meters, and smart cards. Therefore, key-aggregate encryption (KAE) (Chu et al., 2014) was proposed to overcome the issues of ABE in group data sharing.

2.4 Key-aggregate encryption

KAE (Chu et al., 2014) is a public-key cryptosystem that allows data owners to aggregate a set of decryption keys and it generates a constant sized single decryption key. To delegate access rights to the set of files, the data owner is required to share a single aggregate key to the user. In addition, a user is required to submit a single aggregate key to the cloud server for accessing any number of shared files from the same data owner. In KAE, the ciphertext is associated with an index i, given by the data owner at the time of encryption to categorize ciphertexts into different classes. The example shown in Fig. 1 explains the concept of ciphertext class. As shown in the figure, Alice has personal photos and confidential data. Based on whether Alice's research data or finance data are there in the files, she gives an index along with the plaintext as inputs to the encryption algorithm. If Alice wants to delegate access rights to her finance data to her charted accountant (CA), then she has to aggregate all the encryption keys with which the finance data are encrypted and has to share a single aggregate key with her CA.

Using a master-secret key, the secret keys for different classes are extracted. For any set of a document's indices $S \subseteq \{1, 2, \ldots, n\}$, the secret key holder generates a single constant sized aggregate key k_{agg} to delegate the access rights of dataset S to another user. Using the aggregate key k_{agg}, the user decrypts any ciphertext C_i if $i \in S$. More importantly, the size of an aggregate key is the same as the size of a single secret key, and by using an aggregate key, the set of ciphertexts is decrypted (instead of the separate secret keys).

The idea of key-aggregate encryption was first proposed by Chu et al. (2014). After that, many solutions have been proposed for KAE and to improve this method (Mahalle and Pawade, 2014; Banu, 2015; Chame and Kumar, 2015; Firdose and Rebekah, 2015; Patranabis et al., 2015; Dang et al., 2016).

The predefined bound of the maximum number of ciphertext classes is a limitation of KAE (Chu et al., 2014). If the number of outsourced ciphertexts goes beyond the predefined limit n, then public parameters and keys (as they rely on n) must be updated in KAE. Specifically, the KAE method is not flexible in terms of complex classifications of ciphertext classes; i.e., when a data owner wants to classify ciphertexts under more than n (the predefined limit of documents held by a data owner) different classes, then the number of aggregate keys required to be shared is not constant. Therefore, Chu et al. (2014) proposed a variant, i.e., a public-key extension of KAE, to meet the scenario shown in Fig. 2. The data owner can request additional key pairs in case he/she wants to classify ciphertexts into more than n classes. As shown in Fig. 2, an aggregate key k_{agg1} comprises the access rights of secret keys $\{k_1, k_2, \ldots, k_n\}$. If the number of outsourced ciphertexts goes beyond the predefined limit n, then the number of ciphertext classes increases to $n + 1$.

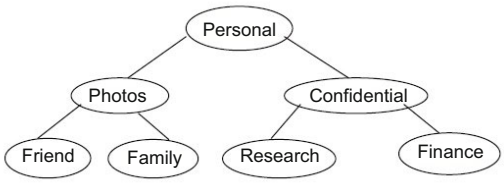

Fig. 1 An example of a ciphertext class

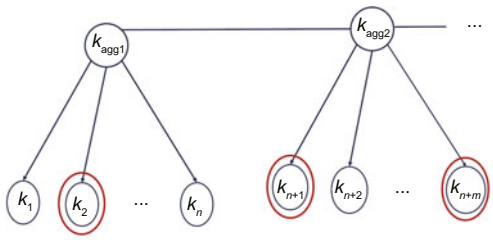

Fig. 2 Classification of ciphertexts under more than n classes in the KAE method

Therefore, additional key pairs are assigned to data owners and a new aggregate key k_{agg2} is generated that comprises the access rights of the keys $\{k_{n+1}, k_{n+2}, \ldots, k_{n+m}\}$. However, using k_{agg1} one cannot access any ciphertext encrypted using the secret keys $\{k_{n+1}, k_{n+2}, \ldots, k_{n+m}\}$. If the data owner wants to delegate the access rights of ciphertexts encrypted using secret keys $\{k_2, k_{n+1}, k_{n+m}\}$ that are from different branches, then he/she is required to share two aggregate keys, i.e., k_{agg1} and k_{agg2}. In KAE (Chu et al., 2014), the aggregate key for the parent node comprises the access rights of the keys for the child nodes. Therefore, in KAE to delegate the access rights of files encrypted using keys from different branches, the number of aggregate keys required to be generated is not constant.

The KAE approach proposed in Mahalle and Pawade (2014) uses the decoy technology. For the most part, the decoy technology stores some fake or wrong data files in the database, but these files have the user's actual data. Therefore, an attacker cannot differentiate between the actual documents and the decoy documents. In Dang et al. (2016), fast reconstruction techniques were proposed to resolve the scalability issue in adopting key aggregation in practical applications having large datasets (e.g., sensor data, especially time-series data which are continuously sensed, encrypted, and streamed to the cloud). The approach proposed in Dang et al. (2016) considers a multidimensional range as well as down-sampling queries on the large datasets. The proposed approach tries to decrease the reconstruction time of the aggregate key for such queries while preserving its provable security. Patranabis et al. (2015) proposed a dynamic key-aggregate encryption method that allows dynamic revocation of user access rights without changing either the public or private parameters, unlike existing key-aggregate methods (Mahalle and Pawade, 2014; Banu, 2015; Chame and Kumar, 2015; Firdose and Rebekah, 2015; Dang et al., 2016) that are static in their access right delegation policies. The method proposed in Patranabis et al. (2015) uses an additive subgroup that helps achieve massive reductions in the ciphertext as well as the key sizes, in contrast to the existing methods because they use bilinear pairings across multiplicative elliptic curve subgroups. In existing methods (Soubhagya et al., 2013; Chu et al., 2014; Sun et al., 2014; Sumalatha et al., 2015), the number of ciphertext classes is predefined to some fixed value n. Therefore, to overcome this issue of the static value of n, KAE (Guo et al., 2017) has been proposed. The value of n was kept variable and an aggregate key was independent of the number of maximum possible ciphertext classes. The KAE (Guo et al., 2017) also allows the cloud server to check the authenticity of the user who requests access to an outsourced ciphertext. However, KAE does not allow the user to search for a keyword across the shared data. Moreover, the size of the aggregate key is two group elements instead of one group element in the existing methods.

The KAE methods proposed in Firdose and Rebekah (2015) and Sumalatha et al. (2015) improved the performance of KAE introduced by Chu et al. (2014) and provided efficient as well as secure solutions. Wang and Zhou (2016) offered the solution of a leakage-resilient KAE; i.e., the attacker cannot recover any information about the master secret key no matter how many bits of the aggregate key are leaked. However, their solution is not efficient and protects only the master secret key with leakage resiliency. Therefore, the construction of a more secure and efficient leakage-resilient KAE method remains an open problem. Recently, Patranabis et al. (2017) proposed two provably secure KAE methods that achieved chosen plaintext attack (CPA) security and chosen ciphertext attack (CCA) security, respectively. They also extended the basic KAE framework and combined it with broadcast encryption methods for distributing the aggregate key among an arbitrary number of data users in a real-life data-sharing environment. However, Patranabis et al. (2017) did not show how the data owner can revoke the delegated rights of selected users in case the same aggregate key is distributed among a group of users using broadcast encryption. Wang (2019) proposed a leakage resilient and provably secure KAE method based on the KAE constructed by Patranabis et al. (2017).

However, existing solutions for KAE (Singhal, 2001; Chu et al., 2014; Mahalle and Pawade, 2014; Chame and Kumar, 2015; Firdose and Rebekah, 2015; Patranabis et al., 2015, 2017; Dang et al., 2016; Wang and Zhou, 2016; Guo et al., 2017; Wang et al., 2017) do not support searching across encrypted data. If an authorized user needs to retrieve the encrypted data or file containing the given keywords, then he/she must carry out the following

steps: (1) Download the data; (2) Retrieve the plaintext through decryption; (3) Search for particular keywords. Because the number of files or quantity of data stored on the cloud is high, such retrieval, as well as the search process, becomes infeasible for real-time use. Therefore, to allow authorized users to perform efficient data retrieval, equipping an encryption method with keyword-search capability is required.

2.5 Searchable encryption

Data retrieval from terabytes of encrypted data stored on a cloud server is a challenging task. Typically, a user may want to selectively retrieve files instead of retrieving all the encrypted files for which he/she is authorized to access. For efficient data retrieval from encrypted data outsourced on the cloud, we require a keyword-based search technique known as searchable encryption (SE) (Song et al., 2000). Equipping an encryption method with keyword-search capability makes the authorized user's task easier such that an authorized user can efficiently search for an encrypted file having particular keywords by checking whether the label attached to the ciphertext and keywords associated with the user's trapdoor are matched or not. If both the keyword and the label are matched, then the user can retrieve and decrypt that particular file only, instead of downloading and decrypting a number of files available on the cloud.

As shown in Fig. 3, an SE cryptosystem consists of three parties, a cloud storage server, data owners (DO), and users (the individuals who search the encrypted data). DOs are the ones who own the data files and encrypt them before uploading. The user is the party who wants to execute a keyword search and obtain a result. The cloud storage server stores the data uploaded by a DO and performs a search on behalf of the user and then returns the result to the query generator. In short, an SE method allows a server to search encrypted data on behalf of a client without learning the information about the plaintext data. Currently, the state-of-the-art SE constructions achieve different trade-offs among security, efficiency, and query expressiveness.

Song et al. (2000) proposed for the first time a practical method for ciphertext retrieval using symmetric encryption. However, their method suffers from an efficiency issue because the communication

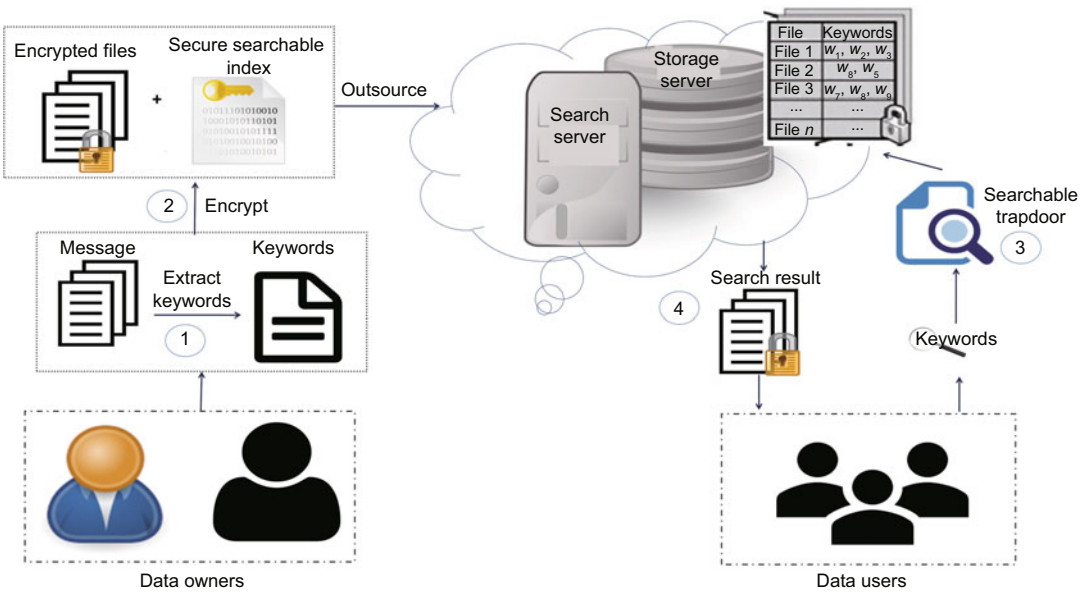

Fig. 3 System model of searchable encryption

Steps: (1) The data owner extracts keywords from the payload message; (2) The data owner generates the ciphertext (encrypted message) as well as a searchable encrypted index (encrypted keywords using a searchable secret key) and stores them on the cloud server; (3) The data user generates a searchable trapdoor using a query keyword and secret key, and then sends the generated trapdoor to the server; (4) The cloud server retrieves the files using the submitted trapdoor and sends the search results to the query requestor

overhead increases linearly with the file size. The SE method proposed in Song et al. (2000) leaks information related to the query keyword. Therefore, to provide keyword privacy, the SE methods (Goh, 2003; Chang and Mitzenmacher, 2005; Curtmola et al., 2011) provide efficient as well as secure solutions based on an encrypted index for searching across outsourced encrypted data. Boneh et al. (2004) addressed the problem of searching encrypted data and gave a solution based on a public-key encryption method. However, early works (Goh, 2003; Chang and Mitzenmacher, 2005; Yang et al., 2006; Hwang and Lee, 2007; Curtmola et al., 2011) support only a single-keyword search. In addition, the above-mentioned methods do not support access control on the encrypted data. To give access privileges and different search capabilities to multiple users, user authorization should be enforced.

2.6 Multi-user searchable encryption

Hwang and Lee (2007) proposed the first SE method based on public-key encryption in a multi-user setting. There are solutions in the literature to manage a group of users in SE methods (Wang et al., 2007, 2008a, 2008b; Bao et al., 2008; Huang et al., 2016). The method proposed in Wang et al. (2008b) supports a static group of users, whereas the methods discussed in Wang et al. (2007, 2008a) and Bao et al. (2008) manage a dynamic group of users. Huang et al. (2016) proposed an SE method based on an inverted index in a multi-user setting. However, their approach leaks the user access control information to the cloud server. To provide access control on searchable encrypted data, Wang et al. (2013) and Sun et al. (2014) proposed a searchable CP-ABE method that allows data owners to encrypt the index using an access policy, which determines who can search the index. Xiong et al. (2013) integrated the traditional CP-ABE method and a homomorphic encryption method to provide searchable ciphertext with CP-ABE access control. In the method proposed in Liao et al. (2013), a user can search for a keyword if and only if the user's attributes satisfy the access policy of the encrypted keyword. Zhang et al. (2013) used predicate encryption to achieve a fine-grained keyword search. The method also supports multi-keyword search with disjunction or conjunction queries across encrypted data. Padhya and Jinwala (2014) proposed a searchable CP-ABE with a hidden access policy and constant key length.

2.7 Multi-key searchable encryption

The multi-key searchable encryption (MKSE) approach proposed in Popa and Zeldovich (2013) allows the user to search for data encrypted under different encryption keys. It allows the user to submit a single search trapdoor to the cloud server, and the server then searches for the received query keyword across the stored documents encrypted under different keys. While searching, a server obtains only information about whether a given query keyword matches a word in a document or not.

2.8 Key-aggregate searchable encryption

The first solution for KASE was proposed by Cui et al. (2016). In a KASE method, the owner has to share only a single aggregate key with a user to allow search rights across a set of documents, and the user is required to submit only a single trapdoor for searching the dataset shared by the same owner. The main difference between the KASE concept and MKSE is that KASE allows data owners to delegate the right to a keyword search to a set of users by sharing the aggregate key with them, whereas the aim of MKSE is to ensure that the cloud server can carry out the keyword search using a single trapdoor across the different documents for which the user is authorized. Cui et al. (2016) proposed the novel idea of KASE for the first time in 2014. After that, the KASE schemes (Lambhate and Patil, 2016; Li et al., 2016; Pansare et al., 2016) were proposed to improve the performance of the one introduced by Cui et al. (2016). However, security analysis of chosen message attacks and chosen keyword attacks is not discussed in the existing methods (Cui et al., 2016; Li et al., 2016). Moreover, the existing KASE methods do not provide data and trapdoor privacy, as discussed in Kiayias et al. (2016). The KASE (Cui et al., 2016; Li et al., 2016) methods use BDHE for the security assumption. However, SDH or its variants such as BDHE do not guarantee the security of specific parameters, as discussed in Cheon (2006). Furthermore, when a user submits an aggregate trapdoor Tr to the cloud server to carry out a search across a set S ($|S| = m$) of files, the server first requires generating a trapdoor Tr_i for each index $i \in S$ using the submitted trapdoor Tr before searching. The

predefined bound on the maximum number of possible ciphertext classes at the time of system setup makes the KASE (Cui et al., 2016; Li et al., 2016) methods impractical for real-time use. Therefore, Kiayias et al. (2016) proposed a multi-user SE method with a constant size ciphertext and trapdoor. However, in their method, each secret key contains 13 group elements, each trapdoor contains 6 group elements for the single query, and each keyword ciphertext contains 7 elements. Other constructions for KASE methods were given in Lambhate and Patil (2016), Pansare et al. (2016), Li et al. (2018), and Zhou et al. (2018). Zhou et al. (2018) considered industrial IoT (IIoT) applications and proposed a KASE method in the file-centric framework for IIoT applications. They also described an attack on the KASE method (Cui et al., 2016), in which the attacker can guess the authorized user's key with the help of an inside adversary. However, this approach suffers from low performance. In this method, each user's public key has $(3n+1)$ elements and secret key has $(n+3)$ elements. Here, n is the maximum number of documents held by a data owner.

To the best of our knowledge, none of the existing KASE methods allow searching across multi-owner data with minimum overhead and the ability to prevent a chosen message attack, chosen keyword attack, and cross-pairing attack.

3 Preliminaries

In this section, we discuss basic assumptions and cryptographic concepts to understand the proposed MULKASE method.

3.1 Bilinear map

A bilinear map is a map $e: G\times G \to G_T$ with the following properties: (1) bilinearity—for all $u, v \in G$ and $a, b \in Z_p^*$, we have $e(u^a, v^b) = e(u,v)^{ab}$; (2) non-degeneracy—$e(g,g) \neq 1$, where g is a generator of group G; (3) computability—there is an efficient algorithm to compute $e(u,v)$ for any $u, v \in G$.

3.2 Computational assumption

The security assumption of MULKASE is DBDH (Definition 1). We use DBDH instead of the BDHE security assumption. The reason for selecting the DBDH assumption is that SDH and its variants are solvable in polynomial time assuming that $g, g^{\alpha}, g^{\alpha^d}$ for a positive divisor d of $p-1$ are known and secret $\alpha \in Z_p$, as proved by Cheon (2006).

Definition 1 (DBDH assumption) The DBDH problem is in group G of prime order p (according to the security parameter). For input of a tuple (g, g^a, g^b, g^c, R), where $a, b, c \in Z_p$ are chosen at random and g is a generator of G, decide $R = e(g,g)^{abc}$ or not. Algorithm A has advantage ϵ in solving a DBDH problem in G if $\mathrm{Adv}_{\mathrm{DBDH}}(A) = \left|\Pr\left(A(g, g^a, g^b, g^c, e(g,g)^{abc}) = 0\right) - \Pr\left(A(g, g^a, g^b, g^c, R) = 0\right)\right| \geq \epsilon(\kappa)$, where $e(g,g)^{abc} \in G_T$. We say that the DBDH assumption holds in G if no probabilistic polynomial time (PPT) algorithm has an advantage of at least ϵ in solving the DBDH problem in G.

3.3 Notations

The notations used throughout the paper are given in Table 1.

3.4 Problem statement

To simulate the search requirements across multi-owner data using a constant sized single

Table 1 Notations used for MULKASE terms

Term	Meaning
λ	Security parameter
n	Number of documents held by a data owner
B	Bilinear map group system
SP	System parameter
T^*	Keyword token
ks	Keyword space
PubK	Public parameters
pk	Public key
msk	Master secret key
doc_i	The ith document
k_i	The ith searchable secret key
i	Index of a document or a file
S	The set of indices of documents, $S \subseteq \{1, 2, \ldots, n\}$
C	Ciphertext
C_i	The ith ciphertext
C_{KW}	Ciphertext of keyword KW
C_{m}	Ciphertext of data (or message)
δ_i	Public information of the ith ciphertext (C_1 and C_2)
m	Message or plaintext
k_{agg}	Aggregate key
Tr	Trapdoor used to search query keyword w
R	Search result
w	Query keyword attached with a searchable trapdoor
KW	Keyword attached with a ciphertext
KV	Keyword value

searchable encryption key, consider the scenario of a personal health record (PHR) storage system. A PHR storage system allows patients to store their personal and health data on the cloud. It is expected that a PHR system securely stores and maintains patients' data on the cloud. The sources of a patient's data are diverse, including data transmitted from Internet-connected devices like heart monitoring implants connected to the patient's body or sensors deployed in the patient's living area. Patient controlled encryption (PCE) was also considered one of the applications of KASE in Chu et al. (2014).

In Figs. 4 and 5, we illustrate a scenario with only two patients named Alice and Bob to simplify an understanding of the idea. However, there are thousands of patients in real-world scenarios. Each patient is assumed to have multiple health records (HRs) denoted as $\{\mathrm{HR}_{\mathrm{Alice},i}\}_{i=1}^{m}$, $\{\mathrm{HR}_{\mathrm{Bob},j}\}_{j=1}^{n}$, and each record is encrypted under different encryption keys $\{k_{\mathrm{Alice},i}\}_{i=1}^{m}$, $\{k_{\mathrm{Bob},j}\}_{j=1}^{n}$, respectively. Here, m and n are the total numbers of health records for Alice and Bob, respectively. If Alice wants to delegate search and access rights to her set S ($|S| = x$) of HRs to a health-care professional, then KASE allows the data owner to share a single aggregate key with the user to delegate search rights across multiple files. If a health-care professional, say a doctor, wants to retrieve the HRs of Alice, i.e., $\{\mathrm{HR}_{\mathrm{Alice},i}\}_{i=1}^{x}$ (encrypted under different encryption keys $\{k_{\mathrm{Alice},i}\}_{i=1}^{x}$), then the doctor has to generate trapdoor $\mathrm{Tr}_{\mathrm{Alice}}$ using an aggregate key $k_{\mathrm{Alice,agg}}$. However, if a doctor wants to search all of his/her patients' records, containing the keyword "Disease = Malaria," then he/she has to send multiple trapdoors $\mathrm{Tr}_{\mathrm{Alice}}$, $\mathrm{Tr}_{\mathrm{Bob}}$, and so on, to the cloud. Therefore, as shown in Fig. 4, the user has to maintain multiple keys and has to generate multiple trapdoors to search over multi-owner data. Therefore, Li et al. (2016) proposed the KASE scheme to support multi-owner search using a single trapdoor and also provided verification of search results using an aggregate key. However, as shown in Fig. 4, in the existing methods (Cui et al., 2016; Li et al., 2016), when a user submits an aggregate trapdoor Tr to the cloud

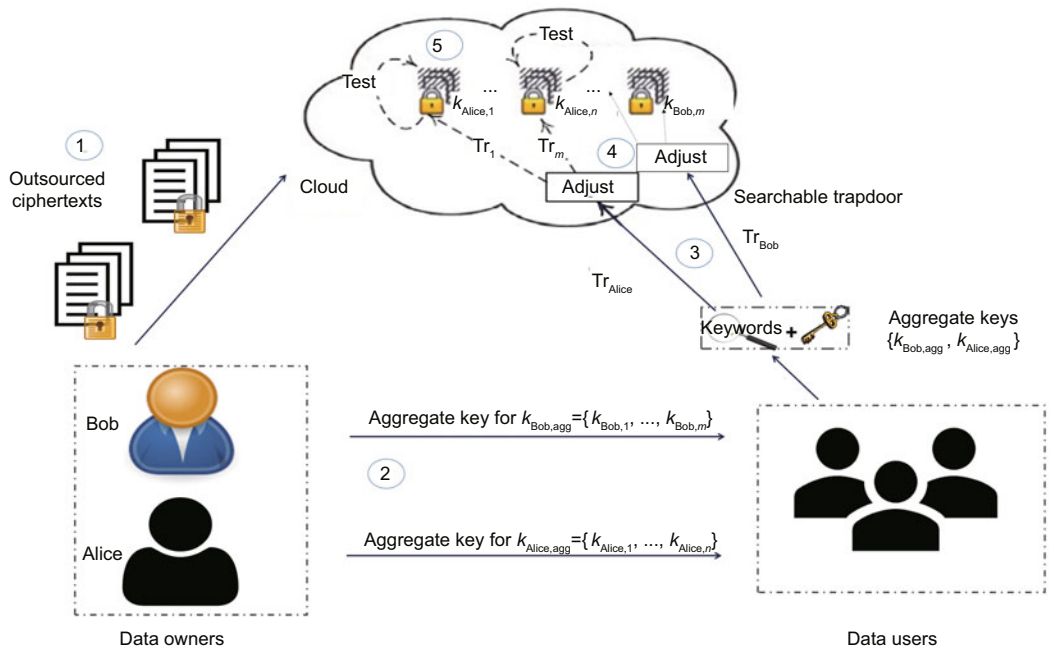

Fig. 4 Keyword search across the set of data owned by multiple users: the existing KASE approach

Steps: (1) Data owners (Alice, Bob, and other people) upload a ciphertext onto the cloud server; (2) Each data owner sends the aggregate key to the data user to share set S of documents; (3) The data user generates searchable trapdoors using each aggregate key received from the different data owners and a query keyword, and then submits all the trapdoors to the server; (4) The cloud server transforms each input aggregate trapdoor Tr to several Tr_i for each $i \in S$ (each Tr_i is an actual trapdoor to perform a keyword search across the ith document); (5) The cloud server checks whether the ith document contains the query keyword or not using the individual Tr_i

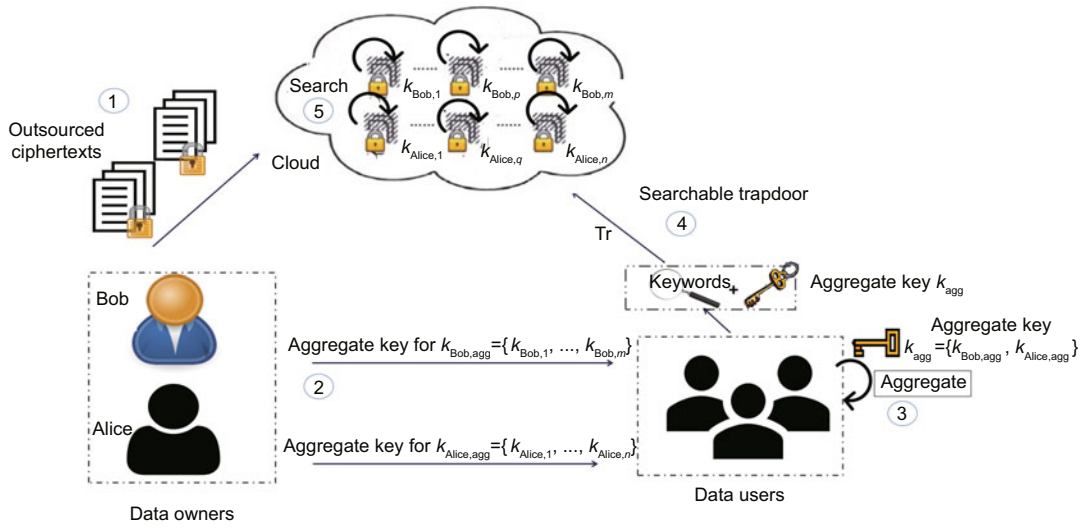

Fig. 5 Keyword search across the set of data owned by multiple users: the MULKASE approach

Steps: (1) Data owners (Alice, Bob, and other people) upload a ciphertext onto the cloud server; (2) Each data owner sends the aggregate key to the data user to share set S of documents; (3) The data user generates a single aggregate key using aggregate keys received from the different data owners; (4) The data user constructs a searchable trapdoor using an aggregate key and a query keyword, and then submits a trapdoor to the server; (5) The cloud server checks whether the ith document contains the query keyword or not using Tr

server to carry out a search across m files, the server first requires the generation of a trapdoor Tr_i for each index $i \in S$ using submitted trapdoor Tr before the search. Here, S is a set of ciphertext indices. The process of generating individual trapdoors from the aggregate trapdoor at the time of keyword search adds an extra computational overhead of $O(|S|)$ on the server side. Therefore, we require an encryption method that allows an authorized user to search and access the set of data owned by multiple users using a single aggregate key with the minimum computation and storage overhead.

As shown in Fig. 5, the MULKASE method allows a doctor (or an authorized user) to perform a keyword search across all the shared data $\{\text{HR}_{\text{Alice},i}\}_{i=1}^{x}$, $\{\text{HR}_{\text{Bob},j}\}_{j=1}^{n}$ using a single trapdoor even though the data are owned by the different patients, Alice and Bob. Using the submitted trapdoor, the cloud server carries out the search across the shared documents using the aggregate trapdoor only (without requiring the generation of individual trapdoors Tr_i from the aggregate trapdoor).

3.5 MULKASE framework

The MULKASE framework consists of the following algorithms:

1. Setup(1^λ, n)

The cloud service provider runs this algorithm. It takes the following two parameters as input: (1) security parameter λ; (2) the number of documents held by a data owner (n). The algorithm outputs the system parameters (SP).

2. Keygen

The data owner runs this algorithm to produce a public/master-secret key pair (pk, msk).

3. Encrypt(pk, i, m, KW)

The data owner encrypts the ith document and keyword KW using this algorithm and produces its data-ciphertext, keyword-ciphertext, and public information (δ_i), i.e., $C_i = (\delta_i, C_m, C_{\text{KW}})$. It takes the following four parameters as input: (1) a public key pk; (2) an index i denoting the ciphertext class; (3) message m; (4) keyword KW. The algorithm outputs ciphertexts of data as well as a keyword. The data owner outsources ciphertext C_i on the cloud server.

4. Extract(msk, S)

To delegate the keyword search and data access rights for a selected group of documents to a selected group of users, the data owner runs this algorithm to produce an aggregate searchable encryption key. It takes the following two parameters as input: (1) the data owner's master-secret key msk; (2) a set S of document indices. The algorithm outputs the

aggregate key k_{agg}.

5. Aggregate($k_{\text{agg1}}, k_{\text{agg2}}, \ldots, k_{\text{agg}k}, S_1, S_2, \ldots, S_k$)

This algorithm generates a single aggregate key, when an authorized user receives multiple aggregate keys shared by different data owners. When given input of aggregate keys $k_{\text{agg1}}, k_{\text{agg2}}, \ldots, k_{\text{agg}k}$ (shared by k different data owners) and sets of document indices $S_1, S_2, \ldots, S_k$, this algorithm outputs a single aggregate key k_{agg}. The resultant aggregate key k_{agg} contains the rights to access and search across all the documents in the range of set $S = S_1 \cup S_2 \cup \ldots \cup S_k$.

6. Trapdoor(k_{agg}, w)

An authorized user generates a keyword trapdoor using this algorithm. It takes as input the aggregate searchable encryption key k_{agg} and query keyword w. The algorithm outputs an aggregate single trapdoor Tr.

7. Test(Tr, S, i)

The cloud server runs this algorithm to perform a keyword search over an encrypted document by giving the trapdoor Tr, the set S of document indices, and the document index i as inputs. If $i \in S$, then this algorithm outputs true or false to denote whether the keyword KW attached to ciphertext C_i is the same as the query keyword w or not.

8. Decrypt($k_{\text{agg}}, S, i, C_i$)

An authorized user runs this algorithm to access the retrieved documents using an aggregate key k_{agg} given by the data owner. The inputs to the algorithm are as follows: (1) aggregate key k_{agg}; (2) the set S; (3) an index i denoting the ciphertext class of C_i; (4) ciphertext C_i. The algorithm outputs the decrypted result m if $i \in S$.

3.6 Security and functional goals

The proposed MULKASE method aims to achieve the following security and functional objectives:

1. Compactness

The size of the aggregate key in the MULKASE method should be independent of the number of files that are within its range. The aggregate key is a single secret key that aggregates the keyword search and data decryption rights of a set of shared ciphertexts. Moreover, when the data owner wants to delegate access and search rights to all the files classified under different classes, the size of the aggregate key should be independent of the number of ciphertext classes. Similarly, the size of the ciphertext should be constant.

2. Correctness

For any message m containing query keyword w, if $C \leftarrow$ Encrypt(pk, i, m, KW) and Tr=Trapdoor(k_{agg}, w), then 1←Test(Tr, S, i).

3. Delegation

Another goal of our proposed method is the delegation of search and decryption rights of any number of shared ciphertexts using a single aggregate key. The user having an aggregate key can perform both search and decryption on the set of shared ciphertexts.

4. Query privacy

The MULKASE method must not leak any information about the query keyword to any third party or server. The user sends a query trapdoor to the server to search for a keyword, but the server must not acquire any information about the query keyword.

5. Controlled searching

Attackers or unauthorized users are not allowed to perform a keyword search without the data owner's authorization. Specifically, one cannot carry out a keyword search on a set of documents that are not within the range of the known aggregate searchable encryption key. In addition, one cannot create new aggregate keys for another set of documents from the known one.

6. Collision resistance

The MULKASE method must be collision-resistant. A collision attack is carried out by combining multiple aggregate keys to obtain more information than each aggregate key individually contains. For example, if a user (say a doctor) has the aggregate key k_{agg1} to decrypt X-ray reports of patients {"A," "B," "C"} and another aggregate key k_{agg2} to decrypt HIV reports of patients {"D," "E"}, then he/she cannot access other medical reports (by combining k_{agg1} and k_{agg2}).

3.7 System model

Fig. 6 illustrates the MULKASE system model. The system model consists of three entities: data owners, data users, and cloud servers. The characteristics of each entity are discussed below:

1. Data owners

Data owners are the users who want to outsource large amounts of data (documents/files) $D =$

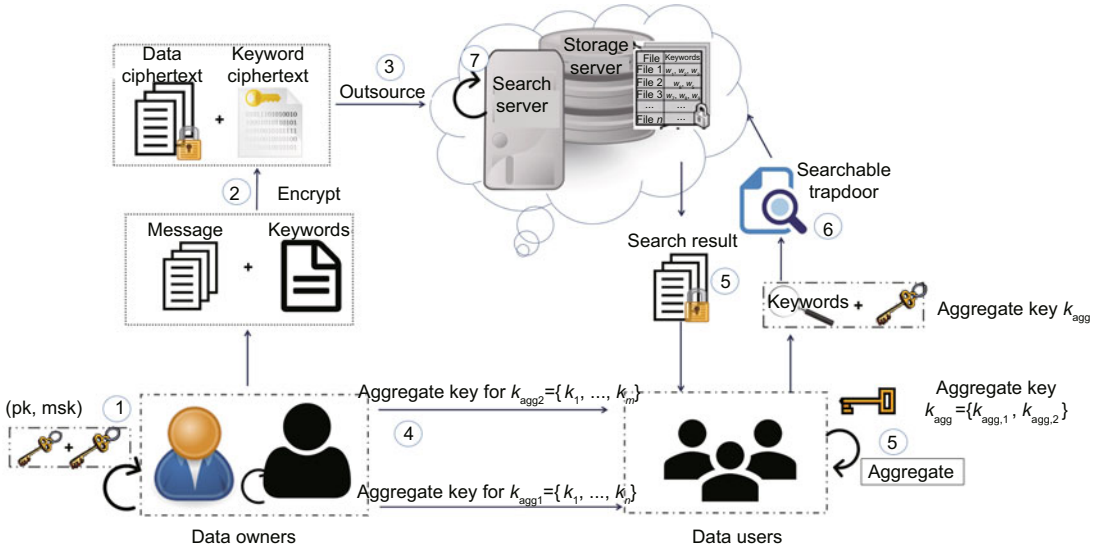

Fig. 6 System model of MULKASE

Steps: (1) The cloud server sets up the system's public parameters and the data owner generates a key pair (public, master-secret); (2) Data owners encrypt the payload message (m) along with the related keyword (KW) using the public key (pk) and searchable encryption key (k_i), and generate the data-ciphertext, keyword-ciphertext, and public information (δ_i) ($C_i = (\delta_i, C_m, C_{\mathrm{KW}})$); (3) Data owners upload ciphertexts (C_i) onto the cloud server; (4) Each data owner sends the aggregate key (k_{agg}) to the data user for sharing set S of their documents; (5) The data user generates a single aggregate key (k_{agg}) using each aggregate key ($k_{\mathrm{agg},1}, k_{\mathrm{agg},2}, \ldots, k_{\mathrm{agg},m}$) received from the different data owners; (6) The data user constructs a searchable trapdoor Tr using an aggregate key (k_{agg}) and a query keyword (w), and submits a trapdoor to the server; (7) The cloud server runs the proposed Test($\cdot$) algorithm to check whether the ith document contains the query keyword (w) or not using the submitted trapdoor Tr

$\{\mathrm{doc}_1, \mathrm{doc}_2, \ldots, \mathrm{doc}_n\}$ to the cloud servers. The data owner encrypts data D along with the related keyword KW before outsourcing the data on the cloud server. Using the public key pk and searchable encryption key k_i for $i = \{1, 2, \ldots, n\}$, the data owner generates $D' = C_i = (\delta_i, C_m, C_{\mathrm{KW}}) = \mathrm{Encrypt}_{k_i}(\mathrm{pk}, i, \mathrm{doc}_i, \mathrm{KW})$, $\forall i \in \{1, 2, \ldots, n\}$. The data owner generates a single aggregate key k_{agg} from the searchable encryption keys $\{k_1, k_2, \ldots, k_n\}$. The data owner then shares an aggregate searchable secret key k_{agg} with the users to delegate search and data access rights to the documents $\{\mathrm{doc}_1, \mathrm{doc}_2, \ldots, \mathrm{doc}_n\}$.

2. Data users

An authorized user who wants to search for a keyword w across the shared dataset D generates a corresponding aggregate trapdoor Tr using an aggregate key k_{agg}. Using the aggregate trapdoor Tr, the user can efficiently retrieve selected documents that contain the given query keyword w. The user submits the trapdoor Tr to the cloud server.

3. Cloud servers

Cloud servers provide massive storage and computation resources to perform the keyword search query. Initially, the cloud server sets up the system's public parameters. The cloud server uses the submitted trapdoor and public information to carry out a keyword search across the set of encrypted data for which the user is authorized and returns the search results.

3.8 Threat model

We consider cloud servers to be "honest but curious," which is consistent with most related works on key-aggregate encryption (Chu et al., 2014; Cui et al., 2016; Li et al., 2016). In addition, the data user is curious and may try to obtain additional information beyond his/her query about the data or other users' privacy. However, the data user's capabilities within the system are limited by both storage space and computing power. Moreover, communication channels involving the server are assumed to be insecure. We consider the following types of attacks: (1) An attacker may try to learn a query keyword from the submitted trapdoor; (2) An attacker

can try to learn a keyword in a document from the stored keyword-ciphertexts and the relevant public information; (3) An authorized user or server can try to perform the keyword search across any set of documents not within the range of the user's aggregate key; (4) The collision of a malicious authorized user with the cloud server is possible; (5) An attacker can try to generate a new aggregate key for any new set S' of indices from the known aggregate key.

However, the security analysis discussed in Section 5.1 will show that the proposed method is secure against the above-mentioned attacks.

4 The proposed method: MULKASE

4.1 Construction of MULKASE

Based on the framework described in Section 3.5, we propose the construction of MULKASE as follows:

1. Setup(1^λ, n)

The cloud server runs this algorithm to initialize and publish the system parameters SP=(B, PubK, T^*). The details of each system parameter are as given below: (1) Bilinear map group system B=$(p, G, G_T, e(\cdot))$, where p is the order of G and $2^\lambda \le p \le 2^{\lambda+1}$. (2) PubK=($g$, g_1, …, g_n) $\in G_{n+1}$, where n is the number of documents D= $\{\mathrm{doc}_1, \mathrm{doc}_2, \ldots, \mathrm{doc}_n\}$ that belong to a data owner. If the data owner uploads one more document onto the cloud, then n increases by one and one public parameter g_{n+2} is added into PubK without affecting the other system parameters. Therefore, PubK is updated to (g, g_1, … , g_n, g_{n+1}). In a similar way, when the number of ciphertext classes increases to $n+m$, then PubK is updated to (g_1, g_2, …, g_n, g_{n+2}, g_{n+3}, …, g_{n+m}). Here, m is the number of documents the data owner uploads beyond the maximum number of ciphertext classes n. As shown in Fig. 7, if the number of outsourced ciphertexts goes beyond the predefined limit of n, then MULKASE allows the data owner to classify ciphertexts under more than n different classes. Moreover, by sharing a single aggregate key, the data owner can delegate search and access rights to the set of ciphertexts classified under different classes. Therefore, our method is not restricted to a predefined value of the maximum number of possible ciphertexts. $g_i = g^{\alpha^i} \in G$ for $i = \{1, 2, \ldots, n\}$, for a random generator $g \in G$ and a random $\alpha \in Z_p$. (3) The cloud server defines the hash functions to map the given keyword with a unique value. (4) Assume that there are m categories of keywords in the system. Therefore, keyword space ks=$\{w_1, w_2, \ldots, w_m\}$ and KV_i=$\{\mathrm{KV}_1, \mathrm{KV}_2, \ldots, \mathrm{KV}_m\}$ for $1 < i < m$, for the set of all possible values of keywords $\{w_1, w_2, \ldots, w_m\}$. (5) $H(\mathrm{KV}_i)$. Given a keyword w_i having value KV_i, the hash function proceeds as follows: If $w_i = \mathrm{KV}_i$ has not been queried before, then select a random value $\mathrm{kv}_i \in_R Z_p$ and compute $T_i^* = g^{\mathrm{kv}_i}$. Add the tuple $< w_i, \mathrm{KV}_i, \mathrm{kv}_i, T_i^* >$ to table T_{kw} and return T_i^*.

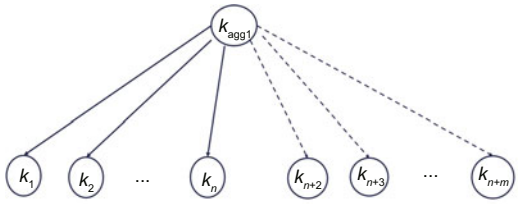

Fig. 7 Classification of ciphertexts under more than n classes in MULKASE

2. Keygen

The data owner runs this algorithm to generate a key pair (pk, msk), pk=v=g^γ, msk=γ, where $\gamma \in Z_p$.

3. Encrypt(pk, i, m, KW)

To upload the ith document onto the cloud, the data owner encrypts it and generates its data-ciphertext as well as the keyword-ciphertext by following the steps shown below: (1) Set the searchable encryption key k_i of the ith document as k_i= t, where $t \in Z_p$. (2) Compute the public information $\delta_i = (C_1, C_2)$ for k_i, $C_1 = g^t$, C_2=$e(v, g_i)$=$e(g^\gamma, g_i)$. (3) For a keyword KW, the algorithm outputs the keyword-ciphertext C_{KW} as C_3=C_{KW}=$(g^{\mathrm{kv}_i})^t$. (4) To generate the data-ciphertext for a message m=doc_i, the algorithm computes $C_4 = C_m$=$m \cdot e(g_1, g_n)^t$. The term $g_{n+1} = g^{\alpha^{n+1}}$ is missing in the public parameters PubK. Therefore, an attacker cannot compute the value of $e(C_1, g_{n+1})$ to obtain the value of $e(g_1, g_n)^t$. An attacker cannot obtain any information related to message m. The data owner stores the ciphertext $C_i = (\delta_i, C_{\mathrm{KW}}, C_m) = (C_1, C_2, C_3, C_4)$ on the cloud server.

4. Extract(msk, S)

The data owner runs this algorithm to generate an aggregate searchable encryption key. When given a subset $S \subseteq \{1, 2, \ldots, n\}$ of the document

indices and the owner's master-secret key msk as inputs, the algorithm outputs an aggregate key k_{agg} by computing $k_{\text{agg}} = \pi_{j\in S} g_j^{\gamma}$. The data owner securely sends k_{agg} and a set S to the user, to delegate the keyword search and data access rights to the ciphertexts within the range of set S.

5. Aggregate($k_{\text{agg1}}, k_{\text{agg2}}, \ldots, k_{\text{agg}k}, S_1, S_2, \ldots, S_k$)

An authorized user generates a single aggregate key by aggregating multiple aggregate keys shared by different data owners. When given an input of aggregate keys $\{k_{\text{agg1}}, k_{\text{agg2}}, \ldots, k_{\text{agg}k}\}$ (shared by k different data owners) and a set of document indices $\{S_1, S_2, \ldots, S_k\}$, this algorithm outputs a single aggregate key k_{agg} that contains the rights to access as well as search across all the documents within the range of set $S = S_1 \cup S_2 \cup \ldots \cup S_k$. Assume that an authorized user has aggregate key k_{agg1} for the set of document indices S_1, an aggregate key k_{agg2} for set S_2, and so on. Moreover, keys $k_{\text{agg1}}, k_{\text{agg2}}, \ldots, k_{\text{agg}k}$ are generated using master-secret key msk=$\{\gamma_1, \gamma_2, \ldots, \gamma_k\}$ respectively. The Aggregate($\cdot$) algorithm generates a single aggregate key k_{agg} by computing $k_{\text{agg}} = k_{\text{agg1}} \cdot k_{\text{agg2}} \cdot \ldots = \pi_{j\in S_1} g_j^{\gamma_1} \cdot \pi_{j\in S_2} g_j^{\gamma_2} \cdot \ldots = \pi_{j\in S, i\in\{1,2,\ldots,k\}} g_j^{\gamma_i} = k_{\text{agg}}$.

6. Trapdoor(k_{agg}, w)

The user who wants to perform a keyword search across the shared data runs this algorithm and generates a single aggregate trapdoor Tr. If the set of documents S is in the range of an aggregate key k_{agg}, then the user having trapdoor Tr can search over any document in the range of set S. This algorithm computes a searchable trapdoor for the given keyword w: $\text{Tr} = (\text{Tr}_0, \text{Tr}_1) = (k_{\text{agg}} \cdot (g^{\text{kv}_i})^b, g^b)$, where $b \in Z_p$. Then the user submits (Tr, S) to the cloud server.

7. Test(Tr, S, i)

On receiving the trapdoor $\text{Tr} = (\text{Tr}_0, \text{Tr}_1)$ from the query requestor, the cloud server runs this algorithm to perform a search for a keyword over the ith document. If $i \notin S$, the Test($\cdot$) algorithm outputs NULL. Otherwise, it returns the search result $R \in \{0, 1\}$ by following the rules: $R = 1$ if document doc_i contains keyword w; otherwise, $R = 0$. To check if document doc_i contains the query keyword w or not, the cloud server follows the steps shown below: (1) The cloud server calculates $C' = \pi_{j\in S} C_{2,j} = \pi_{j\in S} e(v, g_j)$. To search across multi-owner data using an aggregate key k_{agg} (which comprises the power of multiple aggregate keys $\{k_{\text{agg1}}, k_{\text{agg2}}, \ldots, k_{\text{agg}k}\}$ shared by k different data owners), the cloud server generates C' using $C_{2,j}|\ \forall j \in S$ where $S = S_1 \cup S_2 \cup \ldots \cup S_k$. For efficiency consideration, C' for set S is computed only once. (2) Using public information δ_i=(C_1, C_2) as well as searchable ciphertext C_{KW} related to searchable encryption k_i and subset S, the Test($\cdot$) algorithm outputs true or false by judging 0 or $1 = ? = e(C', C_1) \cdot e(C_3, \text{Tr}_1)/e(\text{Tr}_0, C_1)$.

8. Decrypt(k_{agg}, S, i, C)

If $i \notin S$, the Decrypt($\cdot$) algorithm outputs NULL. Otherwise, it returns the message. On receiving the access request from an authorized user, the cloud server follows the steps shown below: (1) The cloud server calculates $C' = \pi_{j\in S} C_{2,j} = \pi_{j\in S} e(v, g_j)$. (2) The decryption algorithm outputs $m = C_4 e(\text{k}_{\text{agg}} \pi_{j\in S, j\neq n} g_{j+1}, C_1)/e(\text{pub}\cdot C', C_1)$, where $\text{pub} = \pi_{j\in S} g_{j+1}$.

For correctness of the Decrypt($\cdot$) algorithm, consider the scenario as discussed below: Suppose that Bob has an aggregate key k_{agg} that is generated by aggregating m different aggregate keys $\{k_{\text{agg1}}, k_{\text{agg2}}, \ldots, k_{\text{agg}m}\}$ shared by different data owners. Therefore, an aggregate key k_{agg} comprises the rights to access m different data owners' selected shared documents in the range of set $S = S_1 \cup S_2 \cup \ldots \cup S_m$. When given an input of aggregate key k_{agg} with a set of indices S, the Decrypt($\cdot$) algorithm first computes $C' = \pi_{j\in S}\, C_{2,j} = \pi_{j\in S, i\in\{1,2,\ldots,m\}} e(g^{\gamma_i}, g_j)$ and outputs

$$
\begin{aligned}
& C_4 \cdot e(k_{\text{agg}} \pi_{j\in S, j\neq n} g_{j+1}, C_1)/e(\text{pub}\cdot C', C_1) \\
&= \frac{C_4 \cdot e(k_{\text{agg}} \cdot \pi_{j\in S, j\neq n} g_{j+1}, g^t)}{e(\pi_{j\in S} g_{j+1} \cdot \pi_{j\in S,\ i\in\{1,2,\ldots,m\}}\ e(g^{\gamma_i}, g_j), g^t)} \\
&= \frac{C_4 \cdot e(k_{\text{agg}}, g^t) \cdot e(\pi_{j\in S, j\neq n} g_{j+1}, g^t)}{e(\pi_{j\in S} g_{j+1}, g^t) \cdot e(\pi_{j\in S,\ i\in\{1,2,\ldots,m\}}\ e(g^{\gamma_i}, g_j), g^t)} \\
&= \frac{m \cdot e(g_1, g_n)^t \cdot \frac{e(\pi_{j\in S} g_{j+1}, g^t)}{e(g_{n+1}, g^t)}}{e(\pi_{j\in S} g_{j+1}, g^t)} \\
&= \frac{m \cdot e(g_1, g_n)^t}{e(g_n, g_1)^t} \\
&= m.
\end{aligned}
$$

4.2 Description of the method

4.2.1 Scenario 1: search across a single data owner's data

Suppose a user, say Alice, wants to store her personal data (documents/files) $D = \{\text{doc}_1,$

$\mathrm{doc}_2, \ldots, \mathrm{doc}_m\}$ on the cloud server. Alice encrypts her document doc_i under key k_i for $i = \{1, 2, \ldots, m\}$. Alice stores public information called δ_i related to k_i along with data-ciphertext and keyword-ciphertext C_i on the cloud server. Alice delegates search rights of all $|S| = m$ documents to Bob by generating and sharing a single searchable aggregate key k_{agg} with Bob. If Bob wants to search for a keyword w across all the documents shared by Alice, he generates a trapdoor for keyword w using aggregate key k_{agg} and submits it to the cloud server. The cloud server uses stored public information δ_i and an aggregate trapdoor Tr to search for a keyword w across all the data encrypted under $\{k_1, k_2, \ldots, k_m\}$ while receiving only one trapdoor from Bob. In KASE methods (Cui et al., 2016; Li et al., 2016), when a user submits an aggregate trapdoor Tr to carry out a search, the server first requires generation of a trapdoor Tr_i for each index $i \in S$ using the submitted trapdoor Tr before searching. The process of generating individual trapdoors from the aggregate trapdoor at the time of a keyword search adds an extra overhead on the server side. However, we overcome this overhead on the server side and allow the server to search using the submitted aggregate trapdoor only (without requiring generation of individual trapdoors Tr_i from the aggregate trapdoor).

4.2.2 Scenario 2: search across the data of multiple owners

Consider the PHR example discussed in Section 3.4 and shown in Fig. 5. Suppose that patients Alice, Bob, and Carol have multiple HRs denoted as $\{\mathrm{HR}_{\mathrm{Alice},x}\}_{x=1}^m$, $\{\mathrm{HR}_{\mathrm{Bob},y}\}_{y=1}^n$, and $\{\mathrm{HR}_{\mathrm{Carol},z}\}_{z=1}^k$, respectively. The patients (Alice, Bob, and Carol) first encrypt the PHR records using searchable encryption keys denoted as $\{k_{\mathrm{Alice},x}\}_{x=1}^m$, $\{k_{\mathrm{Bob},y}\}_{y=1}^n$, and $\{k_{\mathrm{Carol},z}\}_{z=1}^k$, respectively. Alice wants to delegate search and access rights to her set $S(|S| = p)$ of HRs to a health-care professional. Therefore, Alice generates an aggregate key $k_{\mathrm{Alice,agg}}$ to delegate search and access rights on a selected set of PHR records $S_{\mathrm{Alice}} = \{1, 2, \ldots, p\}$ out of her m records. Similarly, Bob and Carol generate keys $k_{\mathrm{Bob,agg}}$ and $k_{\mathrm{Carol,agg}}$ that comprise rights to access and search across the set of PHR records $S_{\mathrm{Bob}} = \{1, 2, \ldots, q\}$ and $S_{\mathrm{Carol}} = \{1, 2, \ldots, r\}$, respectively. Alice, Bob, and Carol share their keys $k_{\mathrm{Alice,agg}}$, $k_{\mathrm{Bob,agg}}$, and $k_{\mathrm{Carol,agg}}$ with the doctor. On receiving the aggregate keys, the doctor first generates a single aggregate key k_{agg} by aggregating multiple keys ($k_{\mathrm{Alice,agg}}$, $k_{\mathrm{Bob,agg}}$, $k_{\mathrm{Carol,agg}}$) using the MULKASE Aggregate(·) algorithm shown below:

$$\begin{aligned}
&\mathrm{Aggregate}(k_{\mathrm{Alice,agg}}, k_{\mathrm{Bob,agg}}, k_{\mathrm{Carol,agg}}, S_{\mathrm{Alice}},\\
&S_{\mathrm{Bob}}, S_{\mathrm{Carol}}) = k_{\mathrm{Alice,agg}} \cdot k_{\mathrm{Bob,agg}} \cdot k_{\mathrm{Carol,agg}}\\
&= \pi_{j \in S_{\mathrm{Alice}}} g_j^{\gamma_1} \cdot \pi_{j \in S_{\mathrm{Bob}}} g_j^{\gamma_2} \cdot \pi_{j \in S_{\mathrm{Carol}}} g_j^{\gamma_3}\\
&= \pi_{j \in S, i \in \{1,2,3\}} g_j^{\gamma_i}\\
&= k_{\mathrm{agg}},
\end{aligned}$$

where $S = S_{\mathrm{Alice}} \cup S_{\mathrm{Bob}} \cup S_{\mathrm{Carol}}$.

Using the resultant single aggregate key k_{agg}, the doctor can access as well as search all the PHRs for which Alice, Bob, and Carol have individually delegated search rights. Suppose that a doctor wants to search for keyword w in all the documents shared by Alice, Bob, and Carol. He/She generates a trapdoor for the word w using aggregate key k_{agg} and submits it to the cloud server. The cloud server uses public information and the aggregate trapdoor Tr to search for the word w in all the data encrypted under $\{k_{\mathrm{Alice},x}\}_{x=1}^m$, $\{k_{\mathrm{Bob},y}\}_{y=1}^n$, and $\{k_{\mathrm{Carol},z}\}_{z=1}^k$ while receiving only one trapdoor from the doctor. Furthermore, the cloud server sends the search results to the doctor. The doctor can decrypt the retrieved PHRs using an aggregate key k_{agg}.

5 Security and theoretical analysis

In this section, we first describe MULKASE security analysis. Then we provide a theoretical analysis of the proposed method.

5.1 Security analysis

To analyze the security of our method and especially to show that the method is secure against the defined threat model and also satisfies the security requirements, we define and prove its security against an active attacker A using the security game shown in Figs. 8 and 9. We define the security game between the challenger and the attacker. We give the proof in a generic group model using the DBDH hardness assumption. We also prove the security of our method in terms of controlled searching and query privacy.

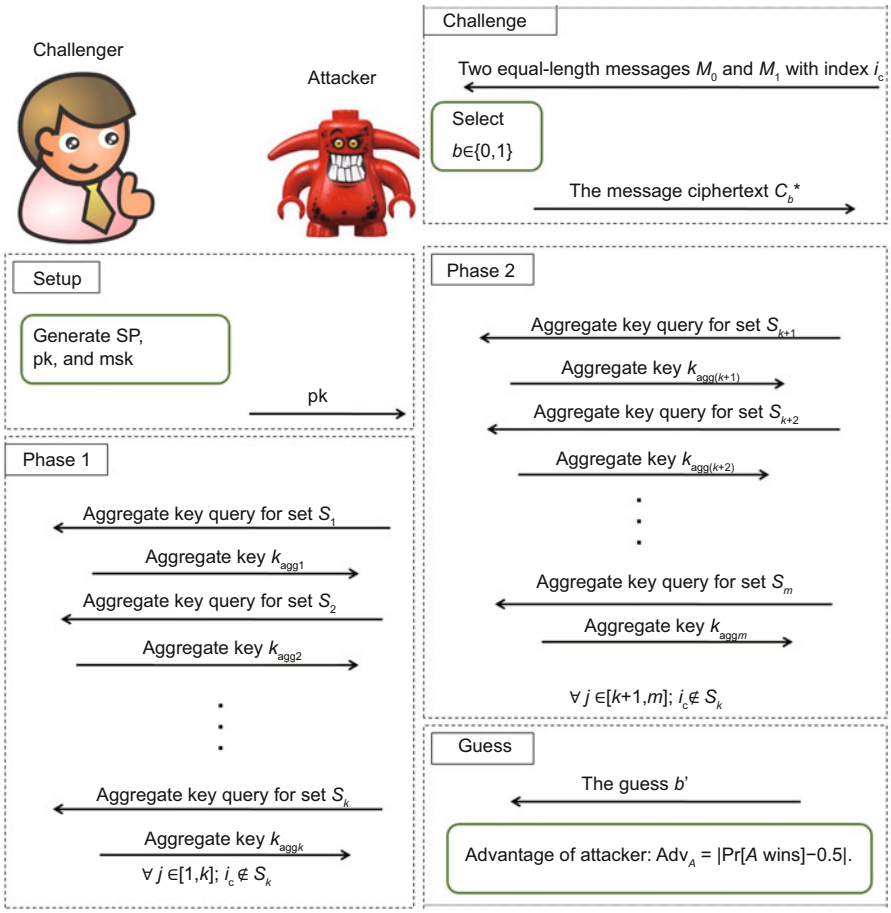

Fig. 8 IND-CPA security game for the MULKASE method

Steps: (1) Setup phase. The challenger runs Setup(1^λ, n) to obtain a public key pk and a master-secret key msk. The challenger issues public key pk to adversary A. (2) Query phase 1. Adversary A adaptively queries the oracle to generate an aggregate key corresponding to set S of the document indices of his/her choice. The oracle answers with an aggregate key k_{agg} for a given set S. (3) Challenge phase. Attacker A sends the challenger two equal-length messages M_0 and M_1 on which the attacker wishes to be challenged with a set of keywords and an index i_c. The only restriction is that the attacker should not have previously asked for the aggregate key corresponding to a set S which includes an index i_c. The challenger chooses b randomly from {0,1} and runs Encrypt(pk, i_c, M_b, KW). The challenger returns ciphertext C_b^* to the adversary. (4) Query phase 2. Identical to that in phase 1, the adversary asks for the aggregate key for a set S of his/her choice but he/she cannot ask for the aggregate key corresponding to a set S that includes an index i_c and a Decrypt query on C_b^*. (5) Guess. Adversary A outputs its guess $b' \in \{0,1\}$ for b and wins the game if $b = b'$

5.1.1 Security definition: IND-CPA game

We define IND-CPA for the proposed MULKASE method. The KASE methods do not prove their security against an IND-CPA attack (Cui et al., 2016; Li et al., 2016). The game proceeds as shown in Fig. 8.

The advantage of the adversary in this game is defined as $|\Pr[b = b'] - \frac{1}{2}|$, where the probability is taken over the random bits used by the challenger and the adversary.

Definition 2 The proposed MULKASE method is said to be semantically secure if $\text{Adv}_A < \epsilon$ is negligible with respect to the security parameter for any polynomial time adversary.

Theorem 1 The proposed MULKASE method is IND-CPA secure under the DBDH assumption, if there is no polynomial time adversary A who can win the game with a non-negligible advantage $\text{Adv}_A(l)$ in security parameter l.

Proof We consider a challenger C, a simulator SIM, and a polynomial time adversary A. We assume that adversary A has a non-negligible

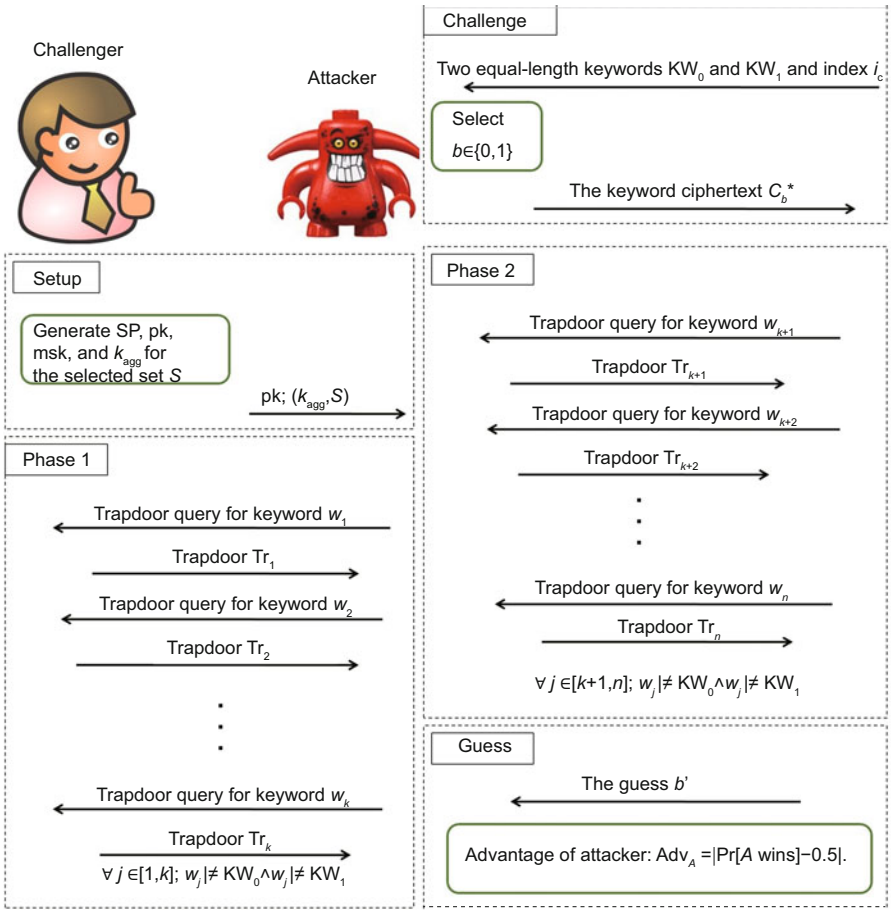

Fig. 9 IND-CKA security game for the MULKASE method

Steps: (1) Setup phase. The challenger runs Setup(1^λ, n) to obtain a public key pk and a master-secret key msk. The challenger issues public key pk and an aggregate key k_{agg} to adversary A to delegate search rights over a selected set of ciphertexts C_i where $i \in S$. (2) Query phase 1. Adversary A adaptively queries the oracle using an aggregate key k_{agg} to generate a trapdoor corresponding to keyword w of his/her choice. The oracle answers with a trapdoor Tr for querying a keyword. (3) Challenge phase. Attacker A sends the challenger two keywords KW_0 and KW_1 on which the attacker wishes to be challenged and an index i_c where $i_c \in S$. The only restriction is that the attacker should not have previously asked for the trapdoor corresponding to a keyword KW_0 or KW_1. The challenger chooses b randomly from {0,1} and runs Encrypt(pk, i_c, M, KW_b). The challenger returns keyword-ciphertext C_b^* to the adversary. (4) Query phase 2. Identical to that in phase 1, the adversary asks for the trapdoor for the keyword of his/her choice but he/she cannot ask for the trapdoor corresponding to a keyword KW_0 or KW_1 and Test(·) query on ciphertext C_b^*. (5) Guess. Adversary A outputs its guess $b' \in \{0,1\}$ for b and wins the game if $b = b'$

advantage $\epsilon(l)$ to break the privacy of our method. Then we can construct simulator SIM that breaks the decisional DBDH problem $\varsigma = (g,\ g^a,\ g^b, g^c,\ R)$ with advantage $\epsilon(l)/2$. The simulation proceeds as follows:

On DBDH input (g, g^a, g^b, g^c, R), simulator SIM aims to decide if $R = e(g,g)^{abc}$. Challenger C generates $a, b, c, z \in_R Z_p$, bilinear groups G, G_T with prime order p and the mapping $G \times G \to G_T$. Here, g is a generator of group G. Challenger C computes v as follows:

$$R = \begin{cases} e(g,g)^{abc}, & v = 0, \\ e(g,g)^{z}, & v = 1. \end{cases}$$

The challenger gives instance $(g,\ g^a,\ g^b,\ g^c,\ R) \in G_T$ to simulator SIM. SIM interacts with A (SIM simulates C for A) and starts the simulation as follows:

1. Setup

Simulator SIM generates system parameters SP$=(g,\ g_1, \ldots, g_n)$. Here, $g_i = (g^a)^{\alpha^i} \in G$ for

$i = \{1, 2, \ldots, n\}$. Moreover, SIM simulates the hash oracles for keyword, $O_{\mathrm{H}}(\mathrm{KV})_i$. Given keyword w_i, having value KV_i, the hash function proceeds as follows: (1) If $w_i = \mathrm{KV}_i$ has not been queried before, then SIM tosses a random coin $c_i \in \{0, 1\}$ with the probability that $\Pr|c_i = 0| = 1/(q_T + 1)$, where q_T is a very large number. We require that q_T should be larger than the number of oracle queries for the aggregate key. If $c_i = 0$, then select a random value $\mathrm{kv}_i \in_R Z_p$ and compute $T_i^* = (g^{\mathrm{kv}_i})^c$. Otherwise, compute $T_i^* = (g^{\mathrm{kv}_i})^{bc}$. Add the tuple $< w_i, \mathrm{KV}_i, c_i, T_i^* >$ to table T_{kw} and return T_i^*. (2) Otherwise, retrieve T_i^* from table T_{kw} with respect to KV_i and return T_i^*.

SIM sets public key $\mathrm{pk} = v = (g^c)^r$, where $r \in_R Z_p$. Finally, SIM records the tuple $< \mathrm{pk}, r >$ in table T_k. Table T_k is used to record the tuple $< \mathrm{pk},\ r >$, and these records will be used in other oracles to give a response to the queries. Simulator SIM sends the public key pk to A.

2. Query phase 1

A issues an aggregate key query for set S. For a given set S of the document indices, SIM computes an aggregate key $k_{\mathrm{agg}} = \pi_{j \in S}\ g_j^r$ where $r \in_R Z_p$, and sends it to A.

3. Challenge

A outputs two equal-length messages M_0 and M_1 on which it wishes to be challenged with a set of keywords KW, an index i_{c}, and a public key pk^*. If pk^* is not in table T_k, then simulator SIM terminates. Here, the restriction is that the attacker had not previously asked for the aggregate key corresponding to set S where $i_{\mathrm{c}} \in S$. Otherwise, SIM randomly chooses a bit $u \in \{0,1\}$ and computes ciphertext C_u^*.

Challenger C sets searchable encryption key $k_i = t = bc/a^{1+n-1}$ and computes ciphertext

$$\begin{aligned}
&C_u^* = (C_1, C_2, C_4) \\
&= \left(g^{bc/a^{i_{\mathrm{c}}-1}}, e(v, g_{i_{\mathrm{c}}}), m_u e(g_1, g_n)^{bc/a^{1+n-1}}\right) \\
&= \left(g^{bc/a^{1+n-1}}, e(v, g_{i_{\mathrm{c}}}), m_u e(g^{a^1}, g^{a^n})^{bc/a^{1+n-1}}\right) \\
&= \left(g^{bc/a^{1+n-1}}, e(v, g_{i_{\mathrm{c}}}), m_u e(g, g)^{(a^n a^1)(bc/a^{1+n-1})}\right).
\end{aligned}$$

4. Query phase 2

The simulator answers the queries as in phase 1. However, adversary A is not allowed to issue an aggregate key query for set S which includes an index i_{c} and a decrypt query on C_u^*.

5. Guess

A outputs a guess $u' \in_R \{0, 1\}$.

If $u' = u$ $(R = e(g, g)^{abc})$, then SIM outputs $v' = 0$. If $u' \neq u$, then SIM outputs $v' = 1$ to indicate that the ciphertext is a random element. Therefore, A gains no information about v, and in turn, $\Pr[u \neq u'|v = 1] = 1/2$. As simulator SIM guesses $v' = 1$ when $u \neq u'$, $\Pr[v = v'|v = 1] = 1/2$.

If $v = 0$, then A is able to view the valid encryption components with advantage $\epsilon(l)$, a negligible quantity in the security parameter in l. Therefore, $\Pr[u = u'|v = 0] = 1/2 + \epsilon(l)$. Similarly, simulator SIM guesses $v' = 0$ when $u = u'$, and in turn, $\Pr[v = v'|v = 0] = 1/2 + \epsilon(l)$. The overall advantage of the simulator in the DBDH game is

$$\begin{aligned}
&\frac{1}{2}\Pr[v = v'|v = 1] + \frac{1}{2}\Pr[v = v'|v = 0] - \frac{1}{2} \\
=&\frac{1}{2} \times \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \epsilon(l)\right) - \frac{1}{2} \\
=&\frac{\epsilon(l)}{2}.
\end{aligned}$$

Therefore, if A has a non-negligible advantage $\epsilon(l)$ in the above game, then we can build a simulator (SIM) which can break the DBDH problem with non-negligible quantity $\epsilon(l)/2$, which is an intractable problem.

5.1.2 Security definition: IND-CKA game

We need to ensure that Test (Tr, S, i) does not reveal any information about keyword w unless trapdoor Tr is available. Therefore, we provide a security model to prove that the proposed MULKASE is secure against a chosen keyword attack (CKA). We define security against an active attacker who is able to obtain trapdoor Tr for any keyword w of his/her choice. Even under such an attack, the attacker should not be able to distinguish an encryption of a keyword KW_0 from an encryption of a keyword KW_1 for which he/she did not obtain the trapdoor. This is the most appropriate security concept in KASE. The KASE methods (Cui et al., 2016; Li et al., 2016) do not prove their security against an IND-CKA attack. The game proceeds as shown in Fig. 9.

The advantage of the adversary in this game is defined as $|\Pr[b = b'] - \frac{1}{2}|$, where the probability is taken over the random bits used by the challenger and the adversary.

Definition 3 The proposed MULKASE is

semantically secure against an adaptive chosen keyword attack if all polynomial time adversaries have at most a negligible advantage in this security game.

Theorem 2 The proposed MULKASE method is IND-CKA secure, assuming that the DBDH problem is hard to solve.

Proof We consider a challenger C, a simulator SIM, and a polynomial time adversary A. We assume that adversary A has a non-negligible advantage ϵ to break the privacy of MULKASE. Then we can construct a simulator SIM that breaks the decisional DBDH problem $\varsigma = (g, g^a, g^b, g^c, z)$ with advantage $\frac{\epsilon}{2}\left(1-\frac{N^2}{p}\right)$.

Here, we assume that for trapdoor Tr_w which denotes the search query of keyword w and $\mathrm{Tr}_{w'}$ which denotes a query of keyword w', $\mathrm{Tr}_w \neq \mathrm{Tr}_{w'}$. If there exists w and w', $w \neq w'$ such that $\mathrm{Tr}_w = \mathrm{Tr}_{w'}$, then Tr_w can search in keyword-ciphertext $C_{w'}$ where $\mathrm{Tr}_{w'} \models C_{w'}$ and $\mathrm{Tr}_w \not\models C_{w'}$. This assumption holds with probability

$$\frac{p(p-1)\ldots(p-(N-1))}{p^N} > \frac{(p-(N-1))^N}{p^N}$$
$$=\left(1-\frac{N-1}{p}\right)^N > \left(1-\frac{N(N-1)}{p}\right) > \left(1-\frac{N^2}{p}\right),$$

where N is the number of possible keywords in the system and p is the order of group G. On DBDH input (g, g^a, g^b, g^c, R), simulator SIM aims to decide if $R = e(g,g)^{abc}$.

Challenger C generates $a, b, c, z \in_R Z_p$, bilinear groups G, G_T with prime order p and the mapping $G \times G \to G_T$. Here, g is a generator of group G. Challenger C computes v as follows:

$$R = \begin{cases} e(g,g)^{abc}, & v = 0, \\ e(g,g)^{z}, & v = 1. \end{cases}$$

The challenger gives instance $(g, g^a, g^b, g^c, R) \in G_T$ to simulator SIM. SIM interacts with A (SIM simulates C for A) and starts the simulation as follows:

1. Setup

This is mostly the same as that in the IND-CPA game. For the selected set S of document indices, the challenger generates aggregate key $k_{\mathrm{agg}} = \pi_{j\in S} g_j^{\tau}$. SIM sends the public key pk and aggregate key with a selected set of document indices (k_{agg}, S) to A.

2. Query phase 1

A issues a Trapdoor$(\cdot)$ query for a keyword w. SIM computes a trapdoor $\mathrm{Tr} = (\mathrm{Tr}_0, \mathrm{Tr}_1) = (k_{\mathrm{agg}} \cdot (g^{a^i})^r, g^r)$ where $r \in Z_p$, and sends it to A.

3. Challenge

A outputs two keywords KW_0 and KW_1 on which it wishes to be challenged and index i_c. We consider only the case where $w \not\models \mathrm{KW}_0 \wedge w \not\models \mathrm{KW}_1$. The reason for this is if $\mathrm{KW}_0 = \mathrm{KW}_1$ or $(w \models \mathrm{KW}_0 \wedge w \models \mathrm{KW}_1)$, then SIM simply aborts and takes a random guess. The probability that SIM aborts is $\Pr[\mathrm{abort}] = N^2/p$. Otherwise, SIM randomly chooses a bit $u \in \{0,1\}$ and computes keyword-ciphertext for KW_u.

Challenger C sets $t = a/a^i$ and computes $C_b^* = (C_1, C_2, C_3) = \left(g^{a/a^i}, e(v, g_i), (H(w_j))^{a/a^i}\right) | H(w_j) \in \mathrm{KW}_u$.

4. Query phase 2

The simulator answers the queries as in phase 1. A is not allowed to issue a Trapdoor$(\cdot)$ query on keywords KW_0 and KW_1 and a Test$(\cdot)$ query on C_u^*.

5. Guess

A outputs a guess $u' \in_R \{0,1\}$.

If $u' = u$, then SIM outputs $v' = 0$. If $u' \neq u$, then SIM outputs $v' = 1$. Based on this, there will be two cases as follows: (1) If $v = 0$, then $R = e(g,g)^{abc}$ and hence the challenge ciphertext is a correct ciphertext of keyword KW_u. Therefore, A outputs $u' = u$ with an advantage ϵ, $\Pr[u' = u | v = 0 \wedge \overline{\mathrm{abort}}] = 1/2 + \epsilon$, $\Pr[v' = v | v = 0 \wedge \overline{\mathrm{abort}}] = 1/2 + \epsilon$ because SIM guesses $v' = 0$ when $u' = u$. (2) If $v = 1$, then the challenge ciphertext is independent of KW_0 and KW_1, so that A cannot obtain any information of u. Therefore, A outputs $u' \neq u$ with no knowledge, $\Pr[u' \neq u | v = 1 \wedge \overline{\mathrm{abort}}] = 1/2$, $\Pr[v' = v | v = 1 \wedge \overline{\mathrm{abort}}] = 1/2$ because SIM guesses $v' = 1$ when $u' \neq u$.

From cases (1) and (2), it follows that SIM's advantage in this DBDH game can be computed as

$$\begin{aligned}
&\Pr[v' = v] - \frac{1}{2} \\
=&\Pr[v = 0]\Pr[v' = v | v = 0] \\
&+ \Pr[v = 1]\Pr[v' = v | v = 1] - \frac{1}{2} \\
=&\frac{1}{2}\Pr[v' = v | v = 0] + \frac{1}{2}\Pr[v' = v | v = 1] - \frac{1}{2} \\
=&\frac{1}{2}\big(\Pr[v' = v | v = 0] + \Pr[v' = v | v = 1] - 1\big)
\end{aligned}$$

$$
\begin{aligned}
=&\frac{1}{2}\big(\Pr[\text{abort}]\Pr[v'=v|v=0\wedge\text{abort}]\\
&+\Pr[\overline{\text{abort}}]\Pr[v'=v|v=0\wedge\overline{\text{abort}}]\\
&+\Pr[\text{abort}]\Pr[v'=v|v=1\wedge\text{abort}]\\
&+\Pr[\overline{\text{abort}}]\Pr[v'=v|v=1\wedge\overline{\text{abort}}]-1\big).
\end{aligned}
$$

Because the event "abort" is independent of the DBDH challenge, we have

$$
\begin{aligned}
&\Pr[v'=v|v=0\wedge\text{abort}]\\
=&\Pr[v'=v|v=1\wedge\text{abort}]\\
=&\frac{1}{2}\left[\frac{N^2}{p}\cdot\frac{1}{2}+\left(1-\frac{N^2}{p}\right)\left(\frac{1}{2}+\epsilon\right)+\frac{N^2}{p}\cdot\frac{1}{2}\right.\\
&\left.+\left(1-\frac{N^2}{p}\right)\cdot\frac{1}{2}-1\right]\\
=&\frac{1}{2}\left[\left(1-\frac{N^2}{p}\right)\epsilon\right]\\
=&\frac{\epsilon}{2}\left(1-\frac{N^2}{p}\right).
\end{aligned}
$$

Therefore, if A has a non-negligible advantage ϵ in the above game, then we can build a simulator (SIM) that can break the DBDH problem with non-negligible quantity $=\frac{\epsilon}{2}\left(1-\frac{N^2}{p}\right)$, which is an intractable problem.

Theorem 3 The proposed MULKASE method can achieve the goal of query privacy.

Proof To launch an attack, the attackers may try to obtain some information that they are not authorized to access. For example, the curious cloud server can obtain the stored keyword-ciphertexts, the δ_i associated with the searchable encryption key of the ith document, the submitted trapdoor, and so on. The malicious authorized user, say, Alice, can have an aggregate key k_{agg} and the ability to perform the keyword search across a set of documents owned by Bob. However, even if the malicious user can obtain the information, MULKASE can be proved to achieve the goal of query privacy. The result of Theorem 3 can be derived from the following lemmas:

Lemma 1 An attacker is not able to learn a given keyword in a query from the submitted trapdoor.

Proof Suppose attacker A wants to learn a keyword given in a query from the submitted trapdoor $\text{Tr}=(\text{Tr}_0,\text{Tr}_1)=(k_{\text{agg}}\cdot(g^{\text{kv}_i})^b,g^b)$ where $b\in Z_p$. If he/she can guess the aggregate key k_{agg}, he/she will achieve success. In this case, A can know the system parameters $\text{SP}=(B,\text{PubK},\text{Tr}^*)$ where $\text{PubK}=(g,g_1,\ldots,g_n)\in G_{n+1}$ and the set S of indices.

However, to generate k_{agg}, for each $j\in S$, attacker A must compute g_j^{γ}. As γ is the data owner's master-secret key msk, which is protected from leakage, there is a very negligible probability that A can obtain msk. As a result, A cannot fulfill his/her goal.

Lemma 2 An attacker cannot learn a keyword in a document from the stored keyword-ciphertexts and the relevant public information.

Proof The curious cloud server A may try to determine something from the stored encrypted data. With the knowledge of $\text{PubK}=(g,g_1,\ldots,g_n)\in G_{n+1}$, and public information $\delta_i=(C_1,C_2)$ where $C_1=g^t$, $C_2=e(v,g_i)=e(g^{\gamma},g_i)$, and $C_3=C_{\text{KW}}=(g^{kv_i})^t$, A may try to launch attacks by retrieving the value of t from the known C_1 or C_3. However, as per the discrete logarithm problem, A is unable to obtain the value of t. In fact, this result is confirmed by the hardness assumption of the BDHE problem. Therefore, an attacker is unable to determine the content from the stored information.

The proofs of Lemmas 1 and 2 clearly show that the proposed MULKASE method achieves query privacy.

Theorem 4 The proposed MULKASE method supports controlled searching.

Proof Theorem 4 requires that a user cannot carry out a keyword search on a set of documents that are not applicable by the known aggregate searchable encryption key. In addition, one cannot create new aggregate keys for another set of documents from the known one. Theorem 4 can be deduced from the following lemmas:

Lemma 3 Each user who has the valid aggregate key can carry out a successful keyword search.

Proof Lemma 3 proves the correctness of the proposed method. On receiving the aggregate trapdoor Tr from the query generator, the cloud server executes the test algorithm of the proposed MULKASE scheme to perform a keyword search without requiring transform of Tr. Suppose that the user has a trapdoor $\text{Tr}=(\text{Tr}_0,\text{Tr}_1)=(k_{\text{agg}}\cdot(g^{\text{kv}_i})^b,g^b)$ for keyword w_i having value kv_i. Trapdoor Tr is generated using an aggregate key k_{agg}. Let us say that an aggregate key k_{agg} comprises the power to search across m different data owners' selected shared documents within the scope of a set of indices $S=S_1\cup S_2\ldots\cup S_m$. On receiving the aggregate trapdoor

Tr, the cloud server first calculates $C' = \pi_{j\in S}\ C_{2,j} = \pi_{j\in S,\ i\in\{1,2,\ldots,m\}}\ e(g^{\gamma_i}, g_j)$, and searches for a given query keyword by checking

$$\begin{aligned}
&e(C', C_1)e(C_3, \mathrm{Tr}_1)/e(\mathrm{Tr}_0, C_1)\\
=&\frac{e(\pi_{j\in S,\ i\in\{1,2,\ldots,m\}}\ e(g^{\gamma_i}, g_j), g^t)\cdot e((g^{\mathrm{kv}_i})^t, g^b)}{e(k_{\mathrm{agg}}\cdot(g^{\mathrm{kv}_i})^b, g^t)}\\
=&\frac{e(\pi_{j\in S,\ i\in\{1,2,\ldots,m\}}\ e(g^{\gamma_i}, g_j), g^t)\cdot e((g^{\mathrm{kv}_i})^t, g^b)}{e(k_{\mathrm{agg}}, g^t)\cdot e((g^{\mathrm{kv}_i})^b, g^t)}\\
=&1.
\end{aligned}$$

Therefore, the user with the valid aggregate key can carry out a successful keyword search.

Lemma 4 An authorized user or server is not able to carry out a keyword search across any set of documents, not in the range of the user's aggregate key.

Proof From Lemma 3, we can see that if C' is generated by a wrong set S', the expression $e(k_{\mathrm{agg}}, g^t)$ will not be equal to the expression $e(C', g^t)$. Therefore, the two expressions cannot be canceled out and result in an unsuccessful keyword search. The C' must be computed by the same set S of the aggregate key; otherwise, the Test$(\cdot)$ algorithm returns false for any document not in the range of the user's aggregate key.

Lemma 5 Even when a malicious authorized user collides with the cloud server, he/she is not able to carry out a keyword search across any document that is not within the range of the user's aggregate key.

Proof In the case of collision, if a curious cloud server and a malicious authorized user collide, then they may try to perform a keyword search across documents that are not within the range of their respective aggregate key. In MULKASE, the size of an aggregate key is constant; i.e., it consists of a single group of elements irrespective of the number of data owners and the number of shared documents. Therefore, an adversary cannot extract the individual master secret key or keyword from the single aggregate key and trapdoor.

Furthermore, if a user has two or more aggregate keys shared by different data owners (or by colliding with another user), he/she may try to search the data in the documents that are not within the range of the individual aggregate key. Suppose that the user has an aggregate key $k_{\mathrm{agg}} = k_{\mathrm{agg},A}\cdot k_{\mathrm{agg},B} = \pi_{j\in S} g_j^{\gamma_i}$ where $S = S_A \cup S_B$ and $i \in \{1, 2\}$, which comprises the power to search in two different data owners' documents in the range of set $S = S_A \cup S_B$. Consider $S_A = \{1, 2, 5\}$ and $S_B = \{3, 4\}$. Then an authorized user having an aggregate key k_{agg} can search for "doc$_2$" of data owner A. As $2 \notin S_B$ using k_{agg} the user is unable to search for document "doc$_2$" of data owner B. Because a user does not have msk, he/she is unable to calculate $g_j^{\gamma_2}$ for $j = 2$. Therefore, the user cannot infer more information than each aggregate key individually contains, even after combining multiple aggregate keys.

Hence, different users cannot collide to decrypt or search in the ciphertext(s) not in the range of their respective aggregate keys.

Lemma 6 An attacker is not able to generate the new aggregate key for any new set S' of indices from the known aggregate key.

Proof A malicious authorized user A has an aggregate key k_{agg} for a set S. He/She may try to compute a new aggregate key for a new set S' to access other documents not in the range of set S. To compute the new key, A should know the value of g_j^{γ} for all $j \in S'$. However, A knows $k_{\mathrm{agg}} = \pi_{j\in S}\ g_j^{\gamma}$, but he/she cannot obtain an individual multiplier from the product. Each multiplier of an aggregate key is protected by the data owner's master-secret key γ. Therefore, A cannot fulfill his/her goal.

From the proofs of Lemmas 3–6, we can say that the proposed MULKASE supports controlled searching.

Theorem 5 MULKASE is secure against a cross-pairing attack.

Proof Attacker A (the curious cloud server or an authorized user) may try to learn more information from the stored encrypted data that are beyond attackers' authorization. With the knowledge of PubK $= (g,\ g_1, \ldots, g_n) \in G_{n+1}$ and public information $\delta_i = (C_1, C_2)$ where $C_1 = g^t$, $C_2 = e(v,\ g_i) = e(g^{\gamma},\ g_i)$, A may try to obtain information about message m by computing the value of $e(g_1,\ g_n)^t$. The term $g_{n+1} = g^{\alpha^{n+1}}$ is missing in the public parameters PubK. Therefore, an attacker cannot compute the value of $e(C_1, g_{n+1})$ to obtain the value of $e(g_1, g_n)^t$. Note that A cannot obtain the value of $e(g_1,\ g_n)^t$ by cross-pairing the available public information PubK and δ_i. Moreover, data-ciphertext C_m is secured with the value of t, and A is unable to obtain the value of t as per the discrete logarithm problem. In fact, this result is confirmed by the hardness assumption of the BDHE problem. In

addition, using the keyword-ciphertext and public information, an attacker cannot launch cross-pairing attack to learn whether doc_i contains the keyword w or not.

In MULKASE, both the parameters of trapdoor $\mathrm{Tr} = (\mathrm{Tr}_0, \mathrm{Tr}_1)$ are protected using the same value $b \in_R Z_p$. Therefore, cross-pairing of two different trapdoors Tr and Tr' is not possible, as the value of b is used differently to generate each trapdoor. Cross-pairing of different trapdoors can generate incompatibility.

Therefore, MULKASE is secure against a cross-pairing attack.

5.2 Theoretical analysis: comparison of significant characteristics

The comparison of KASE methods (Chu et al., 2014; Cui et al., 2016) with the proposed MULKASE method based on significant characteristics is shown in Table 2. The KAE methods do not allow delegation of search rights across the shared data (Singhal, 2001; Chu et al., 2014; Mahalle and Pawade, 2014; Firdose and Rebekah, 2015; Patranabis et al., 2015; Dang et al., 2016; Guo et al., 2017). Therefore, a comparison of KAE methods with MULKASE is inappropriate. In a similar way, comparison of the method for KASE proposed in Zhou et al. (2018) with MULKASE is inappropriate because it does not support search of multi-owner data.

As shown in Table 2, KASE (Cui et al., 2016) does not support search of multi-owner data using a single trapdoor. On the other hand, the KASE method proposed in Li et al. (2016) provides a solution for searching multi-owner data using a single trapdoor and also allows verification of the search results using an aggregate key. However, the KASE methods (Cui et al., 2016; Li et al., 2016) require trapdoor transformation at the time of keyword search, adding an additional computational and storage overhead on the server side. Additionally, from Table 2, we can see that no KASE methods listed in the table provide the other important characteristics such as security against message and keyword indistinguishability, security against cross-pairing attacks, or federated searching using a single trapdoor. In contrast, MULKASE allows the user to search data owned by multiple owners using a single trapdoor with security against message and keyword indistinguishability as well as cross-pairing attacks. Furthermore, MULKASE supports searching data distributed on a federated cloud using a trapdoor and reducing overhead on the server side at the time of the keyword search (because the trapdoor transformation is not required). The size of an aggregate key in MULKASE is independent of the number of documents held by a data owner.

5.3 Theoretical analysis: comparison of storage complexity

We can compare the storage overhead of the public parameters (PubK), aggregate key (k_{agg}), trapdoor (Tr), and ciphertext (C) in terms of the size of an element from the bilinear group shown in Table 3.

As we consider the DBDH security assumption instead of the 1-BDHE assumption, we are required to generate n public parameters $\{g, g_1, \ldots, g_n\}$ instead of $2n + 1$ (Cui et al., 2016; Li et al., 2016). Therefore, the size of PubK in the MULKASE method is $n + 1$ instead of $2n + 1$.

In the MULKASE method, the aggregate key is a constant size; i.e., it consists of a single group of elements irrespective of the number of data owners and documents.

The size of a trapdoor in MULKASE is greater compared to the KASE method for searching across

Table 2 Comparative analysis of significant characteristics

Method	MO	TT	KSI	FS	IND-CPA	IND-CKA	CPrA	SRV
Cui et al. (2016)	×	✓	×	×	×	×	×	×
Li et al. (2016)	✓	✓	×	×	×	×	×	✓
MULKASE	✓	×	✓	✓	✓	✓	✓	×

MO: search across multi-owner data using a single aggregate key; TT: trapdoor transformation required; KSI: the aggregate key size being independent of the maximum number of ciphertexts held by a data owner; FS: support federated search using an aggregate trapdoor; IND-CPA: secure against message indistinguishability attacks; IND-CKA: secure against keyword indistinguishability attacks; CPrA: secure against cross-pairing attacks; SRV: verification of search result using an aggregate key

single-owner data (Cui et al., 2016). However, with such increased storage overhead, we provide trapdoor and data privacy in contrast to the KASE methods (Cui et al., 2016; Li et al., 2016). For searching across multi-owner data in the KASE method (Cui et al., 2016), the number of trapdoors required equals the number of data owners. On the other hand, KASE (Li et al., 2016) allows searching across multi-owner data using a single trapdoor, but it requires auxiliary values that are the same as the number of data owners. In MULKASE, the size of the trapdoor remains constant in both cases, searching single-owner data and searching multi-owner data.

The proposed method provides the novel feature of search ability across multi-owner data. However, the size of the ciphertext in the MULKASE method is the same as that of the KASE method (Cui et al., 2016). The storage overhead for ciphertext in Li et al. (2016) is greater in comparison to MULKASE.

5.4 Theoretical analysis: comparison of computational complexity

The expected computational time based on the input parameters for the proposed MULKASE method and methods proposed in Cui et al. (2016) and Li et al. (2016) is shown in Table 4. We analyze the computational overhead in terms of major operations such as scalar multiplication (M), bilinear pairings (P), and exponentiations (E) involved in the listed methods.

The results given in Table 4 clearly indicate that in the MULKASE method, the number of pairing operations that are required for encryption as well as for keyword search operations is constant irrespective of the number of shared documents and number of data owners.

The proposed method provides the novel feature of search ability across multi-owner data. However, the encryption algorithm in our method requires less computation time compared to the KASE methods (Cui et al., 2016; Li et al., 2016).

The computation cost required to generate an aggregate key in MULKASE, in both scenarios of searching across multi-owner data and single-owner data, is the same as that of KASE (Li et al., 2016). The computation time to generate an aggregate key to search multi-owner data relies on the number of data owners, i.e., U in the MULKASE and KASE methods (Li et al., 2016).

The computation cost required to generate a trapdoor from an aggregate key and query keyword is greater in MULKASE compared to the KASE methods (Cui et al., 2016; Li et al., 2016) for searching single-owner data. However, with such an extra

Table 3 Comparison of storage overhead

Method	PubK	k_{agg}	Trapdoor		Ciphertext
			Searching single-owner data	Searching multi-owner data	
Cui et al. (2016)	$(2n+1)\|G\|$	$\|G\|$	$\|G\|$	$U\|G\|$	$2G+G_T$
Li et al. (2016)	$(2n+1)\|G\|$	$\|G\|$	$\|G\|$	$\|G\|+U\cdot\text{AV}$	$3G+G_T$
MULKASE	$(n+1)\|G\|$	$\|G\|$	$2\|G\|$	$2\|G\|$	$2G+G_T$

n: the number of documents that belong to the data owner; U: the number of different data owners (i.e., k_{agg} aggregates the power to search across U data owner's documents); AV: the size of the auxiliary value used to search across multi-owner data; $|G|$: the bit length of the element that belongs to G; G and G_T: bilinear groups

Table 4 Comparison of computational overhead

Method	Encryption*	Aggregate key generation		Trapdoor generation**		Keyword search***	
		Single-owner	Multi-owner	Single-owner	Multi-owner	Single-owner	Multi-owner
Cui et al. (2016)	$2E+3P+2M$	$\|S\|\cdot M$	–	M	$U\cdot M$	$(\|S\|\cdot M)+(\|S\|\cdot M+2P)$	$(\|S\|\cdot M)+U(\|S\|\cdot M+2P)$
Li et al. (2016)	$2E+3P+2M$	$\|S\|\cdot M$	$U\cdot M$	M	$(M+E)+U(M+E)$	$(\|S\|\cdot M)+(\|S\|\cdot M+2P)$	$U(\|S\|\cdot M)+U(\|S\|+2M)+2P$
MULKASE	$2E+2P+M$	$\|S\|\cdot M$	$U\cdot M$	$2E+M$	$2E+M$	$3P+\|S\|\cdot M$	$3P+\|S\|\cdot M$

Single-owner: searching single-owner data; Multi-owner: searching multi-owner data. E: exponentiation; M: scalar multiplication; P: pairing; U: number of data owners; $|S|$: size of set S. *: data-ciphertext+keyword-ciphertext; **: trapdoor+auxilliary value; ***: trapdoor transformation+search

overhead, MULKASE provides trapdoor privacy in contrast to the KASE methods (Cui et al., 2016; Li et al., 2016) which leak data related to the keyword from the trapdoor. In MULKASE, to search multi-owner data, the computation time required to generate the trapdoor is constant, i.e., $O(1)$ and independent of the number of data owners U. On the other hand, for KASE methods (Cui et al., 2016; Li et al., 2016) the computational cost to generate a trapdoor to search data owned by multiple owners depends on U.

MULKASE uses the fixed number of pairing operations during keyword search. The computation time required to carry out a keyword search in MULKASE remains the same in both cases, i.e., searching multi-owner data or single-owner data. Moreover, in the KASE methods (Cui et al., 2016; Li et al., 2016), when an authorized user submits the trapdoor to the cloud server to perform a search, the total time required to search for a keyword on the cloud server is: time required to transform aggregate trapdoor Tr to generate actual trapdoor Tr_i to search the ith document + time required to execute the Test $(\cdot)$ algorithm. This trapdoor transformation process used in the methods (Cui et al., 2016; Li et al., 2016) adds an extra computational and storage overhead on the server side. The MULKASE method allows search of shared documents using an aggregate trapdoor without requiring generation of an individual trapdoor Tr_i from the aggregate trapdoor. Therefore, we overcome the extra overhead required on the server side. As shown in Table 4, we improve the query performance compared to existing KASE methods (Cui et al., 2016; Li et al., 2016) by achieving constant computational cost in both cases, searching multi-owner data and searching single-owner data. Moreover, in MULKASE, the computational cost to search multi-owner data is independent of the number of data owners U, whereas in KASE methods (Cui et al., 2016; Li et al., 2016) it is linear with U. With such constant computational overhead, MULKASE performs better than the KASE methods (Cui et al., 2016; Li et al., 2016).

5.5 Theoretical analysis: comparison of communication complexity

Table 5 shows the comparison of the communication overhead required to submit a search request to the cloud server. We observe that in both scenarios, searching single-owner data and multi-owner data, MULKASE requires a constant communication cost (i.e., $O(1)$) to send a search request by submitting a trapdoor to the cloud server. This constant overhead makes MULKASE efficient compared to KASE methods (Cui et al., 2016; Li et al., 2016). The KASE methods require a communication overhead proportional to the number of data owners (i.e., $O(U)$ for U data owners), as they are required to submit U trapdoors to search the multi-owner data.

Table 5 Comparison of communication overhead

Method	Communication overhead	
	Single-owner	Multi-owner
Cui et al. (2016)	T	$U \cdot T$
Li et al. (2016)	T	$T + U \cdot \mathrm{AV}$
MULKASE	T	T

T: number of trapdoors, U: number of data owners, AV: number of auxiliary values used to search over multi-owner data

6 Empirical evaluation

6.1 Implementation details

To evaluate the performance, we implemented the prototype for our method as well as the methods of KASE (Cui et al., 2016) and VKASE (Li et al., 2016). We conducted the experiments on two different platforms, a computer and a mobile device, using the Java pairing based cryptographic (JPBC) library (de Caro and Iovino, 2011). The computer configuration was a 64-bit, Intel® Core™ i5-7200U CPU @ 2.50 GHz with Windows 10 OS. To evaluate the computational cost in the resource-constrained device, we simulated each cryptographic operation on a Xiaomi Redmi 3S mobile device with Android OS 6. We used type A pairing for evaluation and it was the fastest (symmetric) pairing among all types of curves. Type A pairings were constructed on the curve $y^2 = x^3 + x$ across the field F_q for some prime $q = 3 \bmod 4$. The order of G_1 is p. To compare the performance of the proposed method with those of the existing methods, we considered the following parameters: (1) number of keywords; (2) number of users (data owners); (3) number of keyword ciphertexts in set S; (4) number of documents held by the data owner. The values of the simulation parameters are as given in Table 6.

For each simulation value we ran the Setup$(\cdot)$,

Encrypt(·), Trapdoor(·), and Test(·) algorithms multiple times on the computer and mobile device, and considered their average results. To calculate the computation cost, we measured the time required by an algorithm on all given inputs of the simulation parameters. In MULKASE, Setup(·), KeyGen(·), and TestQ were computed by the cloud server. Encrypt(·), Trapdoor(·), and Extract(·) were computed by the data owner (using the mobile device, the sensor, or the computer). Therefore, we evaluated the following computational costs and storage costs of our method and compared them with those of the existing KASE methods (Cui et al., 2016; Li et al., 2016): (1) Evaluate and compare the storage overhead of system parameters (with respect to the number of documents held by a data owner); (2) Evaluate and compare the storage overhead of the query trapdoor (with respect to the number of data owners in case of searching multi-owner data); (3) Evaluate and compare the computation cost of Encrypt(·) (with respect to the number of keywords attached with keyword ciphertext); (4) Evaluate and compare the computation cost of Test(·) (with respect to the number of keyword ciphertexts in set S and number of data owners in case of searching multi-owner data).

Table 6 Simulation parameters

Parameter	Values for simulation
Number of keywords	{1, 4, 10, 33, 100, 210, 500, 1000, 3000, 4000}
Number of users	{500, 1000, 2000, 3000, 4000, 5000}
Number of keyword ciphertexts in set S	{10, 50, 100, 500, 800, 1000, 2500, 4000, 5000}
Number of documents held by the data owner	{10, 50, 100, 500, 800, 1000, 2500, 4000, 5000}

The theoretical analysis confirms that for the KASE methods considered, i.e., MULKASE and KASE (Li et al., 2016), the size of an aggregate key and computation cost of Extract(·) are the same. The computation time to generate an aggregate key to search multi-owner data relies on the number of data owners in the MULKASE and KASE methods (Li et al., 2016). Therefore, we did not test the computation cost of the Extract(·) algorithm and storage overhead of an aggregate key. Although KASE allows search of multi-owner data using a single trapdoor (Li et al., 2016), it requires auxiliary values that are the same as the number of data owners. In MULKASE, the size of the trapdoor remains constant in both cases, searching single-owner data and searching multi-owner data.

6.2 Performance analysis

In this subsection we compare the computational cost of Encrypt(·) (with respect to the number of keywords attached to keyword ciphertext) and Test(·) (with respect to the number of keyword ciphertexts in set S and the number of data owners in case of searching across multi-owner data) algorithms, storage overhead of the query trapdoor (with respect to the number of data owners in case of searching across multi-owner data), and system parameters (with respect to the number of documents held by the data owner) of our method with those of the existing KASE methods (Cui et al., 2016; Li et al., 2016).

6.2.1 Comparison of storage complexity

In this analysis of storage overhead, we used 80-bit advanced encryption standard (AES) (Daemen and Rijmen, 2001) key size security level. Therefore, the size of q is 512 bits and the size of an element in group G_1 is 1024 bits using an elliptic curve with 160 bits p. However, the size of an element in group G_1 can be reduced to 520 bits (i.e., 65 bytes) using a compression technique proposed in Shim (2012). The size of an element in group G_2 is 1024 bits.

The results shown in Fig. 10a indicate that the size of the trapdoor increases linearly with the number of users (i.e., data owners), for both the existing KASE methods (Cui et al., 2016; Li et al., 2016). The size of the trapdoor remains constant in our method for both cases, searching single-owner data and searching multi-owner data. For searching multi-owner data in the KASE method (Cui et al., 2016), the number of trapdoors required equals the number of data owners. On the other hand, KASE (Li et al., 2016) allows searching multi-owner data using a single trapdoor, but it requires auxiliary values that are the same as the number of data owners. Therefore, for both the KASE methods considered, data owners and users are challenged in generating and storing large numbers of query trapdoors in resource-constrained devices when searching multi-owner data. Additionally, the size of the system parameters is $n+1$ in our method, whereas it is $2n+1$

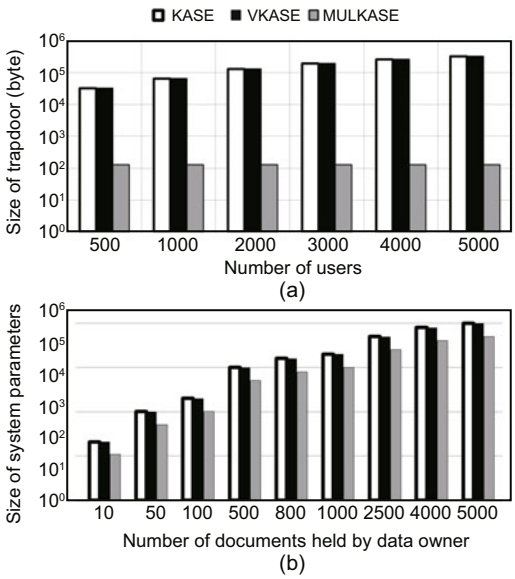

Fig. 10 Storage overhead: (a) size of trapdoors; (b) size of system parameters

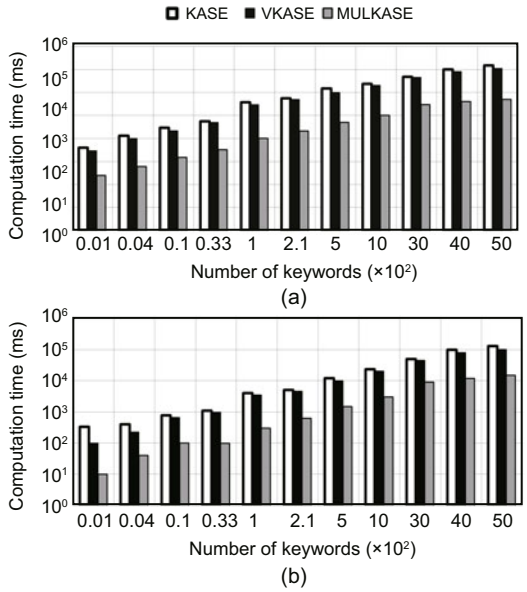

Fig. 11 Computation time of Encrypt(·): (a) mobile device; (b) computer

in the existing KASE methods (Cui et al., 2016; Li et al., 2016) (Fig. 10b) (n is the number of documents held by the data owner). From the storage analysis shown in Fig. 10, we can clearly say that the existing KASE methods require more computation time to generate $(2n + 1)$ system parameters compared to the $(n + 1)$ system parameters in MULKASE. In a similar way, the existing KASE methods require more computation time to generate U trapdoors to search U different data owners' data compared to a single trapdoor in our method.

6.2.2 Comparison of computational complexity

The results of the performance analysis clearly indicate that the performance analysis concurs with the theoretical analysis of MULKASE. Fig. 11 shows that the computational overhead of the encryption algorithm in MULKASE, as well as Cui's KASE and Li's KASE methods, is linear with the number of keywords attached to the ciphertext. However, the computation cost in our method is lower compared to the existing KASE methods for both mobile devices and computers. The reason for the low computation cost of the encryption algorithm in MULKASE is that we are using fewer pairing operations and multiplication operations compared to the existing KASE methods. In addition, the mathematical operation required to generate a keyword ciphertext is the pairing, whereas in MULKASE we are using exponentiation to generate a keyword ciphertext. Rouselakis and Waters (2015) showed that for different security levels (i.e., 80, 112, and 128 bits) and prime order groups, the group exponentiation (of a random group element with a random exponent) requires significantly less time compared to the pairing operations (on random group elements). Therefore, it is not feasible to upload the document with many keywords using a mobile phone in existing KASE methods as discussed in Cui et al. (2016). On the other hand, we are reducing the computation time of the encryption algorithm in our method compared to the existing KASE methods. Therefore, if the data owner needs to attach multiple keywords to one file, then MULKASE results in an efficient option compared to the existing methods.

The computation time required to carry out a keyword search in the proposed method is the same in the two cases, i.e., searching multi-owner data and searching single-owner data (Fig. 12a). Moreover, in the KASE methods, when an authorized user submits a trapdoor to the cloud server to perform a search, the total time required to search for a keyword on the cloud server is: time required to transform aggregate trapdoor Tr to generate actual trapdoor Tr_i for searching across the ith document + time required to execute the Test $(\cdot)$ algorithm. This trapdoor transformation process used in the methods (Cui et al., 2016; Li et al., 2016) adds an extra

computational and storage overhead on the server side. The proposed method allows searching across shared documents using an aggregate trapdoor without requiring generation of an individual trapdoor Tr_i from the aggregate trapdoor. Therefore, we overcome the extra overhead required on the server side (Fig. 12b) and improve the query performance compared to existing KASE methods. Furthermore, as shown in Fig. 12, MULKASE achieves a constant computational cost in both cases.

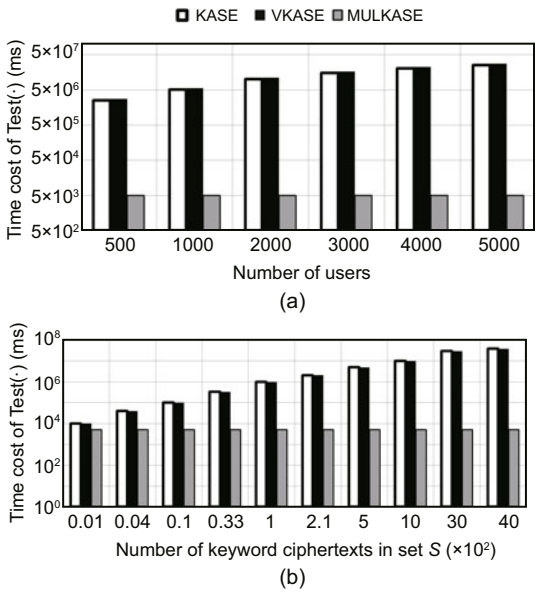

Fig. 12 Computation time of Test(·): (a) number of users; (b) number of keyword ciphertexts in set S

7 Federated search using MULKASE

A comprehensive survey of the literature for KASE (Cui et al., 2016; Lambhate and Patil, 2016; Li et al., 2016; Pansare et al., 2016; Zhou et al., 2018) shows the following research issues: (1) Is it possible to allow searching across different data records scattered across different cloud servers using minimal key requirements? (2) How is a search performed across the federated cloud? (3) Is it feasible to use the existing KASE to carry out a search across a federated cloud? Given these issues, we first give an overview of the federated cloud and a federated search. Then we discuss a solution to perform a federated search using the proposed MULKASE.

A federated cloud or a cloud federation (Massonet et al., 2015; Pawar et al., 2015) refers to the interconnection of two or more cloud service providers (public or private cloud) to create one hybrid cloud to meet commercial needs and to get functional-, location-, or cost-based benefit. The cloud federation allows an organization to distribute workloads around the world and move data between different cloud service providers. The client can therefore achieve a great deal of flexibility when using services through the federated cloud, as one does not need to rely upon a single service provider. To explain the idea of the federated cloud, consider a scenario where Alice has a cloud-based online shop. Alice chooses to federate her load across multiple cloud providers (considering both cost and location issues) to expand her capacity in handling multiple clients during peak periods. Assume that Alice has an India-based service, but to expand sales in the United States (US) region, she chooses a local cloud service provider to federate her load across regions.

A search process for such distributed and heterogeneous datasets that returns a concatenated search result is called a "federated search" (Pawar et al., 2015; Sinha et al., 2015). There are two models to employ a federated search (Diaz et al., 2010; Arya et al., 2015), search-time merging and index-time merging. In search-time merging, a query federator (trusted third party) receives the query and forwards it to multiple search engines. The federator then waits to receive replies from the search engines, and when received, returns a concatenated search result. Each search engine runs their individual search function. Search-time merging is an easier solution to implement because the query federator has to merge or concatenate the received results without requiring indexing of content centrally. However, if the query generator needs search results based on some relevancy criteria (e.g., keyword frequency in the file), then merging of search results received from different search engines into a unified result is difficult because each search engine scores relevancy in a different way. In the index-time merging model, the contents of search results that are received from different search engines are indexed centrally. As the contents are indexed centrally, a relevancy algorithm is applied centrally, which results in a more accurate search result for the user query. Therefore, traditional enterprise search systems use this model.

To apply the proposed MULKASE method to carry out a federated search across distributed datasets, we can consider the federated cloud as a

multi-owner model. Because the federated cloud is a combination of multiple clouds, the aggregate trapdoor allows search of data stored on any of the cloud servers. Considering the pros and cons of both search models (search-time merging and index-time merging), a hybrid search model is a better option to implement a federated search. In a hybrid search model, the search engine follows either the search-time merging or index-time merging model, based on whether the search engine can sustain the cost of getting its contents indexed centrally or not.

The system model to perform a federated search using the MULKASE method is shown in Fig. 13. The data owner first generates an aggregate key that is composed of the power to search across all the data distributed across multiple clouds. Furthermore, the data owner shares an aggregate key with the user to delegate search and access rights to a set of data, scattered across multiple clouds. If a user receives aggregate keys shared by different data owners, he/she generates a single aggregate key using the Aggregate(·) algorithm from MULKASE. An authorized user then generates a trapdoor for the query keyword and submits it to the query federator. On receiving the trapdoor, the query federator carries out a federated search (as per the hybrid search model) and sends the unified search result to the user.

Moreover, in MULKASE the aggregate key is of constant size; i.e., it consists of single group elements irrespective of the number of data owners and documents. For both search scenarios, i.e., searching data owned by multiple users or a single user, the size of the trapdoor remains constant in MULKASE. The theoretical and empirical analyses confirm that the MULKASE method is efficient in terms of key size, trapdoor size, number of pairing operations required for keyword search, and computation time to perform the search. The flexibility of MULKASE in terms of search and access rights delegation with complex classifications of ciphertext classes makes it more suitable for carrying out a federated search using a single aggregate trapdoor.

8 Conclusions and future extension

The issue of keyword search across multi-owner data using a single searchable secret key

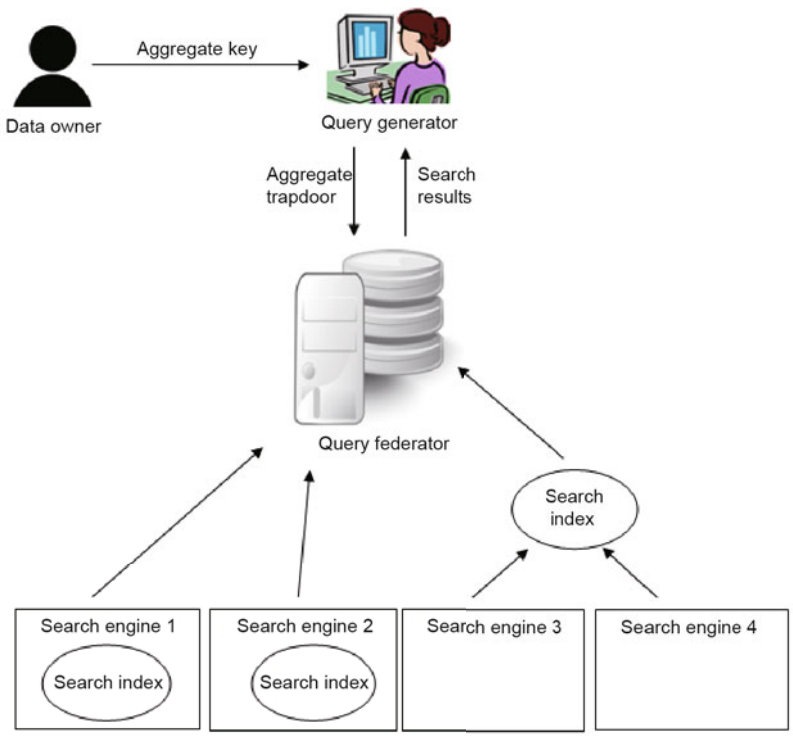

Fig. 13 System model for a federated search using MULKASE

is considered for the first time in this paper. We proposed a novel MULKASE method that allows an authorized user to search for a keyword across a set of data owned by different users using a single aggregate trapdoor. Security and efficiency analyses confirmed that the proposed MULKASE provides an efficient, secure, and searchable data-sharing system for public cloud storage. In addition, the proposed MULKASE method is applicable for carrying out a federated search. To the best of our knowledge, this is the first efficient, secure, and collision-resistant KASE method that allows search of multi-owner data and provides flexibility as well in terms of the delegation of search rights using an aggregate key even if the classifications of ciphertext classes are complex. In the future, we will work on revocation of the delegated search and access rights. Additionally, the construction of a secure and efficient leakage-resilient KASE method is an interesting research direction.

Compliance with ethics guidelines

Mukti PADHYA and Devesh C. JINWALA declare that they have no conflict of interest.

References

Akl SG, Taylor PD, 1983. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans Comput Syst*, 1(3):239-248. https://doi.org/10.1145/357369.357372

Arya D, Ha-Thuc V, Sinha S, 2015. Personalized federated search at LinkedIn. Proc 24th ACM Int Conf on Information and Knowledge Management, p.1699-1702. https://doi.org/10.1145/2806416.2806615

Atallah MJ, Blanton M, Fazio N, et al., 2009. Dynamic and efficient key management for access hierarchies. *ACM Trans Inform Syst Secur*, 12(3), Article 18. https://doi.org/10.1145/1455526.1455531

Ateniese G, de Santis A, Ferrara AL, et al., 2006. Provably-secure time-bound hierarchical key assignment schemes. Proc 13th ACM Conf on Computer and Communications Security, p.288-297. https://doi.org/10.1145/1180405.1180441

Banu AS, 2015. Efficient data sharing in cloud medium with key aggregate cryptosystem. *Netw Commun Eng*, 7(3):118-121.

Bao F, Deng RH, Ding XH, et al., 2008 Private query on encrypted data in multi-user settings. Proc 4th Int Conf on Information Security Practice and Experience, p.71-85. https://doi.org/10.1007/978-3-540-79104-1_6

Bethencourt J, Sahai A, Waters B, 2007. Ciphertext-policy attribute-based encryption. Proc Symp on Security and Privacy, p.321-334. https://doi.org/10.1109/SP.2007.11

Boneh D, di Crescenzo G, Ostrovsky R, et al., 2004. Public key encryption with keyword search. Int Conf on the Theory and Applications of Cryptographic Techniques, p.506-522. https://doi.org/10.1007/978-3-540-24676-3_30

Chame SD, Kumar A, 2015. A noval approach key aggregate cryptosystem for resizable data sharing in cloud storage. *Int Res J Eng Technol*, 7(2):508-512.

Chang YC, Mitzenmacher M, 2005. Privacy preserving keyword searches on remote encrypted data. Proc 3rd Int Conf on Applied Cryptography and Network Security, p.442-455. https://doi.org/10.1007/11496137_30

Cheon JH, 2006. Security analysis of the strong Diffie-Hellman problem. Proc 24th Annual Int Conf on the Theory and Applications of Cryptographic Techniques, p.1-11. https://doi.org/10.1007/11761679_1

Cheung L, Newport C, 2007. Provably secure ciphertext policy ABE. Proc 14th ACM Conf on Computer and Communications Security, p.456-465. https://doi.org/10.1145/1315245.1315302

Chu CK, Chow SSM, Tzeng WG, et al., 2014. Key-aggregate cryptosystem for scalable data sharing in cloud storage. *IEEE Trans Parall Distrib Syst*, 25(2):468-477. https://doi.org/10.1109/TPDS.2013.112

Cui BJ, Liu ZL, Wang LY, 2016. Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage. *IEEE Trans Comput*, 65(8):2374-2385. https://doi.org/10.1109/TC.2015.2389959

Curtmola R, Garay J, Kamara S, et al., 2011. Searchable symmetric encryption: improved definitions and efficient constructions. *J Comput Secur*, 19(5):895-934. https://doi.org/10.3233/JCS-2011-0426

Daemen J, Rijmen V, 2001. The Design of Rijndael. AES—the Advanced Encryption Standard. Springer Berlin Heidelberg.

Dang H, Chong YL, Brun F, et al., 2016. Practical and scalable sharing of encrypted data in cloud storage with key aggregation. Proc 4th ACM Workshop on Information Hiding and Multimedia Security, p.69-80. https://doi.org/10.1145/2909827.2930795

Daza V, Herranz J, Morillo P, et al., 2010. Extensions of access structures and their cryptographic applications. *Appl Algebr Eng Commun Comput*, 21(4):257-284. https://doi.org/10.1007/s00200-010-0125-1

de Caro A, Iovino V, 2011. jPBC: Java pairing based cryptography. Proc Symp on Computers and Communications, p.850-855. https://doi.org/10.1109/ISCC.2011.5983948

Diaz F, Lalmas M, Shokouhi M, 2010. From federated to aggregated search. Proc 33rd Int ACM SIGIR Conf on Research and Development in Information Retrieval, p.910. https://doi.org/10.1145/1835449.1835682

Dodis Y, Fazio N, 2003. Public key broadcast encryption for stateless receivers. ACM CCS-9 Workshop on Digital Rights Management, p.61-80. https://doi.org/10.1007/97835404499355

Fiat A, Naor M, 1993. Broadcast encryption. Proc 13th Annual Int Cryptology Conf on Advances in Cryptology, p.480-491. https://doi.org/10.1007/3-540-48329-2_40

Firdose HF, Rebekah RDC, 2015. A key aggregate construction with adaptable offering of information in cloud. *Int J Comput Eng Res Trends*, 2(5):355-358.

Goh EJ, 2003. Secure Indexes. Cryptology ePrint Archive, Report 2003/216. https://eprint.iacr.org/2003/216

Goyal V, Pandey O, Sahai A, et al., 2006. Attribute-based encryption for fine-grained access control of encrypted data. Proc 13th ACM Conf on Computer and Communications Security, p.89-98. https://doi.org/10.1145/1180405.1180418

Guo C, Luo NQ, Bhuiyan ZA, et al., 2017. Key-aggregate authentication cryptosystem for data sharing in dynamic cloud storage. Proc 14th Int Symp on Pervasive Systems, Algorithms and Networks & 11th Int Conf on Frontier of Computer Science and Technology & 3rd Int Symp of Creative Computing, p.242-249. https://doi.org/10.1109/ISPAN-FCST-ISCC.2017.43

Huang HP, Du JP, Wang H, et al., 2016. A multi-keyword multi-user searchable encryption scheme based on cloud storage. Proc IEEE Trustcom/BigDataSE/ISPA, p.1937-1943. https://doi.org/10.1109/trustcom.2016.0296

Hwang YH, Lee PJ, 2007. Public key encryption with conjunctive keyword search and its extension to a multi-user system. Proc 1st Int Conf on Pairing-Based Cryptography, p.2-22. https://doi.org/10.1007/978-3-540-73489-5_2

Kiayias A, Oksuz O, Russell A, et al., 2016. Efficient encrypted keyword search for multi-user data sharing. Proc 21st European Symp on Research in Computer Security, p.173-195. https://doi.org/10.1007/978-3-319-45744-4_9

Kurosawa K, Yoshida T, Desmelt Y, 2000. Inherently large traceability of broadcast encryption scheme. Proc IEEE Int Symp on Information Theory, p.464. https://doi.org/10.1109/isit.2000.866762

Lambhate S, Patil S, 2016. A survey on cloud group data sharing using key-aggregate searchable encryption (KASE) scheme. *Int J Sci Res Sci Eng Technol*, 2(1): 182-185.

Li T, Liu ZL, Li P, et al., 2016. Verifiable searchable encryption with aggregate keys for data sharing in outsourcing storage. Proc 21st Australasian Conf on Information Security and Privacy, p.153-169. https://doi.org/10.1007/978-3-319-40367-0_10

Li T, Liu ZL, Jia CF, et al., 2018. Key-aggregate searchable encryption under multi-owner setting for group data sharing in the cloud. *Int J Web Grid Serv*, 14(1):21-43. https://doi.org/10.1504/IJWGS.2018.088358

Liao ZH, Wang JM, Lang B, 2013. Ciphertext-policy hidden vector encryption for multi-user keyword search. Proc 3rd Int Conf on Internet & Cloud Computing Technology.

Mahalle RV, Pawade PP, 2014. A review of secure data sharing in cloud using key aggregate cryptosystem and decoy technology. *Int J Sci Res*, 3(12):2694-2697.

Massonet P, Levin A, Celesti A, et al., 2015. Security requirements in a federated cloud networking architecture. Workshops of ESOCC Advances in Service-Oriented and Cloud Computing, p.79-88. https://doi.org/10.1007/978-3-319-33313-7_6

Padhya M, Jinwala D, 2014. A novel approach for searchable CP-ABE with hidden ciphertext-policy. Proc 10th Int Conf on Information Systems Security, p.167-184. https://doi.org/10.1007/978-3-319-13841-1_10

Pansare N, Somkuwar A, Shaikh A, et al., 2016. Key-aggregate searchable encryption (KASE) for user revocation in cloud storage. *Int J Eng Tech*, 2(1):68-70.

Park JH, Lee DH, 2008. A new public key broadcast encryption using Boneh-Boyen-Goh's HIBE scheme. Proc 4th Int Conf on Information Security Practice and Experience, p.101-115. https://doi.org/10.1007/978-3-540-79104-1_8

Patranabis S, Shrivastava Y, Mukhopadhyay D, 2015. Dynamic key-aggregate cryptosystem on elliptic curves for online data sharing. Proc 16th Int Conf in Cryptology in India Progress in Cryptology, p.25-44. https://doi.org/10.1007/978-3-319-26617-6_2

Patranabis S, Shrivastava Y, Mukhopadhyay D, 2017. Provably secure key-aggregate cryptosystems with broadcast aggregate keys for online data sharing on the cloud. *IEEE Trans Comput*, 66(5):891-904. https://doi.org/10.1109/TC.2016.2629510

Pawar PS, Sajjad A, Dimitrakos T, et al., 2015. Security-as-a-service in multi-cloud and federated cloud environments. Proc 9th IFIP Int Conf on Trust Management, p.251-261. https://doi.org/10.1007/978-3-319-18491-3_21

Pirretti M, Traynor P, McDaniel P, et al., 2010. Secure attribute-based systems. *J Comput Secur*, 18(5):799-837.

Popa RA, Zeldovich N, 2013. Multi-key Searchable Encryption. Cryptology ePrint Archive, Report 2013/508. https://eprint.iacr.org/2013/508

Ragab-Hassen H, 2010. Efficient key management model and scheme for content access control in hierarchies. Proc IEEE Globecom Workshop, p.1492-1496. https://doi.org/10.1109/glocomw.2010.5700187

Rivest RL, Shamir A, Adleman L, 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun ACM*, 21(2):120-126. https://doi.org/10.1145/359340.359342

Rouselakis Y, Waters B, 2015. Efficient statically-secure large-universe multi-authority attribute-based encryption. Proc 19th Int Conf on Financial Cryptography and Data Security, p.315-332. https://doi.org/10.1007/978-3-662-47854-7_19

Sahai A, Waters B, 2005. Fuzzy identity-based encryption. Proc 24th Annual Int Conf on the Theory and Applications of Cryptographic Techniques Advances in Cryptology, p.457-473. https://doi.org/10.1007/11426639_27

Shim KA, 2012. CPAS: an efficient conditional privacy-preserving authentication scheme for vehicular sensor networks. *IEEE Trans Veh Technol*, 61(4):1874-1883. https://doi.org/10.1109/TVT.2012.2186992

Singhal A, 2001. Modern information retrieval: a brief overview. *IEEE Data Eng Bull*, 24(4):35-43.

Sinha A, Kale CV, Douglas JL, et al., 2015. Federated Search. US Patent App. 14/503, 138.

Song DX, Wagner D, Perrig A, 2000. Practical techniques for searches on encrypted data. Proc IEEE Symp on Security and Privacy, p.44-55. https://doi.org/10.1109/SECPRI.2000.848445

Soubhagya B, Mini VG, Celin JA, 2013. A homomorphic encryption technique for scalable and secure sharing of personal health record in cloud computing. *Int J*

Comput Appl, 67(11):40-44.
https://doi.org/10.5120/11443-7344

Sumalatha MR, Begam MBR, Priya ED, et al., 2015. Secure data sharing using aggregate key for sensitive data. *Int Res J Eng Technol*, 2(4):40-45.

Sun WH, Yu SC, Lou WJ, et al., 2014. Protecting your right: attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. Proc IEEE Conf on Computer Communications, p.226-234.
https://doi.org/10.1109/INFOCOM.2014.6847943

Wang CJ, Li WT, Li Y, et al., 2013. A ciphertext-policy attribute-based encryption scheme supporting keyword search function. Proc 5th Int Symp on Cyberspace Safety and Security, p.377-386.
https://doi.org/10.1007/978-3-319-03584-0_28

Wang PS, Wang HX, Pieprzyk J, 2007. Common secure index for conjunctive keyword-based retrieval over encrypted data. Proc 4th VLDB Workshop on Secure Data Management, p.108-123.
https://doi.org/10.1007/978-3-540-75248-6_8

Wang PS, Wang HX, Pieprzyk J, 2008a. Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups. In: Franklin MK, Hui LCK, Wong DS (Eds.), Cryptology and Network Security. Springer Berlin Heidelberg, p.178-195.
https://doi.org/10.1007/978-3-540-89641-8_13

Wang PS, Wang HX, Pieprzyk J, 2008b. Threshold privacy preserving keyword searches. Proc Int Conf on Current Trends in Theory and Practice of Computer Science, p.646-658.
https://doi.org/10.1007/978-3-540-77566-9_56

Wang ZW, 2019. Provably secure key-aggregate cryptosystems with auxiliary inputs for data sharing on the cloud. *Fut Gener Comput Syst*, 93:770-776.
https://doi.org/10.1016/j.future.2017.09.041

Wang ZW, Zhou LY, 2016. Leakage-resilient key-aggregate cryptosystem with auxiliary input. Proc 25th Int Conf on Computer Communication and Networks, p.1-5.
https://doi.org/10.1109/ICCCN.2016.7568536

Wang ZW, Cao C, Yang NH, et al., 2017. ABE with improved auxiliary input for big data security. *J Comput Syst Sci*, 89:41-50. https://doi.org/10.1016/j.jcss.2016.12.006

Waters B, 2011. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In: Catalano D, Fazio N, Gennaro R, et al. (Eds.), Public Key Cryptography-PKC 2011. Springer Berlin Heidelberg, p.53-70.
https://doi.org/10.1007/978-3-642-19379-8_4

Xiong AP, Gan QX, He XX, et al., 2013. A searchable encryption of CP-ABE scheme in cloud storage. Proc 10th Int Computer Conf on Wavelet Active Media Technology and Information Processing, p.345-349.
https://doi.org/10.1109/ICCWAMTIP.2013.6716664

Yang ZQ, Zhong S, Wright RN, 2006. Privacy-preserving queries on encrypted data. European Symp on Research in Computer Security, p.479-495.
https://doi.org/10.1007/11863908_29

Yao DF, Fazio N, Dodis Y, et al., 2004. ID-based encryption for complex hierarchies with applications to forward security and broadcast encryption. Proc 11th ACM Conf on Computer and Communications Security, p.354-363.
https://doi.org/10.1145/1030083.1030130

Zhang LH, Yang WH, Liao LZ, 2013. On an efficient implementation of the face algorithm for linear programming. *J Comput Math*, 31(4):335-354.

Zhou R, Zhang XS, Du XJ, et al., 2018. File-centric multi-key aggregate keyword searchable encryption for industrial Internet of Things. *IEEE Trans Ind Inform*, 14(8):3648-3658. https://doi.org/10.1109/TII.2018.2794442