Verifiable Key-Aggregate Searchable Encryption with a Designated Server in Multi-Owner Setting

Jinlu Liu, Zhongkai Wei, Jing Qin, Bo Zhao, Jixin Ma

Abstract—Key-aggregate searchable encryption (KASE) schemes support selective data sharing and keyword-based ciphertext searching by using the constant-size shared key and trapdoor, making these schemes attractive for resource-constrained users to store, share, and search encrypted data in public clouds. However, most previously proposed KASE schemes suffer from our proposed "off-line keyword guessing attack (KGA)" and some other weaknesses. Consequently, they fail to gain the keyword ciphertext indistinguishability and trapdoor indistinguishability, which are vital security goals of searchable encryption. Inspired by the relationship of public key encryption with keyword search (PEKS) and KASE, we design a new KASE scheme called key-aggregate searchable encryption with a designated server (dKASE). The dKASE scheme achieves our proposed keyword ciphertext indistinguishability against chosen keyword attack (KC-IND-CKA) and keyword trapdoor indistinguishability against keyword guessing attack (KT-IND-KGA) security models, where the latter model captures off-line KGA. Then, we extend the dKASE scheme to verifiable dKASE in multi-owner setting (dVKASEM) scheme. With dVKASEM, when multiple data owners authorize a user to access data, the user merely needs to store his single key and generate a single trapdoor to query these owners' data. Besides, the adoption of the aggregate signature significantly reduces the overhead of verifying whether data has been tampered with. Performance analysis illustrates that our schemes are efficient.

Index Terms—Selective data sharing, key-aggregate searchable encryption, KGA, multi-owner, verifiable

1 Introduction

Service computing plays an important role in modern computing and has a profound impact on building flexible, scalable, and reusable distributed applications and systems. Cloud computing plays a pivotal role in the realm of service computing, enhancing its capabilities and enabling the realization of its principles. It offers the necessary infrastructure, tools, and services that align with the principles of service computing, enabling the creation, deployment, and management of reusable and distributed software services. With the popularity of cloud computing, large amounts of local data are increasingly being outsourced to remote cloud servers to reduce the burden of data management and maintenance [1], [2]. Besides, to protect data privacy against untrusted servers, users opt to store their encrypted data on the cloud, thereby presenting a challenge in achieving efficient data utilization. The proposal of searchable encryption (SE) technology [3], [4] solves the problem of ciphertext retrieval. This technology enables data owners to upload encrypted data to the cloud securely. The cloud server then conducts searches based on trapdoors provided by authorized users. Subsequently, the server returns matched documents to users without compromising the confidentiality of the underlying plaintext data.

In addition to providing users with convenient data storage services, another vital function of the cloud system is data sharing. To ensure the effective utilization and management of data resources within the proper scope, the ability to selectively share encrypted data becomes an essential part. In essence, this entails allowing the data owner to share distinct subsets of ciphertext with different users. The desired selective sharing demands distinct encryption keys for individual documents. This implies that the expenses associated with securely distributing the key by the data owner, the secure storage of the key and the computation of the trapdoor by the user all escalate proportionally with the count of shared documents. This approach will be impractical as the quantity of shared documents grows. To further illustrate this problem, we consider the following concrete collaborative data sharing scenario.

Multiple hospitals intend to use a cloud service to share patients' data for collaboration. For example, hospital A_1 uploads all of its patients' data to cloud storage. Suppose all data is divided into the following classes based on disease type: $contagion$, $nervous\ system$, $digestive\ system$, $circulatory\ system$, and $dermatosis$. Since these data contain highly sensitive information of patients, A_1 encrypts distinct data classes using different keys before uploading to cloud. Additionally, keyword ciphertext is generated based on the disease name, enabling efficient data searches when needed. At some point, A_1 aims to share data subset $S_1 = \{contagion, digestive\ system, circulatory\ system\}$

- *Jinlu Liu is with School of Mathematics, Shandong University, Jinan, 250100, Shandong, China. E-mail: jinlulmath@163.com*
- *Zhongkai Wei is with School of Mathematics, Shandong University, Jinan, 250100, Shandong, China. E-mail: 17865196829@163.com*
- *Jing Qin is with School of Mathematics, Shandong University, Jinan, 250100, Shandong, China. She is also with Skate Key Laboratory of Cryptology, P.O.Box 5159, Beijing, 100878, China. E-mail: qinjing@sdu.edu.cn*
- *Bo Zhao is with Chip and Security Department, Huakong Qingjiao Information Technology Co., Ltd., Beijing, 100080, China. E-mail: zhaobomath@163.com*
- *Jixin Ma is with School of Computing and Mathematical Sciences University of Greenwich, London, UK. E-mail:j.ma@greenwich.ac.uk.*

Corresponding author: Jing Qin.

with hospital R to collaborate (Subsequently, A_1 may also need to share data subset S_1' to hospital R', share data subset S_1'' to hospital R'', and so on). It needs to distribute keys of these classes to R, and R must store these keys securely and generate trapdoors for each class in set S_1 using the received keys.

We observe that key-aggregate searchable encryption (KASE) [5] is an excellent mechanism to address the above-mentioned challenge. In KASE, the data owner can extract an aggregate key from any number of different classes using his/her secret key. This aggregate key is a single key but aggregates the ability to search ciphertexts of these different classes. With KASE, to share data subset S_1, hospital A_1 just securely provides a single aggregate key K_{S_1} to R. Subsequently, R solely submits a single trapdoor generated by K_{S_1} to the server to query these shared data.

However, we find that most previous KASE schemes suffer from the security vulnerability caused by the off-line keyword guessing attack (KGA). Specifically, when the channel between the user and the server is public, the trapdoors that the user submits to the server are also publicly available. Since keywords are usually chosen from low-entropy spaces, an attacker, after obtaining a trapdoor, can exhaustively guess the keyword and off-line determine whether the guessed keyword is correct or not through some equations to reveal the keyword corresponding to the trapdoor. After revealing the keyword, the attacker can proceed to compute the aggregate key used to generate the trapdoor, to access the data classes corresponding to the aggregate key. Besides, Kiayias et al. [6] showed that the ciphertexts of most KASE schemes are vulnerable to the cross pairing attacks, so the ciphertext cannot achieve indistinguishable security. Further, if many other hospitals, like A_2, A_3, and A_4, also separately share patients' data subset S_2, S_3, and S_4 with hospital R, then R must securely store aggregate keys sent by these hospitals and generate separate trapdoors to access each hospital's data. That is, in the multi-owner setting, the user's overhead increases linearly with the number of data owners, which is challenging for resource-constrained users. In addition, due to financial incentives, the server in a commercial cloud computing environment may temper with the data. Thereby, the patient data received by hospital R might have been tempered with. Therefore, it is necessary to design a mechanism to verify the correctness of returned data by the server. A trivial way is that the data owner generates a signature for each ciphertext data and outsources these signatures along with the keyword ciphertexts to the cloud server. When the cloud server returns matched ciphertexts to the user, it also returns the corresponding signatures. Then, the user can verify the correctness of each data in turn using signatures. However, the computation and communication overhead is linearly related to the amount of matched data, which is impractical.

Therefore, it is imperative to propose a KASE scheme that can uphold the following properties simultaneously: (1) resist off-line KGA and meet ciphertext indistinguishability; (2) keep the user's overhead at a constant size in multi-owner setting; (3) can efficiently verify whether the server has tampered with the ciphertext in the commercial cloud computing environment. Regrettably, no such KASE scheme has been proposed thus far.

1.1 Our Contributions

To achieve secure keyword searching and selective data sharing, we propose a novel KASE system named key-aggregate searchable encryption with a designated server (dKASE). We also present two security models to capture the security of keyword ciphertext and trapdoor respectively, and a concrete KASE scheme. Further, to reduce the overhead of users in multi-owner setting and efficiently verify the correctness of returned data when the cloud server may maliciously tamper with the data, an extended scheme named verifiable key-aggregate searchable encryption with a designated server in multi-owner setting (dVKASEM) is proposed. The formal security analysis proves that our schemes are secure under the proposed security models. Finally, we conduct performance analysis to demonstrate the efficiency of our schemes. Compared with previous KASE schemes, our study mainly has the following contributions:

- *Ciphertext indistinguishability and trapdoor indistinguishability.* Our schemes overcome the security weaknesses discussed earlier. Concretely speaking, they are provably secure under the proposed "keyword ciphertext indistinguishability against chosen keyword attack (KC-IND-CKA)" and "keyword trapdoor indistinguishability against keyword guessing attack (KT-IND-KGA)" security models. We formally define the security against off-line KGA executed by outside attackers (unauthorized users) for the dKASE system. Our KT-IND-KGA security model captures this attack, and analysis indicates that if a dKASE scheme is KT-IND-KGA secure, it can resist off-line KGA.
- *Constant-sized user's overhead in multi-owner setting.* Our extended dVKASEM scheme efficiently reduces the overhead of key management and trapdoor generation in the multi-owner setting. No matter how many data owners authorize access to the data, a user only needs to store his single key and use that key to generate a single trapdoor to search on multi-owner's data. Therefore, the user's overhead is independent of the number of data owners.
- *Efficient verifiable functionality.* In dVKASEM, we introduce the aggregate signature to verify the correctness of returned data. Consequently, regardless of how many matched data there are, the server only returns an aggregate signature to the user for verification. This approach eliminates the need to return individual signatures for each data, thus significantly reducing the computation and communication overhead of verification.

Fig.1 depicts the above hospital collaborative data sharing scenario using our dVKASEM.

1.2 Related Works

1.2.1 Public Key Encryption with Keyword Search against Keyword Guessing Attack

Boneh et al. [7] introduced the notion of PEKS and a concrete scheme. Since then, many PEKS schemes [8], [9] have been proposed. Byun et al. [10] raised that the recent PEKS schemes are vulnerable against the off-line KGA:

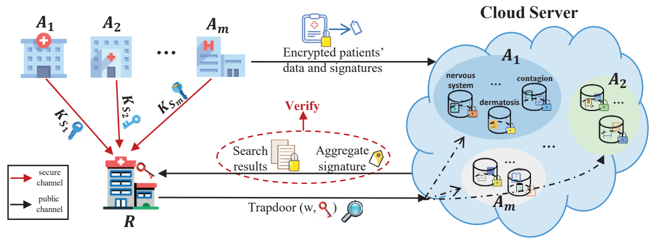


Fig. 1: Hospital collaborative data sharing with dVKASEM

TABLE 1: Functionality and Security Comparison with [25], [26].

	Multi-Owner	Cipher-ind	Trap-ind	Verifiable
[26]	✓	✗	✗	✗
[25]	✗	✗	✗	✓
dKASE	✗	KC-IND-CKA	KT-IND-KGA	✗
dVKASEM	✓	KC-IND-CKA	KT-IND-KGA	✓

Cipher-ind: Ciphertext indistinguishability;
Trap-ind: Trapdoor indistinguishability.

since keywords are usually selected from a low-entropy space, an attacker can recover the keyword from a given trapdoor by exhaustively guessing the keyword off-line. Rhee et al. [11] defined the concept of "trapdoor indistinguishability" and proved that trapdoor indistinguishability is a sufficient condition for resisting off-line KGA of outside adversaries. They also designed a searchable public-key encryption scheme with a designated tester (dPEKS) scheme that satisfies ciphertext indistinguishability and trapdoor indistinguishability. Xu et al. [12] proposed a public-key encryption with fuzzy keyword search scheme to resist off-line KGA from outside adversaries. However, these schemes can not resist the KGA from inside adversaries.

In 2016, Chen et al. [13] proposed a dual-server PEKS scheme to solve the KGA problem from malicious server. Huang and Li [14] introduced a primitive called public authenticated encryption with keyword search (PAEKS) and a corresponding concrete scheme. In PAEKS, the generation of keyword ciphertext needs the secret key of the sender, so that the server cannot encrypt a keyword. Noroozi and Eslami [15] showed that the Huang and Li's PAEKS scheme cannot resist the KGA in the multi-user setting. They also proposed new security model and concrete scheme to fix the problem. Qin et al. [16] revisited the security model of [14] and pointed out that this model did not capture the chosen multi-ciphertext attacks. Therefore, they proposed an improved security model that captures the KGA and chosen multi-ciphertext attacks. They also constructed a PAEKS scheme meeting the proposed security model. In 2022, Liu et al. [17] proposed a quantum-resistant PAEKS scheme based on lattices.

In addition to off-line KGA, how to effectively resist on-line KGA by malicious data senders is also a research hotspot of PEKS [18], [19].

1.2.2 *Key-Aggregate Searchable Encryption*

KASE [5] is a novel cryptographic primitive proposed based on the key-aggregate cryptosystem (KAC) [20]. Recently, Chu et al. [20] first proposed a KAC scheme using the broadcast encryption scheme [21]. In KAC, the data owner encrypts each document class using different keys. When the owner intends to share multiple classes with a user, the corresponding keys can be aggregated and sent to the user. This aggregated key empowers the user to decrypt any documents in these classes. But Chu et al.'s scheme doesn't permit searching. In 2015, Cui et al. [5] combined the SE with KAC and presented the KASE.

Li et al. [22] devised a verifiable KASE scheme using bloom filter (BF) [23] to verify the correctness and completeness of search results. However, as keyword ciphertext and BFs are generated without utilizing secret keys, the server can forge the search results at will. This renders the verification of search results unfeasible [24]. In 2020, [24] proposed a verifiable KASE using BF and Merkle hash tree. Considering that the aggregate key may be used by unauthorized users, a verifiable and authenticated KASE scheme using digital signature was proposed [25]. However, the user needs to interact with the server through two rounds to verify. Liu et al. [26] constructed a verifiable KASE scheme under multi-owner setting based on [22]. The user only needs to securely keep his secret key locally, even if numerous data owners share documents with him. However, its verification mechanism is the same as [22].

Although [5], [22], [24], [25], [26] all realize the functionality of KASE, they did not give security proofs. Moreover, the underlying schemes of [22], [24], [25], [26] are all [5]. Whereas, Kiayias et al. [6] show that [5] is vulnerable to cross pairing attack, so the ciphertext cannot achieve indistinguishable security. Zhou et al. [27] present a trapdoor attack on [5] in which one user can get the aggregate key of another user if two users have search permission for a same document class. This attack is more restrictive than our proposed KGA because anyone can execute the off-line KGA. Additionally, [27] introduced a new aggregate keyword searchable encryption system. The system parameters of the above KASE schemes have a linear relationship with the document class number. The computation cost is large. To address this drawback and the trapdoor attack mentioned in [27], Wang et al. [28] proposed a scheme where the size of system parameters is constant. But there is a linear relationship between the size of keyword ciphertext and the number of document classes, and there is no formal security proof. The aforementioned schemes only supports single keyword search. Recently, Liu et al. [29] proposed a KASE supporting conjunctive queries (KASE-CQ) framework and a concrete KASE-CQ scheme.

We show the comparison between our scheme and existing related KASE schemes in terms of functionality and security in TABLE 1.

2 Preliminaries

2.1 Notations

We let λ denote the security parameter, n denote the maximum number of document classes belonging to a data owner, and PPT denote the probabilistic polynomial time.

2.2 Bilinear Pairings

Let $\mathcal{G}$ and $\mathcal{G}_T$ be two cyclic multiplicative groups of prime order p. Let g be a generator of $\mathcal{G}$. A bilinear pairing is a map $e : \mathcal{G} \times \mathcal{G} \to \mathcal{G}_T$ with the following properties:

- Bilinearity: For any $g, h \in \mathcal{G}$, $\alpha, \beta \in Z_p$, $e(g^\alpha, h^\beta) = e(g, h)^{\alpha\beta}$ is true;
- Non-degeneracy: $e(g, g) \neq 1$;
- Computability: For any $g, h \in \mathcal{G}$, there is an efficient algorithm to calculate $e(g, h)$.

2.3 Hardness Assumptions

Definition 1 (BDH). *The BDH (Bilinear Diffie-Hellman) problem in $\mathcal{G}$ is as follows: given a 4-tuple (g, g^a, g^b, g^c), where a, b, c are randomly chosen from Z_p^*, compute $e(g, g)^{abc}$. The BDH assumption holds in $\mathcal{G}$ if for any PPT algorithm $\mathcal{A}$, there is a negligible function $negl(\cdot)$ such that*

$$Pr[\mathcal{A}(g, g^a, g^b, g^c) = g^{abc}] \leq negl(\lambda).$$

Definition 2 (HDH). *The HDH (Hash Diffie-Hellman) problem in $\mathcal{G}$ is stated as follows: given the hash function $H : \{0,1\}^* \to \{0,1\}^{hLen}$ and a 4-tuple $(g, g^a, g^b, H(g^c))$, where $hLen$ is a number, a and b are randomly chosen from Z_p^*, output "1" if $c = a \cdot b$, output "0" otherwise. The HDH assumption holds in $\mathcal{G}$ if for any PPT algorithm $\mathcal{A}$, there is a negligible function $negl(\cdot)$ such that*

$$|Pr[\mathcal{A}(g, g^a, g^b, H(g^{a \cdot b})) = "1"] \\ - Pr[\mathcal{A}(g, g^a.g^b, \eta) = "1"]| \leq negl(\lambda),$$

where η is a random bit string of length $hLen$.

2.4 Aggregate Signatures

A digital signature that permits aggregation is known as an aggregate signature [30]. We briefly recall the bilinear aggregate signature (BAS) introduced by Boneh et al. [30].

Definition 3 (BAS). *A BAS scheme contains the following PPT algorithms, where $h : \{0,1\}^* \to \mathcal{G}^*$ is a hash function.*

- ***KeyGen.*** *For user u_i, select $\beta_i \in Z_p^*$ at random and calculate $v_i = g^{\beta_i}$. Finally, set u_i's public key to v_i and secret key to β_i.*
- ***Sign.*** *For user u_i, given β_i and a message $m_i \in \{0,1\}^*$, calculate $h_i \leftarrow h(m_i)$ and the signature $\sigma_i \leftarrow h_i^{\beta_i}$.*
- ***Verify.*** *Given a message m, a signature σ, and a public key v, the verifier calculates $h \leftarrow h(m)$ and verifies that $e(g, \sigma) = e(v, h)$ holds. If so, then σ is valid.*
- ***Aggregate.*** *Each u_i in aggregate user set $U = \{u_i\}_{i=1}^k$ offers a signature σ_i of m_i. The m_i must all be different. Compute the aggregate signature as $\sigma \leftarrow \prod_{i=1}^k \sigma_i$.*
- ***AggregateVerify.*** *Given an aggregate signature σ for an aggregate user set U, and original message m_i and public key v_i for each user $u_i \in U$, verify σ:*
  a) *check whether m_i are all different, reject if there is the same, otherwise*
  b) *compute $h_i \leftarrow h(m_i)$ for $1 \leq i \leq k$, and if $e(g, \sigma) = \prod_{i=1}^k e(v_i, h_i)$, then σ is valid.*

Remark 1. *The k signatures can be issued by the same public key v. In this case, one just needs to verify $e(g, \sigma) = e(v, \prod_{i=1}^k h_i)$, and the "**AggregateVerify**" is faster.*

Remark 2. *The "**Aggregate**" can be done incrementally, which means that other signatures can be aggregated based on the already obtained aggregated signature.*

3 Keyword Guessing Attack on Cui et al.'s Scheme

3.1 The Brief Description of Cui et al.'s Scheme

- $Setup(1^\lambda, n)$. Initialize system as follows:
  a) Generate a bilinear map group system $\mathcal{PG} = (p, \mathcal{G}, \mathcal{G}_T, e(\cdot,\cdot))$, where p is the order of $\mathcal{G}$.
  b) Randomly select a generator $g \in \mathcal{G}$ and $\alpha \in Z_p^*$, and compute $g_i = g^{(\alpha^i)}$ for $i = 1, 2, ..., n, n+2, ..., 2n$.
  c) Pick a hash function $H : \{0,1\}^* \to \mathcal{G}^*$.

  Finally, the system parameter is:
  $$param = \{\mathcal{PG}, (g, g_1, ..., g_n, g_{n+2}, ..., g_{2n}), H\}.$$
- $KeyGen$. The data owner randomly selects $\gamma \in Z_p^*$, and sets his master secret key as $msk = \gamma$ and public key as $pk = v = g^\gamma$.
- $Encrypt(pk, j)$. For a document belonging to class j ($j \in [1, n]$), the data owner computes its keyword ciphertext $\{c_1, c_2, c_w\}$ as follows:
  a) Pick a random $t \in Z_p^*$.
  b) Compute the auxiliary value
  $$\Delta_j = \{c_1, c_2\} = \{g^t, (v \cdot g_j)^t\}.$$
  c) Compute keyword w's ciphertext
  $$c_w = e(g, H(w))^t / e(g_1, g_n)^t.$$
- $Extract(msk, S)$. The data owner computes the aggregate key for a subset $S \subseteq [1, n]$ to be $K_S = \prod_{l \in S} g_{n+1-l}^\gamma$ and securely sends $\{K_S, S\}$ to the user.
- $Trapdoor(K_S, w)$. The user generates the trapdoor $Tr = K_S \cdot H(w)$ for keyword w and submits $\{Tr, S\}$ to the server.
- $Adjust(j, S, Tr)$. The cloud server adjusts Tr to actual trapdoor $Tr_j = Tr \cdot \prod_{l \in S, l \neq j} g_{n+1-l+j}$ of document class j ($j \in S$).
- $Test(Tr_j, j)$. For each document in class j, the cloud server determines whether $c_w = \frac{e(Tr_j, c_1)}{e(pub, c_2)}$ (where $pub = \prod_{l \in S} g_{n+1-l}$) is true and outputs "1" if so, "0" otherwise.

3.2 Security Vulnerability

Lemma 1. *Cui et al.'s scheme is vulnerable to off-line KGA.*

Proof: An attacker $\mathcal{A}$ performs off-line KGA as follows:

1. $\mathcal{A}$ captures a trapdoor $Tr = K_S \cdot H(w)$ and its corresponding document class index set S.
2. $\mathcal{A}$ selects a keyword w' and computes $e(g, H(w'))e(pk, \prod_{l \in S} g_{n+1-l})$.
3. $\mathcal{A}$ tests if $e(g, H(w'))e(pk, \prod_{l \in S} g_{n+1-l}) = e(g, Tr)$ is true. If so, output w'; Otherwise, go to step 2.
4. $\mathcal{A}$ computes $Tr/H(w')$ to get the aggregate key K_S.

Since keywords are always picked from space with low entropy, the attacker will be able to accomplish off-line KGA in a reasonable time. □

Remark 3. *We notice that anyone (inside/outside attackers) can carry out the above keyword guessing attack.*

Remark 4. *The trapdoors of [22], [24], [25], [26] are identical to that of [5], so the attack works for them as well.*

4 Problem Formulation

4.1 System & Threat Model

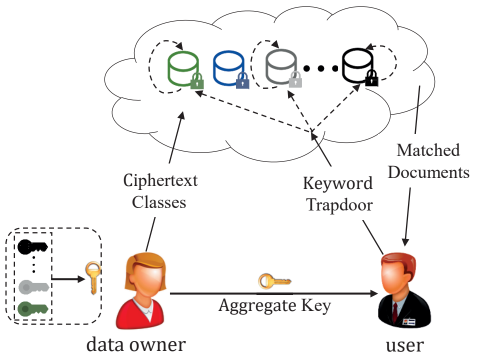


Fig. 2: System Model of dKASE

The dKASE system model is shown in Fig. 2. A dKASE system consists of three types of entities: data owners, users, and the cloud server. We introduce each entity as follows.

- Data Owner. The entity classifies its documents and encrypts different classes using different keys, then outsources the ciphertext to the cloud server. In addition, the entity also generates the aggregate key of the shared document classes set and sends it to the user through the secure channel.
- User. After receiving the aggregate key, this entity can issue search queries based on its interested keywords for the ciphertext classes corresponding to this aggregate key.
- Cloud Server. The entity stores ciphertexts from the data owner, performs the search operation for the user, and returns the matched ciphertext.

The data owner is credible. Data users may collude to get as much as possible secret information by combining the aggregate keys they received. Regarding the cloud server, in the basic dKASE scheme, we consider it to be a semi-trusted entity that honestly conducts the protocols but tries to spy out as much secret information as possible. In the extended dVKASEM scheme, we consider the more realistic commercial cloud computing environment, where the cloud server may maliciously tamper with the data.

4.2 System Definition of dKASE

Definition 4 (dKASE). *The dKASE system, as illustrated in Fig. 3, contains the following algorithms:*

- $\boldsymbol{param \leftarrow Setup(1^{\lambda}, n)}$ **:** *On input λ and n, the data owner initializes the system parameter param.*
- $\boldsymbol{(PK_o, MSK) \leftarrow KeyGen_o}$ **:** *The data owner outputs his public key and master secret key pair (PK_o, MSK).*
- $\boldsymbol{(PK_s, SK_s) \leftarrow KeyGen_s}$ **:** *The cloud server outputs its public key and secret key pair (PK_s, SK_s).*
- $\boldsymbol{CT_j \leftarrow Encrypt(PK_o, PK_s, j, w)}$ **:** *On input PK_o, PK_s, and a document belonging to j-th class and its keyword w, the data owner generates keyword ciphertext CT_j of this document.*
- $\boldsymbol{K_S \leftarrow Extract(MSK, S)}$ **:** *The data owner inputs MSK and a set $S \subseteq [1, n]$, and then outputs the corresponding aggregate key K_S. Finally, the data owner securely transmits $\{K_S, S\}$ to the user.*
- $\boldsymbol{Tr \leftarrow Trapdoor(K_S, PK_s, w)}$ **:** *On input K_S, PK_s, and a keyword w, the user generates a trapdoor Tr and transmits $\{Tr, S\}$ to the cloud server.*
- $\boldsymbol{1/0 \leftarrow Test(SK_s, CT_j, Tr)}$ **:** *On input Tr, SK_s, and a ciphertext CT_j ($j \in S$) under keyword w', the cloud server outputs "1" if $w' = w$, and "0" otherwise.*

4.3 Security Model

Security of dKASE Ciphertext

Now, we define dKASE's ciphertext security as "keyword ciphertext indistinguishability against chosen keyword attack (KC-IND-CKA)". This security model considers two types of attackers: cloud server and outside attacker (including users authorized by the data owner). The aim is to ensure that the server cannot distinguish between the ciphertexts of two challenge keywords, under the situation that it is allowed to query trapdoors for any non-challenge keywords; the outside attacker without SK_s unable to distinguish the ciphertexts of two challenge keywords even though the attacker has the trapdoors of all keywords.

Definition 5 (KC-IND-CKA). *Let $\mathcal{A}_i$ ($i = 1, 2$) be a PPT attacker. We define the following two games between $\mathcal{A}_1$ (or $\mathcal{A}_2$) and a challenger $\mathcal{C}$:*
Game 1: *Suppose $\mathcal{A}_1$ is the cloud server.*

Phase 1-1: *$\mathcal{C}$ runs $Setup(1^{\lambda}, n)$, $KeyGen_o$, and $KeyGen_s$ algorithms to generate param, (PK_o, MSK), and (PK_s, SK_s). Then it gives param, PK_o, and (PK_s, SK_s) to $\mathcal{A}_1$.*

Phase 1-2: *$\mathcal{A}_1$ outputs set $S^* \subseteq [1, n]$ to attack. $\mathcal{C}$ randomly picks $j \in S^*$ and sends it to $\mathcal{A}_1$.*

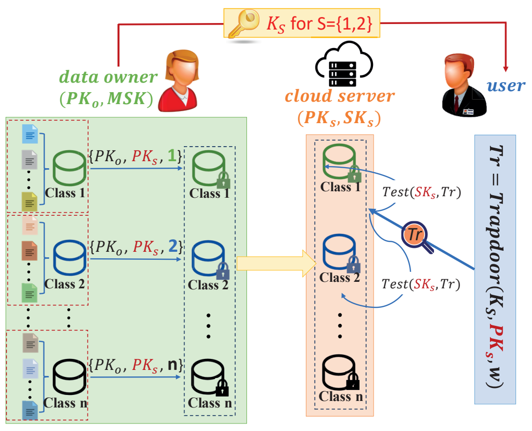


Fig. 3: The dKASE system framework

Phase 1-3: *$\mathcal{A}_1$ makes keyword trapdoor queries over the challenged set S^*. Each keyword queried is denoted as w and the trapdoor obtained is denoted as Tr_w.*

Phase 1-4: *$\mathcal{A}_1$ sends $\mathcal{C}$ two challenge keywords (w_0^*, w_1^*). The constraint is that neither w_0^* nor w_1^* is queried in Phase 1-3. $\mathcal{C}$ randomly picks $\beta \in \{0,1\}$ and computes a challenge ciphertext $CT_j^* = Encrypt(PK_o, PK_s, j, w_\beta^*)$ and returns it to $\mathcal{A}_1$.*

Phase 1-5: *$\mathcal{A}_1$ makes keyword trapdoor queries as in Phase 1-3 on condition that $w \neq w_0^*, w_1^*$.*

Phase 1-6: *$\mathcal{A}_1$ outputs its guess β' of β.*

We define the advantage of $\mathcal{A}_1$ winning Game 1 as $\boldsymbol{Adv_{\mathcal{A}_1}^{Game\ 1}(\lambda)} = |Pr[\beta' = \beta] - 1/2|$.

Game 2: *Suppose $\mathcal{A}_2$ is an outside attacker (including users authorized by the data owner)*

Phase 2-1: *$\mathcal{C}$ runs $Setup(1^\lambda, n)$, $KeyGen_o$, and $KeyGen_s$ algorithms to generate param, (PK_o, MSK), and (PK_s, SK_s). Then it gives param, PK_o, and PK_s to $\mathcal{A}_2$.*

Phase 2-2: *$\mathcal{A}_2$ outputs set $S^* \subseteq [1,n]$ to attack. $\mathcal{C}$ randomly picks $j \in S^*$, computes K_{S^*}, and sends $\{j, K_{S^*}\}$ to $\mathcal{A}_2$.*

Phase 2-3: *$\mathcal{A}_2$ sends $\mathcal{C}$ two challenge keywords (w_0^*, w_1^*). $\mathcal{C}$ randomly picks $\beta \in \{0,1\}$ and computes a challenge ciphertext $CT_j^* = Encrypt(PK_o, PK_s, j, w_\beta^*)$ and sends it back to $\mathcal{A}_2$.*

Phase 2-4: *$\mathcal{A}_2$ outputs its guess β' of β.*

We define the advantage of $\mathcal{A}_2$ winning Game 2 as $\boldsymbol{Adv_{\mathcal{A}_2}^{Game\ 2}(\lambda)} = |Pr[\beta' = \beta] - 1/2|$.

We say that a dKASE scheme is KC-IND-CKA secure if $\boldsymbol{Adv_{dKASE,\mathcal{A}_i}^{KC-IND-CKA}(\lambda)} = \boldsymbol{Adv_{\mathcal{A}_i}^{Game\ i}(\lambda)}$, where i is either 1 or 2, is negligible in λ.

Security of dKASE Trapdoor

In the following, we define dKASE's trapdoor security as "keyword trapdoor indistinguishability against keyword guessing attack (KT-IND-KGA)". The security model guarantees that an outside attacker (excluding the cloud server and authorized users) cannot distinguish the trapdoors of two challenge keywords, under the situation that it is allowed to collude with other outside attackers and query trapdoors for all non-challenge keywords.

Definition 6 (**KT-IND-KGA**). *Let $\mathcal{A}_3$ be a PPT attacker. We define the following game between $\mathcal{A}_3$ and a challenger $\mathcal{C}$.*
Game 3: *Suppose $\mathcal{A}_3$ is an outside attacker (excluding the cloud server and the authorized users).*

Phase 3-1: *$\mathcal{C}$ runs $Setup(1^\lambda, n)$, $KeyGen_o$, and $KeyGen_s$ algorithms to generate param, (PK_o, MSK), and (PK_s, SK_s). Then it gives param, PK_o, and PK_s to $\mathcal{A}_3$.*

Phase 3-2: *$\mathcal{A}_3$ outputs set $S^* \subseteq [1,n]$ to attack. $\mathcal{C}$ computes $K_{\overline{S^*}}$ and sends it to $\mathcal{A}_3$.*

Phase 3-3: *$\mathcal{A}_3$ makes keyword trapdoor queries over the challenged set S^*. Each keyword queried is denoted as w and the trapdoor obtained is denoted as Tr_w.*

Phase 3-4: *$\mathcal{A}_3$ sends $\mathcal{C}$ a pair of keywords (w_0^*, w_1^*). The restriction is that neither w_0^* nor w_1^* is queried in Phase 3-3. $\mathcal{C}$ randomly picks $\beta \in \{0,1\}$ and computes a challenge trapdoor $Tr^* = Trapdoor(K_{S^*}, PK_s, w_\beta^*)$ and returns it to $\mathcal{A}_3$.*

Phase 3-5: *$\mathcal{A}_3$ makes keyword trapdoor queries as in Phase 3-3 on condition that $w \neq w_0^*, w_1^*$.*

Phase 3-6: *$\mathcal{A}_3$ outputs its guess β' of β.*

We define the advantage of $\mathcal{A}_3$ winning Game 3 as $\boldsymbol{Adv_{\mathcal{A}_3}^{Game\ 3}(\lambda)} = |Pr[\beta' = \beta] - 1/2|$.

We say that a dKASE scheme is KT-IND-KGA secure if $\boldsymbol{Adv_{dKASE,\mathcal{A}_3}^{KT-IND-KGA}(\lambda)} = \boldsymbol{Adv_{\mathcal{A}_3}^{Game\ 3}(\lambda)}$ is negligible in λ.

Trapdoor security vs. off-line keyword guessing attack

Next, we will show that a dKASE scheme is secure against off-line KGA of the outside attacker if it is KT-IND-KGA secure. This is similar to dPEKS [11]. What is different is that in dPEKS, outside attackers refer to users other than server and receiver, while in our dKASE, outside attackers refer to unauthorized users, and these users may collude. We first define the security against off-line KGA in dKASE, and then explain its relationship with KT-IND-KGA.

We assume that $\mathcal{A}_4$ is an outside PPT attacker. It attempts to collude with other unauthorized users and perform off-line KGA on a dKASE scheme. In TABLE 2, we give a formal description of $\mathcal{A}_4$'s off-line KGA. We define $\mathcal{A}_4$'s success probability of off-line KGA as

$$Succ_{dKASE,\mathcal{A}_4}^{off-line\ KGA}(\lambda) = Pr[Exp_{dKASE,\mathcal{A}_4}^{off-line\ KGA}(\lambda) = 1].$$

TABLE 2: Off-line KGA in dKASE

$\boldsymbol{Exp_{dKASE,\mathcal{A}_4}^{off-line\ KGA}(\lambda)}$
$param \leftarrow Setup(1^\lambda, n)$;
$(PK_o, MSK) \leftarrow KeyGen_o$;
$(PK_s, SK_s) \leftarrow KeyGen_s$;
$K_{S^*} \leftarrow Extract(MSK, S^*)$;
$Tr \leftarrow Trapdoor(K_{S^*}, PK_s, w)$;
$K_{\overline{S^*}} \leftarrow Extract(MSK, \overline{S^*})$;
$w' \leftarrow \mathcal{A}_4(PK_o, PK_s, Tr, K_{\overline{S^*}})$;
if $w' = w$ then return 1 else return 0

Definition 7 (**Security against off-line KGA of dKASE**). *A dKASE scheme is secure against off-line KGA if for any outside attacker $\mathcal{A}_4$, $Succ_{dKASE,\mathcal{A}_4}^{off-line\ KGA}(\lambda)$ is negligible.*

Theorem 1. *If dKASE is KT-IND-KGA secure, it is secure against off-line keyword guessing attack.*

Proof: This proof is almost the same as that of dPEKS [11], so we omit it here. □

5 The Proposed dKASE Scheme

5.1 Design Rational

In PEKS, the sender uses the receiver's public key PK_r to encrypt keywords, and the receiver uses his secret key SK_r to generate trapdoors. In KASE, the data owner uses his public key PK_o to encrypt keywords, and the user utilizes an aggregate key K_S to generate trapdoors. As shown in Fig. 4, we observe that the user who obtained the aggregate key in KASE can be regarded as the receiver in PEKS, the operation of generating keyword ciphertext using PK_o in KASE can be viewed as generating keyword ciphertext using PK_r in PEKS, and the operation of generating trapdoor using K_S in KASE can be viewed as generating trapdoor using SK_r in PEKS.

Inspired by the above relationship between KASE and PEKS, it is to think of directly using the standard technology of PEKS to solve the off-line KGA of the outside attacker of

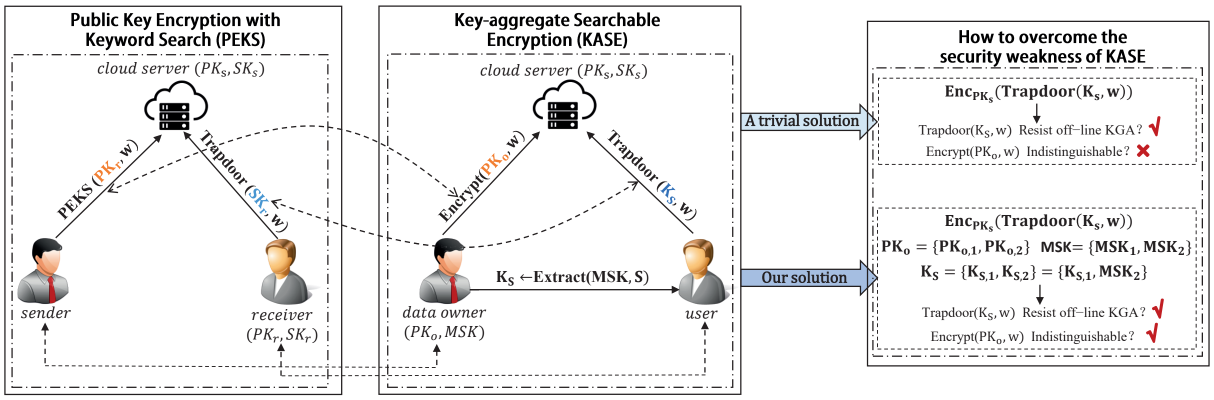

Public Key Encryption with Keyword Search (PEKS)
cloud server (PK_s, SK_s)
PEKS (PK_r, w)
Trapdoor (SK_r, w)
sender
receiver (PK_r, SK_r)
Key-aggregate Searchable Encryption (KASE)
cloud server (PK_s, SK_s)
Encrypt(PK_o, w)
Trapdoor (K_S, w)
$K_S \leftarrow$Extract(MSK, S)
data owner (PK_o, MSK)
user
A trivial solution
Our solution
How to overcome the security weakness of KASE
$Enc_{PK_s}(Trapdoor(K_s, w))$
$Trapdoor(K_S, w)$ Resist off-line KGA? ✓
$Encrypt(PK_o, w)$ Indistinguishable? ✗
$Enc_{PK_s}(Trapdoor(K_s, w))$
$PK_o = \{PK_{o,1}, PK_{o,2}\}$ $MSK = \{MSK_1, MSK_2\}$
$K_S = \{K_{S,1}, K_{S,2}\} = \{K_{S,1}, MSK_2\}$
$Trapdoor(K_S, w)$ Resist off-line KGA? ✓
$Encrypt(PK_o, w)$ Indistinguishable? ✓

Fig. 4: Design rationale of dKASE

KASE, that is, using PK_s to encrypt trapdoors. However, as we mentioned earlier, the keyword ciphertexts of [5], [22], [25], [26], and [24] don't satisfy the indistinguishability, so it is not feasible to use the standard technology of PEKS in such a trivial way. To achieve the ciphertext indistinguishability while solving the off-line KGA, we introduce another pair of public key and master secret key of the data owner into KASE, which are regarded as the public key and secret key of the receiver in PEKS. Then we use this public key to encrypt keyword to make the keyword ciphertext achieve indistinguishable security. The master secret key is as part of the aggregate key to generate trapdoor.

5.2 Concrete Construction

- $\boldsymbol{Setup(1^{\lambda}, n)}$ **:** Initialize system as follows:
  a) Generate a bilinear map group system $\mathcal{PG} = (p, \mathcal{G}, \mathcal{G}_T, e(\cdot,\cdot))$, where p is the order of $\mathcal{G}$.
  b) Randomly select a generator $g \in \mathcal{G}$ and $\alpha \in Z_p^*$, and compute $g_i = g^{(\alpha^i)}$ for $i = 1, 2, ..., n, n+2, ..., 2n$.
  c) Specify hash functions $H : \{0,1\}^* \rightarrow \mathcal{G}^*$, $H_1 : \{0,1\}^* \rightarrow \mathcal{G}^*$, $H_2 : \mathcal{G}_T \rightarrow \{0,1\}^{\lambda}$.

  Finally, the system parameter is:
  $$param = \{\mathcal{PG}, (g, g_1, ..., g_n, g_{n+2}, ..., g_{2n}), H, H_1, H_2\}.$$
- $\boldsymbol{KeyGen_o}$ **:** The data owner randomly chooses $\gamma, y \in Z_p^*$, computes $PK_{o,1} = g^{\gamma}$ and $PK_{o,2} = g^y$, and sets
  $$PK_o = \{PK_{o,1}, PK_{o,2}\},\ MSK = \{\gamma, y\}.$$
- $\boldsymbol{KeyGen_s}$ **:** The cloud server randomly chooses $x \in Z_p^*$ and $Q \in \mathcal{G}^*$, computes $X = g^x$, and sets
  $$PK_s = \{PK_{s,1}, PK_{s,2}\} = \{Q, X\},\ SK_s = x.$$
- $\boldsymbol{Encrypt(PK_o, PK_s, j, w)}$ **:** On input PK_o, PK_s, and a document belonging to class j and its keyword w, the data owner randomly chooses $t \in Z_p^*$ and computes
  $$c_{j,1} = g^t,\ c_{j,2} = (PK_{o,1} \cdot g_j)^t,\ c_{j,3} = e(g_1, g_n)^t,$$
  $$cw_j = H_2((e(H_1(w), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^t).$$
  Finally, he sets $CT_j = \{c_{j,1}, c_{j,2}, c_{j,3}, cw_j\}$.
- $\boldsymbol{Extract(MSK, S)}$ **:** On input MSK and a set $S \subseteq [1, n]$, the data owner computes
  $$K_{S,1} = \prod_{l \in S} g_{n+1-l}^{\gamma},\ K_{S,2} = y.$$
  Finally, he sets $K_S = \{K_{S,1}, K_{S,2}\}$.
- $\boldsymbol{Trapdoor(K_S, PK_s, w)}$ **:** On input K_S, PK_s, and a keyword w, the user randomly chooses $r \in Z_p^*$ and computes
  $$Tr' = K_{S,1} \cdot H_1(w)^{K_{S,2}} \cdot H(PK_{s,2}^r),\ Tr'' = g^r.$$
  Finally, he sets $Tr = \{Tr', Tr''\}$.
- $\boldsymbol{Test(SK_s, CT_j, Tr)}$ **:** On input SK_s, CT_j, and Tr, the cloud server performs the search as follows:
  - It adjusts Tr to trapdoor Tr_j for class j:
    $$Tr_j = \frac{Tr'}{H((Tr'')^{SK_s})} \cdot \prod_{l \in S, l \neq j} g_{n+1-l+j};$$
  - Next, it computes
    $$Tr'_j = \frac{c_{j,3} \cdot e(Tr_j, c_{j,1})}{e(\prod_{l \in S} g_{n+1-l}, c_{j,2})};$$
  - Finally, it verifies whether Formula 1 holds.
    $$H_2(Tr'_j \cdot e(c_{j,1}, PK_{s,1})^{SK_s}) = cw_j \quad (1)$$

    If Formula 1 holds, the cloud server returns "1", else it returns "0".

Correctness. Let CT_j be a valid ciphertext for w' and Tr be a valid trapdoor for w. It follows that,
$$\begin{aligned} Tr_j &= \frac{Tr'}{H((Tr'')^{SK_s})} \cdot \prod_{l \in S, l \neq j} g_{n+1-l+j} \\ &= K_{S,1} \cdot H_1(w)^y \cdot \prod_{l \in S, l \neq j} g_{n+1-l+j}, \end{aligned}$$
$$\begin{aligned} Tr'_j &= \frac{c_{j,3} \cdot e(Tr_j, c_{j,1})}{e(\prod_{l \in S} g_{n+1-l}, c_{j,2})} \\ &= \frac{c_{j,3} \cdot e(K_{S,1} \cdot H_1(w)^y \cdot \prod_{l \in S, l \neq j} g_{n+1-l+j}, g^t)}{e(\prod_{l \in S} g_{n+1-l}, (g^{\gamma} \cdot g_j)^t)} \\ &= \frac{e(g_1, g_n)^t \cdot e(H_1(w)^y, g^t) \cdot e(\prod_{l \in S, l \neq j} g_{n+1-l+j}, g^t)}{e(\prod_{l \in S} g_{n+1-l}, g_j^t)} \\ &= e(H_1(w)^y, g^t), \end{aligned}$$

$$\begin{aligned}&H_2(Tr'_j \cdot e(c_{j,1}, PK_{s,1})^{SK_s})\\ =&H_2(e(H_1(w), g^y)^t \cdot e(g^x, Q)^t)\\ =&H_2((e(H_1(w), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^t),\end{aligned}$$

Clearly, if $w = w'$, then Formula 1 holds.

6 The Proposed dVKASEM Scheme

In this section, based on the above dKASE scheme, we consider two more realistic scenarios. One is that multiple data owners share documents with a user, in such a condition, it is desirable to reduce the cost of storing aggregate keys and generating trapdoors. Another is that in a commercial cloud computing environment, the cloud server may maliciously tamper with the data, at which point it is necessary to verify the correctness of the returned data. We design an extended scheme dVKASEM to address these problems.

6.1 Design Rational

Inspired by the scheme of [26] (but as previously stated, it is insecure), to decrease storage and computation burden in multi-owner setting, we introduce the user's key uk. After getting aggregate keys sent by multiple data owners, the user encrypts each aggregate key with uk to convert it to a value that can be stored publicly in the cloud, while the user merely stores uk securely locally. When searching, the user can generate the trapdoor using only uk, and then the server adjusts the trapdoor to the actual trapdoor for each data owner's data by using the encrypted aggregate key.

For verifying the correctness of data, the signature is a good choice. But for the ordinary signature, we need to verify each returned result, in turn, to know whether the server has tampered with the data, which is not practical. Fortunately, we found that using aggregate signatures can greatly improve efficiency. Data owners encrypt each document with KAC and sign it with the $\boldsymbol{Sign}$ algorithm of Definition 3. When a user accesses multiple data owners' documents, the server searches for each data owner's documents in turn to obtain the matched encrypted documents and corresponding signatures. After searching for a data owner's documents, the server can aggregate the obtained signatures into a single signature because signatures can be produced by the same public key (the same user) in BAS system. Thus multiple aggregate signatures can be obtained after searching for documents of multiple data owners. Further, due to the aggregation can be done incrementally, the server can then reaggregate these aggregate signatures to obtain one aggregate signature. With this aggregate signature, the user can effectively verify search results.

6.2 System Definition of dVKASEM

Definition 8 (dVKASEM). *The dVKASEM system consists of the following algorithms:*

- $\boldsymbol{param \leftarrow Setup(1^{\lambda}, n)}$ **:** *On input λ and n, the data owner initializes the system parameter $param$.*
- $\boldsymbol{(PK_{oi}, MSK_i) \leftarrow KeyGen_{oi}}$ **:** *The i-th data owner outputs his public key and master secret key pair (PK_{oi}, MSK_i).*
- $\boldsymbol{(PK_s, SK_s) \leftarrow KeyGen_s}$ **:** *The cloud server outputs its public key and secret key pair (PK_s, SK_s).*
- $\boldsymbol{(CT_{i,j,k}, c_{i,j,k}, \sigma_{i,j,k}) \leftarrow Encrypt(PK_{oi}, MSK_i, PK_s, f_{i,j,k}, d_{i,j,k}, w)}$ **:** *The i-th data owner inputs PK_{oi}, PK_s, MSK_i, and the k-th document $f_{i,j,k}$ of his j-th class and document identifier $d_{i,j,k}$ and keyword w, and outputs keyword ciphertext $CT_{i,j,k}$, document ciphertext $c_{i,j,k}$, and signature $\sigma_{i,j,k}$.*
- $\boldsymbol{K_{S_i} \leftarrow Extract(MSK_i, S_i)}$ **:** *The i-th data owner inputs MSK_i and a set $S_i \subseteq [1, n]$, and then outputs the corresponding aggregate key K_{S_i}. Finally, the i-th data owner securely transmits $\{K_{S_i}, S_i\}$ to the user.*
- $\boldsymbol{EK_{S_i} \leftarrow EncK_s(uk, K_{S_i})}$ **:** *After receiving K_{S_i}, the user encrypts K_{S_i} with his secret key uk to get EK_{S_i} and sends $\{i, S_i, EK_{S_i}\}$ to the server.*
- $\boldsymbol{Tr \leftarrow Trapdoor(uk, w)}$ **:** *On input uk and keyword w, the user creates a trapdoor Tr, and transmits Tr and the corresponding document class sets to server.*
- $\boldsymbol{Tr_i \leftarrow Adjust(SK_s, Tr, EK_{S_i})}$ **:** *On input SK_s, Tr, and EK_{S_i}, the server adjusts the Tr to the trapdoor Tr_i about document class set S_i of the i-th data owner.*
- $\boldsymbol{1/0 \leftarrow Test(SK_s, CT_{i,j,k}, c_{i,j,k}, \sigma_{i,j,k}, Tr_i)}$ **:** *The cloud server first initializes C, Σ as two empty sets. Then, on input SK_s, Tr_i, and $CT_{i,j,k}$ under keyword w', if $w' = w$, the cloud server outputs "1" and adds $\sigma_{i,j,k}$ to Σ, $\{d_{i,j,k}, c_{i,j,k}\}$ to C, otherwise outputs "0". We denote $DO = \{i|(d_{i,j,k}, c_{i,j,k}) \in C\}$.*
  When all documents related to Tr are searched, the cloud server computes the aggregate signature σ of all signatures in Σ and returns σ, C to the user.
- $\boldsymbol{1/0 \leftarrow Verify(\{PK_{o_i}\}_{i \in DO}, \sigma, C)}$ **:** *To test the correctness of C, the user runs this algorithm. If C passes the verification, output "1", otherwise output "0".*

6.3 Concrete Construction

- $\boldsymbol{Setup(1^{\lambda}, n)}$ **:** This algorithm is the same as that of dKASE, except a hash function $h : \{0, 1\}^* \rightarrow \mathcal{G}^*$ also needs to be specified.
  Finally, the system parameter is:
  $$param = \{\mathcal{PG}, (g, g_1, ..., g_n, g_{n+2}, ..., g_{2n}), H, H_1, H_2, h\}.$$
- $\boldsymbol{KeyGen_{oi}}$ **:** The i-th data owner randomly chooses $\gamma_i, y_i \in Z_p^*$, computes $PK_{oi,1} = g^{\gamma_i}$ and $PK_{oi,2} = g^{y_i}$., and sets
  $$PK_{oi} = \{PK_{oi,1}, PK_{oi,2}\}, MSK_i = \{\gamma_i, y_i\}.$$
- $\boldsymbol{KeyGen_s}$ **:** The algorithm is the same as that of dKASE.
- $\boldsymbol{Encrypt(PK_{oi}, MSK_i, PK_s, f_{i,j,k}, d_{i,j,k}, w)}$ **:** On input PK_{oi}, PK_s, MSK_i, and the k-th document $f_{i,j,k}$ belonging to class j and its keyword w, the i-th data owner randomly chooses $t \in Z_p^*$ and computes
  $$c_{ijk,1} = g^t, c_{ijk,2} = (PK_{oi,1} \cdot g_j)^t, c_{ijk,3} = e(g_1, g_n)^t,$$
  $$cw_{ijk} = H_2((e(H_1(w), PK_{oi,2}) \cdot e(PK_{s,2}, PK_{s,1}))^t).$$
  In addition, for the ciphertext $c_{i,j,k}$ obtained by using KAC to encrypt $f_{i,j,k}$, the i-th data owner generates a signature $\sigma_{i,j,k} = h(d_{i,j,k}||c_{i,j,k})^{\gamma_i}$ ($d_{i,j,k}$ is the identifier of $f_{i,j,k}$). Finally, he sets $CT_{i,j,k} = \{c_{ijk,1}, c_{ijk,2}, c_{ijk,3}, cw_{ijk}\}$ and sends $\{CT_{i,j,k}, c_{i,j,k}, \sigma_{i,j,k}\}$ to the cloud server.

- $\boldsymbol{Extract(MSK_i, S_i)}$ **:** On input MSK_i and a set $S_i \subseteq [1, n]$, the i-th data owner computes

$$K_{S_i,1} = \prod_{l \in S_i} g_{n+1-l}^{\gamma_i}, K_{S_i,2} = y_i.$$

Finally, he sets $K_{S_i} = \{K_{S_i,1}, K_{S_i,2}\}$.

- $\boldsymbol{EncK_s(uk, K_{S_i})}$ **:** On input secret key $uk = \{uk_1, uk_2\}$ and aggregate key K_{S_i}, user computes

$$EK_{S_i,1} = K_{S_i,1} \cdot g^{-uk_1 \cdot K_{S_i,2}},$$

$$EK_{S_i,2} = uk_1 \cdot uk_2 \cdot K_{S_i,2}.$$

Finally, he sets $EK_{S_i} = \{EK_{S_i,1}, EK_{S_i,2}\}$ and sends $\{i, S_i, EK_{S_i}\}$ to the server.

- $\boldsymbol{Trapdoor(uk, w)}$ **:** On input uk and a keyword w, the user randomly chooses $r \in Z_p^*$ and computes

$$Tr' = (g \cdot H_1(w)^{\frac{1}{uk_1}})^{\frac{1}{uk_2}} \cdot H(PK_{s,2}^r), Tr'' = g^r.$$

Finally, he sets $Tr = \{Tr', Tr''\}$.

- $\boldsymbol{Adjust(SK_s, Tr, EK_{S_i})}$ **:** On input SK_s, Tr, and EK_{S_i}, the cloud server computes

$$Tr_i = EK_{S_i,1} \cdot (\frac{Tr'}{H((Tr'')^{SK_s})})^{EK_{S_i,2}}.$$

- $\boldsymbol{Test(SK_s, CT_{i,j,k}, c_{i,j,k}, \sigma_{i,j,k}, Tr_i)}$ **:** The cloud server first initializes C, Σ as two empty sets. Then, on input SK_s, $CT_{i,j,k}$, and Tr_i, it performs the search:

  a) it adjusts the Tr_i to the trapdoor $Tr_{i,j}$ for the document class j of the i-th data owner:

$$Tr_{i,j} = Tr_i \cdot \prod_{l \in S_i, l \neq j} g_{n+1-l+j}.$$

  b) Next, it computes

$$Tr'_{i,j} = \frac{c_{ijk,3} \cdot e(Tr_{i,j}, c_{ijk,1})}{e(\prod_{l \in S_i} g_{n+1-l}, c_{ijk,2})}.$$

  c) Finally, it verifies whether Formula 2 holds.

$$H_2(Tr'_{i,j} \cdot e(c_{ijk,1}, PK_{s,1})^{SK_s}) = cw_{ijk} \quad (2)$$

  If the Formula 2 holds, then the cloud server outputs "1", and adds $\sigma_{i,j,k}$ to Σ, $\{d_{i,j,k}, c_{i,j,k}\}$ to C, otherwise outputs "0".

We denote $DO = \{i | (d_{i,j,k}, c_{i,j,k}) \in C\}$; for $\forall i \in DO$, $DO_i = \{j | (d_{i,j,k}, c_{i,j,k}) \in C\}$; for $\forall j \in DO_i$, $DO_{i,j} = \{k | (d_{i,j,k}, c_{i,j,k}) \in C\}$. When all documents related to Tr are searched, the cloud server computes the aggregate signature of all signatures in Σ:

$$\sigma = \prod_{i \in DO} \prod_{j \in DO_i} \prod_{k \in DO_{i,j}} \sigma_{i,j,k}.$$

Then, it returns σ and C to the user.

- $\boldsymbol{Verify(\{PK_{oi}\}_{i \in DO}, \sigma, C)}$ **:** On input the public keys $\{PK_{oi}\}_{i \in DO}$, aggregate signature σ, identity-ciphertext set C, the user verifies whether Formula 3 holds.

$$e(g, \sigma) = \prod_{i \in DO} e(PK_{oi,1}, \prod_{j \in DO_i} \prod_{k \in DO_{i,j}} h(d_{i,j,k} || c_{i,j,k})) \quad (3)$$

If the Formula 3 holds, then the user outputs "1", otherwise outputs "0".

Correctness. We show the correctness of the "Test" algorithm and the "Verify" algorithm respectively.
Test: Similar to the dKASE scheme, so we omit it here.
Verify: Let C be a valid identity-ciphertext set returned by the cloud server and σ be the aggregate signature corresponding to C. Consequently,

$$\begin{aligned} e(g, \sigma) &= \prod_{i \in DO} e(g, \prod_{j \in DO_i} \prod_{k \in DO_{i,j}} h(d_{i,j,k} || c_{i,j,k}))^{\gamma_i} \\ &= \prod_{i \in DO} e(PK_{oi,1}, \prod_{j \in DO_i} \prod_{k \in DO_{i,j}} h(d_{i,j,k} || c_{i,j,k})) \end{aligned}$$

that is, Formula 3 holds. Therefore, our dVKASEM scheme can correctly verify the search results.

7 Security Analysis

7.1 Security Analysis for dKASE Scheme

Theorem 2. *The dKASE scheme is KC-IND-CKA secure in the random oracle model (ROM) assuming the BDH assumption holds in $\mathcal{G}$.*

Proof: By Lemma 2 and Lemma 3, it is proved. □

Lemma 2. *The dKASE scheme is KC-IND-CKA secure in Game 1 under the ROM assuming the BDH assumption holds.*

Proof: Suppose that there exists a cloud server $\mathcal{A}_1$ that can break the dKASE scheme with $\boldsymbol{Adv_{\mathcal{A}_1}^{Game\ 1}(\lambda)} \geq \varepsilon$ (Assume that $\mathcal{A}_1$ queries random oracle H_1, random oracle H_2, and trapdoor at most q_{H_1}, q_{H_2} and q_t, respectively). We construct a simulator $\mathcal{B}$ that can solve the BDH problem at least with probability $\varepsilon' = \varepsilon/(eq_t q_{H_2})$ (where e is the base of the natural logarithm). Therefore, if the BDH assumption holds, then ε' is negligible and consequently ε must be negligible. Thereby, the dKASE scheme is KC-IND-CKA secure according to Definition 5. Below, we describe in detail how to construct simulator $\mathcal{B}$.

Given a BDH problem instance $(p, \mathcal{G}, \mathcal{G}_T, e(\cdot,\cdot), g, g^a, g^b, g^c)$, $\mathcal{B}$ simulates the challenger $\mathcal{C}$ and interacts with $\mathcal{A}_1$ to run each phase of Game 1 as follows.

Phase 1-1: $\mathcal{B}$ first chooses $\alpha \in Z_p^*$ at random and computes $g_i = g^{(\alpha^i)}$ for $i = 1, ..., n, n+2, ..., 2n$. Then, it chooses $\gamma \in Z_p^*$ at random, computes $PK_{o,1} = g^\gamma$, and sets $PK_{o,2} = g^c$. It also randomly chooses $Q \in \mathcal{G}^*$ and $x \in Z_p^*$, sets $PK_{s,1} = Q$, and computes $PK_{s,2} = X = g^x$. It returns $\{p, \mathcal{G}, \mathcal{G}_T, e(\cdot,\cdot), (g, g_1, ..., g_n, g_{n+2}, ..., g_{2n}), H, H_1, H_2\}$ as system parameter (where H is a hash function randomly chosen by $\mathcal{B}$), $(PK_{o,1}, PK_{o,2})$ as the public key of the data owner, and $(PK_{s,1}, PK_{s,2})$ and x as the cloud server's public key and secret key. The random oracles H_1 and H_2 are controlled by $\mathcal{B}$ in the following way.

H-Query. $\mathcal{A}_1$ makes hash queries at this stage. $\mathcal{B}$ creates two hash lists, H_1List and H_2List, to keep track of all queries and responses, both of which are initially empty.

- For H_1, let w_i be the i-th query. If w_i already belongs to H_1List, $\mathcal{B}$ responds according to H_1List. Otherwise, do the following:
  - Randomly choose $\delta_i \in \{0, 1\}$ such that $Pr[\delta_i = 0] = 1/(q_t + 1)$.
  - Choose $l_i \in Z_p^*$ at random and set $H_1(w_i)$ as

$$H_1(w_i) = \begin{cases} g^b \cdot g^{l_i} & \text{if } \delta_i = 0, \\ g^{l_i} & \text{if } \delta_i = 1. \end{cases}$$

  - $H_1(w_i)$ is returned in response to this query, and $(w_i, H(w_i), l_i, \delta_i)$ is added to H_1List.

- For H_2, let z_i be the i-th query. If z_i is already in H_2List, $\mathcal{B}$ responds according to H_2List. Otherwise, it randomly chooses $Z_i \in \{0,1\}^\lambda$, responds $H_2(z_i) = Z_i$ as the answer, and adds (z_i, Z_i) to H_2List.

Phase 1-2: $\mathcal{A}_1$ sends $\mathcal{B}$ a set $S^* \subseteq [1,n]$ to attack. $\mathcal{B}$ randomly picks $j \in S^*$ and returns it to $\mathcal{A}_1$.
Phase 1-3: $\mathcal{A}_1$ makes trapdoor queries in this phase. $\mathcal{B}$ first computes $K_{S^*,1} = \prod_{l\in S^*} g^\gamma_{n+1-l}$. Then, for a trapdoor query on keyword w_i, $\mathcal{B}$ does the following:

- Let $(w_i, H(w_i), l_i, \delta_i)$ be the corresponding tuple. If $\delta_i = 0$, abort. Otherwise, go to the next step.
- According to the simulation, we can get $H_1(w_i) = g^{l_i}$. $\mathcal{B}$ randomly chooses $r \in Z_p^*$ and computes:

$$T_{w_i} = \left\{ K_{S^*,1} \cdot PK_{o,2}^{l_i} \cdot H(PK_{s,2}^r),\ g^r \right\}.$$

  Note that the T_{w_i} is a valid trapdoor over the challenged set S^* as $K_{S^*,1} \cdot PK_{o,2}^{l_i} \cdot H(PK_{s,2}^r) = K_{S^*,1} \cdot g^{c\cdot l_i} \cdot H(PK_{s,2}^r) = K_{S^*,1} \cdot H_1(w_i)^c \cdot H(PK_{s,2}^r)$.

Phase 1-4: $\mathcal{A}_1$ sends $\mathcal{B}$ a pair of keywords (w_0^*, w_1^*). The restriction is that neither w_0^* nor w_1^* is queried in Phase 1-3. Then, $\mathcal{B}$ performs the following:

- The tuples $(w_0^*, H(w_0^*), l_0^*, \delta_0^*)$ and $(w_1^*, H(w_1^*), l_1^*, \delta_1^*)$ are obtained by running the algorithm of simulating random oracle H_1. If both δ_0^* and δ_1^* are 1, abort. Otherwise, go to the next step.
- Randomly choose $\beta \in \{0,1\}$ so that $\delta_\beta^* = 0$.
- Randomly pick $R \in \{0,1\}^\lambda$ and creat a ciphertext

$$\begin{aligned} CT_j^* &= \{c_{j,1}^*, c_{j,2}^*, c_{j,3}^*, cw_j^*\} \\ &= \left\{ g^a, (g^a)^\gamma \cdot (g^a)^{\alpha^j}, e(g^a, g_n)^\alpha, R \right\} \\ &= \{g^a, (g^\gamma \cdot g_j)^a, e(g_1, g_n)^a, R\}. \end{aligned}$$

- Set $R = H_2((e(H_1(w_\beta^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$.

By the definitions of $PK_{o,2}$, $PK_{s,1}$, and $PK_{s,2}$, we can get

$$\begin{aligned} &(e(H_1(w_\beta^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a \\ =&(e(g^b \cdot g^{l_\beta^*}, g^c) \cdot e(g^x, Q))^a \\ =&e(g,g)^{abc} \cdot e(g^a, g^c)^{l_\beta^*} \cdot e(g^a, Q)^x. \end{aligned}$$

Phase 1-5: $\mathcal{A}_1$ makes keyword trapdoor queries as in Phase 1-3 on condition that $w \neq w_0^*, w_1^*$.
Phase 1-6: Finally, $\mathcal{A}_1$ outputs its guess β' of β. By this time, $\mathcal{B}$ selects a tuple (z, Z) from H_2List at random and returns $\frac{z}{e(g^a,g^c)^{l_\beta^*} \cdot e(g^a,Q)^x}$ as its solution for BDH problem.

The simulation and solution are now complete. Next, we analyze the probability that $\mathcal{B}$ outputs $e(g,g)^{abc}$ correctly is at least ε'. For this purpose, we begin by examining the probability that $\mathcal{B}$ will not "abort" in this simulation process. The following events are defined:
$\mathcal{E}_1$: $\mathcal{B}$ does not abort due to $\mathcal{A}_1{}'$ trapdoor query.
$\mathcal{E}_2$: $\mathcal{B}$ does not abort in Phase 1-4.
$\mathcal{E}_3$: $\mathcal{A}_1$ makes an H_2 query for either $H_2((e(H_1(w_0^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ (shorthand for H_{20}) or $H_2((e(H_1(w_1^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ (shorthand for H_{21}).

Now, we show that all three events $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ occur at a high enough probability.

Claim 1. *$Pr[\mathcal{E}_1] \geq 1/e$.*

Proof: In general, we assume that $\mathcal{A}_1$ does not make trapdoor queries for the same keyword twice. The probability that a trapdoor query of a keyword w_i causes $\mathcal{B}$ to abort is equal to the probability of $\delta_i = 0$, that is, $1/(q_t+1)$, and since $\mathcal{A}_1$ makes at most q_t trapdoor queries, we get $Pr[\mathcal{E}_1] \geq (1 - 1/(q_t+1))^{q_t} \geq 1/e$. □

Claim 2. *$Pr[\mathcal{E}_2] \geq 1/q_t$.*

Proof: If the keywords w_0^* and w_1^* generated by $\mathcal{A}_1$ in Phase 1-4 satisfy $\delta_0^* = \delta_1^* = 1$, $\mathcal{B}$ will abort in this Phase. The δ_0^* and δ_1^* are independent of $\mathcal{A}_1{}'$ current view because $\mathcal{A}$ has not asked the trapdoors of w_0^* and w_1^*. As a result, $Pr[\delta_i^* = 0] = 1/(q_t+1)$ for $i = 0,1$ and since these two values are unrelated, we get $Pr[\delta_0^* = \delta_1^* = 1] = (1-(1/(q_t+1)))^2 \leq 1 - 1/q_t$. Consequently, $Pr[\mathcal{E}_2] \geq 1/q_t$. □

Claim 3. *$Pr[\mathcal{E}_3] \geq 2\varepsilon$.*

Proof: Let $\neg\mathcal{E}_3$ denote the event that $\mathcal{A}_1$ does not issue a query for either one of H_{20} and H_{21} in real attack. If $\neg\mathcal{E}_3$ occurs, then $\beta \in \{0,1\}$ means that $\mathcal{A}_1$'s view has no bearing on whether CT_j^* is w_0^*'s or w_1^*'s ciphertext. So $Pr[\beta' = \beta] = 1/2$. Then, we have

$$\begin{aligned} &Pr[\beta' = \beta] \\ =&Pr[\beta' = \beta|\neg\mathcal{E}_3]Pr[\neg\mathcal{E}_3] + Pr[\beta' = \beta|\mathcal{E}_3]Pr[\mathcal{E}_3] \\ \leq&Pr[\beta' = \beta|\neg\mathcal{E}_3]Pr[\neg\mathcal{E}_3] + Pr[\mathcal{E}_3] \\ =&\frac{1}{2} + \frac{1}{2}Pr[\mathcal{E}_3], \end{aligned} \tag{4}$$

$$\begin{aligned} &Pr[\beta' = \beta] \\ =&Pr[\beta' = \beta|\neg\mathcal{E}_3]Pr[\neg\mathcal{E}_3] + Pr[\beta' = \beta|\mathcal{E}_3]Pr[\mathcal{E}_3] \\ \geq&Pr[\beta' = \beta|\neg\mathcal{E}_3]Pr[\neg\mathcal{E}_3] \\ =&\frac{1}{2} - \frac{1}{2}Pr[\mathcal{E}_3]. \end{aligned} \tag{5}$$

In addition, based on the assumption of $\mathcal{A}_1$, we know:

$$|Pr[\beta' = \beta] - 1/2| \geq \varepsilon. \tag{6}$$

Combining Formulas 4, 5 and 6, we can obtain $\varepsilon \leq |Pr[\beta' = \beta] - 1/2| \leq \frac{1}{2}Pr[\mathcal{E}_3]$. Therefore, $Pr[\mathcal{E}_3] \geq 2\varepsilon$. □

Assuming $\mathcal{B}$ doesn't abort, $\mathcal{B}$ precisely simulates the real attack until $\mathcal{A}_1$ submits a query to H_{20} or H_{21}. According to Claim 3, after simulation ends, $\mathcal{A}_1$ will query for either H_{20} or H_{21} with probability at least 2ε. Hence, $\mathcal{A}_1$ makes query for $H_2((e(H_1(w_b^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ with a probability of at least ε. That is, the value $H_2((e(H_1(w_b^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ will appear in H_2List. The probability that $\mathcal{B}$ chooses the right is $1/q_{H_2}$. Thus, if $\mathcal{B}$ does not abort during the simulation, the probability that $\mathcal{B}$ outputs the correct solution of BDH problem instance is ε/q_{H_2}. Moreover, the likelihood of $\mathcal{A}_1$ not aborting the simulation is $Pr[\mathcal{E}_1 \wedge \mathcal{E}_2]$, and since $\mathcal{A}_1$ does not query the trapdoors of w_0^* and w_1^*, events $\mathcal{E}_1$ and $\mathcal{E}_2$ are independent. So, $Pr[\mathcal{E}_1 \wedge \mathcal{E}_2] \geq 1/(eq_t)$. Finally, we get the probability that $\mathcal{B}$ can solve the BDH problem correctly is $\varepsilon/(eq_tq_{H_2})$.

□

Lemma 3. *The dKASE scheme is KC-IND-CKA secure in Game 2 under the ROM assuming the BDH assumption holds.*

Proof: Suppose that there exists an outside attacker (including users authorized by the data owner) $\mathcal{A}_2$ that can break the dKASE scheme with $\boldsymbol{Adv_{\mathcal{A}_2}^{Game\ 2}(\lambda)} \geq \varepsilon$ (Assume that $\mathcal{A}_2$ queries random oracle H_1, random oracle H_2, and trapdoor at most q_{H_1}, q_{H_2}, and q_t, respectively). We construct a simulator $\mathcal{B}$ that can solve the BDH problem at least with probability $\varepsilon' = \varepsilon/q_{H_2}$. Therefore, if the BDH assumption holds, then ε' is negligible and consequently ε must be negligible. Thereby, the dKASE scheme is KC-IND-CKA secure according to Definition 5. Below, we describe in detail how to construct simulator $\mathcal{B}$.

Given a BDH problem instance $(p, \mathcal{G}, \mathcal{G}_T, e(\cdot,\cdot), g, g^a, g^b, g^c)$, $\mathcal{B}$ simulates the challenger $\mathcal{C}$ and interacts with $\mathcal{A}_2$ to run each phase of Game 2 as follows.

Phase 2-1: $\mathcal{B}$ first chooses $\alpha \in Z_P^*$ at random and computes $g_i = g^{(\alpha^i)}$ for $i = 1, ..., n, n+2, ..., 2n$. Then, it chooses $\gamma, y \in Z_p^*$ at random and computes $PK_{o,1} = g^\gamma$, $PK_{o,2} = g^y$. It also sets $PK_{s,1} = Q = g^b$, $PK_{s,2} = X = g^c$. It returns $\{p, \mathcal{G}, \mathcal{G}_T, e(\cdot,\cdot), (g, g_1, ..., g_n, g_{n+2}, ..., g_{2n}), H, H_1, H_2\}$ as system parameter (where H is a hash function randomly chosen by $\mathcal{B}$), $PK_o = (PK_{o,1}, PK_{o,2})$ as public key of the data owner, and $PK_s = (PK_{s,1}, PK_{s,2})$ as public key of the cloud server. The random oracles H_1 and H_2 are controlled by $\mathcal{B}$ in the following way.

H-Query. $\mathcal{A}_2$ makes hash queries at this stage. $\mathcal{B}$ creates two hash lists, H_1List and H_2List, to keep track of all queries and responses, both of which are initially empty.

- For H_1, let w_i be the i-th query. If w_i already belongs to H_1List, $\mathcal{B}$ responses according to H_1List. Otherwise, it randomly chooses $l_i \in Z_p^*$, computes $H(w_i) = g^{l_i}$ and returns it as answer. Finally, $\mathcal{B}$ adds $(w_i, H(w_i), l_i)$ to H_1List.
- By the same method as Lemma 2, $\mathcal{B}$ can simulate H_2 queries. We omit the details here.

Phase 2-2: $\mathcal{A}_2$ sends $\mathcal{B}$ a set $S^* \subseteq [1, n]$ to attack. $\mathcal{B}$ randomly picks $j \in S^*$, computes $K_{S^*} = \{K_{S^*,1}, K_{S^*,2}\} = \{\prod_{l \in S^*} g_{n+1-l}^\gamma, y\}$, and returns $\{j, K_{S^*}\}$ to $\mathcal{A}_2$.

Phase 2-3: $\mathcal{A}_2$ sends $\mathcal{B}$ a pair of keywords (w_0^*, w_1^*). $\mathcal{B}$ randomly chooses $\beta \in \{0, 1\}$, generates a ciphertext

$$\begin{aligned} CT_j^* &= \{c_{j,1}^*, c_{j,2}^*, c_{j,3}^*, cw_j^*\} \\ &= \left\{g^a, (g^a)^\gamma \cdot (g^a)^{\alpha^j}, e(g^a, g_n)^\alpha, R\right\} \\ &= \{g^a, (g^\gamma \cdot g_j)^a, e(g_1, g_n)^a, R\}, \end{aligned}$$

and sets $R = H_2((e(H_1(w_\beta^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$. By the definition of $PK_{o,2}$, $PK_{s,1}$ and $PK_{s,2}$, we can get

$$\begin{aligned} R =& (e(H_1(w_\beta^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a \\ &= (e(g^{l_\beta^*}, g^y) \cdot e(g^c, g^b))^a \\ &= e(g^{l_\beta^*}, g^a)^y \cdot e(g, g)^{abc}. \end{aligned}$$

Phase 2-4: Finally, $\mathcal{A}_2$ outputs a guess β' of β. By this time, $\mathcal{B}$ randomly picks a tuple (z, Z) from H_2List and returns $\frac{z}{e(g^{l_\beta^*}, g^a)^y}$ as its solution for BDH problem instance.

The simulation and solution are now complete. Next, we analyze the probability that $\mathcal{B}$ outputs $e(g, g)^{abc}$ correctly is at least ε'. We define $\mathcal{E}$ be an event that $\mathcal{A}_2$ issues a query for either one of $H_2((e(H_1(w_0^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ or $H_2((e(H_1(w_1^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ in the real attack. Let $\neg\mathcal{E}$ denote that $\mathcal{E}$ doesn't occur. We notice that if $\neg\mathcal{E}$ occurs, $Pr[\beta' = \beta] = 1/2$. Therefore, similar Formulas 4, 5, and 6, we can get $\varepsilon \leq |Pr[\beta' = \beta] - 1/2| \leq \frac{1}{2}Pr[\mathcal{E}]$, $Pr[\mathcal{E}] \geq 2\varepsilon$. Hence, $\mathcal{A}_2$ issues a query for $H_2((e(H_1(w_b^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ with probability at least ε. That is, the value $H_2((e(H_1(w_b^*), PK_{o,2}) \cdot e(PK_{s,2}, PK_{s,1}))^a)$ will appear in H_2List. The probability that $\mathcal{B}$ chooses the right is $1/q_{H_2}$. Therefore, the probability that $\mathcal{B}$ outputs the correct solution of the BDH problem instance is ε/q_{H_2}. □

Theorem 3. *The dKASE scheme is KT-IND-KGA secure assuming the HDH assumption holds.*

Proof: Suppose that there exists an outside attacker (excluding the cloud server and the authorized users) $\mathcal{A}_3$ that can break the dKASE scheme with $\boldsymbol{Adv_{\mathcal{A}_3}^{Game\ 3}(\lambda)} \geq \varepsilon$. We construct a simulator $\mathcal{B}$ that can solve the HDH problem at least with probability ε. Therefore, if the HDH assumption holds, then ε must be negligible. Thereby, the dKASE scheme is KT-IND-KGA secure according to the Definition 6. Below, we show in detail how to construct simulator $\mathcal{B}$.

Given an HDH problem instance: (g, g^a, g^b, η) and a hash function $H : \{0,1\}^* \to \mathcal{G}^*$, where η is either $H(g^{ab})$ or a random element of $\mathcal{G}^*$, $\mathcal{B}$ simulates the challenger $\mathcal{C}$ and interacts with $\mathcal{A}_3$ to run each phase of Game 3 as follows.

Phase 3-1: $\mathcal{B}$ first chooses $\alpha \in Z_P^*$ at random and computes $g_i = g^{(\alpha^i)}$ for $i = 1, ..., n, n+2, ..., 2n$. Then, it chooses $\gamma, y \in Z_p^*$ at random and computes $PK_{o,1} = g^\gamma$, $PK_{o,2} = g^y$. It also randomly chooses $Q \in \mathcal{G}^*$ and sets $PK_{s,1} = Q$, $PK_{s,2} = X = g^a$. It returns $\{p, \mathcal{G}, \mathcal{G}_T, e(\cdot,\cdot), (g, g_1, ..., g_n, g_{n+2}, ..., g_{2n}), H, H_1, H_2\}$ as system parameter (where H_1 and H_2 are hash functions randomly chosen by $\mathcal{B}$), $PK_o = (PK_{o,1}, PK_{o,2})$ as public key of the data owner, and $PK_s = (PK_{s,1}, PK_{s,2})$ as public key of the cloud server.

Phase 3-2: $\mathcal{A}_3$ sends $\mathcal{B}$ a set $S^* \subseteq [1, n]$ to attack. $\mathcal{B}$ computes $K_{\overline{S^*}} = \left\{K_{\overline{S^*,1}}, K_{\overline{S^*,2}}\right\} = \{\prod_{l \in \overline{S^*}} g_{n+1-l}^\gamma, y\}$ and returns it to $\mathcal{A}_3$.

Phase 3-3: $\mathcal{A}_3$ makes trapdoor queries in this phase. For a trapdoor query on keyword w_i, $\mathcal{B}$ does the following:

- Compute $K_{S^*,1} = \prod_{l \in S^*} g_{n+1-l}^\gamma$.
- Choose $r \in Z_p^*$ at random.
- Compute $T_{w_i} = \{K_{S^*,1} \cdot H_1(w)^y \cdot H((g^a)^r), g^r\}$.

Phase 3-4: $\mathcal{A}_3$ sends $\mathcal{B}$ a pair of keywords (w_0^*, w_1^*). The restriction is that neither w_0^* nor w_1^* is queried in Phase 3-3. $\mathcal{B}$ randomly picks $\beta \in \{0, 1\}$, and computes $Tr^* = \left\{K_{S^*,1} \cdot H_1(w_\beta^*)^y \cdot H(\eta), g^b\right\}$ and returns it to $\mathcal{A}_3$.

Phase 3-5: $\mathcal{A}_3$ makes keyword trapdoor queries as in Phase 3-3 on condition that $w \neq w_0^*, w_1^*$.

Phase 3-6: $\mathcal{A}_3$ outputs a guess β' of β. If $\beta' = \beta$, $\mathcal{B}$ outputs "1", otherwise outputs "0".

This completes the simulation and solution. Next, we analyze that the probability of $\mathcal{B}$ correctly solving the HDH problem is at least ε. If $\eta = H(g^{ab})$ in a given instance of HDH problem, simulation and real attack are indistinguishable, and therefore $\mathcal{A}_3$ guesses $\beta' = \beta$ with at least probability $\frac{1}{2} + \varepsilon$. If η is a random elament of $\mathcal{G}$, $\mathcal{A}_3$ guesses

TABLE 3: Computation Comparison with [25], [26].

Algorithm \ Scheme	[26]	[25]	dKASE	dVKASEM
Setup	$(2n-1)E_{\mathcal{G}}+(2n-2)M_{\mathcal{Z}_p}$	$(2n-1)E_{\mathcal{G}}+(2n-2)M_{\mathcal{Z}_p}$	$(2n-1)E_{\mathcal{G}}+(2n-2)M_{\mathcal{Z}_p}$	$(2n-1)E_{\mathcal{G}}+(2n-2)M_{\mathcal{Z}_p}$
KeyGen$_\mathbf{o}$	$E_{\mathcal{G}}$	$2E_{\mathcal{G}}$	$2E_{\mathcal{G}}$	$2E_{\mathcal{G}}$
KeyGen$_\mathbf{s}$	$\perp$	$\perp$	$E_{\mathcal{G}}$	$E_{\mathcal{G}}$
Encrypt	$2P+4E_{\mathcal{G}}+M_{\mathcal{G}}+M_{\mathcal{G}_T}$	$2P+6E_{\mathcal{G}}+2M_{\mathcal{G}}+M_{\mathcal{G}_T}$	$3P+2E_{\mathcal{G}}$ $+2E_{\mathcal{G}_T}+M_{\mathcal{G}}+M_{\mathcal{G}_T}$	$3P+3E_{\mathcal{G}}$ $+2E_{\mathcal{G}_T}+M_{\mathcal{G}}+M_{\mathcal{G}_T}$
Extract	$(\vert S\vert-1)M_{\mathcal{G}}+E_{\mathcal{G}}$	$(\vert S\vert-1)M_{\mathcal{G}}+E_{\mathcal{G}}$	$(\vert S\vert-1)M_{\mathcal{G}}+E_{\mathcal{G}}$	$(\vert S\vert-1)M_{\mathcal{G}}+E_{\mathcal{G}}$
Trapdoor	$E_{\mathcal{G}}+M_{\mathcal{G}}$	$mM_{\mathcal{G}}$	$m(2E_{\mathcal{G}}+2M_{\mathcal{G}})$	$4E_{\mathcal{G}}+2M_{\mathcal{G}}$
Search	$m\vert S\vert(2P+$ $(2\vert S\vert-1)M_{\mathcal{G}}+M_{\mathcal{G}_T})$	$m\vert S\vert(2P+$ $(2\vert S\vert-2)M_{\mathcal{G}}+M_{\mathcal{G}_T})$	$m\vert S\vert(3P+E_{\mathcal{G}}+E_{\mathcal{G}_T}+$ $(2\vert S\vert-2)M_{\mathcal{G}}+3M_{\mathcal{G}_T})$	$m\vert S\vert(3P+2E_{\mathcal{G}}+E_{\mathcal{G}_T}+$ $2\vert S\vert M_{\mathcal{G}}+3M_{\mathcal{G}_T})+mdM_{\mathcal{G}}$
Verify	$\perp$	$m(2P+(2d+3)E_{\mathcal{G}}+$ $2dM_{\mathcal{G}}+dM_{\mathcal{Z}_p})$	$\perp$	$(m+1)P+m(d-1)M_{\mathcal{G}}$ $+(m-1)M_{\mathcal{G}_T}$

$\beta' = \beta$ with probability $\frac{1}{2}$. Therefore, we have

$$\begin{aligned}&|Pr[\mathcal{A}(g,g^a,g^b,H(g^{a\cdot b}))="1"]-Pr[\mathcal{A}(g,g^a.g^b,\eta)="1"]|\\ \geq&(\frac{1}{2}+\varepsilon)-\frac{1}{2}\\ =&\varepsilon.\end{aligned}$$

□

7.2 Security Analysis for dVKASEM Scheme

In this subsection, we prove the ciphertext indistinguishability and trapdoor indistinguishability of dVKASEM in Theorem 4 and Theorem 5, respectively. The proof idea is basically the same as that of dKASE, so we do not analyze in detail but focus on the differences. In theorem 6, we show that dVKASEM can effectively verify search results.

For the page limit of the journal, we defer the proofs of the following theorems to the appendix A, B and C, which is provided as the supplemental material.

Theorem 4. *The dVKASEM scheme is KC-IND-CKA secure in the ROM assuming the BDH assumption holds.*

Theorem 5. *The dVKASEM scheme is KT-IND-KGA secure assuming the HDH assumption holds.*

Theorem 6. *For the cloud server, it is computationally infeasible to forge a valid aggregate signature to pass the "Verify" algorithm when the group $\mathcal{G}$ is a bilinear group for Diffie-Hellman.*

8 Performance Analysis

8.1 Theoretical Analysis

We compare our scheme with [26] and [25] in the field of computation and communication cost. TABLE 3 shows the computation comparison, and TABLE 4 shows the communication comparison. As we pointed out earlier, we do not consider the verifiability of [26]. In addition, the authentication method of [25] is universal, so we do not consider its authentication and focus on its verifiability.

TABLE 4: Communication Comparison with [25], [26].

Component \ Scheme		[26]	[25]	dKASE	dVKASEM
Ciphertext (Data Owner→Server)		$2\vert\mathcal{G}\vert+\vert\mathcal{G}_T\vert$	$2\vert\mathcal{G}\vert+\vert\mathcal{G}_T\vert$	$2\vert\mathcal{G}\vert+\vert\mathcal{G}_T\vert+\vert\lambda\vert$	$3\vert\mathcal{G}\vert+\vert\mathcal{G}_T\vert+\vert\lambda\vert$
Aggregate Key (Date Owner→User)		$\vert\mathcal{G}\vert$	$\vert\mathcal{G}\vert$	$\vert\mathcal{G}\vert+\vert\mathcal{Z}_p\vert$	$\vert\mathcal{G}\vert+\vert\mathcal{Z}_p\vert$
Encrypted Aggregate Key (User→Server)		$\vert\mathcal{G}\vert$	$\perp$	$\perp$	$\vert\mathcal{G}\vert+\vert\mathcal{Z}_p\vert$
Trapdoor (User→Server)		$\vert\mathcal{G}\vert$	$m\vert\mathcal{G}\vert$	$2m\vert\mathcal{G}\vert$	$2\vert\mathcal{G}\vert$
Verify	Challenging Value (User→Server)	$\perp$	$md\vert\mathcal{Z}_p\vert$	$\perp$	$\perp$
	proof (Server→User)	$\perp$	$2\vert\mathcal{G}\vert+\vert\mathcal{Z}_p\vert$	$\perp$	$\vert\mathcal{G}\vert$

"$|\mathcal{G}|$": the bit-length of an element in group $\mathcal{G}$;
"$|\mathcal{G}_T|$": the bit-length of an element in group $\mathcal{G}_T$;
"$|\lambda|$": the length of security parameter λ.

Since the system frameworks of these schemes differ we will unify the algorithms contained in each scheme as $\{Setup, KeyGen_o, KeyGen_s, Encrypt, Extract, Trapdoor, Search, Verify\}$, where $KeyGen_o$ denotes that the data owner generates its public key and master secret key, $KeyGen_s$ denotes that the server generates its public/secret key pair, $Search$ denotes that the server uses trapdoor to search for keyword ciphertext, including trapdoor adjustment and specific test algorithm.

For simplicity, we assume that there are m data owners sharing documents with the user, and they all share $|S|$ classes, with each class containing one document. Following the search, each data owner has d documents that meet the search criteria. Furthermore, we only consider some time-consuming operations, including the bilinear pairing operation P, exponentiation operation $E_{\mathcal{G}}$ and $E_{\mathcal{G}_T}$ in $\mathcal{G}$ and $\mathcal{G}_T$, respectively, and multiplication operation $M_{\mathcal{G}_1}$, $M_{\mathcal{G}_T}$, and $M_{\mathcal{Z}_p}$ in $\mathcal{G}$, $\mathcal{G}_T$, and $\mathcal{Z}_p$, respectively.

TABLE 3 shows that the main difference in the computation cost of these four schemes is in $KeyGen_s$, $Trapdoor$, and $Verify$. In order to resist the off-line KGA, our schemes introduce the public key and secret key of the server. For $Trapdoor$, in dVKASEM and [26], a trapdoor generated by the user's secret key can be used to query the data of multiple data owners, so the computation cost is independent of m. In addition, our $Verify$ algorithm is more efficient than that of [25], and [26] can not realize verification.

TABLE 4 shows that the trapdoor size of our dVKASEM is constant, but that of [25] is related to m. Moreover, during verification in dVKASEM, the server merely needs to send the user an element of group $\mathcal{G}$, namely the aggregate signature, but in [25] the communication complexity of verification is linear with the number of matched documents.

8.2 Practical Analysis

Our experiments are performed on Ubuntu 20.04.1 LTS operating system and are implemented by GCC 9.4.0 compile on a Linux Server with AMD Ryzen 5 4600H with Radeon Graphics CPU@ 3.00 GHz and 3GB memory by using C++ programming language based on the Paring Based Cryptography (PBC) Library. We simulate the time costs of schemes [25], [26] and our schemes in different settings.

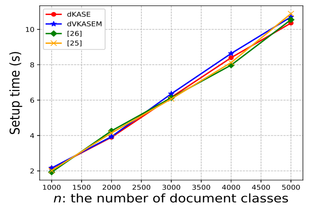


Fig. 5: Setup Time

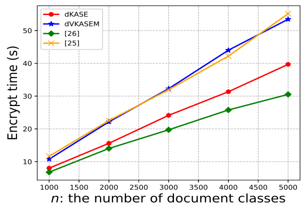


Fig. 6: Encrypt Time

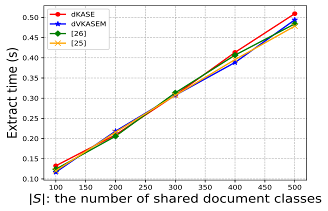


Fig. 7: Extract Time

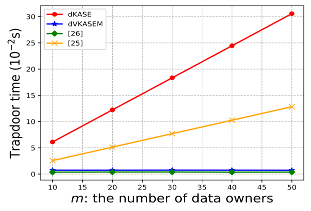


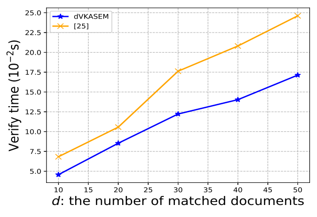
Fig. 8: Trapdoor Time

Fig. 5 shows the time cost for **Setup**, indicating that the efficiency of the setup algorithm in these four schemes is nearly identical and has almost a linear relationship with the number of document classes, which is consistent with the theoretical analysis. The time cost of **Encrypt** is shown in Fig. 6. The encryption time of dVKASEM and [25] is longer than that of dKASE and [26] because both dVKASEM and [25] need to generate additional signatures for verification. Although our dKASE scheme is slightly time-consuming than [26], this is for higher security. In Fig. 7, we provide the trend of **Extract**'s time consumption relative to the number of shared document classes. The efficiency and growth rate of these schemes are the same. In Fig. 8, we demonstrate the **Trapdoor** generation time. The simulation results reveal that our extension scheme dVKASEM greatly improves the efficiency of trapdoor generation. Compared to [26], our scheme improves security with little sacrifice to the efficiency of trapdoor generation.

Fig. 9 shows the time cost of **Search**, where in Fig. 9 (a) we fix $m = 1$, $|S|$ varies from 100 to 500, and in Fig. 9 (b) we fix $|S| = 100$, m varies from 10 to 50. We can see that the search time of our schemes is a bit larger than that of [25], [26]. The extra time cost is due to what we intend to achieve KC-IND-CKA and KT-IND-KGA security. Fig. 10 shows the time cost of **Verify**, where in Fig. 10 (a) we fix $m = 1$, d varies from 10 to 50, and in Fig. 10 (b) we fix $d = 10$, m varies from 10 to 50. The adoption of aggregate signature effectively improves our verification efficiency.

9 Conclusion

We propose a notion of dKASE and design a concrete scheme. Our dKASE scheme can realize secure ciphertext searching and selective data sharing at the same time. Furthermore, we present an extended scheme to support multi-owner and verifiability of search results, which is applicable to cloud outsourcing model in practice. We also define two security models on dKASE including KC-IND-CKA and KT-IND-KGA. Formal security analysis demonstrates our schemes achieve the desired security properties. Experimental results show our schemes' efficiency.


(a)

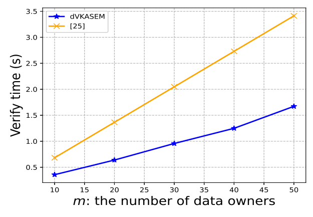
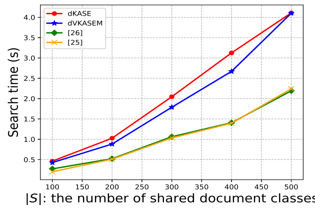

(b)

Fig. 10: Verify Time


(a)

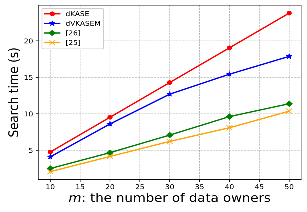

(b)

Fig. 9: Search Time

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No.62072276, No.61772311).

References

[1] J. Liu, B. Zhao, J. Qin, X. Zhang, and J. Ma, "Multi-keyword ranked searchable encryption with the wildcard keyword for data sharing in cloud computing," *The Computer Journal*, 2021.

[2] W. Shen, J. Yu, M. Yang, and J. Hu, "Efficient identity-based data integrity auditing with key-exposure resistance for cloud storage," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[3] J. Gharehchamani, Y. Wang, D. Papadopoulos, M. Zhang, and R. Jalili, "Multi-user dynamic searchable symmetric encryption with corrupted participants," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[4] J. Ning, J. Chen, K. Liang, J. K. Liu, C. Su, and Q. Wu, "Efficient encrypted data search with expressive queries and flexible update," *IEEE transactions on services computing*, vol. 15, no. 3, pp. 1619–1633, 2020.

[5] B. Cui, Z. Liu, and L. Wang, "Key-aggregate searchable encryption (kase) for group data sharing via cloud storage," *IEEE Transactions on computers*, vol. 65, no. 8, pp. 2374–2385, 2015.

[6] A. Kiayias, O. Oksuz, A. Russell, T. Qiang, and W. Bing, "Efficient encrypted keyword search for multi-user data sharing," in *European Symposium on Research in Computer Security*, 2016.

[7] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 506–522.

[8] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, "Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions," in *ACRYPTO 2005*.

[9] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," in *International Workshop on Information Security Applications*. Springer, 2004, pp. 73–86.

[10] J. W. Byun, H. S. Rhee, H.-A. Park, and D. H. Lee, "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *Workshop on secure data management*. Springer, 2006, pp. 75–83.

[11] H. S. Rhee, R. H. Park, R. Susilo, and R. H. Lee, "Trapdoor security in a searchable public-key encryption scheme with a designated tester," *Journal of Systems & Software*, vol. 83, no. 5, pp. 763–771, 2010.

[12] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack," *IEEE Transactions on computers*, vol. 62, no. 11, pp. 2266–2277, 2012.

[13] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "Dual-server public-key encryption with keyword search for secure cloud storage," *IEEE transactions on information forensics and security*, vol. 11, no. 4, pp. 789–798, 2015.

[14] Q. Huang and H. Li, "An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks," *Information Sciences*, vol. 403, pp. 1–14, 2017.

[15] M. Noroozi and Z. Eslami, "Public key authenticated encryption with keyword search: revisited," *IET Information Security*, vol. 13, no. 4, pp. 336–342, 2019.

[16] B. Qin, Y. Chen, Q. Huang, X. Liu, and D. Zheng, "Public-key authenticated encryption with keyword search revisited: Security model and constructions," *Information Sciences*, vol. 516, pp. 515–528, 2020.

[17] Z.-Y. Liu, Y.-F. Tseng, R. Tso, M. Mambo, and Y.-C. Chen, "Public-key authenticated encryption with keyword search: Cryptanalysis, enhanced security, and quantum-resistant instantiation," in *Proceedings of the 2022 ACM on Asia conference on computer and communications security*, 2022, pp. 423–436.

[18] Y.-C. Chen, "Speks: secure server-designation public key encryption with keyword search against keyword guessing attacks," *The Computer Journal*, vol. 58, no. 4, pp. 922–933, 2015.

[19] M. Noroozi and Z. Eslami, "Public-key encryption with keyword search: a generic construction secure against online and offline keyword guessing attacks," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, pp. 879–890, 2020.

[20] C.-K. Chu, S. S. Chow, W.-G. Tzeng, J. Zhou, and R. H. Deng, "Key-aggregate cryptosystem for scalable data sharing in cloud storage," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 2, pp. 468–477, 2013.

[21] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Annual international cryptology conference*. Springer, 2005, pp. 258–275.

[22] T. Li, Z. Liu, P. Li, C. Jia, Z. L. Jiang, and J. Li, "Verifiable searchable encryption with aggregate keys for data sharing in outsourcing storage," in *Australasian Conference on Information Security and Privacy*. Springer, 2016, pp. 153–169.

[23] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[24] J. Li, Q. Gan, F. Wang, and X. Wang, "Provably secure key aggregate searchable encryption with verifiability for online data sharing in cloud computing," *Available at SSRN 3992940*.

[25] Z. Liu and Y. Liu, "Verifiable and authenticated searchable encryption scheme with aggregate key in cloud storage," in *2018 14th International Conference on Computational Intelligence and Security (CIS)*, 2018.

[26] Z. Liu, T. Li, P. Li, C. Jia, and J. Li, "Verifiable searchable encryption with aggregate keys for data sharing system," *Future Generation Computer Systems*, vol. 78, pp. 778–788, 2018.

[27] R. Zhou, X. Zhang, X. Du, X. Wang, G. Yang, and M. Guizani, "File-centric multi-key aggregate keyword searchable encryption for industrial internet of things," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2018.

[28] X. Wang, Y. Xie, X. Cheng, and Z. Jiang, "An efficient key-aggregate keyword searchable encryption for data sharing in cloud storage," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019.

[29] J. Liu, B. Zhao, J. Qin, X. Hou, and J. Ma, "Key-aggregate searchable encryption supporting conjunctive queries for flexible data sharing in the cloud," *Information Sciences*, p. 119336, 2023.

[30] D. Boneh, "Aggregate and verifiably encrypted signatures form bilinear maps," *EUROCRYPT 2003*, 2003.

Jinlu Liu received the BS degree from the School of Mathematical Sciences, Shanxi University, P.R.China, in 2019. She is currently working toward the PhD degree in the School of Mathematics, Shandong University. Her research interests include cloud computing and applied cryptography.

Zhongkai Wei received the BS degree from the School of Physics, Shandong University, P.R.China, in 2018. He is currently working toward the PhD degree in the School of Mathematics, Shandong University. His current research interests focus on searchable encryption and cloud computing security.

Jing Qin received the BS degree from Information Engineering University, Zhenzhou, P.R.China, in 1982, and the PhD degree from the School of Mathematics, Shandong University, P.R.China, in 2004. She is a professor with the School of Mathematics, Shandong University P.R.China. Her research interests include information security, design and analysis of security about cryptologic protocols. She has coauthored 2 books and has published more than 30 professional research papers. She is a senior member of Chinese association for Cryptologic Research (CACR) as well as China Computer Federation (CCF).

Bo Zhao received the BS degree from the School of Mathematics and Information Science Hebei Normal University, P.R.China, in 2019, and the MS degree from the School of Mathematics, Shandong University, P.R.China, in 2022. He currently works for Huakong Qingjiao Technology Co., Ltd., BeiJing, China. His research interests include searchable encryption and secure multi-party computation.

Jixin Ma received the BSc and MSc degrees in mathematics, in 1982 and 1988, respectively, and the PhD degree in computer sciences, in 1994. He is a reader of computer science with the Department of Computing and Information Systems, and the director of the Centre for Computer and Computational Science, as well as the leader of Artificial Intelligence Research Group, at the University of Greenwich, United Kingdom. His main research areas include artificial intelligence, software engineering and information systems, with special interest in information security.