Received December 6, 2019, accepted December 16, 2019, date of publication December 20, 2019, date of current version January 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2961169

Efficient Verifiable Key-Aggregate Keyword Searchable Encryption for Data Sharing in Outsourcing Storage

XUQI WANG[1], **XIANGGUO CHENG**[1], **AND YU XIE**[2]
[1]College of Computer Science and Technology, Qingdao University, Qingdao 266071, China
[2]Department of Computer Science, Tongji University, Shanghai 201804, China

Corresponding author: Xiangguo Cheng (15964252399@163.com)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61572267 and Grant 61303197, and in part by the Shandong Provincial Natural Science Foundation under Grant ZR2016FQ02.

ABSTRACT In a secure data sharing system, users can selectively retrieve encrypted files by performing keyword search over the ciphertext of data. Most of the existing searchable encryption schemes can provide security protection for both data owner and users. Nevertheless, three pivotal issues need to be addressed. Firstly, the cloud might return a wrong result or incomplete result for some reasons, e.g., saving the computing resources. Secondly, users need to store massive keys to generate trapdoors and decrypt the ciphertext of data, which brings great challenges to users' key management. Thirdly, when users perform keyword search over a large number of files, they need to generate and submit massive trapdoors, which is unrealistic. Proceeding from these points, in this paper, we propose an efficient verifiable key-aggregate keyword searchable encryption (EVKAKSE) scheme. In this scheme, the data owner distributes only one aggregate key to users for keyword search, decryption and verification, who can use the aggregate key to generate a single trapdoor for keyword search over shared files. Generally, we define the requirements of the scheme, analyze the threat models and give a valid construction. Furthermore, our security analysis and experimental evaluation demonstrate that the scheme is efficient and secure.

INDEX TERMS Searchable encryption, cloud storage, trapdoor privacy, security.

I. INTRODUCTION

With the increasing storage demand of enterprises and individuals, cloud storage has been widely used in recent years. Enterprises usually store business data through commercial outsourcing storage systems (such as Dropbox and Syncany) and share it with employees, while individuals use social applications (e.g., Facebook and Twitter) to share personal videos and photos with family and friends.

The security issues caused by cloud storage such as inadvertent data leaks cannot be overlooked. A common approach is that the data owner upload encrypted files to the cloud, and only authorized users can retrieve and decrypt these files using decryption keys. However, encrypted data makes it challenging for users to search. Searchable encryption(SE) [1] is a common solution in which the data owner is required to encrypt potential keywords and upload the ciphertexts of keywords to the cloud together with encrypted data. The majority of existing searchable symmetric encryption (SSE) schemes [6], [7], [12], [13] and public key encryption with keyword search (PEKS) schemes [14]–[16] can ensure the privacy and confidentiality when performing keyword search. Besides, attribute based encryption (ABE) schemes [17]–[19] are applied to achieve fine-grained access control aware keyword search. Nevertheless, the data owner needs to distribute a large number of keys linearly related to the number of files to users who need to store these keys accordingly, which will increase the difficulty of key management when the number of keys is large. Additionally, when users perform keyword search over the shared files, they also need to submit the massive trapdoors to the cloud. Some recent work [20]–[22] focuses on multi-user searchable encryption (MUSE) which mainly solves the access control of users for files, whereas how to reduce the number of shared keys and trapdoors is still not considered. Note that Liu *et al.* [23] seems to solve the problems of key redundancy and efficiency, but how to control which user can access which file is not considered. Therefore, in the premise of

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaochun Cheng.

controlling the user's access to the file, the storage overhead, computational overhead of users and communication costs are supposed to be considered.

In addition, the commercial cloud [25] may perform keyword search over partial shared files instead of the whole for saving computing resources, and then return the corresponding results. Furthermore, malicious clouds may directly return wrong results, so it is necessary to add authentication mechanism in SE scheme.

Recently, many key-aggregate searchable encryption schemes [4], [5], [26], [30] have been proposed. Although these schemes solve the key redundancy for keyword search, the number of decryption keys is still linear with the number of shared files. However, their actual operation is not efficient enough, and the cloud need to bear a large computational overhead especially in the establishment phase. Besides, there are also numerous key-aggregate searchable encryption schemes that lack effective verification mechanisms. Only honest clouds are considered in those schemes, and semi-honest (even untrustworthy) clouds are not discussed in their threat model.

In this paper, we propose an efficient verifiable key-aggregate keyword searchable encryption (EVKAKSE). In this scheme, the data owner does not need to generate a large number of keys, but only generate an aggregate key containing file access privileges for users, and the aggregate key should be stored by users for keyword search, decryption and verification. Users only need to generate an aggregate trapdoor to perform keyword search over all privileged files. We address some of the security issues that exist in the key-aggregate schemes. In fact, our goal is to settle how the data owner delegates the access right of the files to the user through an aggregate key in a secure and low-cost case. The contribution summarized as follows:

- We give a concrete construction of key-aggregate searchable encryption named EVKAKSE. In the construction, we design an algorithm that the data owner only needs to generate a single aggregate key for keyword search, decryption and verification.
- The EVKAKSE scheme greatly reduces computational overhead of cloud server and data owner, which is more efficient than the most of existing searchable encryption with aggregate key.
- We address eavesdropping problem which is an internal attack, and we prefer to term it trapdoor attack, i.e., a malicious user can infer the aggregate key through the trapdoor submitted by valid users.
- We present the corresponding security definitions and prove their security. And we conduct related performance evaluation which confirms that the EVKAKSE scheme is efficient and practical.

II. RELATED WORK

A. MULTI-USER SEARCHABLE ENCRYPTION

In recent years, many multi-user searchable encryption schemes including SSE [8]–[10] and PEKS [15], [16] have been proposed. The data owner always wants to share their files with a group of authorized users who can perform keyword search over the shared files. In MUSE schemes [11], [20]–[22], [24], different keys are always used for different files during both search and decryption for confidentiality considerations. Generally, users always store massive keys and the key management is difficult. The target of MUSE is how to control the user's access right for files. Specifically, an authorized user cannot access a file out of scope of privileges.

B. VERIFIABLE SEARCHABLE ENCRYPTION

Chai and Gong [25] presented a threat model, in which a ''semi-honest but curious'' server satisfies: (1) the server does not modify the stored ciphertexts of data; (2) the cloud may try to learn some information from the uploaded encrypted files and the keyword ciphertexts as well as the trapdoors submitted by users; (3) the cloud may perform keyword search over partial shared files for saving computing resources or other reasons and return the corresponding results.

Some verifiable keyword search over plaintext schemes [31], [32] have been proposed, but those scheme are weak in this threat model. To ensure the privacy of keyword, the access control of the verification is necessary [33].

C. KEY-AGGREGATE METHOD

The key-aggregate searchable encryption scheme is mainly constructed by the technology of broadcast encryption [28]. To solve the key redundancy in a data sharing system, Chu *et al.* [2] used broadcast encryption proposed by Boneh *et al.* [3] to construct a key-aggregate encryption (KAE) scheme, in which a user can use a single key to access files with his/her shared privileges, but it does not support the keyword search. Cui *et al.* [4] presented a key-aggregate searchable encryption (KASE) scheme based on KAE. Afterwards, Li *et al.* [5] proposed a similar construction (VSEAK) in which the data owner generates a aggregate key to users for both search and verification. However, Zhou *et al.* [26] pointed a eavesdropping problem which is an internal attack, a malicious user can calculate other valid users' aggregate keys by their submitted trapdoors and proposed a viable scheme based on Kiayias's scheme [29]. In fact, Zhou's scheme focuses on the application of searchable encryption with aggregate key under the industrial Internet of things (IIoT) [27]. And Shu *et al.* [30] used the crowdsourcing structure to construct the a new key-aggregate searchable encryption scheme. The idea of our study is mainly inspired by the key-aggregate method.

III. SYSTEM MODEL AND SECURITY DEFINITIONS

A. SYSTEM MODEL

As shown in Figure.1, The EVKAKSE system model involves four different entities: the data owner, the user, the aid server and the cloud server. For ease of understanding, we use an example to describe the system. Suppose that Alice

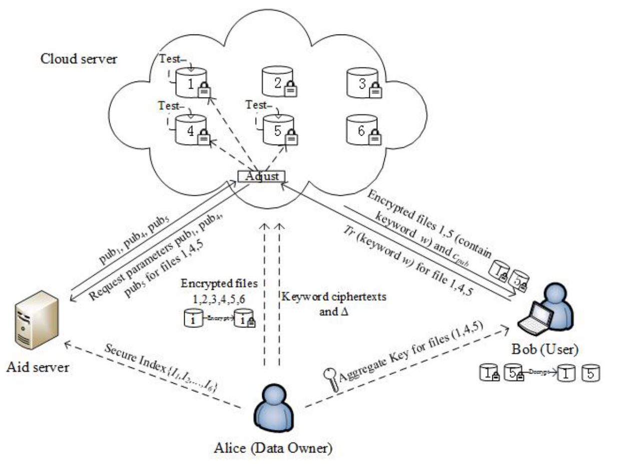

Cloud server
Test
1
2
3
Test
4
Test
5
6
Adjust
pub_1, pub_4, pub_5
Request parameters pub_1, pub_4, pub_5 for files 1,4,5
Encrypted files 1,5 (contain keyword w) and c_{pub}
Tr (keyword w) for file 1,4,5
Encrypted files 1,2,3,4,5,6
Encrypt
Keyword ciphertexts and Δ
Aid server
Bob (User)
Decrypt
Secure Index $\{I_1, I_2, \ldots, I_6\}$
Aggregate Key for files (1,4,5)
Alice (Data Owner)

FIGURE 1. System model.

(data owner) encrypts six files and uploads these encrypted files and related information to the cloud server. After that, Alice sends an aggregate key for file 1,4,5 to Bob (user) securely and submits the secure index of six files to the aid server through secure channel. Bob may perform keyword search with the aggregate key and generate the aggregate trapdoor Tr of a keyword w, then submits the Tr to cloud server. When the cloud server receives the Tr of Bob, it will request some parameters from the aid server. After confirming Bob's shared file set S, the aid server generates $|S|$ parameters pub_1, pub_4, pub_5 and sends them to the cloud. The cloud runs the algorithm for the ciphertexts of the encrypted files 1,4,5 with the Tr and the received pub_1, pub_4, pub_5 to detect whether the file contains the keyword. If a file contains the keyword, the cloud will send the encrypted files and related parameter to Bob. Subsequently, Bob uses his aggregate key and received parameters to decrypt the received ciphertexts of data and verify the integrity of data.

Note that we can set the data owner's private server as an aid server. It always returns the correct parameters to clouds and will never offer some sensitive information to clouds or users.

The cloud server uses the **Setup** algorithm to generate public parameters to set up the scheme. The data owner uses the **KeyGen** algorithm to produce the public/master-secret key pair. The secure index of each file is generated by the data owner using the **BuildIndex** algorithm. And the data owner uses the **Encrypt** algorithm to generate a Δ_i and symmetric key, encrypt keywords and files. While the data owner uses the **TagGen** algorithm to generate verification information. If the data owner shares a group of selected files with a user, he/she can use his/her master-secret key and secure index to generate an aggregate key by the **Extract** algorithm and the aggregate key will be securely distributed to the user. A valid user performs keyword search, he/she will use his/her aggregate key and a keyword w to produce an aggregate trapdoor by the **Trapdoor** algorithm and the trapdoor will be submitted to the cloud server. Receiving the submitted trapdoor, the cloud server will request some parameters from the aid server by the **Adjust** algorithm and perform keyword search according to the shared file set of the user by the **Test** algorithm. After receiving encrypted files and related parameters, the user will use **Decrypt** algorithm to decrypt the ciphertexts of data. Finally, the user can use **Verify** algorithm to verify data integrity.

Formally, the general framework of our scheme is composed of eleven algorithms (**Setup**, **KeyGen**, **BuildIndex**, **Encrypt**, **TagGen**, **Extract**, **Trapdoor**, **Adjust**, **Test**, **Decrypt**, **Verify**).

1) **Setup**(k): With the input of a security parameter k, the cloud server executes this algorithm to generate the system parameters.
2) **KeyGen**: The data owner uses this algorithm to generate his public/mater-secret key pair (pk, msk).
3) **BuildIndex**: The data owner uses this algorithm to produce secure index req_{index} for all shared files. And the req_{index} should be stored in aid server securely.
4) **Encrypt**(pk, req_{index}, i): The data owner uses this algorithm to generate a symmetric encryption key and encrypt the i^{th} file with this key. Then the data owner encrypts keywords of the i^{th} file with the pk and the req_{index}. After that, the data owner produces a Δ_i. Finally, the algorithm outputs the Δ_i, encrypted files and ciphertexts of keywords.
5) **TagGen**($W, \mathcal{F}w$): W denotes the keyword set and $\mathcal{F}w$ denotes the file set containing same keyword. The data owner uses this algorithm to generate the verification tags for all uploaded files. And these tags should be publicly stored in the aid server.
6) **Extract**(msk, req_{index}, S): The data owner uses this algorithm to generate an aggregate searchable encryption key for the users with the access right of a selected files set S. It takes as input the master-secret key msk, the secure index req_{index} and the set S, outputs the aggregate key K_{agg}. Eventually, the aggregate key will be securely distributed to the user.
7) **Trapdoor**(K_{agg}, w): The user uses this algorithm to produce a single trapdoor Tr with his/her aggregate key K_{agg} and a specified keyword w. After that, the Tr is submitted to the cloud server.
8) **Adjust**(i, S, req_{index}): The aid server uses this algorithm to generate $|S|$ parameters $\{pub_i | i \in S\}$ for all certain files in set S. After that, the aid server sends these parameters to the cloud server.
9) **Test**($Tr, i, S, pub_i, \Delta_i$): Receiving the submitted Tr and pub_i, the cloud server uses this deterministic algorithm to find that if the files in set S contain the keyword w which derived from the Tr. If a encrypted file contains the w, the cloud will send this encrypted file C and a related parameter c_{pub_i} to the user.
10) **Decrypt**(i, c_{pub_i}, C): Receiving the returned C and c_{pub_i} from the cloud, the user uses this algorithm to compute the symmetric key. Then the algorithm uses the symmetric key to decrypt the encrypted file and obtain primitive file F.

11) **Verify**(w, F_i): F_i denotes the i^{th} primitive file F. The user uses this algorithm to verify the integrity of retrieved files by comparing the computing result to the tags stored in the aid server. If there is a tag_w that equals the computing result, we can say that the received data is complete.

B. THREAT MODELS

According to the behaviors of the cloud and users, the security threats in this system can be categorized into three models:

1) SEMI-HONEST BUT CURIOUS CLOUD SERVER

Assume that the public cloud is "semi-honest but curious" ("honest but curious"), it will execute the algorithm to complete the whole (or a fraction of) query task submitted by the user and return the corresponding results. However, the cloud may try to learn some information from the uploaded encrypted files and the keyword ciphertexts as well as the trapdoors submitted by users.

2) MALICIOUS CLOUD SERVER

The cloud server can arbitrarily deviate from the prescribed protocol, and attempt to learn sensitive information as much as it can. Worse still, it may intentionally return wrong files to the user. If the malicious cloud colludes with the malicious user, it will attempt to decrypt the unauthorized files or speculate the keywords contained in the files. Obviously, if an algorithm is secure under the "malicious" model, it is still secure under the "honest but curious" model.

3) TRAPDOOR ATTACK

It is an internal attack. Assume that a malicious user eavesdrops a trapdoor submitted from a valid user. The malicious user may try to calculate the aggregate key of the valid user from the trapdoor.

C. DESIGN GOALS

The EVKAKSE scheme should satisfy five requirements as follows.

- **Keyword search.** This scheme should enable users to generate desired trapdoors for any given keyword to perform keyword search over encrypted files. And the cloud always returns the correct result by running **Test** algorithm.
- **Efficiency.** Each algorithm of the scheme should be efficient. In particular, the data owner and users do not have to bear significant computing and storage overhead.
- **Compactness.** It requires that the size of the aggregate key should be independent of the maximum of files.
- **Access Authorization.** This scheme can generate an aggregate key for a valid user, which contains the keyword search right for shared files. It means that the data owner can delegate keyword search right to a user through an aggregate key.
- **Verification.** This requirement is central to detect if the cloud has malicious fraud behavior. Users can verify the integrity of received files. Note that this process needs the aid server to provide the correct verification tags generated by the data owner.

D. SECURITY DEFINITIONS

As introduced in Section I, a well-designed EVKAKSE scheme should satisfy three security requirements. We reformulate their definitions as follows.

*Definition 1 (Correctness): Given some search task for the keyword $w \in \mathcal{W}$ of some files, a secure key-aggregate searchable encryption scheme is correct if $(g, h, \mathcal{B}, H_0, H_1, \mathcal{E}nc, \mathcal{D}ec) \leftarrow$ **Setup**(k), $(pk, msk) \leftarrow$ **KeyGen**, $req_{index} \leftarrow$ **BuildIndex**, $(C, \Delta_i, c_w) \leftarrow$ **Encrypt**(pk, req_{index}, i), $tag_w \leftarrow$ **TagGen**$(W, \mathcal{F}w)$ and $K_{agg} \leftarrow$ **Extract**(msk, req_{index}, S) such that, for any valid trapdoors submitted by an authorized user $(Tr, S) \leftarrow$ **Trapdoor**(K_{agg}, w), the cloud will request adjust parameters from the aid server $\{pub_i | i \in S\} \leftarrow$ **Adjust**(S) and perform keyword search over certain encrypted files $c_{pub_i} \leftarrow$ **Test**$(Tr, i, S, req_{index}, pub_i, \Delta_i)$. After receiving the encrypted files returned from the cloud, the user will correctly decrypt ciphertext of data $F \leftarrow$ **Decrypt**(i, c_{pub_i}, C) and successfully verify the integrity of data ture $\leftarrow$ **Verify**(w, F_i).*

Definition 2 (Privacy): For users, given some search task for keywords $w \in \mathcal{W}$ of some files, a secure key-aggregate searchable encryption scheme satisfy privacy if the keyword w and aggregate key K_{agg} cannot be calculated from Tr with published information. And the cloud cannot decrypt the encrypted files even in collusion with the unauthorized malicious user.

Definition 3 (Search Control): A valid user cannot perform keyword search for unauthorized files successfully with the known aggregate key, and the **Test** *algorithm will always return false.*

IV. PRELIMINARIES

In this section, we review the cryptology concepts and complexity assumption used in this paper.

A. BILINEAR MAP

Let $\mathcal{G}$ and $\mathcal{G}_1$ be two multiplicative cyclic group of prime order p and g be a generator of $\mathcal{G}$. A bilinear map e is a map $e : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}_1$ with the following properties:

- Bilinearity: for all $u, v \in \mathcal{G}$ and $a, b \in Z_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$;
- Non-degeneracy: $e(g, g) \neq 1$;
- Computability: There is an efficient algorithm to compute $e(u, v)$ for any $u, v \in \mathcal{G}$;

B. TRUNCATED DECISIONAL Q-AUGMENTED BILINEAR DIFFIE-HELLMAN EXPONENT ASSUMPTION

The truncated decisional q-augmented bilinear Diffie-Hellman exponent assumption [34] has been used to design broadcast encryption scheme [35] and the scheme has been proved IND-CCA2 security under standard model.

Given a group $\mathcal{G}$ of prime order q, randomly choose two generators $g, g' \in \mathcal{G}$ and a random $\alpha \in_R Z_q$. Given a tuple $Tu = (g', g'_{q+2}, g, g_1, \cdots, g_q)$ and $Z \in \mathcal{G}$, where we use g_i, g'_i donate $g^{(a^i)}, g'^{(a^i)}$, the truncated decisional q-ABDHE problem is to decide whether Z equals to $e(g', g_{q+1})$ or to a random element of $\mathcal{G}_1$.

C. DECISIONAL BILINEAR DIFFIE-HELLMAN ASSUMPTION

Given a group G of prime order q, choose a generator $g \in \mathcal{G}$ and $a, b, c \in Z_q$ randomly. Given a tuple (g^a, g^b, g^c) and $Z \in \mathcal{G}_1$, the decisional BDH problem is to decide whether Z equals to $e(g, g)^{abc}$ or a random element of G_1.

V. THE PROPOSED SCHEME

For convenience, we summarize some common notations throughout the paper shown in Table.1.

TABLE 1. Some frequently used symbols.

Symbols	Descriptions
S	The selected (shared) file set
$\mathcal{F}w$	The file set containing same keyword
$\mathcal{F}t$	The retrieved file set
W	The keyword set
w	The arbitrarily selected keyword in W
c_w	The ciphertext of w
F	The arbitrarily selected file
F_i	The i-th file
C	The encrypted file of F
k_s	The symmetric key
req_{index}	The secure index
I_i	The i-th random in req_{index}
tag_w	The verification tag for w

Inspired by [4], [26] and [36] schemes, we propose a concrete EVKAKSE scheme. It is described as follows.

1) **Setup**(k): given the security parameter k, the cloud server executes operations as follows:
   - Generate a bilinear map group system $\mathcal{B} = (p, \mathcal{G}, \mathcal{G}_1, e(\cdot, \cdot))$, where $\mathcal{G}$ is a cyclic multiplicative group of prime order p, while $|p| \leq k$ and we set the maximum number of files as n while $n = p$.
   - Randomly choose generators $g, h \in \mathcal{G}$
   - Select two one-way hash functions $H_0 : \{0, 1\}^* \to Z_p$ and $H_1 : \{0, 1\}^* \to \mathcal{G}$.
   - Select a symmetric encryption algorithm $\mathcal{E}nc$ and a corresponding symmetric decryption algorithm $\mathcal{D}ec$.

   Finally, Output the system parameters $params = (g, h, \mathcal{B}, H_0, H_1, \mathcal{E}nc, \mathcal{D}ec)$.
2) **KeyGen**: the data owner randomly chooses $\alpha \in_R Z_p$, let $v = g^{\alpha}$, outputs:
$$pk = v, msk = \alpha$$
   The pk is the public key and the msk is the master-secret key.
3) **BuildIndex**: the data owner chooses n different randoms $I_i \in_R Z_p$ for each file to generate secure index $i \in \{1, 2, \ldots, n\}$:
$$req_{index} = (I_1, I_2, \cdots, I_n)$$
   The data owner is supposed to send the req_{index} to the aid server securely.
4) **Encrypt**(pk, req_{index}, i): the data owner generates the symmetric key and uses this key to encrypt i^{th} file by the symmetric encryption algorithm $\mathcal{E}nc$. Then the data owner encrypts the keywords in this file. This algorithm takes as input the file set index $i \in \{1, 2, \ldots, n\}$, the pk and the req_{index}, computes that:
   - randomly chooses a $\tau \in_R Z_p$ for the i^{th} file.
   - generates the symmetric encryption key k_s which is an element in $\mathcal{G}_1$:
$$k_s = \frac{e(g, h)^{\tau}}{e(g, g)^{\tau}}$$
   - encrypts the file as:
$$C = \mathcal{E}nc(F, k_s)$$
   - generates $\Delta_i = (c_0, c_1)$ for τ by computing:
$$c_0 = v^{\tau}, c_1 = g^{-\tau I_i}$$
   - generates ciphertext c_w for a keyword w as:
$$c_w = \frac{e(g, h)^{\tau}}{e(g, g)^{\tau H_0(w)}}$$

   Note that the Δ_i and c_w should be stored in cloud server. For saving the computational overhead, the $e(g, h)$ and $e(g, g)$ should be pre-computed.
5) **TagGen**$(W, \mathcal{F}w)$: The data owner uses this algorithm to generate verification tag for each $w \in W$:
$$tag_w = (\prod_{i \in \mathcal{F}w} H_1(F_i))^{H_0(w)}$$
   Note that the tag_w should be stored in aid server.
6) **Extract**(msk, req_{index}, S): the data owner generates the aggregate searchable encryption key for a selected set S of the universe of files. This algorithm takes as input master-secret keys msk, the req_{index}, the set S and outputs the aggregate key $K_{agg} = (k_{agg,0}, k_{agg,1})$ by computing:
$$k_{agg,0} = g^{-1/(\alpha - \prod_{i \in S} I_i)},$$
$$k_{agg,1} = h^{1/(\alpha - \prod_{i \in S} I_i)}$$
   Note that if $\prod_{i \in S} I_i = \alpha$, the **Extract** algorithm aborts. The data owner sends K_{agg} to the authorized user and delegates keyword search right of the file set S to the user.
7) **Trapdoor**(K_{agg}, w): the user generates only one trapdoor Tr of keyword w and performs keyword

search for shared files associated with K_{agg} by this algorithm as:

$$Tr = {k_{agg,0}}^{H(w)} k_{agg,1}$$

Then, the user sends (Tr, S) to the cloud server.

8) **Adjust**(i, S, req_{index}): this algorithm takes as input the selected set S and generates the certain parameters for keyword search by the aid server. And this process includes two steps:
   - Step 1: the cloud server requests adjust parameters $\{pub_i | i \in S\}$ from the aid server according to the set S of users.
   - Step 2: the aid server returns adjust parameters $\{pub_i | i \in S\}$ to the cloud server.

   The aid server computes pub_i with req_{index}:

$$pub_i = \prod_{j \in S, j \neq i} I_j$$

9) **Test**$(Tr, i, S, pub_i, \Delta_i)$: the cloud server uses this algorithm to perform keyword search over the i^{th} file in the set S. This algorithm takes as input the trapdoor Tr, the req_{index}, the $\Delta_i = (c_0, c_1)$ relevant to its secret parameter τ, and verifies the equation:

$$c_w \overset{?}{=} e\left(c_0 c_1^{pub_i}, Tr\right)$$

   If the equation holds, let $c_{pub_i} = c_0 c_1^{pub_i}$ and the cloud will send C and c_{pub_i} to the user.

10) **Decrypt**(i, c_{pub_i}, C): the user uses this algorithm to decrypt the files retrieved from the cloud. The user obtains the symmetric key k_s by computing:

$$k_s = e(c_{pub_i}, k_{agg,0} k_{agg,1})$$

   And the user can decrypt the ciphertext of i^{th} file with k_s:

$$F = \mathcal{D}ec(C, k_s)$$

11) **Verify**$(w, F_i, \mathcal{F}t)$: the user uses this algorithm to verify the integrity of received files. The algorithm takes as input the keyword w and primitive files and computes:

$$tag'_w = (\prod_{i \in \mathcal{F}t} H_1(F_i))^{H_0(w)}$$

   The algorithm needs to find that if there is a $tag_w = tag'_w$ existing in the aid server, and output *true* or *false* according to the result.

If the file set S only contains one file, it is a practical public key encryption with keyword search scheme.

VI. THE SPECIAL CASE

Sometimes, an online and honest aid server setting is difficult to deploy, so we modify the proposed scheme and present a special case.

As shown in Figure.2, when a new user (Bob) joins in this system, the data owner (Alice) will generate an aggregate key and send it to the user. Meanwhile, the data owner will generate all related parameters of the user at once and upload them to the cloud server.

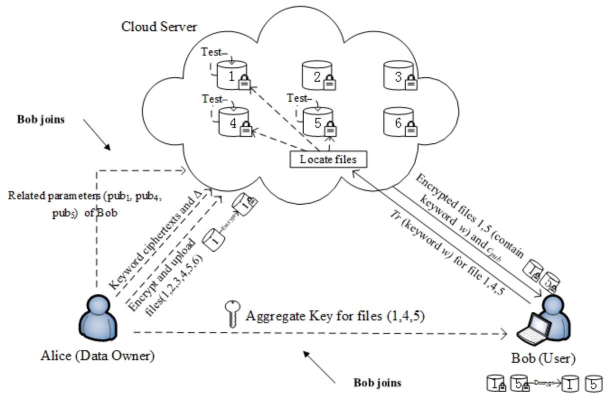

Cloud Server
Test
Test
Test
1
2
3
4
5
6
Locate files
Bob joins
Related parameters (pub1, pub4, pub5) of Bob
Keyword ciphertexts and Δ
Encrypt and upload files(1,2,3,4,5,6)
Encrypt
Encrypted files 1,5 (contain keyword w) and c_pub
Tr (keyword w) for file 1,4,5
Aggregate Key for files (1,4,5)
Alice (Data Owner)
Bob (User)
Bob joins
Decrypt

FIGURE 2. Special system model.

To remove the aid server, we modify the **Extract** algorithm and remove the **Adjust** algorithm. Notice that verification tags are still generated by **TegGen** algorithm, but those tags should be stored in the cloud server. The other algorithms are the same as before.

Extract(msk, req_{index}, S): the data owner generates the aggregate searchable encryption key for a selected set S of the universe of files. This algorithm takes as input master-secret keys msk, the req_{index}, the set S and outputs the aggregate key $K_{agg} = (k_{agg,0}, k_{agg,1})$ by computing:

$$k_{agg,0} = g^{-1/(\alpha - \prod_{i \in S} I_i)},$$
$$k_{agg,1} = h^{1/(\alpha - \prod_{i \in S} I_i)}$$

Note that if $\prod_{i \in S} I_i = \alpha$, the **Extract** algorithm aborts. Next, the data owner generates the related parameters pub_i for each $i \in S$ by computing:

$$pub = \prod_{j \in S} I_j$$

(pub only needs to be computed once for each user.)

$$pub_i = \frac{pub}{I_i}$$

The data owner sends K_{agg} to the authorized user and uploads all related parameters $\{pub_i | i \in S\}$ to the cloud server.

When the cloud server receives a trapdoor submitted from a user, it will perform keyword search with the stored parameters directly instead of requesting related parameters from an aid server.

Considering of the security, the set S should contain at least three files.

VII. CORRECTNESS AND SECURITY ANALYSIS

In this section, we give strictly proofs on the correctness of our scheme, and analyze its security according to the definitions modeled in section III-D.

A. CORRECTNESS

Informally, the correctness means that:

1) The **Test** algorithm performs keyword search correctly with submitted (Tr, S) and returns the correct result for users.
2) Users can decrypt the ciphertexts of the downloaded files correctly with his/her own aggregate key.
3) Users can use the decrypted information to verify whether the cloud is fraudulent.

Theorem 1: The proposed construction is correct.

Proof: The correctness of both the keyword search function and verification function is equivalent to which of the scheme. The user uses **Trapdoor** algorithm to generate the correct trapdoor Tr, and submits (Tr, S) to the cloud. After that, the cloud server uses **Test** algorithm to perform keyword search. This process is as follows:

$$\begin{aligned}
&e\left(c_{pub_i}, Tr\right)\\
&= e\left(c_0 c_1^{\prod_{j\in S, j\neq i} I_j}, Tr\right)\\
&= e\left(v^{\tau} g^{-\tau\prod_{i\in S} I_i}, k_{agg,0}{}^{H(w)} k_{agg,1}\right)\\
&= e\left(v^{\tau} g^{-\tau\prod_{i\in S} I_i}, \left(hg^{-H(w)}\right)^{1/(\alpha-\prod_{i\in S} I_i)}\right)\\
&= \frac{e\left(g^{\alpha\tau} g^{-\tau\prod_{i\in S} I_i}, h^{1/(\alpha-\prod_{i\in S} I_i)}\right)}{e\left(g^{\alpha\tau} g^{-\tau\prod_{i\in S} I_i}, g^{H(w)/(\alpha-\prod_{i\in S} I_i)}\right)}\\
&= \frac{e\left(g^{\tau(\alpha-\prod_{i\in S} I_i)}, h^{1/(\alpha-\prod_{i\in S} I_i)}\right)}{e\left(g^{\tau(\alpha-\prod_{i\in S} I_i)}, g^{H(w)/(\alpha-\prod_{i\in S} I_i)}\right)}\\
&= \frac{e\left(g^{\tau}, h\right)}{e\left(g^{\tau}, g^{H(w)}\right)}\\
&= \frac{e\left(g, h\right)^{\tau}}{e\left(g, g\right)^{\tau H(w)}}\\
&= c_w
\end{aligned}$$

The user can perform a keyword search successful with his/her own aggregate key.

To get the symmetric key for decryption, the user uses his/her aggregate key and received parameter c_{pub_i} to compute k_s for the i^{th} file. We can see that:

$$\begin{aligned}
&e\left(c_{pub_i}, k_{agg,0} k_{agg,1}\right)\\
&= e\left(c_0 c_1^{\prod_{j\in S, j\neq i} I_j}, k_{agg,0} k_{agg,1}\right)\\
&= e\left(v^{\tau} g^{-\tau\prod_{i\in S} I_i}, (hg)^{-1/(\alpha-\prod_{i\in S} I_i)}\right)\\
&= \frac{e\left(g^{\alpha\tau} g^{-\tau\prod_{i\in S} I_i}, h^{1/(\alpha-\prod_{i\in S} I_i)}\right)}{e\left(g^{\alpha\tau} g^{-\tau\prod_{i\in S} I_i}, g^{1/(\alpha-\prod_{i\in S} I_i)}\right)}\\
&= \frac{e\left(g^{\tau}, h\right)}{e\left(g^{\tau}, g\right)}\\
&= \frac{e\left(g, h\right)^{\tau}}{e\left(g, g\right)^{\tau}}\\
&= k_s
\end{aligned}$$

After getting the k_s, the user can decrypt ciphertext C and obtain the decrypted file by computing:

$$F = \mathcal{D}ec(C, k_s)$$

Finally, for verification, the user computes that:

$$tag'_w = \Big(\prod_{i\in \mathcal{F}t} H_1(F_i)\Big)^{H_0(w)}$$

If the set of received files $\mathcal{F}t$ is equal to the actual set of files $\mathcal{F}w$. We can deduce that $tag'_w = tag_w$, Thus whether the cloud perform keyword search operations completely is known.

As shown above, the EVKAKSE scheme is correct. □

B. PRIVACY

In this section, we prove that the cloud server may learn something from the submitted trapdoor and the published information, but it will not affect the privacy of the user and data owner. Kiayias *et al.* [29] and Zhou *et al.* [26] presented the trapdoor attack problem detailed, and we analyze the security issue of trapdoor based on above work.

Theorem 2: The cloud server is unable to determine a keyword from any published information.

Proof: The cloud server may obtain some information from submitted trapdoor or public parameters stored in cloud according to Definition 2. Theorem 2 can be deduced from the following lemmas: □

Lemma 1: The cloud server cannot determine the keyword from submitted trapdoor.

Proof: Assume a valid user who has access to a file set S submits Tr for the keyword w search, while $Tr = k_{agg,0}^{H(w)} k_{agg,1}$ is an element in group $\mathcal{G}$. We can see that $Tr = \left(hg^{-H(w)}\right)^{1/(\alpha-\prod_{i\in S} I_i)}$, while h is an element in group G generated from g. Let $g^{\lambda} = h$, while $\lambda \in_R Z_p$. Then $Tr = g^{(\lambda-H(w))/(\alpha-\prod_{i\in S} I_i)}$. Note that the attacker cannot calculate $g^{1/(\alpha-\prod_{i\in S} I_i)}$ from the public parameters without the master-secret keys. Even the cloud server has K_{agg}, he cannot know the keyword contained in the Tr because of discrete logarithm problem. Actually, this result is ensured by the assumption of the intractability of truncated decisional q-ABDHE problem. □

Lemma 2: The cloud server cannot determine the keyword from the ciphertext of keyword and the related published information.

Proof: The c_w is a random element in group $\mathcal{G}_1$. If the cloud server determines the keyword w from keyword ciphertexts $c_w = e\left(g, h\right)^{\tau}/e\left(g, g\right)^{\tau H(w)}$ that it needs to solve discrete logarithm problem. Obviously, there is no polynomial time algorithm can solve this problem. □

Lemma 3: The cloud server cannot calculate the symmetric key from any published information.

Proof: From the conclusions of lemma 1 and lemma 2, the cloud server cannot calculate g^{τ} or τ from v^{τ}. So, there is no way to use g^{τ} to generate $k_s = e(g, h)^{\tau}/e(g, g)^{\tau}$. If the cloud want to calculate k_s in terms of $c_w = e\left(g, h\right)^{\tau}/e\left(g, g\right)^{\tau H(w)}$, it needs to know τ and the keyword w. The cloud cannot calculate k_s from published information in this case. □

Theorem 3: Ideally, an insider attacker cannot obtain a valid user's aggregate key from the trapdoor by eavesdropping.

Proof: Known by Lemma 1, it is difficult to calculate the keyword and aggregate key from a single trapdoor. However, when the attacker gets a bunch of trapdoors from a single user, the situation will be different. Suppose there is an insider attacker who continuously eavesdrops on the trapdoors submitted by a valid users.

Assume the attacker obtains $Tr_0 = k_{agg,0}^{H(w_0)} k_{agg,1}$ and $Tr_1 = k_{agg,0}^{H(w_1)} k_{agg,1}$ from a valid user, he/she can easily compute $Tr_\alpha = \frac{Tr_0}{Tr_1} = k_{agg,0}^{H(w_0)-H(w_1)}$.

If the valid user's set of keywords is F_1 ($w_0, w_1 \in F_1$) and the insider attacker's set of keywords is F_2, and $F_1 \subseteq F_2$. Let $t = |F_2|$, the attacker need to calculate two exact values $\frac{1}{H(w_0)-H(w_1)}$ and $H(w_0)$ to get $k_{agg,0} = T_\alpha^{1/(H(w_0)-H(w_1))}$ and $k_{agg,1} = \frac{Tr_0}{k_{agg,0}^{H(w_0)}}$. The probability of successful attack is $\frac{1}{t^2}$. When t is large enough, the probability can be negligible. Notice that if $F_1 \nsubseteq F_2$, the probability of successful attack depends on whether the discrete logarithm problem can be solved. □

C. SEARCH CONTROL

A valid user may try to access files which is authorized or unauthorized. Informally, we need assume that if there are malicious users even collude with the cloud server, it will have a no-neglected impact on aggregate key security and search control.

Theorem 4: A valid user cannot perform a keyword search for an unauthorized file with his/her aggregate key.

Proof: Assume that a user Bob performs a keyword w search for a file set S' of Amy with his aggregate key K_{agg}, while the file set of Bob is S, and $S \neq S'$. Bob submits (Tr, S') to the cloud server and the cloud uses **Test** algorithm to perform keyword search for each file in the file set S'. We should consider two scenarios. In the first case, the cloud colludes with Bob and requests the parameters for S' from the aid server by **Adjust** algorithm, and computes as follows:

$$e\left(c_0 c_1^{\prod_{j\in S', j\neq i} I_j}, Tr\right) = \frac{e\left(g^{\tau\left(\alpha-\prod_{j\in S'} I_j\right)}, h^{1/\left(\alpha-\prod_{j\in S} I_j\right)}\right)}{e\left(g^{\tau\left(\alpha-\prod_{j\in S'} I_j\right)}, g^{H(w)/\left(\alpha-\prod_{j\in S} I_j\right)}\right)}$$

Note that $(\alpha - \prod_{i\in S} I_i)$ is not equal to $(\alpha - \prod_{i\in S'} I_i)$, and $(\alpha - \prod_{i\in S'} I_i)$ cannot be canceled out of the equation.

In the second case, the the cloud requests the parameters for S of Bob from the aid server by **Adjust** algorithm, and computes as follows:

$$e\left(c_0 c_1^{\prod_{j\in S} I_j}, Tr\right) = \frac{e\left(g^{\tau\left(\alpha-I_i\prod_{j\in S} I_j\right)}, h^{1/\left(\alpha-\prod_{j\in S} I_j\right)}\right)}{e\left(g^{\tau\left(\alpha-I_i\prod_{j\in S} I_j\right)}, g^{H(w)/\left(\alpha-\prod_{j\in S} I_j\right)}\right)}$$

And this algorithm still cannot get a certain value that is equal to the c_w. The **Test** algorithm will not return anything for Bob regardless of whether w is valid. □

Theorem 5: Even in collusion with the cloud, a malicious user cannot perform keyword search and decryption for unauthorized files successfully.

Proof: To perform keyword search for an unauthorized file i successfully, the malicious user needs to submit a valid trapdoor of a keyword with his/her aggregate key. And the cloud needs to compute an element $c_1' = g^{\tau I_j} \in \mathcal{G}$ from $c_1 = g^{\tau I_i}$. Note that j is an arbitrary file in shared file set of the malicious user. Without the $I_i, I_j \in req_{index}$, the cloud cannot help malicious user to perform keyword search for an unauthorized file. Similarly, the cloud cannot generate the correct c_{pub_i} and help malicious user to decrypt unauthorized files either. □

VIII. EFFICIENCY ANALYSIS AND PERFORMANCE EVALUATION

In this section, We compare the computational overhead of EVKAKSE scheme with other similar schemes in detail and conduct experimental simulations.

A. EXPERIMENTAL ANALYSIS

Before the analysis and simulations, we define some symbols frequently used in this section which are shown in Table.2.

We provide the experimental simulation evaluation of the proposed scheme. The scheme is simulated on Ubuntu 18.04 LTS with 2.20GHz Intel i5-5200U and 4GB memory by using the latest GMP library and PBC library. We choose the Type-A pairing to complete a concrete algorithm just like KASE scheme [4] implemented.

Our scheme is compared with KASE scheme, VSEAK scheme [5] and Fc-MKA-KSE scheme [26] through main nine different algorithms in computation complexity. The results show as Table.3.

VSEAK considers the verification function based on KASE, which makes searchable encryption with aggregate method more comprehensive. Fc-MKA-KSE addresses the trapdoor attack from malicious users which has not been discussed before. According to Table.3, the data owner executes

TABLE 2. Notions.

Symbols	Descriptions
N	The maximum number of files
N_c	The maximum number of ciphertexts
M_1	The multiplication operation in Z_p
M_2	The multiplication operation in $\mathcal{G}$
M_3	The multiplication operation in $\mathcal{G}_1$
E_1	The exponentiation operation in Z_p
E_2	The exponentiation operation in $\mathcal{G}$
E_3	The exponentiation operation in $\mathcal{G}_1$
P	The pairing operation of $e : \mathcal{G} \times \mathcal{G} \to \mathcal{G}_1$

TABLE 3. Computation complexity with similar schemes.

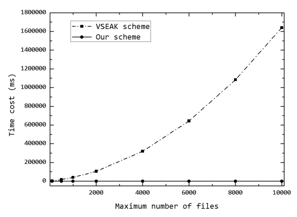
Algorithm	KASE	Fc-MKA-KSE	VSEAK	EVKAKSE
Setup	$2N(E_1+E_2)$	NE_2	$2N(E_1+E_2)$	1
KeyGen	E_2	$4N(E_1+E_2)$	E_2	E_2
Extract	$\|S\|(E_2+M_2)$	$\|S\|(E_2+M_2)$	$\|S\|(E_2+M_2)$	$\|S\|M_1+2E_2$
Encrypt	$2E_2+M_2+M_3+E_3+P$	$4E_2+M_2$	$2E_2+M_2+2M_3+E_3+P$	$2E_2+2M_1+2E_3+M_3$
Trapdoor	M_2	$3E_2+M_2$	M_2	E_2+M_2
Adjust	$\|S\|M_2$	$\|S\|M_2$	$\|S\|M_2$	$\|S\|M_1$
Test	M_3+2P	$\|S\|M_2+2M_3+E_3+4P$	M_3+2P	$\|S\|M_1+M_2+E_2+P$
Decrypt (Retrieve)	-	-	$\|S\|M_2+2P+2M_3$	P
Verify	-	-	$\|S\|(P+M_3)$	$\|\mathcal{F}t\|M_2+E_2$

KeyGen algorithm, and the execution time of **KeyGen** algorithm is linear with the maximum number of files. Therefore, when the maximum number of files is large, the computational overhead will be considerable, i.e., the data owner has to bear considerable computational overhead which is not what we expect.

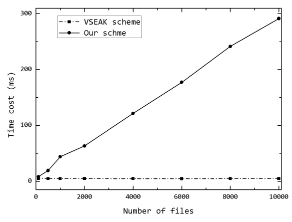
To add verification function and decryption function, as well as address the trapdoor attack without adding extra computational overhead, we use a new method to construct system and make the algorithm execution more efficient. In our scheme, the **Setup** algorithm and **Test** algorithm are completed in the cloud server, the **KeyGen** algorithm, **BuildIndex** algorithm, **TagGen** algorithm, **Encrypt** algorithm and **Extract** algorithm are completed by the data owner, the **Trapdoor** algorithm, **Decrypt** algorithm and **Verify** algorithm are completed by users. And the **Adjust** algorithm is completed by the cloud and the aid server. We compare our scheme with VSEAK scheme through a simulation experiment to reflect the efficiency advantage of our scheme. To simplify the representation, we combine the **KeyGen** algorithm with **BuildIndex** algorithm and use **KeyGen** algorithm to represent. Actually, when the number of files is the same, the **TagGen** algorithm has the same computational overhead as the **Verify** algorithm. So, we will use the time cost of the **Verify** algorithm to represent the time cost of the **TagGen** algorithm. As shown in Figure.3,

we can see that:

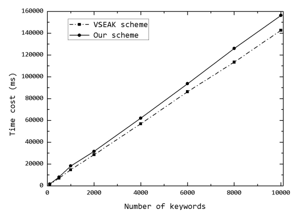
1) The time cost of the **Setup** algorithm in our scheme is a constant, i.e., $0.012s$, but in the VSEAK scheme the time cost is linear with the maximum number of files. When the number of files grows up to 10000, it is restricted that the **Setup** algorithm in VSEAK needs $1641.043s$.
2) The time cost of the **KeyGen** algorithm in our scheme is linear with the maximum number of files but it is reasonable. The **KeyGen** algorithm only needs $0.271s$ to complete when maximum number of files grows up to 10000.
3) In the **Encrypt** algorithm, the time cost of our scheme is similar to that of the VSEAK scheme. When the number of keywords grows up to 10000 (we also assume that the number of files is 10000), the **Encrypt** algorithm needs $156.221s$ to complete, which is acceptable.
4) The time cost of the **Extract** algorithm in our scheme is linear with the number of selected files. When the number of shared files grows up to 10000, the **Extract** algorithm only needs $0.013s$ to complete, and the **Extract** algorithm in VSEAK needs $48.336s$ to complete. The Figure.3(d) shows that the **Extract** algorithm of our scheme is more efficient.
5) The time cost of the **Trapdoor** algorithm in our scheme is a constant, independent of the number of files searched. When the number of files the user searches grows up to 10000, the **Trapdoor** algorithm only needs $0.05s$ to complete. Considering the security, the extra computational overhead is necessary compared with VSEAK. But the **Trapdoor** algorithm in our scheme is more efficient than that in Fc-MKA-KSE scheme.
6) The time cost of the **Adjust** algorithm in our scheme is linear with the number of shared file set S. When the number of shared files grows up to 10000, the **Adjust** algorithm only needs $0.004s$ to complete. We simplified the **Adjust** algorithm of VSEAK, but it still needs $0.321s$ to complete in the same case. Obviously, our algorithm is more efficient.
7) In the **Test** algorithm, the time cost of our scheme is similar to that of the VSEAK scheme. When the number of keyword ciphertexts grows up to 10000, the **Test** algorithm needs $89.751s$ to complete in cloud server. Notice that the cloud server performance can affect actual computational overhead, which means that the **Test** algorithm will perform better in the commercial public cloud.
8) The time cost of the **Decrypt** algorithm in our scheme is linear with the number of retrieved files set $\mathcal{F}t$. When the number of retrieved files grows up to 10000, the **Decrypt** algorithm needs $43.168s$ to complete. However, the **Retrieve** algorithm (similar to the **Decrypt** algorithm) in VSEAK needs $2760.413s$ to complete. In this part of the simulation, we assume that $S = \mathcal{F}t$. If $|S| < |\mathcal{F}t|$, the time cost of the **Decrypt** algorithm in VSEAK will be reduced, but still not efficient enough.
9) The time cost of the **Verify** algorithm in our scheme is linear with the number of retrieved files set $\mathcal{F}t$. When the number of retrieved files grows up to 10000, the **Verify** algorithm only needs $0.281s$ to complete which is extremely efficient.



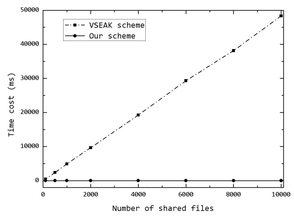
(a) Time cost of **Setup**



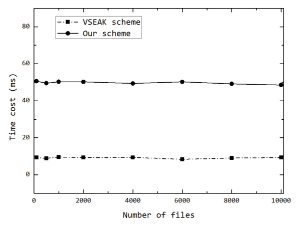
(b) Time cost of **KeyGen**



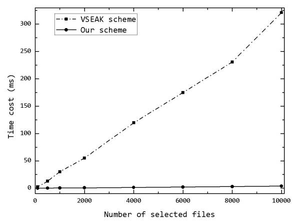
(c) Time cost of **Encrypt**



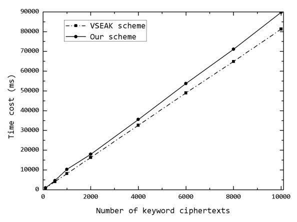
(d) Time cost of **Extract**



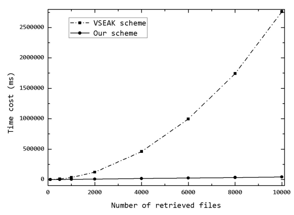
(e) Time cost of **Trapdoor**



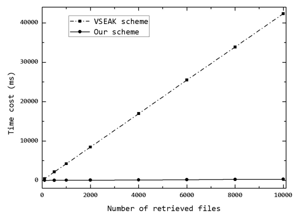
(f) Time cost of **Adjust**



(g) Time cost of **Test**



(h) Time cost of **Decrypt** (**Retrieve** in VSEAK)



(i) Time cost of **Verify** (**TagGen**)

FIGURE 3. Time cost comparison.

In order to eliminate the difference in the time cost of hash functions on files of different sizes, we stipulate that the time cost of hash functions in this simulation is negligible.

Compared with VSEAK, the simulation results show that EVKAKSE provides a new aggregate key searchable encryption method with decryption and verification. Moreover, this scheme reduces the computing overhead of cloud server and data owner, and also addresses trapdoor attack. Additionally, EVKAKSE has obvious advantages when the number of files is large. During the system setup phase, the cloud server does not require a lot of computation to generate a large number of parameters.

IX. CONCLUSION

In this paper, we design an efficient verifiable scheme for data sharing via the public cloud. And we address a decryption key redundancy problem that exists in most searchable encryption scheme with aggregate key. Theoretical analysis and experimental evaluation indicate that the proposed scheme gives better consideration to both security and efficiency. In this scheme, the cloud server does not need to set a maximum number of files during the setup phase. We also address the trapdoor attack without extra computational overhead. The evaluation of the performance demonstrates the construction's efficiency. However, the security of aid server determines the security of EVKAKSE in a way. Our future work is to adjust the algorithm to cancel the design of the aid server without compromising security.

REFERENCES

[1] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy (S P)*, Nov. 2002, pp. 44–55.

[2] C.-K. Chu, S. S. M. Chow, W.-G. Tzeng, J. Zhou, and R. H. Deng, "Key-aggregate cryptosystem for scalable data sharing in cloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 468–477, Feb. 2014.

[3] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Proc. CRYPTO*, 2005, pp. 258–275.

[4] B. Cui, Z. Liu, and L. Wang, "Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2374–2385, Aug. 2016.

[5] T. Li *et al.*, "Verifiable searchable encryption with aggregate keys for data sharing in outsourcing storage," in *Proc. Australas. Conf. Inf. Secur. Privacy*. Cham, Switzerland: Springer, 2016, pp. 153–169.

[6] E.-J. Goh, "Secure Indexes," IACR, New Delhi, India, Tech. Rep. 2003/216, 2003, pp. 1–19.

[7] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, 2006, pp. 79–88.

[8] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. Annu. Int. Cryptol. Conf.* New York, NY, USA: Springer-Verlag, 2006, pp. 535–552.

[9] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 965–976.

[10] X. Ge, J. Yu, H. Zhang, C. Hu, Z. Li, Z. Qin, and R. Hao, "Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification," *IEEE Trans. Depend. Sec. Comput.*, to be published.

[11] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[12] X. Jiang, X. Ge, J. Yu, F. Kong, X. Cheng, and R. Hao, "An efficient symmetric searchable encryption scheme for cloud storage," *J. Internet Serv. Inf. Secur.*, vol. 7, no. 2, pp. 1–18, 2017.

[13] X. Ge, J. Yu, C. Hu, H. Zhang, and R. Hao, "Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *IEEE Access*, vol. 6, pp. 45725–45739, 2018.

[14] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. EUROCRYPT*, 2004, pp. 506–522.

[15] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Proc. Int. Conf. Pairing-Based Cryptogr.* Berlin, Germany: Springer, 2007, pp. 2–22.

[16] J. Li *et al.*, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. IEEE INFOCOM*, 2010, pp. 1–5.

[17] R. Meng, Y. Zhou, J. Ning, K. Liang, J. Han, and W. Susilo, "An efficient key-policy attribute-based searchable encryption in prime-order groups," in *Proc. Int. Conf. Provable Secur.* Cham, Switzerland: Springer, 2017, pp. 39–56.

[18] K. Liang and W. Susilo, "Searchable attribute-based mechanism with efficient data sharing for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 9, pp. 1981–1992, Sep. 2015.

[19] J. K. Liu, M. H. Au, W. Susilo, K. Liang, R. Lu, and B. Srinivasan, "Secure sharing and searching for real-time video data in mobile cloud," *IEEE Netw.*, vol. 29, no. 2, pp. 46–50, Mar. 2015.

[20] Z. Liu, Z. Wang, X. Cheng, C. Jia, and K. Yuan, "Multi-user searchable encryption with coarser-grained access control in hybrid cloud," in *Proc. 4th Int. Conf. Emerging Intell. Data Web Technol.*, Sep. 2013, pp. 249–255.

[21] J. Li *et al.*, "Efficient keyword search over encrypted data with fine-grained access control in hybrid cloud," in *Proc. Int. Conf. Netw. Syst. Secur.* Berlin, Germany: Springer, 2012, pp. 490–502.

[22] F. Zhao, T. Nishide, and K. Sakurai, "Multi-user keyword search scheme for secure data sharing with fine-grained access control," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Berlin, Germany: Springer, 2011, pp. 406–418.

[23] X. Liu, G. Yang, Y. Mu, and R. Deng, "Multi-user verifiable searchable symmetric encryption for cloud storage," *IEEE Trans. Dependable Secure Comput.*, to be published.

[24] X. Liu, Y. Zhang, B. Wang, and J. Yan, "Mona: Secure multi-owner data sharing for dynamic groups in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1182–1191, Jun. 2013.

[25] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 917–922.

[26] 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 Trans. Ind. Inf.*, vol. 14, no. 8, pp. 3648–3658, Aug. 2018.

[27] D. He, M. Ma, S. Zeadally, N. Kumar, and K. Liang, "Certificateless public key authenticated encryption with keyword search for industrial Internet of Things," *IEEE Trans. Ind. Inf.*, vol. 14, no. 8, pp. 3618–3627, Aug. 2018.

[28] A. Fiat and M. Naor, "Broadcast encryption," in *Proc. Crypto*, in LNCS, vol. 773. New York, NY, USA: Springer-Verlag, 1993, pp. 480–491.

[29] A. Kiayias *et al.*, "Efficient encrypted keyword search for multi-user data sharing," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2016, pp. 173–195.

[30] J. Shu, X. Jia, K. Yang, and H. Wang, "Privacy-preserving task recommendation services for crowdsourcing," *IEEE Trans. Serv. Comput.*, to be published.

[31] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Proc. CRYPTO*, in LNCS, vol. 6841, P. Rogaway, Ed. Heidelberg, Germany: Springer, 2011, pp. 111–131.

[32] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 501–512.

[33] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 3025–3035, Nov. 2014.

[34] C. Gentry, "Practical identity-based encryption without random oracles," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, May 2006, pp. 445–464.

[35] Y. Yang, "Broadcast encryption based non-interactive key distribution in MANETs," *J. Comput. Syst. Sci.*, vol. 80, no. 3, pp. 533–545, May 2014.

[36] Y. Yang, M. Ma, and B. Lin, "Proxy re-encryption conjunctive keyword search against keyword guessing attack," in *Proc. Comput., Commun. IT Appl. Conf. (ComComAp)*, Hong Kong, Apr. 2013, pp. 125–130.

XUQI WANG received the B.E. degree in information security from Qingdao University, Qingdao, China, in 2017, where he is currently pursuing the master's degree with the College of Computer Science and Technology. His research interests include cloud computing security and broadcast encryption.

XIANGGUO CHENG received the B.S. degree in mathematics science from Jilin University, in 1992, the M.S. degree in applied mathematics science from Tongji University, in 1998, and the Ph.D. degree with the State Key Laboratory of Integrated Services Network, Xidian University, in 2006. He is currently a Professor with the College of Computer Science and Technology, Qingdao University. His research interests include computer security, public key cryptosystems, and their applications.

YU XIE received the B.E. degree from the College of Computer Science and Technology, Qingdao University, Qingdao, Shandong, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology, Tongji University, Shanghai. His current research interests include credit card fraud detection, machine learning, deep learning, big data, ensemble learning, and natural language processing.