AESM² Attribute-Based Encrypted Search for Multi-Owner and Multi-User Distributed Systems

Wang, Mingyue; Miao, Yinbin; Guo, Yu; Huang, Hejiao; Wang, Cong; Jia, Xiaohua

Published in:
IEEE Transactions on Parallel and Distributed Systems

Published: 01/01/2023

Document Version:
Post-print, also known as Accepted Author Manuscript, Peer-reviewed or Author Final version

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/TPDS.2022.3216320

Publication details:

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

Download date: 13/10/2023
https://doi.org/10.1109/TPDS.2022.3216320.
AESM\textsuperscript{2}: Attribute-based Encrypted Search for Multi-owner and Multi-user Distributed Systems

Mingyue Wang, Yinbin Miao, Yu Guo, Hejiao Huang, Cong Wang, Fellow, IEEE, and Xiaohua Jia, Fellow, IEEE

Abstract—With the rapid development of cloud computing, it is popular for data owners to outsource massive data to the cloud server for data sharing. To protect the privacy of sensitive data, many searchable encryption schemes are proposed. However, most of the existing studies focus on the single-owner model. In practice, users need to query data from distributed owners one by one, which inevitably brings great communication and computation overheads. Moreover, it lacks a secure scheme that realizes the access control requirements of individual owners. In this paper, we propose AESM\textsuperscript{2}, a new attribute-based encrypted search with ownership enhancement scheme for multi-owner and multi-user distributed systems. Our design enables users to search data from authorized owners with only one trapdoor. Owners can enforce owner level permission on users and encrypt their data individually with fine-grained attribute level permission. For practical consideration, we further devise an efficient revocation method of the owner level permission for users, where ciphertexts do not need to be updated. We formally define and prove the security of our design. Moreover, we implement a system prototype and analyze the performance from theoretical and experimental aspects. The evaluation results demonstrate that our scheme is effective and efficient.

Index Terms—Searchable encryption, attribute-based encryption, access control, multi-owner distributed systems

1 INTRODUCTION

Cloud computing has been widely used by organizations and individuals to have data storage and access services in the big data era. By outsourcing a large volume of data to the cloud server, such as Google Cloud, Aliyun, and Amazon Cloud, the storage burden of the user side can be greatly reduced. However, there are great risks of privacy leakage or unauthorized data access in data outsourcing. With the increasing concerns about data privacy, the encryption-before-outsourcing mechanism is widely adopted in cloud computing.

Data utilization is an important issue with encrypted data. Searchable Encryption (SE) technique [1] provides a promising solution, which allows users to search over the encrypted data without decryption directly. To enrich the functionality and security of SE, many studies have been proposed, such as conjunctive keyword search [2], order-revealing encryption [3], and range queries [4]. However, these schemes only support the single-owner and single-user model, where the user that queries data is also the data owner that outsources data to the server. A more common data-sharing model is that multiple users access data owned by multiple distributed owners (multi-owner/multi-user model). An application example is that many geographically distributed hospitals choose to outsource their data to the same cloud service provider (server). For better diagnosis and treatment, a user (e.g., a doctor) in one hospital needs to access the medical data from other hospitals for the same patient. The common practice is to search data from multiple distributed hospitals.

The multi-owner model can be easily achieved by letting users submit a query to each owner one by one. But it brings high costs in communication and computation for the users. The ultimate goal of the multi-owner and multi-user model is to match a user query with the ciphertexts encrypted by multiple distributed owners. Intuitively, there are two ways to do it. One approach is to let all owners share the same secret key for data encryption. This is not desirable for owners as they would worry about data abuse due to the share of the secret key. The other method is to allow owners to use their own secret keys but requires a trusted third party to convert all ciphertexts with an intermediate key [5]–[7]. This method is inefficient because it is time-consuming to convert ciphertexts, which incurs a high cost of re-encryption. There is also some literature that supports secure multi-owner data sharing [8]–[11]. In [9], it focuses on set computation of multiple owners. The scheme in [10] deals with fairness during the search process. However, none of them can meet the access control requirements for individual owners’ datasets. Thus, it is a challenging issue to develop a secure and practical data sharing system for the multi-owner and multi-user model.

Unlike the single-owner model, the privacy issue in the multi-owner model is much more complex, including data privacy and data ownership. The data sharing scheme
should protect data privacy and also follow access control requirements of owners. For instance, in the medical example mentioned before, different hospitals (owners) have their own access authorization mechanisms, including owner level permission and attribute level permission. Doctors (users) with attributes can access data records from multiple hospitals. For a user, owner level permission means whether it is allowed to search the data of certain owners. The attribute level permission is made by each owner to specify the availability of data record level based on attribute. In such a model, asymmetric cryptography is more suitable for generating ciphertexts with different public keys. For instance, [12] proposed an attribute-based keyword search scheme for the multi-owner and multi-user model. However, it requires the server to hold the user list for each owner. Moreover, owner level permission revocation is another difficult issue in the multi-owner model. The challenge lies in how to update the access capability of users in an efficient manner when their keys are revoked.

In this paper, we propose AESM², a privacy-preserving Attribute-based Encrypted Search scheme for Multi-owner and Multi-user distributed systems. In our design, we enhance owners’ ownership by utilizing a two-level access control mechanism, i.e., the owner level permission and attribute level permission. To enable users to search over multiple ciphertexts encrypted by different owners, we design a key aggregate search protocol by using the aggregate key technique in [13], [14]. With the authorized aggregate key, the encrypted keyword (trapdoor) from a user can be converted to match multiple ciphertexts, where the access permission of each owner will be checked. For the attribute level permission, keywords of data records are encrypted by the access policy tree structure [15] of different owners. Furthermore, we devise a version-based scheme for efficient key updating when the revocation happens in owner level permission set of the user. It can prevent revoked users from accessing data with old keys without involving any re-encryption. We summarize our contributions as follows:

- We propose a secure attribute-based encrypted search scheme with ownership enhancement for multi-owner and multi-user distributed systems, where users can search the data of authorized owners with one trapdoor.
- We design an efficient revocation method of the owner level permission that prevents revoked users from accessing data with the old keys.
- We define the security model and give a formal security analysis. We implement a system prototype and analyze the experimental evaluation using the real-world dataset, which shows the efficiency of our design.

The rest of this paper is organized as follows. Section 2 surveys related work. Section 3 presents preliminaries. Section 4 describes the problem formulation. We present the detailed construction of AESM² in Section 5. The security analysis is shown in Section 6. We evaluate the performance by theoretical and experimental analysis in Section 7. Finally, Section 8 concludes this paper.

## 2 RELATED WORK

This paper involves several techniques from applied cryptography. In this section, we review Searchable Encryption and Attribute-based Encryption technologies.

### 2.1 Searchable Encryption

Research on Searchable Encryption (SE) has experienced a big boom with the rapid development of cloud computing [16]. SE enables the untrusted server to search directly on the outsourced encrypted data without decryption. Song et al. [1] first proposed the concept of symmetric searchable encryption (SSE), which motivated a great quantity of further research works in this field [2], [17], [18]. In the following, Boneh et al. [17] presented the first public-key encryption with keyword search. In [18], Li et al. enriched the functionality of fuzzy encrypted keyword search. Cash et al. [2] devised the first SSE scheme for boolean keyword search with sublinear complexity. However, these designs only apply to the single-user setting where the data owner (i.e., user) outsources and searches data with its own secret key. To bridge the gap, multi-user SE [19]–[21] has also been extensively studied, where multiple users can perform search operations. For example, Sun et al. [20] devised a non-interactive authorization SE scheme for multiple clients.

However, these works do not provide fine-grained access control on data records for users. To achieve such a goal, Popa et al. [22] proposed the notion of multi-key searchable encryption, which allowed authorized users to search over multiple files encrypted by different keys with a single request. Unfortunately, Tang et al. [23] showed the security vulnerability when the server colluded with a malicious user and proposed an improved scheme. Following this direction, Cui et al. [13] designed the Key-Aggregate Searchable Encryption (KASE) scheme, which can support data sharing within a group. Later, Zhou et al. [14] pointed out that there existed keyword guessing attacks by unauthorized inside users in [13] and solved this problem. But these schemes only allow a single owner to outsource encrypted data, which does not apply to the multi-source data scenarios (e.g., task recommendation services of crowdsourcing [24]). In the multi-owner and multi-user data-sharing model, users need to access data from multiple data owners. To achieve this goal, Bao et al. [19] introduced the proxy re-encryption technique. Along this research direction, Zhang et al. [5] adopted a trusted administration server as the middleware to transform indexes and trapdoors, which brings expensive computation. According to application scenarios, in [8], owners can share their data with users in the same group. Huang et al. [25] proposed a conditional dissemination scheme for co-owner data group sharing. Very recently, Wang et al. [11] designed a hybrid searchable encryption technique where multiple writers (owners) serve one specific reader (user). Nevertheless, these studies mentioned above do not consider individual access control requirements of each owner’s dataset, which is the focus of this paper (AESM²). We do not consider the access pattern issue, which can be extended with ORAM [26] or volume hiding [27] techniques.

### 2.2 Attribute-based Encryption

Due to the property of fine-grained access control for multiple users, there have been extensive research findings on attribute-based encryption (ABE) [29]–[32]. ABE allows
users with appropriate secret keys to decrypt ciphertexts encrypted under access policies. According to how the ciphertext is encrypted, there are two complementary forms of ABE: Key-Policy ABE (KP-ABE) [33] and Ciphertext-Policy ABE (CP-ABE) [29], [34]. In KP-ABE, attributes are used to describe ciphertexts, and access policies over these attributes are associated with secret decryption keys. Instead, in CP-ABE, attributes are utilized to describe secret keys for users, and access policies over these policies are built into ciphertexts.

Goyal et al. [33] first proposed KP-ABE scheme based on the access tree. In the follow-up design, Bethencourt et al. [34] constructed the first CP-ABE, which motivated many works [35]–[38] in data access control applications. Later, a decentralized multi-authority CP-ABE scheme [29] was proposed to solve the access control issue under multiple authorities. To enrich the functionality and improve the security, revocable CP-ABE was presented in [39], where users’ attributes under multiple authorities can be effectively revoked. Next, Yang et al. [40] constructed constant-size attribute secret keys, which also achieved server-enabled user revocation by using the proxy re-encryption technique in ABE. Zhang et al. [36] proposed the attribute privacy protection scheme with fast decryption. To enable efficient intended data access for outsourced data, ABE is integrated into secure keyword search schemes to directly locate targeted results satisfying access control policies. Zheng et al. [15] devised the attribute-based keyword search scheme, where fine-grained access was enforced in keyword search process. However, it is designed for the single owner model, which is not scalable on owners. In [12], Sun et al. constructed the authorized keyword search with multiple owners and users. Nevertheless, their scheme requires the server to maintain the user lists for each dataset all the time. Besides, this scheme can be simplified as the single-owner setting. Recently, Miao et al. [28] designed the attribute-based keyword search scheme ABKS-SM for the shared multi-owner setting, where data were mutually maintained by multiple owners. Wang et al. [41] proposed the authorized attribute-based keyword search scheme for the multi-owner model. However, it did not solve dynamic authorization issues. Thus, there is a great need to design a privacy-preserving scheme with effective authorization revocation for multi-owner and multi-user distributed systems. Table 1 summarizes properties of the proposed AESM².

### Table 1: Encrypted Multi-owner and Multi-user (M²) Data Sharing Schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>M²</th>
<th>Fine-grained Access Control</th>
<th>Revocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mona [8]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>QYWY [25]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ABKS-SM [28]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Omnes [11]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>AESM²</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

3.1 Cryptographic Assumptions

**Bilinear Pairings.** $G$ and $G_T$ are both multiplicative cyclic groups of prime order $p$. Let $g$ and $g_T$ denote the generators of $G$ and $G_T$, respectively. A bilinear pairing $e: G \times G \rightarrow G_T$ has three properties: (1) Bilinearity. For all $u, v \in G$, and $a, b \in \mathbb{Z}_p$, it satisfies $e(a u, v^b) = e(u, v)^{ab}$. (2) Non-degeneracy. $e(g, g) \neq 1$. (3) Computability. It is efficient to compute $e(u, v)$ for any $u, v \in G$.

**Computational Diffie-Hellman (CDH) Assumption.** Let $G$ be a group of prime order $p$. Given the triple $(g, g^{z_1}, g^{z_2}) \in G$, the goal of adversary $A$ is to compute $g^{z_1 z_2}$. We can say that the CDH assumption holds if there exists no adversary that can solve the CDH problem with non-negligible advantage.

**Generic Bilinear Group Model.** In additive group $\mathbb{Z}_p^*$, we consider two random injective maps $\psi_0, \psi_1: \mathbb{Z}_p^* \rightarrow \{0, 1\}^n$, where $n > 3 \log(p)$. In the groups $G, G_T$ of order $p$, we are given the oracle to compute bilinear map $e: G \times G \rightarrow G_T$, where $G = \{\psi_0(x) | x \in \mathbb{Z}_p^*\}, G_T = \{\psi_1(x) | x \in \mathbb{Z}_p^*\}$. We define $G$ as a generic bilinear group.

3.2 Access Policy Tree

Access policy can be represented by the access trees [33], consisting of leaf nodes and inner (non-leaf) nodes. A leaf node is associated with an attribute and an inner node is a threshold gate. In access tree $T$, let $T_v$ denote the subtree of $T$ with root at node $v$ (e.g., $T_{root} = T$). $att(f)$ represents the attribute of leaf node $f$. The number of children in node $v$ is $num_v$, and index the children as $ind(v) \in \{1, \ldots, num_v\}$. The threshold value of node $v$ is $k_v$, where $1 \leq k_v \leq num_v$. When $k_v = 1$, it is the OR gate. The AND gate can be represented by $k_v = num_v$. Let $F(Atts, T_v)$ be 1 represent that the attribute set $Atts$ of $att(f)$ satisfies the access policies of subtree $T_v$.

Given the access tree $T$, we can hide the secret value $s$ according to $T$ by

$$q_f(0)_{f \in \mathcal{L}(T)} \leftarrow \text{Share}(S, T),$$

where $\mathcal{L}(T)$ denotes the set of leaf nodes. For each node $v$, it generates a polynomial $q_v$ of degree $k_v − 1$ (leaf node $k_v = 1$) in a top-down manner. If node $v$ is the root of $T$, $q_v(0) = s$. Otherwise, it sets $q_v(0) = q_{p(v)}(\text{ind}(v))$, where $p(v)$ denotes the parent node of $v$.

Given a value set $\{E_{u_1}, \ldots, E_{u_m}\}$ with the leaf node set $\{u_1, \ldots, u_m\}$ of access tree $T$. For $j \in \{1, \ldots, m\}$, $q_{u_j}(0), \ldots, q_{u_m}(0)$ are the secret shares of the secret value $s$ distributing according to $T$. $F(Atts, T) = 1$, $E_{u_j} = e(g, g)^{q_{u_j}(0)}$, the element secured in root node $E_{root} = e(g, g)^{q_{root}(0)} = e(g, g)^s$ can be reconstructed in a bottom-up manner as

$$e(g, g)^s = \text{Combine}(T, \{E_{u_1}, \ldots, E_{u_m}\}),$$

where $e$ is the bilinear pairing with generator $g$.

4 Problem Formulation

In this section, we formally describe the target problem of this paper. We present the system model and threat model. The notations used are listed in TABLE 2.
4.1 System Model

In the multi-owner and multi-user model, there are four entities involved, as depicted in Fig. 1. They are Trusted Authority (authority), Data Owners (owners), Cloud Server (server), and Data Requesters (users).

The authority is responsible for issuing keys to owners, the server, and users. It generates the aggregate owner level permission key for users corresponding to the access permissions from multiple owners. Meanwhile, it grants users attributes according to their identities or roles in the system and assigns the attribute keys.

Each owner has a collection of data records $R_i$ and the extracted keyword set $W_i$. It defines the access control requirements, including the owner level permission and fine-grained attribute level permission on $R_i$ for multiple users. To enable efficient queries on the encrypted records, the owner builds searchable indexes on the keyword set $W_i$. In this process, the owner’s predefined access policies are embedded in these indexes to enforce the record-level access control. Then, the owner uploads the encrypted data records and searchable indexes to the server.

The server provides storage and search services. It stores the encrypted records and indexes from owners. After receiving search requests from users, it performs the search protocol and returns the matched encrypted records as results to users.

Each user is associated with a set of attributes and assigned owner level access permission according to its identity and role in the system. When a user searches a keyword $\hat{w}$, it calculates the corresponding trapdoor (encrypted $\hat{w}$) and submits it to the server. It can get all matched results of multiple owners with only single query. Then, it will decrypt these records. The record decryption process is out of the scope of this paper. Readers can refer excellent works in [30], [42].

4.2 Threat Model

In our threat model, we make following threat assumptions regarding to each entity involved:

- **Authority** is considered as fully trusted, and all information is securely exchanged with the authority.
- **Server** is assumed as honest-but-curious as in the previous work [43]. It could perform the designated search operations honestly while it is eager to obtain sensitive information from the ciphertexts of data records and trapdoors. It also attempts to launch keyword guessing attacks using available information. We also assume that the server would not collude with other parties (especially malicious users).

- **Owners** are considered to be fully trusted in the sense that they will protect their keys privately and submit the valid ciphertexts to the server.
- **Users** are assumed as malicious. They could generate valid trapdoors to search for their interested keywords while intend to guess what other users can search (unauthorized insiders guess the private keys of other users) and obtain unauthorized data access.

1. We refer to those users who intend to access data from datasets that owner level permissions have been revoked.
5 Proposed AESM² Scheme

In this section, we first present the overview of our design. Then, we propose the detailed construction of AESM² scheme. After that, we describe revocation method to deal with dynamic changes of owner level permission in multi-owner and multi-user distributed systems.

5.1 Design Overview

In the outsourced multi-owner and multi-user data sharing setting, apart from data privacy, data access authorization is also a fundamental problem. There are two challenging issues yet to be solved. One is how to enable a user to search all encrypted indexes from multiple owners with only one trapdoor. Each owner has individual access control requirements, including the owner level permission and attribute level permission. That is, only when a user has the owner’s access permission can it search data according to the fine-grained attribute level permission. This is difficult because the indexes are encrypted by multiple owners under different access control mechanisms. Intuitively, it can be achieved by submitting search trapdoors to the server one by one. Each trapdoor can be used to search the data of a certain owner. However, this direct method is burdensome for users to generate trapdoor repeatedly (see performance analysis in section 7). Existing solutions [5], [6], [28] did not consider the access requirements of individual owners. The other issue is to enable effective and reliable revocation for users. The challenge lies in how to prevent the user from accessing the data where its owner level permission from some owners is revoked. As owner level permission for the user are flexible, it is hard for the server to judge whether the trapdoor contains the revoked owner level permissions (i.e., trapdoor is generated with the key used before revocation).

To solve the first issue, we start with file-based sharing schemes [13], [22], where the trapdoor can be converted to search multiple files. However, these schemes cannot satisfy access control requirements as desired in our multi-owner setting. To this end, we design a matching protocol based on two-level access permissions. In the owner level permission, we adopt the aggregate key [13], [44] technique to enable the single trapdoor to adapt to multiple indexes. Different from the single-owner setting [13], [14], in our design, these index ciphertexts are encrypted by different keys of multiple owners. Besides, each owner generates its own random nonce to build encrypted indexes. A trusted authority is introduced to aggregate all authorized owner’s access permissions into a single key for the user. During searching, the server uses the aggregate key to identify which datasets of these owners can be accessed by this user. In the attribute level permission, the keyword-record sets are encrypted under the fine-grained access policies by each owner. The queried keyword will be matched if the attributes of the user satisfy the access policy, where the secret value secured by the policy tree can be recovered. The server needs to perform the match operations with the trapdoor and the authorized datasets in the owner level permission to return all matched results for the user.

To deal with the second issue, we assign a version number for each period of revocation in the owner level permission. When the revocation happens, only secret keys need to be updated for users. There are no re-encryption operations on ciphertexts, which incurs no cost for owners in the revocation phase. When the owner level permission of a user (revoked user) is revoked by corresponding owners (revoking owners), the authority generates a new version number and updates all of the users’ secret keys. With the updated key, the revoked users cannot search data from the revoking owners anymore. Non-revoked users can continue having the service with updated secret keys. To improve the system efficiency, we delegate the server to update the system version number for related parameters, such that non-revoked users can also search all the data with unchanged access privileges. Moreover, all users only need to keep the latest secret key instead of all previous keys.

5.2 Construction of AESM²

The construction of proposed AESM² scheme consists of five phases: System Initialization, Key Generation, Index Encryption, Trapdoor Generation and Multi-owner Multi-user Encrypted Search.

5.2.1 System Initialization

The authority sets up the system by running the Setup algorithm with the security parameter as input. It first selects two multiplicative groups $G, G_T$ of the same order $p$ and the bilinear pairing $e : G \times G \rightarrow G_T$. To model the security in the random oracles, the authority also needs to choose two hash functions $H_1 : \{0, 1\}^* \rightarrow G, H_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_p^*$. $H_1$ maps the strings to elements of group $G$, and $H_2$ is a one-way hash function. Then, it chooses several random elements $a, b, c, \beta_2 \in \mathbb{Z}_p^*$ as the system master key $msk = (a, b, c, \beta_2)$. The authority also chooses random elements $\alpha \in \mathbb{Z}_p^*$ as the common secret parameters for all owners. It accordingly computes $v = g^{\beta_2}, g_t = g^{\alpha^t}(t \in \{1, \ldots, 2n\})$, where $n$ is the maximum number of owners in the system. The system public parameter $pp$ is generated as

$$pp = (g, v, g^a, g^b, g^c, \{g_t\}, H_1, H_2).$$

5.2.2 Key Generation

The key generation algorithm KeyGen contains the server’s key generation, owner’s key generation, and user’s key generation.

Server’s Key Generation: To avoid users’ keyword guessing attack [45] by inside attackers [14], it is necessary to assign the server with keys in our scheme AESM². By running KeyGen_{cs} algorithm, the authority first generates a random element $\beta_1 \in \mathbb{Z}_p^*$ as the secret key $sk_{cs} = \beta_1$ for the server. It then computes the corresponding public key as

$$pk_{cs} = u = g^{\beta_1}.$$

Owner’s Key Generation: For each owner $o_i$, the authority chooses two random elements $\gamma_i, 1, \gamma_i, 2 \in \mathbb{Z}_p^*$ and generates the secret key through the KeyGen_{owner} algorithm as

$$sk_{o_i} = (\gamma_i, 1, \gamma_i, 2, ek_{i, 1}, ek_{i, 2}),$$

where $ek_{i, 1} = u^{\gamma_i, 1} = g^{\beta_1 \gamma_i, 1}, ek_{i, 2} = v^{\gamma_i, 1} = g^{\beta_2 \gamma_i, 1}$ are the private encryption keys of $o_i$. The authority then computes...
\[ h_{t,1} = g_{\gamma_{1}^{i}}^{\gamma_{1}^{i}}(t \in \{1, \ldots , n\}) , h_{t,2} = g_{\gamma_{2}^{i}}^{\gamma_{2}^{i}}(t \in \{1, \ldots , 2n\}) \]
and generates the public keys for owner \( o_i \) as
\[ pk_{o_i} = (\{ h_{t,1} \}_{t \in \{1, \ldots , n\}} , \{ h_{t,2} \}_{t \in \{1, \ldots , 2n\}}) . \]

User’s Key Generation: According to the access control requirements of multiple owners (as introduced in subsection 5.1), the user’s authorization key consists of two parts, owner level permission key and attribute level permission key. It is generated by running the KeyGen\textsubscript{user} algorithm for a user.

After choosing a permission version number as VN = \( ver \), the authority first generates the aggregate key \( sk_{u,ap} \) of owner level permissions from each owner for an authorized user \( u \) as
\[
  sk_{u,ap} = \prod_{k \in S} g_{n+1-k}^{\beta_{ver}} ,
\]
where \( S \in \{1, \ldots , n\} \) represents the owner set authorized to \( u \). If a user gets no access permissions from owners (i.e., \( S = \emptyset \)), it will not obtain this secret key. It is worth noticing that the version number is designed to support owner level Atts that entitles a set of attributes \( u \) to \( S \).

The aggregate key \( sk_{u,ap} \) generates random elements \( r \in \mathbb{Z}_p^{*} \) and generates the public keys for owner \( o_i \) or level access permission of \( r \). To encrypt the owner level permission, it chooses a random \( sk \) for a user. \( sk \) is generated by running the \text{IndEnc} algorithm.

\[
  Ap_{u,2} = sk_{u,ap}^x , Ap_{u,2} = pk_{ca}^x .
\]

The introduction of \( Ap_{u,2} \) is to resist the keyword guessing attack as mentioned earlier. For the access control in attribute level, user \( u \) chooses another random element \( s \in \mathbb{Z}_p^{*} \). To match the encrypted keyword in the index, \( u \) computes the encrypted token of query keyword \( \hat{w} \) as
\[
  Tk_{u,\hat{w}} = (tok_1 = (g^{c} g^{bH_{2}(\hat{w})})^s , tok_2 = g^{sc} ,
  tok_3 = A^s = g^{acx-rs}/b) .
\]

To specify the authorization of its attribute, \( u \) also needs to generate the trapdoor for attributes accordingly with the attribute secret key \( sk_{u,at} \) (shown in Eq. 2) as
\[
  Tk_{u,at} = (\forall at_j \in Atts : \{ A'_j = A_j^s , B'_j = B_j^s \}) .
\]

User \( u \) then uploads the secure trapdoor \( Tp_{u,\hat{w}} = (Atts, S, (Ap_{u,1}, Ap_{u,2}), Tk_{u,\hat{w}}, Tk_{u,at}) \) to the server.

5.2.5 Multi-owner Multi-user Encrypted Search

Based on the encrypted indexes from owners and the secure trapdoors from users, the server performs the two-level matches to find results by running the \text{Search} algorithm, where both owner level and attribute level permissions are satisfied as required by all owners.

1. Owner Level Permission Verification

After receiving the trapdoor from user \( u \), the server first converts the owner level permission trapdoor \( Ap_{u,1} \) (in Eq. 6) into \( Ap_i \) for each owner \( o_i \) as
\[
  Ap_i = Ap_{u,1} \cdot \prod_{k \in S , k \neq i} h_{n+1-k+i,2}^{ver} ,
\]
where \( S \) represents owners that give access permissions to \( u \) and \( ver \) is the version number VN from the authority. Given the encrypted index of owner level permission \( C_{i,ap} \) (as shown in Eq. 3) of \( o_i \) and the trapdoor \( Ap_{u,2} \) (in Eq. 6) of \( u \), the server then checks whether the dataset of current \( o_i \) is accessible to \( u \) as
\[
  e(\text{pub}, c_{i,3})^{\beta_1} . e(c_{i,2}, Ap_{u,2}) = e(h^{ver}_{n+1-k+i,1}) ,
\]
where \( pub = \prod_{k \in S} h_{n+1-k,1}^{ver} \) is the public key of owner \( o_i \) and \( h_{n+1,2}^{ver} \) is the public key of owner \( o_i \). Then, the server will continue the following fine-grained attribute level match. Otherwise, the search result of owner
Proof. Only the user who has the owner level permission aggregate key $s_{u,ap}$ can pass the verification process. We present the correctness of Eq. 10 as follows.

$$e(pub, c_{i,2})^\beta_i \cdot e(c_{i,2}, Ap_{u,2}) = e\left(\prod_{k \in S} g_{n+1-k}^{\gamma_{i,1,v}} \cdot g_{2x}^{\gamma_{i,1,r}} \cdot g_{\delta_1}^{\gamma_{i,1,2}}\right)$$

$$= e\left(\prod_{k \in S} g_{n+1-k}^{\gamma_{i,1,v}} \cdot g_{2x}^{\gamma_{i,1,r}} \cdot g_{\delta_1}^{\gamma_{i,1,2}}\right)$$

$$= e\left(\prod_{k \in S} g_{n+1-k}^{\gamma_{i,1,v}} \cdot g_{2x}^{\gamma_{i,1,r}} \cdot g_{\delta_1}^{\gamma_{i,1,2}}\right)$$

$$= e\left(\prod_{k \in S} g_{n+1-k}^{\gamma_{i,1,v}} \cdot g_{2x}^{\gamma_{i,1,r}} \cdot g_{\delta_1}^{\gamma_{i,1,2}}\right)$$

$$= e\left(\prod_{k \in S} g_{n+1-k}^{\gamma_{i,1,v}} \cdot g_{2x}^{\gamma_{i,1,r}} \cdot g_{\delta_1}^{\gamma_{i,1,2}}\right)$$

Here is an example to explain how owner level permissions are verified. As shown in Fig. 2, there are 3 owners in this system, and the authorized user $u$ has access permissions of owner set $S = \{1, 2\}$. In this process, the server will perform computations accordingly upon receiving the trapdoor $(Ap_{u,1}, Ap_{u,2})$ from $u$. It needs to transform this trapdoor for access permission verification of each owner. For example, the server first generates parameters $pub$ and $Ap_1$ for owner $o_1$. Then, it verifies the owner level permission of $o_1$ by checking Eq. 10. Here, it can be observed that owner $o_1$ can pass the verification for $u$. It is the same for owner $o_2$ since they are both accessible to $u$. However, the verification process will fail for owner $o_3$ due to the value of $Ap_3$. This is because the set $S$ and corresponding authorized aggregate key only contain $o_1$ and $o_2$.

Discussions: In AESM\(^2\), the version number VN is given to the server. Another possible design is to embed VN into the public keys of owners, where the server will not know the version number. However, in this way, it is extremely time-consuming to update the public keys for each owner when the revocation happens in the owner level permission.

2) Attribute Level Match

When the adjusted trapdoor $Ap_i$ of owner $o_i$ can pass the owner level access permission verification (Eq. 10), the server will perform the attribute level match for the same trapdoor $T_{pu, w}$ of the user $u$. Since each owner has its fine-grained policies on the keywords, the search results depend on the attributes that the user possesses. For the $Atts$ in the trapdoor $T_{pu, w}$, the server first checks if there exists an attribute set $Atts_X$ that satisfies the access policy tree $T_i$ in encrypted index $I_{i, w}$ of owner $o_i$. For $at_i \in Atts_X$, according to Eq. 5 and Eq. 8, the server will compute

$$E_f = e(A'_j, W_j)/e(B'_j, D_j) = e(g, g)^{\text{attr}_j(0)}$$

where $\text{attr}_j(f) = at_j$ for $f \in \mathcal{L}(T_i)$ (leaf node). The server then combines all the $E_f$ based on $T_i$ to recover the secret value $E_{root}$ in a bottom-up fashion as $E_{root} = e(g, g)^{\text{attr}_0(0)} = e(g, g)^{\text{attr}_{r,2}}$.

With the encrypted index $I_{i, w}$ (in Eq. 4) and trapdoor $T_{pu, w}$ (in Eq. 7), the secret value $E_{root}$ can be used to check whether the query keyword $w$ is matched as

$$e(W_{i,3, tok_2}) = e(W_{i,1, tok_1})E_{root}e(tok_3, W_{i,2}).$$

If the query keyword $\check{w}$ exists in the encrypted index (i.e., $\check{w} = w$), the Eq. 12 holds and it returns 1, which indicates that the server will return the corresponding records of queried keyword of owner $o_i$. Otherwise, it returns 0. The server needs to perform the attribute level match for each owner in the authorized set $S$ of $u$ in owner level permission, and then returns all the match records of owners.

5.3 Owner Level Permission Revocation in AESM\(^2\)

In multi-owner and multi-user distributed systems, the user’s access privileges may dynamically change. There are two requirements when the owner level permission revocation happens. First, the revoked users can only search with the new owner level permission set $S'$. It cannot search data from revoking owners even if it submits the trapdoor generated with the previous secret key to the server. Second, non-revoked users with the unchanged owner level...
permission set $S$ can also search as before without affection. For example, a user $\hat{u}$ has access permissions from owner $\{A, B, C\}$. When the owner $B$ revokes the permission for the user $\hat{u}$, $\hat{u}$ cannot get any data from owner $B$ any longer. Other users with the access permission from owner $B$ can continue accessing dataset of $B$ as before.

Suppose an access permission $p_\ell$ is revoked for the user $\hat{u}$ by owner $o_\ell$. We use the term of Revoked Permission to denote permission $p_\ell$. The user $\hat{u}$ is denoted as the Revoked User. The revoked user’s permission set from owners is $S^*$ after revocation. We denote the users without owner level permission revocation as Non-revoked Users. The owner level permission revocation method includes three phases: Update Key Generation, Secret Key Update of Users, and System Version Number Update by Server.

5.3.1 Update Key Generation

When the permission $p_\ell$ of owner $o_\ell$ is revoked from a user $\hat{u}$, the authority will generate a new version number $VN^* = ver^* \oplus (ver^* \neq ver)$. The authority generates a unique update key $UK^r$, by running the ApUKKey algorithm for non-revoked users and the server to update the version number of the system as

$$UK^r = \frac{ver^*}{ver^*}.$$  \hspace{1cm} (13)

5.3.2 Secret Key Update of Users

The authority needs to run SKUpd algorithm to generate the latest version of secret keys for both revoked users and non-revoked users. It generates the new secret key for the revoked user (with permission set $S^*$) as

$$sk_{u, ap}^* = \prod_{k \in S^*} g_{n+1-k}^{2ver^*}.$$ The authority transmits the new secret key $sk_{u, ap}^*$ and $S^*$ to the user $\hat{u}$ securely.

To avoid non-revoked users leaking the update key to the revoked user, the secret key update of non-revoked users is run by the authority (users do not get the update key). Based on the previous secret key ($sk_{u, ap}$ shown in Eq. 1), namely, there is no need for the authority to generate new keys for non-revoked users from scratch. It can generate the updated secret key for non-revoked users as

$$sk_{u, ap}^* = sk_{u, ap}^R.$$ The owner level permission sets are kept unchanged for non-revoked users. The authority will give each $sk_{u, ap}^*$ to these non-revoked users through a private channel. All users will use the updated secret key to generate search trapdoors. The revocation function can be processed in batches according to time division, for example, every month. The authority will only be involved periodically after system initialization.

5.3.3 System Version Number Update by Server

To ensure that the revoked users cannot search with the previous secret key (before revocation), all the parameters (adapted trapdoors and search parameters) related to the owner level permission set are required to be updated to the latest version. The server conducts the update operation by using the proxy re-encryption method without decryption before updating. Because the index of each owner and the owner level permission set are decoupled in our design, we do not need to update the index ciphertexts generated by the owners, which greatly improves the update efficiency.

The server updates the system version number by running the algorithm VNUpd during search after owner level permission revocation. Upon receiving the update key $UK^r$ (in Eq. 13) from the authority, the server takes as inputs the trapdoors from the user associated with the owner level permission set and the update key $UK^r$ to update the converted trapdoor $Ap_i$ (in Eq. 9) for owner $o_i$ as

$$Ap_i^* = Ap_{i, 1} \cdot \prod_{k \in S, k \neq i} h_{n+1-k+i, 2}^{ver^*}.$$ The system parameters are also updated to the latest version during owner level permission verification (in Eq. 10) with the update key $UK^r$ as

$$e\left(\frac{pub^*, c_{i, 1}}{c_{i, 2}, Ap_{i, 2}}\right) \approx e\left(h_{n+1, 2}^{ver^*}, c_{i, 1}\right),$$ where $pub^* = pub_{UK^r}$, $h_{n+1, 2}^{ver^*} = (h_{ver^*})_{UK^r}$.

Discussions: In the two-level access control mechanism, the user’s revocation issues can also be divided into two levels. Here, we only focus on the flexible access revocation for users in the owner level permission. For the attribute level permission revocation, there are two kinds of update methods, i.e., access policy update and attribute revocation. Some excellent works [32], [39] have been proposed to cope with this problem. They can be easily integrated into the attribute level of AESM².

6 Security Analysis

In this section, we first define the security model. Then, we formally prove the security of our protocol, which is selectively secure against CKA and achieves access control based search.

6.1 Security Model

Our protocol should achieve the selectively secure against CKA (Chosen-Keyword Attack) for any PPT (Probabilistic Polynomial-Time) adversary. We formalize this security model by using the selective CKA game between an adversary $A$ and a challenger $C$.

- Setup. $A$ first chooses a non-trivial challenge $\mathcal{T}^*$ that can be satisfied by any user without valid attributes, then gives it to $C$ that will run Setup to output public parameters $pp$ and master key $msk$.
- Phase 1. $A$ issues the following oracles for polynomial times, and $C$ initializes an empty keyword list $\mathcal{L}_w$.
  - $O_{KeyGen}$. If the user’s attribute set $Atts$ satisfies the specified access tree $\mathcal{T}^*$, $C$ aborts. Otherwise, $C$ calls KeyGen and returns the secret key $sk_{u, att}$ for corresponding $Atts$.
  - $O_{TdGen}$. $C$ first calls TdGen to generate the trapdoor for $A$, then adds the keyword $w$ in $\mathcal{L}_w$ if the attribute set $Atts$ satisfies the access tree $\mathcal{T}^*$.
- Challenge. $A$ selects two challenging keywords $w_0, w_1 \notin \mathcal{L}_w$. $C$ first chooses a random bit $b \in \{0, 1\}$, then calls IndEnc to generate the index $I_{w_b}$ and sends it to $A$. It is
worth noticing that the requirement \(w_0, w_1 \notin L_w\) is to prevent \(A\) from trivially guessing the bit \(b\) based on the oracle \(O_{T'dGen}\).

- **Phase 2.** \(A\) repeats the process of **Phase 1**, but one limitation is that \(w_0, w_1\) cannot be queried to the oracle \(O_{T'dGen}\) if \(Atts\) satisfies \(T'\).
- **Guess.** \(A\) outputs a guess \(b'\) and wins the above game if \(b = b'\).

**Definition 1.** Let \(|Pr[b' = b] - \frac{1}{2}|\) be \(A\)'s probability of winning above security game, and our protocol is selectively secure against CKA if \(|Pr[b' = b] - \frac{1}{2}|\) is negligible in security parameter \(\lambda\).

In the public-key setting, it is difficult to prevent \(A\) from launching keyword guessing attack. \(A\) may first encrypt a keyword of its choice, and then check whether the resulting ciphertext and target trapdoor are derived from the same keyword. Thus, we form this security property as *keyword secrecy*, which requires that \(A\)'s probability in telling keyword ciphertext of its choice from target trapdoor is negligible. The keyword secrecy game simulated between adversary \(A\) and challenger \(C\) is demonstrated as follows.

- **Setup.** \(C\) calls **Setup** to generate the public parameters \(pp\) and master key \(msk\), then it sends \(pp\) to \(A\) and keeps \(msk\) itself.
- **Phase 1.** \(A\) makes polynomial time queries to the following oracles:
  - \(O_{KeyGen}\). \(C\) submits an attribute set \(Atts\) to \(C\). \(C\) calls **KeyGen** to return the corresponding secret key \(sk_{u,att}\) to \(A\). In addition, \(C\) adds \(Atts\) into the list \(L_{att}\) which is initially empty.
  - \(O_{T'dGen}\). \(A\) submits a keyword \(w\) to \(C\). \(C\) calls **TdGen** to generate the trapdoor \(T_{pu,w}\) for \(A\).
- **Challenge.** \(A\) submits a non-trivial access policy \(T'^*\) to \(C\). \(C\) first selects a keyword \(w^*\) and an attribute set \(Atts^*\) that satisfies \(T'^*\), then calls **IndEnc** and **TdGen** to output \(I_{i,w^*}, T_{pu,w^*}\), respectively. While the requirement is that \(Atts\) does not satisfy \(T^*\) for any \(Atts \in L_{att}\).
- **Phase 2.** This phase is similar to **Phase 1**.
- **Guess.** After guessing \(g\) different keywords, \(A\) outputs a keyword \(w'\). \(A\) wins this security game if \(w' = w\). Otherwise, it fails.

**Definition 2.** Our protocol can achieve the keyword secrecy if \(A\)'s advantage in winning above security game is at most \(\frac{1}{|W|} + \epsilon\), where \(|W|\) is the keyword space, and \(\epsilon\) is a negligible advantage.

### 6.2 Security Proof

**Theorem 1.** Our protocol is selectively secure against CKA in the generic bilinear group model [34], where \(H_1\) is modeled as a random oracle and \(H_2\) is a one-way hash function.

**Proof:** In the selective CKA game, the goal of \(A\) is to distinguish \(g^{a(r_1+\ldots+r_2)}b^{r_1}H_2(w_0)\) from \(g^{a(r_1+\ldots+r_2)}b^{r_1}H_2(w_1)\). Given a random element \(\theta \in \mathbb{Z}_p^\ast\), the probability of distinguishing \(g^\theta\) from \(g^{a(r_1+\ldots+r_2)}b^{r_1}H_2(w_1)\) is equal to that of distinguishing \(g^\theta\) from \(g^{a(r_1+\ldots+r_2)}b^{r_1}H_2(w_1)\). Thus, \(A\) has an advantage \(\epsilon/2\) in distinguishing \(g^\theta\) from \(g^{a(r_1+\ldots+r_2)}b^{r_1}H_2(w_0)\) if it can break the selective CKA game with an advantage \(\epsilon\). To prove above theorem, we consider a CKA game where \(A\) can distinguish \(g^{a(r_1+\ldots+r_2)}\) from \(g^\theta\), shown as follows.

- **Setup.** \(C\) selects three random elements \(a, b, c \in \mathbb{Z}_p^\ast\) and returns partial public parameters \((e, g, p, g^a, g^b, g^c)\) to \(A\). Then, \(A\) chooses an access tree \(T^*\) and sends it to \(C\). \(H_i(at_j)\) is simulated as follows. If the attribute \(at_j\) has been queried before, \(C\) chooses \(\pi_j \in \mathbb{Z}_p^\ast\) and adds \((at_j, \pi_j)\) to \(O_{H_i}\) before returning \(g^{\pi_j}\). Otherwise, \(C\) returns \(g^{\pi_j}\) by using \(\pi_j\) in \(O_{H_i}\).
- **Phase 1.** \(A\) issues \(O_{KeyGen}, O_{T'dGen}\) as follows.
  - \(O_{KeyGen}\). \(C\) chooses \(r \in \mathbb{Z}_p^\ast\) and computes \(A = g^{(ac-r)/b}\). For each attribute \(at_j \in Atts\), \(C\) first selects \(r_j \in \mathbb{Z}_p^\ast\) and computes \(A_j = g^{\pi_j r_j}, B_j = g^{\pi_j^2}\), then returns \((Atts, \{A, \{A_j, B_j\}\})\).
  - \(O_{T'dGen}\). \(C\) first issues the oracle \(O_{KeyGen}\) to get the secret key for attribute set \(Atts\), then selects \(s \in \mathbb{Z}_p^\ast\) and computes \(tok_1 = (g^{a}b^{H_2(w)})^s, tok_2 = g^s, tok_3 = A^s, A^j = A^{j}_{i^2}, B^j = B_{j^2}\) for keyword \(w\). If \(Atts\) satisfies \(T^*\), \(C\) stores \(w\) in \(L_w\).
- **Challenge.** Given two challenging keywords \(w_0, w_1 \notin L_w\) with the same length, \(C\) first chooses \(r_1, r_2 \in \mathbb{Z}_p^\ast\) and computes the secret shares of \(r_1, r_2\) for leaf nodes in the specified access tree \(T^*\). Then, \(C\) selects a random bit \(b \in \{0, 1\}\) and computes \(W_{i,1} = g^{cr_1}, W_{i,2} = g^0, W_{i,2} = g^{br_2}, W_f = g^{\psi_1(0)}, D_f = g^{\psi_1(0)}\) if \(b = 0\). Otherwise, \(C\) returns \(W_{i,1} = g^{cr_1}, W_{i,2} = g^{ar_1+\ldots+r_2}, W_{i,2} = g^{br_2}, W_f = g^{\psi_1(0)}, D_f = g^{\psi_1(0)}\).
- **Phase 2.** This phase has the same process as Phase 1 in the selective CKA game.
- **Guess.** Assume that \(A\) can construct \(e(g, g)^{ba(r_1+\ldots+r_2)}\) for certain term \(g^\theta\) that can be constructed according to above oracle outputs, then it can utilize it to distinguish \(g^\theta\) from \(g^{a(r_1+\ldots+r_2)}\). Thus, we still need to show that \(A\) can construct \(e(g, g)^{ba(r_1+\ldots+r_2)}\) for certain term \(g^\delta\) with a negligible advantage. That is, \(A\) cannot break the selective CKA game with non-negligible advantage.

In the generic group model \((G, G_T)\), where \(G = \{\psi_0(x) | x \in \mathbb{Z}_p^\ast\}, G_T = \{\psi_1(x) | x \in \mathbb{Z}_p^\ast\}\), \(A\)'s advantage in guessing an element in the images of \(\psi_0\) and \(\psi_1\) is negligible. Next, we consider \(A\)'s probability in constructing \(e(g, g)^{ba(r_1+\ldots+r_2)}\) from oracle outputs it has queried, where \(\delta \in \mathbb{Z}_p^\ast\). Finally, let us consider how to construct \(e(g, g)^{ba(r_1+\ldots+r_2)}\) for certain term \(\delta\). As the random element \(r_{1,1}\) only appears in the term \(cr_{1,1}\), the term \(\delta\) should contain \(c\) for constructing \(e(g, g)^{ba(r_1+\ldots+r_2)}\). Assume that \(\delta = \delta^\prime c\). \(A\) needs to construct \(\delta^\prime ac_{r_1,2}\) by using terms \(br_{1,2}\) and \((ac-r)/b\). Due to \(br_{2,2}(ac-r)/b = ac_{r_1,2} - rr_{1,2}\), \(A\) should cancel \(rr_{1,2}\) according to terms \(\pi_{j}, (r + \pi_{j}r), q_f(0), q_f(0)\pi_{j}\), where \(q_f(0)\) is the secret share of \(r_1,2\) according to the access tree \(T^*\). However, it is difficult to construct \(rr_{1,2}\) based on above terms, as \(rr_{1,2}\) can only be constructed on condition that the attributes corresponding to \(r_1,2\) satisfies \(T^*\). Thus, we derive that \(A\) cannot gain a non-negligible advantage in above modified selective CKA game. That is, \(A\) just gains a negligible advantage in breaking the above selective CKA game.

\[\square\]

**Theorem 2.** Our protocol guarantees the keyword secrecy in the random oracle model, where \(H_2\) is a one-way hash function.

**Proof:** We construct a challenger \(C\) that simulates the
keyword secrecy game as follows.

- \( C \) selects random elements \( a, b, c \in \mathbb{Z}_p \) and calls \( \text{Setup} \) to output the public parameters \( pp = (g, g^a, g^b, g^c, H_1, H_2) \) and master key \( msk \), where \( H_2 \) is a one-way hash function. Then, \( C \) simulates the random oracle \( O_{H_1} \) as the same way as Theorem 1. Specifically, if the attribute \( a \) has been queried before, \( C \) chooses \( \pi_j \in \mathbb{Z}_p \) and adds \((a_j, \pi_j) \) to \( O_{H_1} \), before returning \( g^{\pi_j} \). Otherwise, \( C \) returns \( g^{\pi_j} \) by using \( \pi_j \) in \( O_{H_1} \). Finally, \( C \) sends \( pp \) to \( A \).
- **Phase 1.** \( A \) adaptively makes queries to the following oracles for polynomially-many times.

  - \( O_{KeyGen}. \) \( A \) submits an attribute set \( Atts \) to \( C. \) \( C \) calls \( \text{KeyGen} \) to generate the secret key \( sk_{u,att} \) to \( A \). Then, \( C \) adds \( Atts \) into the list \( L_{att} \) which is initially empty.
  
  - \( O_{TdGen}. \) \( A \) submits the keyword \( w \) to \( C. \) \( C \) calls \( \text{TdGen} \) to generate the trapdoor \( T_{p,u,w} \) by using the secret key \( sk_{u,att} \). Then, \( C \) sends \( T_{p,u,w} \) to \( A \).

- **Challenge.** \( A \) submits an access policy tree \( T_{p}^* \) to \( C. \) \( C \) first chooses an attribute set \( Atts^* \) that satisfies \( T_{p}^* \) and calls \( \text{KeyGen} \) to output the secret key \( sk_{u,att}^* \). Then, \( C \) randomly chooses a keyword \( w^* \) from the keyword space \( |W| \), and outputs the index \( i_{1,w^*} \) and trapdoor \( T_{p,u,w^*} \) by running \( \text{IndEnc} \) and \( \text{TdGen} \), respectively. It requires that \( T_{p}^* \) should satisfy the requirement specified in the keyword secrecy game.
- **Phase 2.** This phase is similar to Phase 1.
- **Guess.** \( A \) guesses a keyword \( w' \) and sends it to \( C. \) \( C \) calls \( \text{IndEnc} \) to generate the index \( i_{1,w'} \). If \( i_{1,w'} \) matches with \( T_{p,u,w^*} \), \( A \) wins this secrecy game. Otherwise, \( A \) fails.

Assume that \( A \) has issued \( q \) different keywords before guessing the keyword \( w' \), then the size of remaining keyword space is \( |W| - q \). In addition, as \( H_2(\cdot) \) is a one-way hash function, deriving \( w^* \) from the hash value \( H_2(w^*) \) is negligible. Thus, \( A \)'s advantage in winning the above secrecy game is at most \( \frac{1}{|W|} + \epsilon \). That is, the probability of \( A \) in winning the above secrecy game is negligible. Therefore, our protocol guarantees the keyword secrecy in the random oracle.

Our protocol not only prevents attackers (i.e., outsiders or honest-but-curious cloud server) deducing keyword secrecy from indexes but also disables them from launching keyword guessing attacks on trapdoors, even though they can obtain the secret key of cloud server.

**Theorem 3.** Our protocol can resist the keyword guessing attack from attackers on condition that CDH assumption holds.

**Proof:** We take the server as an example. When launching the keyword guessing attacks, the server can obtain the public parameters, its secret key \( sk_{cs} \) and submitted trapdoors. Suppose the server attempts to guess the keyword \( \tilde{w} \) from trapdoor \( T_{p,u,w} \). Given the trapdoor components \((Ap_{u,1}, Ap_{p,2}) = (sk_{u,ap}v^r, w^z)\), the server does the guessing attacks as follows.

- On input the public parameters \( u, w^r, v, \) the server wants to deduce \( v^r \) from the term \( sk_{u,ap}v^r \). For \( u, v \in G \), we set \( v = u^r \), where \( r \in \mathbb{Z}_p \). That is, given a tuple \((u, u^r, w^z)\), the server attempts to deduce \( w^z \). If the server can compute \( w^z \), then it can deduce the authorized key \( sk_{u,ap} \) from multiple owners. The server then can obtain \( sk_{u,ap} \) which makes it successfully pass the owner level permission verification. However, The above case conflicts with the CDH assumption.
- If the server cannot deduce \( sk_{u,ap} \), then it does not have opportunity for attribute level matching. Even though it can pass the owner level permission verification, it still cannot pass the attribute level match verification if it does not have the attribute level permission key \( sk_{u,att} \), which has proved by Theorem 1.

Thus, the cloud server does not distinguish \( \tilde{w} \) from those embedded in trapdoors, which means that it cannot have chance to successfully launching keyword guessing attack under the CDH assumption.

**Theorem 4.** Our protocol guarantees confidentiality of owner level permission ciphertexts under the CDH assumption, which means that the unauthorized users cannot deduce any sensitive information from owner level permission ciphertexts.

**Proof:** Suppose the unauthorized users are attackers that may be curious to deduce the sensitive information from available ciphertext \( C_{ap} = (c_{i,1}, c_{i,2}, c_{i,3}) \) where \( c_{i,1} = g^{b_1, i, 1}, c_{i,2} = g^{b_2, i, 1}, c_{i,3} = g^{b_2, i, 1} \). For ease of presentation, we define each unauthorized user as an attacker \( A \), then we simulate \( A \)'s attacks as follows.

- Given the ciphertext \((c_{i,1}, c_{i,2}, c_{i,3})\), the goal of \( A \) is to deduce \( g^{b_1, i, r_1} \). Due to \( u, v \in G \), we set \( v = u^{z_i}, g_i = u^{z_2} \) where \( z_1, z_2 \in \mathbb{Z}_p \). Then, \( A \) will use \((u^{r_1}, u^{z_1}, u, u^{z_2})\) to deduce \( u^{r_2, z_2} \). If \( A \) successfully deduces the term \( u^{r_2, z_2} \), then it can solve the CDH problem, which conflicts the CDH assumption.
- Given the ciphertext \((c_{i,2}, c_{i,3}, v)\), the goal of \( A \) is to deduce the term \( v^{r_1} \). Let \( v = z_1^{r_1} \), where \( z \in \mathbb{Z}_p \). Then, \( A \) will use \((z_1^{r_1}, z_2^{r_1})\) to deduce \( z_1^{r_2, r_1} \). If \( A \) successfully deduces the term \( z_1^{r_2, r_1} \), it can break the CDH assumption.

Thus, \( A \) cannot deduce sensitive information from the available owner level permission ciphertexts under the CDH assumption. That is, our protocol can guarantee the confidentiality of owner level permission ciphertexts.

**Theorem 5.** Our protocol also achieves controlled searching, which means that the users cannot access data out of the scopes of their privileges.

**Proof:** In our proof, the authorized users issue search queries over encrypted records owned by specified data owner set \( S \), but cannot search records outside this set. In addition, each user cannot generate the valid aggregate key for the new data owner set according to the known one. The proof of Theorem 5 is on the basis of following two lemmas.

**Lemma 1.** Even though the honest-but-curious server colludes with a malicious user, the attacker still cannot issue search queries over encrypted records not in the scope of compromised aggregate key. In the Search algorithm, the at-
algorithms Ours ASBKS [46] ABDKS [47]

KeyGen \((mS + n_j + 2)E + n_j H_1\) \((3n_a n_c + n_a + 1)E + n_a n_c H_1\) \((2n_j + 6)E\)

IndEnc \((7m + 2mn_a)E + mn_a H_1\) \((2mn_a n_c + 5m)E + mn_a n_c H_1\) \((2mn_j + 8m)E + 2mE_T + mP\)

TdGen \((2n_j + 6)E\) \(2(n_a n_c + n_a + 4)E\) \((2n_j + 4)E\)

Search \((2mn_s + m)E + (4m + 2mn_a n_j + 3mn_s)P + mE_T\) \((2mn_a n_c + 4m)P\) \((2mn_a n_c + mn + 5m)P + 3mE_T\)

Notes: ‘m’: number of owner, ‘mS’: number of system attribute, ‘n_a’: number of possible values for each attribute, ‘n_j’: attribute number of user.

Table 3: Computational Cost Comparison

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Ours</th>
<th>ASBKS [46]</th>
<th>ABDKS [47]</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen</td>
<td>((2n_j + 2)E)</td>
<td>((2n_a n_c + n_a + 1)E)</td>
<td>((2n_j + 4)E)</td>
</tr>
<tr>
<td>IndEnc</td>
<td>((2mn_j + 6m)G)</td>
<td>((2mn_a n_c + 4m)G)</td>
<td>((2mn_j + 7m)G + 2mG_T)</td>
</tr>
<tr>
<td>TdGen</td>
<td>((2n_j + 5)G)</td>
<td>((2n_a n_c + n_a + 3)G)</td>
<td>((2n_j + 4)G +</td>
</tr>
<tr>
<td>Search</td>
<td>(3m(G + (4m + 2mn_a n_S +mn_s)G_T))</td>
<td>((2mn_a n_c + 5m)G_T)</td>
<td>((2mn_a n_c + mn + 6m)G_T)</td>
</tr>
</tbody>
</table>

Table 4: Storage Cost Comparison

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Ours</th>
<th>ASBKS [46]</th>
<th>ABDKS [47]</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen</td>
<td>((2n_j + 2)G)</td>
<td>((2n_a n_c + n_a + 1)G)</td>
<td>((2n_j + 4)G)</td>
</tr>
<tr>
<td>IndEnc</td>
<td>((2mn_j + 6m)G)</td>
<td>((2mn_a n_e + 4m)G)</td>
<td>((2mn_j + 7m)G + 2mG_T)</td>
</tr>
<tr>
<td>TdGen</td>
<td>((2n_j + 5)G)</td>
<td>((2n_a n_e + n_a + 3)G)</td>
<td>((2n_j + 4)G +</td>
</tr>
<tr>
<td>Search</td>
<td>(3m(G + (4m + 2mn_a n_S +mn_s)G_T))</td>
<td>((2mn_a n_e + 5m)G_T)</td>
<td>((2mn_a n_e + mn + 6m)G_T)</td>
</tr>
</tbody>
</table>

Notes: ‘m’: number of owner, ‘mS’: number of system attribute, ‘n_a’: number of possible values for each attribute, ‘n_j’: attribute number of user.

The attacker can obtain \(\chi = \prod_{k \in S'} \prod_{k \in S, k \neq i} g_{n+1-k+i}^{\gamma_k} \) and test whether \(e(s_{k,ap}, h_{u,2}) = e(\chi, v)\), but it cannot cancel out \(e(\prod_{k \in S'} \prod_{k \in S, k \neq i} g_{n+1-k+i}, z)\) based on Eq. 11. Thus, the attacker cannot generate the new aggregate key for any new data owner set from the known aggregate keys.

**Theorem 6.** Our protocol achieves the forward secrecy, which prevents revoked users to make valid search queries.

**Proof.** With the version number \(ver^r\), the revoked users can just use the old information \((A_{A_1}, A_{A_2})\) and trapdoor \(T_{k,u,att}\) to make search queries. The server conducts the owner level permission verification as follows.

- Compute \(A_{p_1} = A_{u_1} \cdot \prod_{k \in S, k \neq i} h_{n+1-k+i}^{\gamma_k} \cdot \prod_{k \in S, k \neq i} g_{n+1-k+i, 2} = A_{A_1} \cdot \prod_{k \in S} g_{n+1-k+i}^{\gamma_k} \cdot \prod_{k \in S, k \neq i} g_{n+1-k+i}^{\gamma_k} \).
- Compute \(pub = \prod_{k \in S} h_{n+1-k+i}^{\gamma_k} = \prod_{k \in S} g_{n+1-k+i}^{\gamma_k} \).
- Call Eq. 10 to execute the verification.

Because the version number embedded in \(A_{p_1}\) is \(ver\) rather than \(ver^r\), then Eq. 10 does not hold. Thus, our protocol prevent revoked users to make valid search queries using old secret keys, which achieves the forward secrecy.

### 7 Implementation and Evaluation

#### 7.1 Theoretical Evaluation

For theoretical evaluation, we will analyze our scheme (AESM²) and the schemes ASBKS [46], ABDKS [47] from the computational cost (shown in Table 3) and storage cost (shown in Table 4). In computational cost analysis, we only consider the costly operations: modular exponentiation operation \(E\) or \(E_T\) in group \(G\) or \(G_T\), hash operation \(H_1\), which maps the arbitrary string into elements in group \(G\) and pairing operation \(P\). For the theoretical storage cost, we define the element length of \(G\), \(G_T\) and \(Z_p\) as \(|G|\), \(|G_T|\) and \(|Z_p|\), respectively.

From Table 3, we can find that the proposed AESM² outperforms both ASBKS and ABDKS in KeyGen algorithm. This is because AESM² involves fewer modular exponentiation operations than those in ASBKS and ABDKS. In the execution of IndEnc algorithm, AESM² is more efficient than ASBKS since there are multiple transitional nodes of the access tree in ASBKS. Although our scheme AESM² has a higher computational cost than ABDKS, it does not bring much burden as this is a one-time computation. Comparing the costs of different schemes in TdGen process, we can easily learn that the performance of AESM² is similar to that of ABDKS, and both perform better than ASBKS when generating one trapdoor. However, when searching from multiple owners, both ASBKS and ABDKS need to iterate multiple times. The proposed AESM² is much more efficient, which can be revealed from the following experimental evaluation. In the Search algorithm, AESM² will be the most efficient when the owner level permission set number \(m_s\) is much smaller than the number of owners \(m\) in the system. All the computational operations in ASBKS and ABDKS are proportional to the owner’s number, while AESM² is affected more by the size of owner level permission set \(m_s\).

In Table 4, we present the storage cost comparison of our scheme AESM² with ASBKS and ABDKS. In the KeyGen process, AESM² is approximately equal to the cost of ABDKS. Both schemes outperform the ASBKS since each attribute has multiple possible values in ASBKS scheme. In the process of IndEnc algorithm, the storage cost of the proposed AESM² is less than those in ASBKS and ABDKS.

---

2. For trapdoor, the storage cost is also the communication cost.
3. In this work, the element length in \(G_T\) is the same as that in \(G\).
4. We compare performances of schemes in users’ key generation.
(b) User key generation (w.r.t. $m_S$)

(c) Owner key generation

When a user queries data from multiple owners, the storage cost of AESM$^2$ in TdGen is more efficient than those in ASBKS and ABDKS since AESM$^2$ aggregates multiple owners’ permissions into one trapdoor, which reveals the advantage of our design. In Search algorithm, the storage cost of AESM$^2$ is decided by the number of owner $m$ and the size of permitted owner set $m_S$, while the ASBKS and ABDKS schemes are both proportional to the number of owner $m$. So, our scheme AESM$^2$ costs much less than those of the other two schemes when $m_S$ is much smaller than $m$. Thus, the proposed AESM$^2$ will be much storage-saving in the system with a large scale of owners.

### 7.2 Experimental Evaluation

To practically evaluate the performance, we simulate our scheme AESM$^2$ and other related works using the real-world dataset $^5$. The simulations are implemented in Python on the Ubuntu server 18.04.3 with Intel Core i7-9700 CPU of 3.00GHz. We use the rapid cryptographic prototyping tool Crypto Charm $^{[48]}$ to implement the underlying operations. The bilinear pairing is initialized as Type A pairing by setting the symmetric curve SS512 in Charm. In our implementation, $|G| = |G_T| = 1024$ bits, $|\mathbb{Z}_p| = 160$ bits. We choose 10000 files and randomly extract 1000 keywords to test four main algorithms: KeyGen algorithm, IndEnc algorithm, TdGen algorithm, and Search algorithm in AESM$^2$ and related researches ASBKS and ABDKS scheme, respectively.

Fig. 3(a) and Fig. 3(b) show the performance of computational costs of three schemes in user key generation (KeyGen) with different number of owner and user’s attribute, respectively. In Fig. 3(a), for comparison, we set the attribute number $n_a = 10$ in all three schemes. We can find that both the costs in ASBKS and ABDKS are proportional to the owner’s number when querying data from multiple owners. In contrast, the user key generation cost of our scheme AESM$^2$ is not influenced by the number of the owner. In Fig. 3(b), along with the different number of attribute of user, the cost of ABDKS is more efficient than those in AESM$^2$ and ASBKS since the additional hash operation $H_1$ is more time-consuming than exponentiation $E$. For example, when setting $n_j = 15$, AESM$^2$ needs 75.85ms to generate keys, and ASBKS needs 142.09ms while ABDKS only costs 35.42ms. From Fig. 3(c), we can see that the time cost of owner key generation of AESM$^2$ increases when the number of the owner rises. In this stage, each owner generates their own public and secret key pairs, and this is a one-time cost. In owner level permission revocation, it is more efficient to update keys than to generate new keys for the users in AESM$^2$.

For IndEnc process, we also evaluate three schemes with regard to different number of the owner and the system attribute. Assuming the number of system attribute is $n_a = 10$ (the number of possible values $n_v = 2$ for each attribute in ASBKS) in Fig. 4(a), the time costs of index generation of three schemes are all proportional to the number of owners. As our scheme AESM$^2$ needs to encrypt the owner level permission and the keyword with attribute simultaneously, it takes extra $3mE$ exponentiation operations compared to the ABDKS scheme, while AESM$^2$ is still more efficient than the ASBKS scheme due to the consuming hash operation in ASBKS. For instance, setting the number of the owner $m = 20$, AESM$^2$ needs 61.10s to encrypt all 1000 keywords, while it takes 81.06s and 40.49s for ASBKS and ABDKS, respectively. Since the index generation algorithm is a one-time operation, it is acceptable for AESM$^2$ in practice. From Fig. 4(b), we can observe that all of the three schemes are not obviously influenced by the number of system attribute when the number of the owner is set to $m = 10$. This is because $m$ will become the dominant factor with a large number of indexes in each owner. In AESM$^2$, the revocation of owner level permission will not influence the index generation phase (no re-encryption with the ciphertexts), which greatly reduces the system burden during revocation.

Fig. 4(c) demonstrates the efficiency of TdGen algorithm varying with different number of user’s attribute. The time cost is for one trapdoor generation. Our scheme AESM$^2$ is around the same cost as the ABDKS scheme, while both are less consuming than the ASBKS scheme. It is worth noting that AESM$^2$ is apparently much more efficient than the other two schemes when searching from multiple owners, as they both need to iterate multiple times. Practically, AESM$^2$ will bring a better search experience for the users. For example, setting $m = 10$, $n_j = 10$, AESM$^2$ consumes 26.25ms, while ASBKS and ABDKS individually need 359.5ms, 261.4ms.

In Fig. 4(d), we evaluate the computational efficiency of our scheme AESM$^2$ with regard to different number of the owner and the permitted number of owner level permission set $S$. We can find that the number of the owner does

---

5. Enron Email Dataset: https://www.cs.cmu.edu/~enron/.
not have a big impact on the efficiency when the size of the permitted owner set $m_S$ is constant. It is the $m_S$ that plays a major role in the search process. Compared with the other two schemes, when the number of the owner in the system increases, the time consumption of AESM\(^2\) does not change much when the number in $S$ is set to 10, as shown in Fig. 4(e). For instance, when $m$ varies from 20 to 60, the time costs of ASBKS and ABDKS increase from 40.50s to 121.20s, and 20.89s to 63.35s, respectively. Nevertheless, the cost of AESM\(^2\) remains almost unchanged, which varies from 15.98s to 16.45s. This advantage of AESM\(^2\) can be significantly efficient when there are many owners, and the permitted set $S$ for the user is relatively small. The efficiency of Search algorithm is presented in Fig. 4(f), where the number of the owner is set to 20, and the size of $S$ in AESM\(^2\) is 10. We can observe that the number of system attribute $n_a$ does not influence much on all three schemes as the size of the index in each owner is large in the evaluation. Furthermore, our scheme AESM\(^2\) outperforms the other two schemes. For instance, the time consumption of AESM\(^2\) is 15.49s when $n_a = 20$, while ASBKS and ABDKS cost 42.02s and 20.2s, respectively. It is worth mentioning that the system efficiency of search is almost the same for the owner level permission revocation in AESM\(^2\).

In Fig. 5, we analyze the performance of TdGen and Search algorithms of three schemes for storage cost. We first evaluate the TdGen algorithm with different number of the owner in Fig. 5(a), where our scheme AESM\(^2\) is obviously much more efficient than ASBKS and ABDKS. Both the other schemes need to generate a trapdoor for each owner, while AESM\(^2\) can query from multiple owners with one trapdoor submission. The storage overhead will not increase along with the owner’s number the same as the other two schemes. For example, setting $n_a = 10, n_v = 2, n_j = 10,$
when \( m \) doubles from 20 to 40, the storage overheads of ASBKS and ABDKS will change from 921600 bits to 1843200 bits, and 494720 bits to 989440 bits, while AESM\(^2\) remains 25600 bits. This property of aggregate key greatly reduces the storage on the user side in our design. Accordingly, the communication overheads between the users and server of AESM\(^2\) will be relatively smaller than the other schemes. In Fig. 5(b), when the number of the system owner increases, all the storage costs of Search process in three schemes will rise. In AESM\(^2\), there is additional \( 3nG |G| \) storage overhead in the owner level permission verification design. However, AESM\(^2\) still performs more storage-saving than ASBKS and ABDKS. For instance, with \( n_a = 10, n_e = 2, n_S = 10 \), the cost of ASBKS is 921600 bits when the owner number \( m \) is 20, and ABDKS needs 532480 bits space, while AESM\(^2\) only costs 378880 bits. The system efficiency will benefit a lot from AESM\(^2\) when there are many owners in the system with a relatively small permitted owner set. Moreover, we analyze the influence of the system attribute number in different schemes as shown in Fig. 5(c), where storage costs in all schemes rise when the number of system attribute increases. We can observe that our scheme AESM\(^2\) brings less storage overhead than both ASBKS and ABDKS schemes. Thus, AESM\(^2\) is desirable and efficient in the multi-owner and multi-user model.

8 Conclusion

In this paper, we study the data-sharing problem in multi-owner and multi-user distributed systems. We propose an efficient attribute-based encrypted keyword search scheme, named AESM\(^2\), where access control requirements of owners are satisfied. Our design uses a two-level matching structure to achieve owner level permission and attribute level permission. In AESM\(^2\), users can search data from multiple owners with a single search trapdoor. Furthermore, we design an efficient revocation method for users’ owner level permission. We formally prove the security of our scheme and also analyze the performance by comparing it with related works. Through theoretical and experimental comparison analysis, we show that our scheme AESM\(^2\) is more efficient in terms of computation and storage costs in the multi-owner and multi-user model. A limitation of our work is the security assumption of the server. We rely on the server to perform efficient system updating for owner level permission revocation. So, it is necessary to assume that the server cannot help revoked users recover access permissions. Although our design is dynamic for access control, it can only support static databases. It is an interesting direction to design a cryptosystem that can achieve dynamic databases in the multi-owner and multi-user model.

Acknowledgements

This work was funded in part by National Natural Science Foundation of China under Grants 61732022, 62072361, and 62102035, Shenzhen Science and Technology Program under Grants GXWD2022017124827001 and JCYJ20210324132406016, Fundamental Research Funds for the Central Universities under Grant JB211505, the Research Grants Council of Hong Kong under Grants CityU11213920, 11217819, 11217620, 11218521, 11202419, RFS2122-1S04, C2004-21GF, R1012-21, and R6021-20F, and by InnoHK initiative, the Government of the HKSAR, and Laboratory for AI-Powered Financial Technologies.

References


Mingyue Wang received the B.E. degree in Computer Science from Harbin Engineering University in 2018. She is currently pursuing the Ph.D. degree in Department of Computer Science both with Harbin Institute of Technology, Shenzhen, and City University of Hong Kong. Her current research interests include cloud computing security, applied cryptography, and blockchain technology.

Yinbin Miao (M’18) received the B.E. degree with the Department of Telecommunication Engineering from Jilin University, Changchun, China, in 2011, and Ph.D. degree with the Department of Telecommunication Engineering from Xidian University, Xi’an, China, in 2016. He is also a postdoctor in Nanyang Technological University, and a postdoctor in City University of Hong Kong. He is currently an Associate Professor with the Department of Cyber Engineering in Xidian University, Xi’an, China. His research interests include information security and applied cryptography.

Yu Guo (M’19) is currently a Lecturer at the School of Artificial Intelligence, Beijing Normal University. He received his B.E. degree in Software Engineering from Northeastern University in 2013, the M.Sc degree in Electronic Commerce and Ph.D. degree in Computer Science from City University of Hong Kong in 2014 and 2019. He has also been a Postdoctor and Research Fellow at City University of Hong Kong. His research interests include cloud computing security, network security, privacy-preserving data processing, and blockchain technology.
Hejiao Huang graduated from the City University of Hong Kong and received the Ph.D. degree in computer science in 2004. She is currently a professor with the Department of Computer Science in Harbin Institute of Technology (Shenzhen), Shenzhen, China, and previously was an invited professor at INRIA, France. Her research interests include cloud computing, trustworthy computing, formal methods for system design and wireless networks.

Cong Wang (F’21) is currently a Professor in the Department of Computer Science, City University of Hong Kong. His research interests include data and network security, blockchain and decentralized applications, and privacy-enhancing technologies. He has been one of the Founding Members of the Young Academy of Sciences of Hong Kong since 2017, and has been conferred the RGC Research Fellow in 2021. He is a co-recipient of the Best Paper Award of IEEE ICDCS 2020, ICPADS 2018, MSN 2015, the Best Student Paper Award of IEEE ICDCS 2017, and the IEEE INFOCOM Test of Time Paper Award 2020. His research has been supported by multiple government research fund agencies, including National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He has served as associate editor for IEEE TDSC, IEEE TSC, IEEE IoT-J, IEEE Networking Letters, and The Journal of Blockchain Research, and TPC co-chairs for a number of IEEE conferences and workshops. He is a fellow of the IEEE, and member of the ACM.

Xiaohua Jia (F’13) received the BSc and MEng degrees in 1984 and 1987, respectively, from the University of Science and Technology of China, and the DSc degree in 1991 in information science from the University of Tokyo. He is an adjunct in Harbin Institute of Technology (Shenzhen) while performing this work. He is currently the chair professor with the Department of Computer Science at City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks and mobile wireless networks. He is an editor of the IEEE TPDS (2006-2009), Wireless Networks, Journal of World Wide Web, Journal of Combinatorial Optimization, etc. He is the general chair of ACM MobiHoc 2008, TPC co-chair of IEEE MASS 2009, area-chair of IEEE INFOCOM 2010, TPC co-chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symposium, and Panel co-chair of IEEE INFOCOM 2011. He is fellow of the IEEE Computer Society.