Identity-certifying Authority-aided Identity-based Searchable Encryption Framework in Cloud Systems

Zi-Yuan Liu, Yi-Fan Tseng, Raylin Tso, Yu-Chi Chen, and Masahiro Mambo

Abstract—In the era of cloud computing, massive quantities of data are encrypted and uploaded to the cloud to realize a variety of applications and services while protecting user confidentiality. Accordingly, the formulation of methods for efficiently searching encrypted data has become a critical problem. Public-key encryption with keyword search is an efficient solution that allows the data owner to generate encrypted keywords for a given document while also allowing the data user to generate the corresponding trapdoor for searching. Huang and Li proposed a public-key authenticated encryption with keyword search (PAEKS) scheme to resist keyword guessing attacks, where the data owner not only encrypts keywords but also authenticates them. However, existing PAEKS-related schemes carry a trade-off between efficiency, storage cost, and security. In this paper, we introduce a novel framework, called identity-certifying authority-aided identity-based searchable encryption, which has the advantage of reducing storage space while remaining the efficiency and security. We formally define the system model and desired security requirements to represent attacks in a real scenario. In addition, we propose a provably secure scheme based on the gap bilinear Diffie–Hellman assumption and experimentally evaluate our scheme in terms of its performance and theoretical features against its state-of-the-art counterparts.

Index Terms—cloud systems, identity-based searchable encryption, identity-certifying authority, keyword search.

I. INTRODUCTION

WITH the maturation of cloud computing technology, enterprises have increasingly uploaded massive quantities of data to the cloud to reduce their storage and computing burden. For example, convenience store chains upload data from each branch to the cloud for analysis, and hospitals upload patient data to the cloud for management. In addition, since the introduction of the concept of Industry 4.0 by Lasi et al. [1], firms have begun integrating cloud systems into their data collection and production processes. However, privacy concerns remain on shipments, patient records, and even factory inventory are highly sensitive, and companies may be reluctant to directly upload them to cloud systems that they cannot fully trust. Consequently, data are often encrypted before being uploaded to cloud systems in order to avoid information leakage, but such encrypted data pose a computational challenge for cloud systems.

Symmetric searchable encryption (SSE), introduced by Song et al. [2], is one solution to the aforementioned problem. In SSE, a data owner (DO) can generate encrypted keywords for each encrypted file by using a symmetric key that is shared with the data user (DU) before their data are uploaded to the cloud systems. Subsequently, the DU can generate a trapdoor for specified keywords and submit them to the cloud systems to search for encrypted files that are related to these keywords. Because of these properties, SSE is well suited to cloud computing, and various SSE approaches have been proposed [3], [4], [5], [6], [7]. However, SSE is restricted to the key sharing problem of symmetric cryptosystems. Specifically, the DO and DU must agree on a shared key before encrypting keywords and generating trapdoors, respectively.

To further increase the range of application and reduce the communication overhead of negotiating keys, Boneh et al. [8] introduced a searchable encryption method in a public-key setting, called public key encryption with keyword search (PEKS). Instead of using a shared key as done in SSE, in PEKS, the DO encrypts keywords by using the DU’s public key, and the DU generates corresponding trapdoors by using his or her secret key. After the pioneering work by Boneh et al., PEKS immediately caught the attention of researchers, and many studies have applied PEKS to various applications [9], [10], [11], [12], [13], [14], [15], [16]. However, in 2006, Byun et al. [17] observed that because the entropy of keywords is low, any malicious party, through a so-called keyword guessing attack, can randomly select keywords to generate the ciphertext and test whether the ciphertext is passable; thus, the malicious party can obtain the information associated with the keywords in the trapdoor. In particular, to resolve the keyword guessing attack launched by a malicious cloud server (CS) as part of a so-called insider keyword guessing attack (IKGA), Chen et al. [18], [19], [20] have first proposed solutions for dual-server setting, and their methods were improved upon by Tso et al. [21].

Huang and Li [22] recently introduced the concept of public-key authenticated encryption with keyword search (PAEKS) under a single-server setting, where the trapdoor works only for ciphertext that is authenticated by the DO using his or her secret key; therefore, a malicious CS cannot randomly generate ciphertext and further perform IKGA. Inspired by Huang and Li’s work [22], scholars have proposed several PAEKS schemes [23], [24], [25], [26], [27], [28], [29].
However, because PAEKS schemes are based on public key settings and do not provide implicit authentication, they require a trusted public key infrastructure to bind public keys with the respective identities of entities through issuing certificates; nevertheless, this process incurs additional overhead. Several approaches are available for intuitively solving this problem in PAEKS:

1) Identity-based authenticated encryption with keyword search (IBAEKS) [30]: In IBAEKS schemes, the user can apply to a trusted key generation center (KGC) for the secret key of a specific identity; they can then use the identity as their public key. Thus, in IBAEKS, no additional steps are required to prove the validity of the public key.

2) Certificate-based authenticated encryption with keyword search (CBAEKS) [31], [32]: In CBAEKS schemes, the user can apply to a trusted certification authority for the certificate of a pair comprising a specific identity and public key. Unlike the certificates used in PAEKS schemes, the certificate in CBAEKS not only implicitly authenticates the validity of the identity and public key but also acts as a partial secret key.

3) Certificateless authenticated encryption with keyword search (CLAKEKS) [33], [34], [35], [36]: The solution of CLAKEKS is similar to that of IBAEKS, except that the user generates a pair comprising public and secret values, in addition to obtaining the secret key of their identity from the KGC. The user finally uses the identity and public value as the full public key and uses the secret key and secret value as the full secret key.

Although the three aforementioned approaches solve the problem in PAEKS, each approach has its disadvantages in safety, storage, or efficiency. Specifically, in IBAEKS [30], because the KGC can gain access to any user’s secret key, key escrow problems occur when the KGC is malicious. By contrast, although CBAEKS [31], [32] and CLAKEKS [33], [34], [35], [36] solve the key escrow problem, the users in these schemes require an additional generated certificate or a public–secret value pair to encrypt keywords or generate trapdoors. Consequently, users require more storage space to store certificates and keys. It becomes less convenient for the users in CBAEKS and CLAKEKS schemes than in IBAEKS schemes, because the users have to use additional information instead of merely using their identities to encrypt keywords and generate trapdoors. Accordingly, to ameliorate this trade-off, we present a novel identity-certifying authority (ICA)-aided identity-based searchable encryption (IBSE) framework (hereafter referred to as ICA-IBSE).

Our contributions are summarized as follows:

- To master the trade-off between efficiency and security, on the basis of Chow et al. [37] and Emura et al. [38], who have used an ICA in identity-based encryption, we propose our ICA-IBSE scheme, which inherits the advantages of IBAEKS in terms of convenience and storage requirements and eliminates the disadvantage of IBAEKS regarding the key escrow problem.
- We define the system model and security requirements of the ICA-IBSE framework before applying it to a practical case. Moreover, we provide the security proofs to show that under the defined security models, our scheme is secure if the gap bilinear Diffie–Hellman (GBDH) assumption holds.
- We further provide a theoretical comparison and performance evaluation of our scheme against state-of-the-art schemes [24], [25], [30], [31], [32], [33], [35]. The results indicate that our scheme is more efficient and that it performs better at reducing the storage requirement for the public key, ciphertext, and trapdoor.

II. PRELIMINARIES

In this section, we introduce the requisite background, including notations, digital signature, symmetric bilinear groups, and gap bilinear Diffie–Hellman assumption.

A. Notations

For simplicity and readability, TABLE I describes the notations used throughout the paper.
B. Digital Signature

A digital signature scheme Sig with the message space $\mathcal{M}$ comprises three algorithms:

1. Sig.KeyGen($1^\lambda$): The security parameter $\lambda$ is taken as the input, and a verification key $vk_{Sig}$ and signing key $sk_{Sig}$ are the outputs.

2. Sig.Sign($sk_{Sig}, m$): The signing key $sk_{Sig}$ and a message $m \in \mathcal{M}$ are taken as the inputs, and a signature $\sigma_{Sig}$ is the output.

3. Sig.Verify($vk_{Sig}, m, \sigma_{Sig}$): The verification key $vk_{Sig}$, a message $m \in \mathcal{M}$, and a signature $\sigma_{Sig}$ are taken as the inputs, and $1$ is the output if the signature is valid; otherwise, $\bot$ is the output.

A Sig is considered to be correct if for all $\lambda$, all messages $m \in \mathcal{M}$, all $(vk_{Sig}, sk_{Sig}) \leftarrow$ Sig.KeyGen($1^\lambda$), and all $\sigma_{Sig} \leftarrow$ Sig.Sign($sk_{Sig}, m$), the equation $Pr[\text{Sig.Verify}(vk_{Sig}, m, \sigma_{Sig}) = 1] = 1$ holds. In addition, we consider that Sig is secure against an existential unforgeability under an adaptive chosen message attack (EU-CMA) if, for any $A$, the advantage of $A$ wins the EU-CMA game $Adv_{\text{EU-CMA}}(\Phi) := Pr[\text{Sig.Verify}(vk_{Sig}, m^*, \sigma_{Sig}^*) = 1] = 1$ is negligible. Here, the adversary $A$ is allowed to query the signing oracle $O_{\text{Sig}}(\cdot)$ on any message $m \neq m^* \in \mathcal{M}$ and allowed to obtain the corresponding signature $\sigma_{Sig}$.

C. Symmetric Bilinear Groups

Let $q$ be a $\lambda$-bit prime, and let $G_1$ and $G_T$ be two cyclic groups of the same prime order $q$, where $g \in G_1$ is a generator. In addition, let $\hat{e} : G_1 \times G_1 \rightarrow G_T$ be a bilinear pairing (a map). We consider that the tuple $(g, g, G_1, G_T, \hat{e})$ is a symmetric bilinear group if the following properties are satisfied:

- Bilinearity: for all $g_1, g_2 \in G_1$ and $a, b \in \mathbb{Z}_q^*$, $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$;
- Nondegeneracy: $g_T := \hat{e}(g, g)$ is a generator of $G_T$ (i.e., $g_T \neq 1$ holds);
- Computability: $\hat{e}$ is efficiently computable.

D. Gap Bilinear Diffie–Hellman Assumption \cite{39, 40}

Given a symmetric bilinear group tuple $\Phi = (g, g, G_1, G_T, \hat{e})$, we consider that the GBDH assumption holds if, for any probabilistic polynomial-time algorithm (PPT) adversary $A$, the advantage (defined as follows) is negligible:

$$Adv_{\text{GBDH}}(\Phi, A) := Pr[T = \hat{e}(g^a, g^b) | a, b, c \in \mathbb{Z}_q^*, T \leftarrow O_{\text{GBDH}}(\Phi, g^c, g^b, g^c)],$$

where $O_{\text{GBDH}}$ denotes the maximum number of queries by $A$ to $O_{\text{GBDH}}$, and $O_{\text{GBDH}}$ denotes a decision bilinear Diffie–Hellman oracle that takes $(g^a, g^b, g^c, T)$ as its input and outputs $1$ if $\hat{e}(g, g)^{abc} = T$ and $0$ if otherwise.

III. DESCRIPTION OF ENTITIES IN ICA-IBSE

The ICA-IBSE comprises five entities, namely the Identity-certifying Authority (ICA), Key Generation Center (KGC), Cloud Server (CS), Data Owner (DO), and Data User (DU), which are displayed in Fig. 1.

- ICA: This authority is responsible for validating the DO’s and DU’s identities and issuing trapdoor information and an identity certificate to them.
- KGC: By validating the correctness of the identity certificate, the KGC generates the DO’s and DU’s partial secret keys without knowing any information about their identity.
- CS: The CS has sufficient storage and computing capacity and is mainly responsible for storing encrypted data along with the corresponding encrypted keywords and searching for data.
- DO: The DO first requests his/her certificate from the ICA and further generates his/her partial secret key by interacting with the KGC. Finally, the DO obtains his/her secret key by using trapdoor information. The DO can generate massive quantities of encrypted data along with the corresponding encrypted keywords, which are uploaded to the CS to reduce the storage requirement.
- DU: The DU generates his/her secret key as the DO does. Subsequently, the DU can issue trapdoors, generated by using his/her secret key, to the CS to retrieve the encrypted data associated with the specified keyword.

In the following analysis, we consider adversary threats from different perspectives:

- The DO and DU are fully trusted, as all of the case in PAEKS schemes, because DO and DU know the secret information (i.e., trapdoor). Therefore, the DO and DU is also unable to collude with other parties to reveal their secret keys, or the keyword privacy can be directly compromised by other parties.
- The ICA and KGC cannot collude with each other, because it is impossible for KGC to issue any secret

**Fig. 1.** The proposed ICA-IBSE framework
key to a user whose identity cannot be directly or indirectly identified. Therefore, to enable KGC to generate secret keys, there must be some party (e.g., ICA) that can privately authenticate the user’s identity. Then, the user can indirectly validate his/her identity to the KGC. However, if ICA and KGC are collude with each other, they can adaptively generate any user’s secret keys. Therefore, we need to assume that ICA and KGC cannot collude. Although this restriction has to be added, it is still in accordance with the real scenarios. For example, supposing ICA is a government-issued certificate party that sends a certificate showing the user is authenticated, with this certificate, the service provider (e.g., KGC) can provide the secret key to the user only by verifying the certificate without further knowing the user’s true identity, so the service provider is unable to generate the secret key itself. In general, there are usually government regulators to ensure that the government is not colluding with service providers.

- The KGC and CS are honest but curious, which means that they will attempt to retrieve the sensitive information of the keywords from encrypted keywords and trapdoors.
- The ICA is malicious. In addition to attempting to obtain keyword information as the KGC and CS do, the ICA can generate a potentially malicious ICA key pair.
- The communication channels between the DO and DU and the cloud server are insecure, which means that all transmitted information is eavesdropped upon by any one party (e.g., ICA, KGC, and a malicious outsider). However, the communication channels between the DO and DU and the ICA and KGC are secure. In other words, we assume that the communication channels are encrypted and authenticated (e.g., by using TLS 1.3 [41]).

IV. DEFINITION AND SECURITY MODELS OF ICA-IBSE

A. Definition

A typical ICA-IBSE scheme comprises seven algorithms and a protocol, which are described as follows:

Setup(1^λ): The security parameter λ is taken as the input, and the system parameter pp is the output. Here, we assume that the keyword space W and the identity space ID of ICA-IBSE are defined by the system parameter pp.

ICA-Setup(pp): This algorithm is executed by ICA that takes the system parameter pp as input, and outputs the public–secret key pair (pkICA, skICA) of the ICA.

KGC-Setup(pp): This algorithm is executed by KGC that takes the system parameter pp as input, and outputs the public–secret key pair (pkKGC, skKGC) of the KGC.

ICA-Cert(pp, pkICA, skICA, ID): This algorithm is executed by ICA that takes the system parameter pp, ICA’s public key pkICA, ICA’s secret key skICA, and a user’s identity ID ∈ ID as inputs, and outputs a certificate certID and a piece of trapdoor information tfID, which are sent to the ID through a secure channel.

(User-Obtain-Key(pp, pkKGC, ID, certID, tfID), KGC-Issue-Key(pp, pkKGC, skKGC, pkICA)): This is an interactive key-issuing protocol between a user and the KGC that comprises two algorithms: User-Obtain-Key and KGC-Issue-Key. The user first generates the first-round message \(M_1 \leftarrow \text{User-Obtain-Key}(pp, pk_{KGC}, ID, cert_{ID}, tf_{ID})\) and submits it to the KGC. Subsequently, the KGC returns a second-round message \(M_2 \leftarrow \text{KGC-Issue-Key}(pp, pk_{KGC}, sk_{KGC}, pk_{ICA})\) to the user. At the end of the protocol, the user can locally output a secret key sk_{ID} or \(\bot\).

Encrypt(pp, sk_{DO}, DU, w): This algorithm is executed by DO that takes the system parameter pp, DO’s secret key sk_{DO}, DU’s identity DU, and a keyword w ∈ W as inputs, and outputs a searchable ciphertext ct_{w} associated with keyword w is the output.

Trapdoor(pp, DO, sk_{DU}, w): This algorithm is executed by DU that takes the system parameter pp, DU’s identity DU, DU’s secret key sk_{DU}, and a keyword w ∈ W as inputs, and outputs a trapdoor td_{w} associated with keyword w.

Test(pp, ct_{w}, td_{w}): This algorithm is executed by CS that takes the system parameter pp, a searchable ciphertext ct_{w}, and a trapdoor td_{w} as the inputs, and 1 is the output if ct_{w} is matched with td_{w}; otherwise, 0 is the output.

Definition 1 (Correctness and Consistency of ICA-IBSE). For all security parameters \(\lambda \in \mathbb{N}\), all DOs DO ∈ ID, all DUs DU ∈ ID, and all keywords w, w′ ∈ W, ICA-IBSE is defined to be correct if, when w = w′, we have \(\Pr[\text{Test}(pp, ct_{w}, td_{w'}) = 1] = 1\) and ICA-IBSE is defined to be consistent if, when w ≠ w′, we have \(\Pr[\text{Test}(pp, ct_{w}, td_{w'}) = 0] = 1 - \text{negl}(\lambda)\), where pp ← Setup(1^λ): (pkICA, skICA) ← ICA-Setup(pp); (pkKGC, skKGC) ← KGC-Setup(pp); (cert_i, td_i) ← ICA-Cert(pp, pkICA, skICA, i); and sk_i ← (User-Obtain-Key(pp, pkKGC, i, cert_i, td_i), KGC-Issue-Key(pp, pkKGC, skKGC, pkICA)), for i = \{DO, DU\}, ct_{w} = Encrypt(pp, sk_{DO}, DU, w), and td_{w} ← Trapdoor(pp, DO, sk_{DU}, w′).

B. Security Models

To model the different aspects of the attacks described in Section [11] we revise the security model in [25] and [42] to account for multichosen keyword attacks (MCKAs) and IKGAs in the ICA-IBSE framework. Here, MCKAs, recently introduced by Qin et al. [25], ensure that no adversary can obtain any information on keywords from two tuples of encrypted keywords; IKGAs ensure that no insider adversary (e.g., the CS) can obtain any information on keywords from the trapdoor, even when an insider can conduct tests.

In ICA-IBSE framework, the CS can directly obtain the entirety of ciphertext and all the trapdoors from DO and DU; hence, intuitively, the CS has greater attack capability relative to eavesdroppers on the channel. In addition, ICA and KGC are not fully trusted party in this framework. Therefore, we not only define the attack from CS (i.e., MCKA-CS and IKGA-CS), but further define additionally security games to model
the attack from ICA and KGA (i.e., MCKA-ICA, IKGA-ICA, MCKA-KGC, and IKGA-KGC) for the real scenarios.

Informally, in these security models, the prefix (MCKA or IKGA) means that the models are related to the MCKA security or IKGA security, while the suffix (ICA, CS, and KGC) means that which malicious party is being modeled. In more detail, for malicious ICA, since ICA cannot collude to KGA in ICA-IBSE framework, the malicious ICA in MCKA-ICA and IKGA-ICA models is unable to obtain any identities’ secret key by querying oracles. As for malicious CS, it is allowed to query user’s partial secret key (i.e., M2) in MCKA-CS and IKGA-CS models. As for malicious KGC, to model the ability of KGC that can generate any user’s partial secret key, malicious KGC is given the KGC’s secret key in MCKA-KGC and IKGA-KGC models. In addition, the malicious KGC is allowed to query user’s certificate (i.e., M1) and further generates any user’s secret key. However, since KGC cannot obtain the identity information of the user in ICA-IBSE, the malicious KGC also cannot obtain the identity information corresponding to the certificate when querying the oracles.

Before presenting the formal security models, we first define the following oracles that are simulated by the challenger $A$ for the adversary $B$:

Certificate Oracle $O_{cert}$: For any identity $ID \in ID$, $B$ outputs $(cert_{ID}, t_{fID}) \leftarrow ICA-Cert(pp, pkICA, skICA, ID)$ to $A$.

Secret Key Oracle $O_{sk}$: For any first-round message $M_1$, $B$ runs $M_2 \leftarrow KGC-Issue-Key(pp, pkKGC, skKGC, pkICA)$ and returns $M_2$ to $A$.

Ciphertext Oracle $O_{ct}$: For any keyword $w \in W$, $B$ outputs $ct_w \leftarrow Encrypt(pp, skDO, DU, w)$ to $A$.

Trapdoor Oracle $O_{td}$: For any keyword $w \in W$, $B$ outputs $td_w \leftarrow Trapdoor(pp, IDList[do], sk_{IDList[do]}, w)$ to $A$.

Issue Key KG Oracle $O_{k-g}$: When $A$ makes this query, $B$ randomly chooses $ID \leftarrow \{ID\}$ and computes $(cert_{ID}, t_{fID}) \leftarrow ICA-Cert(pp, pkICA, skICA, ID)$. In addition, $B$ runs $M_1 \leftarrow User-Obtain-Key(pp, pkKGC, ID, cert_{ID}, t_{fID})$, and returns $M_2$ to $A$. Furthermore, $B$ stores ID to IDList, and updates $Q_{key} \leftarrow Q_{key} + 1$. Here, $Q_{key}$ is the amount of instances obtained by querying $O_{k-g}$, and IDList is a list of identities whose corresponding $M_1$ values have been obtained by the adversary through querying $O_{k-g}$.

Ciphertext KG Oracle $O_{ct-g}$: For any keyword $w \in W$, DO index $do$, and DU index $dai$, $B$ first checks whether $do \in Q_{key}$ and $dai \in Q_{key}$. If not, $B$ forces $A$ to output a random bit $b' \in \{0, 1\}$. Otherwise, $B$ retrieves the $dai$-th identity’s secret key $sk_{IDList[dai]}$ and $dai$-th identity IDList[do] in IDList and subsequently outputs $ct_w \leftarrow Encrypt(pp, sk_{IDList[dai]}, IDList[do], w)$ to $A$.

Trapdoor KG Oracle $O_{td-g}$: For any keyword $w \in W$, DO index $do$, and DU index $dai$, $B$ first checks whether $do \in Q_{key}$ and $dai \in Q_{key}$. If not, $B$ forces $A$ to output a random bit $b' \in \{0, 1\}$. Otherwise, $B$ retrieves the $dai$-th identity IDList[do] and $dai$-th identity’s secret key $sk_{IDList[dai]}$ and subsequently outputs $td_w \leftarrow Trapdoor(pp, IDList[do], sk_{IDList[dai]}, w)$ to $A$.

The following we give a formal description of these security models. In particular, MCKA and IKGA games are represented using blue solid and red dotted lines, respectively.

First, in the MCKA-CS and IKGA-CS games, the adversary is unable to issue queries to $O_{cert}$ on some challenged identities $DO, DU \in ID$. Furthermore, the adversary cannot issue queries to oracles $O_{td}$ on the challenged keywords $w_{b,i} \in W$ and cannot issue queries to oracle $O_{ct}$ on the challenged identities $DO, DU \in ID$, where $b \in \{0, 1\}$ and $i \in \{1, \ldots, n\}$. We consider that ICA-IBSE is MCKA-CS secure if the advantage

$$Adv^{MCKA-CS}_{ICA-IBSE,A}(\lambda) = |Pr[b = b'] - 1/2|$$

is negligible for any $A$, and we consider that ICA-IBSE is IKGA-CS secure if the advantage

$$Adv^{IKGA-CS}_{ICA-IBSE,A}(\lambda) = |Pr[b = b'] - 1/2|$$

is negligible for any $A$.

<table>
<thead>
<tr>
<th>MCKA-CS / IKGA-CS Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pp \leftarrow Setup()$; $b \leftarrow {0, 1}$; $(pkICA, skICA) \leftarrow ICA-Setup(pp); (pkKGC, skKGC) \leftarrow KGC-Setup(pp);$</td>
</tr>
<tr>
<td>$(w_0 = (w_{0,1}, \ldots, w_{0,n}), w_1 = (w_{1,1}, \ldots, w_{1,n}), DO, DU) \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkICA, pkKGC);$</td>
</tr>
<tr>
<td>$\hat{c}^* = (ct_{1}^<em>, \ldots, ct_{n}^</em>)$, where $ct_{i}^* \leftarrow Encrypt(pp, skDO, DU, w_{0,i}),$</td>
</tr>
<tr>
<td>for $i = 1, \ldots, n$; $b' \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkICA, pkKGC);$</td>
</tr>
<tr>
<td>$w_0, w_1, DO, DU) \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkICA, pkKGC);$</td>
</tr>
<tr>
<td>$td^* \leftarrow Trapdoor(pp, DO, skDO, w_0);$</td>
</tr>
<tr>
<td>$b' \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkICA, pkKGC, \hat{c}^*);$</td>
</tr>
</tbody>
</table>

In MCKA-ICA and IKGA-ICA games, we model a malicious ICA that can generate a potentially malicious ICA key pair $(pkICA, skICA)$. However, to model the state of affairs where the ICA cannot interact with the KGC, the adversary cannot access to $O_{sk}$. In addition, the adversary is also unable to issue queries to oracles $O_{td}$ on challenged keywords $w_{b,i} \in W$ and oracles $O_{ct}$ on challenged keywords $w_b \in W$ for some challenged identities $DO, DU \in ID$, where $b \in \{0, 1\}$ and $i \in \{1, \ldots, n\}$. We consider that ICA-IBSE is MCKA-ICA secure if the advantage

$$Adv^{MCKA-ICA}_{ICA-IBSE,A}(\lambda) = |Pr[b = b'] - 1/2|$$

is negligible for any $A$, and we consider that ICA-IBSE is IKGA-ICA secure if the advantage

$$Adv^{IKGA-ICA}_{ICA-IBSE,A}(\lambda) = |Pr[b = b'] - 1/2|$$

is negligible for any $A$.

<table>
<thead>
<tr>
<th>MCKA-ICA / IKGA-ICA Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pp \leftarrow Setup()$; $b \leftarrow {0, 1}$; $(pkICA, skICA) \leftarrow ICA-Setup(pp); (pkKGC, skKGC) \leftarrow KGC-Setup(pp);$</td>
</tr>
<tr>
<td>$(w_0 = (w_{0,1}, \ldots, w_{0,n}), w_1 = (w_{1,1}, \ldots, w_{1,n}), DO, DU) \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkKGC);$</td>
</tr>
<tr>
<td>$\hat{c}^* = (ct_{1}^<em>, \ldots, ct_{n}^</em>)$, where $ct_{i}^* \leftarrow Encrypt(pp, skDO, DU, w_{0,i}),$</td>
</tr>
<tr>
<td>for $i = 1, \ldots, n$; $b' \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkICA, pkKGC);$</td>
</tr>
<tr>
<td>$(w_0, w_1, DO, DU) \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkICA, pkKGC);$</td>
</tr>
<tr>
<td>$td^* \leftarrow Trapdoor(pp, DO, skDO, w_0);$</td>
</tr>
<tr>
<td>$b' \leftarrow A^{O_{cert}()}(\cdot, ch_{cert}(), \cdot)(pp, pkICA, pkKGC, \hat{c}^*);$</td>
</tr>
</tbody>
</table>
Finally, the MCKA-KGC and IKGA-KGC games differ from the preceding games in terms of their setups. Specifically, the adversary is given KGC’s secret key. Moreover, we use $Q_{key}$ to count the amount of instances obtained by querying ICA KGC Oracle $O_{\text{ICAKGC}}$, and we use $IDL_{\alpha, \beta}$ to store the corresponding identity of the secret key that is returned by this oracle. To simulate the state of affairs where the adversary does not know the identity of the user, after the adversary completes the first query phase, the adversary outputs two arbitrary indices $\alpha, \beta \in [Q_{key}]$ instead of two identities $DO, DU \in ID$. Subsequently, by using the indices, the challenger chooses $IDL_{\alpha}$ and $IDL_{\beta}$ from $IDL_{\alpha, \beta}$ as the DO and DU, respectively. Furthermore, the adversary cannot issue queries to $O_{\text{ICAKGC}}$ on challenged keywords $w_b, i \in W$ and to issuing queries to $O_{\text{ICAKGC}}$ on challenged keywords $w_b \in W$ for the challenged indices $(\alpha, \beta)$, where $b \in \{0, 1\}$ and $i \in \{1, \cdots, n\}$. We consider that ICA-IBSE is MCKA-KGC secure if the advantage

\[ \text{Adv}^{\text{MCKA-KGC}}_{\text{ICA-IBSE}, A}(\lambda) := |\Pr[b = b'] - 1/2| \]

is negligible for any $A$, and we consider that ICA-IBSE is IKGA-KGC secure if the advantage

\[ \text{Adv}^{\text{IKGA-KGC}}_{\text{ICA-IBSE}, A}(\lambda) := |\Pr[b = b'] - 1/2| \]

is negligible for any $A$.

### MCKA-KGC / IKGA-KGC Game

\[
pp \leftarrow \text{Setup}(1^\lambda); \text{IDL}_{\alpha} = \emptyset, Q_{key} := 1; \quad b \leftarrow \{0, 1\}; \\
(pk_{\text{ICA}}, sk_{\text{ICA}}) \leftarrow \text{ICA-Setup}(pp); (pk_{\text{KGC}}, sk_{\text{KGC}}) \leftarrow \text{KGC-Setup}(pp); \\
(w_0, w_1, \alpha, \beta) \leftarrow A_{\text{Setup}}(pp, pk_{\text{ICA}}, pk_{\text{KGC}}, sk_{\text{KGC}}); \\
ct^* \leftarrow \text{Encrypt}(pp, sk_{\text{ICA}}, w_0, \alpha, \beta); \\
ct' \leftarrow A_{\text{KGC}}(pp, pk_{\text{ICA}}, pk_{\text{KGC}}, sk_{\text{KGC}}); \\
ct_{ID} \leftarrow \text{Trapdoor}(pp, \text{IDL}_{\alpha}, [w_0, w_1, \alpha, \beta], \text{IDL}_{\alpha, \beta}, w_3); \\
ct_{ID} \leftarrow A_{\text{KGC}}(pp, pk_{\text{ICA}}, pk_{\text{KGC}}, sk_{\text{KGC}}, \text{IDL}_{\alpha, \beta}, w_3); \\
\]

### V. IDENTITY-CERTIFYING AUTHORITY-AIDED

IDENTITY-BASED SEARCHABLE ENCRYPTION

### FRAMEWORK

In this section, we first propose a concrete scheme and then analyze the correctness and consistency of the proposed scheme.

#### A. Our Construction

Let $ID = Z_q^*$ and $W = \{0, 1\}^n$ for some $n$ be the identity space and keyword space of the ICA-IBSE scheme, respectively. Let $DO \in Z_q^*$ and $DU \in Z_q^*$ be the identities of the DO and DU, respectively. In addition, let $Sig := (\text{Sign}, \text{KeyGen}, \text{Sign}, \text{Verify})$ be an EU-CMA-secure digital signature with message space $M = \{0, 1\}^m$ for some $m$.

Setup(1$^\lambda$): This algorithm chooses two cyclic groups $G_1, G_T$ with a large prime order $q$; a generator $g \in G_1$; a pairing $\hat{\epsilon} : G_1 \times G_1 \rightarrow G_T$; and three cryptographic hash functions $H : Z_q^* \rightarrow G_1$, $h_1 : Z_q^* \times Z_q^* \times G_T \times \{0, 1\}^n \rightarrow Z_q^*$, and $h_2 : Z_q^* \times G_1 \rightarrow Z_q^*$. It sets the system parameter to be

\[ pp := \{1^\lambda, G_1, G_T, \hat{\epsilon}, q, g, H, h_1, h_2\}. \]

ICA-Setup(pp): The ICA runs $(vk_{\text{Sig}}, sk_{\text{Sig}}) \leftarrow \text{Sign}.\text{KeyGen}(1^\lambda)$. It then outputs

\[ pk_{\text{ICA}} = vk_{\text{Sig}}, sk_{\text{ICA}} = sk_{\text{Sig}}. \]

KGC-Setup(pp): The KGC picks $x \leftarrow Z_q^*$ and computes $y = g^x$. It then outputs

\[ pk_{\text{KGC}} := y; sk_{\text{KGC}} := x. \]

ICA-Cert(pp, pk_{\text{ICA}}, sk_{\text{ICA}}, ID): The ICA computes $u_{ID} = H(ID)$, picks $y_{ID, 1} \leftarrow Z_q^*$, and computes $u_{ID, 1} = g^{y_{ID, 1}}$. In addition, it computes $u_{ID, 2} \in G_1$ as $y_{ID, 2} = u_{ID} : u_{ID, 1}$ and $\sigma_{sg} \leftarrow \text{Sign}.\text{Sign}(sk_{\text{ID, 2}})$. Finally, it outputs

\[ cer_{ID} := (u_{ID, 2}, \sigma_{sg}); t_{FD} := y_{ID, 1}. \]

User-Obtain-Key(pp, pk_{\text{KGC}}, sk_{\text{ICA}}, ID): The user and the KGC run the following steps:

1) The user sets $M_1 := \text{cert}_{ID} = (u_{ID, 2}, \sigma_{sg})$ and sends $M_1$ to the KGC.

2) After receiving $M_1$, the KGC verifies the correctness of $\sigma_{sg}$. If $\text{Sig}.\text{Verify}(vk_{\text{Sig}}, u_{ID, 2}, \sigma_{sg}) = \bot$, the KGC sets $M_2 := \bot$. Otherwise, it computes $M_2 := y_{ID, 2} = u_{ID, 2}$. Finally, it returns $M_2$ to the user.

3) If $M_2 = \bot$, the user outputs $\bot$. Otherwise, it computes $e_{ID} = y_{ID, 2} : y^{-y_{ID, 1}}$ and outputs

\[ sk_{ID} := e_{ID}. \]

Encrypt(pp, sk_{DO}, DU, w): The DO randomly selects $r \leftarrow Z_q^*$ and computes $c_1 = g^r\cdot c_2 = g^{y_{ID, 2}}h_1(\text{DO}, \text{DU}, k, w, c_1)$, where $k = \hat{\epsilon}(sk_{DO}, H(DU))$. Subsequently, it outputs

\[ ct_{DO} := (c_1, c_2). \]

Trapdoor(pp, DO, sk_{DU}, w): The DU outputs $td_{w} := h_{1}(\text{DO}, \text{DU}, k, w)$, where $k = \hat{\epsilon}(H(DO), sk_{DU})$.

Test(pp, ct_{w}, td_{w}): The CS checks whether

\[ c_2 = c_1^{h_2(td_{w}, c_1)}. \]

It returns 1 if the equation is satisfied and returns 0 otherwise.

#### B. Correctness and Consistency of ICA-IBSE

The following we analyze the correctness and consistency of the proposed ICA-IBSE scheme:

1) For $i \in \{DO, DU\}$, we have

\[ sk_{i} = y_{i, 2} \cdot Y^{-y_{i, 1}} = u_{i, 2}^{r} \cdot u_{i, 1}^{r} \cdot Y^{-y_{i, 1}} = H(i)^{r} \cdot Y^{w_{i, 1}} \cdot Y^{-y_{i, 1}} = H(i)^{r}. \]
2) Considering the Test algorithm for a ciphertext $ct_w = (c_1, c_2)$ and a trapdoor $td_{w'}$, we have
$$g^{h_2(h_1(DO, DU, k, w), c_1)} = c_2 = g^{\ell_2(td_{w'}, c_1)} = g^{h_2(td_{w'}, c_1)} = g^{h_2(h_1(DO, DU, k', w'), c_1)}$$

3) In addition, we have
$$k = \hat{e}(sk_{DO}, H(DU)) = \hat{e}(H(DO)^{\varphi}, H(DU)) = \hat{e}(H(DO), H(DU)^{\varphi}) = \hat{e}(H(DO), sk_{DU})$$
$$= k'.$$

Because $k = k'$, when $w = w'$, we have $h_1(DO, DU, k, w) = h_1(DO, DU, k', w')$, and $c_2 = \ell_2(td_{w'}, c_1)$; therefore, correctness is satisfied. Conversely, when $w \neq w'$, because the probability that $h_1(DO, DU, k, w) = h_1(DO, DU, k', w')$ is negligibly low, we have $c_2 \neq \ell_2(td_{w'}, c_1)$; therefore, consistency is also satisfied.

VI. SECURITY ANALYSIS OF ICA-IBSE

In this section, we demonstrate that our scheme is secure against various forms of attacks.

Theorem 1. The proposed construction is MCKA-CS secure if the underlying signature scheme $\text{Sig}$ is EU-CMA secure under the hard GBDH assumption.

Proof. Suppose that some PPT algorithm $\mathcal{A}$ can break the MCKA-CS security of the proposed scheme. If so, then the following proof demonstrates that some other algorithm $\mathcal{B}$ can use $\mathcal{A}$ to solve the GBDH assumption.

Before the beginning of the game, $\mathcal{B}$ is given a GBDH instance $(g, g^a, g^b, g^c)$, where $a, b, c \in \mathbb{Z}_p^*$ are random choices.

**Initialization.** $\mathcal{B}$ first generates the system parameter $pp = \{1^k, G_1, G_T, e, q, g, H, h_1, h_2\}$ according to the scheme. Subsequently, $\mathcal{B}$ chooses $\ell_1, \ell_2 \leq q_H$ as the indices of the challenged identities for the DO and DU, respectively. Here, $q_H$ is the maximum number of queries that could query to the $H$-oracle for different identities. $\mathcal{B}$ also runs $(pk_{ICA}, sk_{ICA}) = \text{ICA-Setup}(pp)$ and sets $pk_{KGC} = g^a$. Furthermore, $\mathcal{B}$ initials four lists (cert-list, $H$-list, $h_1$-list, and $h_2$-list) and randomly chooses a bit $b \leftarrow \{0, 1\}$. Finally, $\mathcal{B}$ returns $(pp, pk_{ICA}, pk_{KGC})$ to $\mathcal{A}$.

**Phase 1.** In this phase, $\mathcal{A}$ is allowed to query the following oracles adaptively at polynomially many instances.

- **$h_1$-oracle:** On the $i$th nonrepeated query $ID_i \in TD$, $\mathcal{B}$ first searches the $H$-list for the entry $(i, ID_i, \mu_{ID_i}, u_{ID_i})$. If no such entry exists, then $\mathcal{B}$ executes the following under the following conditions:
  - If $i \notin \{\ell_1, \ell_2\}$: $\mathcal{B}$ randomly chooses $\mu_{ID_i} \leftarrow \mathbb{Z}_q^*$, computes $u_{ID_i} = g^{\mu_{ID_i}}$, adds $(i, ID_i, \mu_{ID_i}, u_{ID_i})$ to $H$-list, and returns $u_{ID_i}$ to $\mathcal{A}$.
  - If $i = \ell_1$: $\mathcal{B}$ sets $u_{ID_i} = g^{\ell_1}$, adds $(i, ID_i, \bot, u_{ID_i})$ to $H$-list, and returns $u_{ID_i}$ to $\mathcal{A}$.
  - If $i = \ell_2$: $\mathcal{B}$ sets $u_{ID_i} = g^{\ell_2}$, adds $(i, ID_i, \bot, u_{ID_i})$ to $H$-list, and returns $u_{ID_i}$ to $\mathcal{A}$.

- **$h_2$-oracle:** On the query $(h, c)$, $\mathcal{B}$ searches $h_2$-list for the entry $(i, h_1, h_2)$ and returns $h$. Note that we assume that $((ID_i, ID_j, k, w), h)$ is identical to $((ID_j, ID_i, k, w), h)$ for $h_1$-list. If no such entry exists, $\mathcal{B}$ runs the following steps:
  - retrieve $u_{ID_i}$ and $u_{ID_j}$ by calling $H(ID_i)$ and $H(ID_j)$, respectively.
  - check if $\mathcal{O}_{DBDH}(pk_{KGC}, u_{ID_i}, u_{ID_j}, k) = 1$.
  - if $\{i, j\} = \{\ell_1, \ell_2\}$ and the DBDH oracle returns 1, return $k$ as the answer to the GBDH problem and abort.
  - randomly choose $h \leftarrow \mathbb{Z}_q^*$, add $(ID_i, ID_j, k, w, h)$ to $h_1$-list, and return $h$ to $\mathcal{A}$.

- **$h_2$-oracle:** On the query $(h, c)$, $\mathcal{B}$ searches $h_2$-list for the entry $(i, h_1, h)$ and returns $h$. If no such entry exists, $\mathcal{B}$ randomly chooses $\hat{h} \leftarrow \mathbb{Z}_q^*$, adds $(h_1, \hat{h})$ to $h_2$-list, and returns $\hat{h}$ to $\mathcal{A}$.

- **$\mathcal{O}_{\text{cert}()}$:** when $\mathcal{A}$ queries for a certificate corresponding to $ID$, $\mathcal{B}$ first goes through $H$-list for a tuple $(*, ID, *, *)$. If no such tuple is found, $\mathcal{B}$ calls $H(ID)$ first and obtains $(i, ID, \mu_{ID}, u_{ID})$. Subsequently, $\mathcal{B}$ samples $y_{ID, 1} \leftarrow \mathbb{Z}_q^*$ and computes $u_{ID, 1} = g^{y_{ID, 1}}$, $u_{ID, 2} = u_{ID} \cdot u_{ID, 1}$, and $\sigma_{sig} \leftarrow \text{Sig}(sk_{ICA}, u_{ID, 2})$. $\mathcal{B}$ also sets $\text{cert}_{ID} = (u_{ID, 2}, \sigma_{sig})$ and $t_{fID} = y_{ID, 1}$ and adds $(\text{cert}_{ID}, t_{fID}, ID)$ to $h_1$-list. Finally, $\mathcal{B}$ returns $(\text{cert}_{ID}, t_{fID})$ to $\mathcal{A}$.

- **$\mathcal{O}_{\text{ske}()}$:** when $\mathcal{A}$ queries for a secret key with a first-round message $M_1$, $\mathcal{B}$ parses $M_1 = (u_{ID, 2}, \sigma_{sig})$. $\mathcal{B}$ returns $\perp$ if $\text{Sig}(pk_{ICA}, u_{ID, 2}, \sigma_{sig}) = 1$. Otherwise, $\mathcal{B}$ extracts $t_{fID} = y_{ID, 1}$ from $(\text{cert}_{ID} = (u_{ID, 2}, \sigma_{sig}), t_{fID} = y_{ID, 1}, ID)$ in $\text{cert}$-list. Here, we note that because $\text{Sig}$ is EU-CMA secure, if the verification passes, it necessarily exists in $\text{cert}$-list. Subsequently, $\mathcal{B}$ goes through $H$-list for the tuple $(i, ID, \mu_{ID}, \bot)$. If $\mathcal{A}$ makes $\mathcal{B}$ to output a random element $G_T$ for $i \notin \{\ell_1, \ell_2\}$, $\mathcal{B}$ computes $y_{ID, 2} = pk_{KGC}^{y_{ID, 1} + \mu_{ID}}$. Subsequently, $\mathcal{B}$ sets $M_2 = u_{ID, 2}$ and returns it to $\mathcal{A}$.

- **$\mathcal{O}_{\text{ce}()}$:** when $\mathcal{A}$ queries for a ciphertext with $(ID_1, ID_2, w)$, $\mathcal{B}$ executes the following steps:
  - retrieve $(i, ID_1, \mu_{ID_1}, u_{ID_1})$ and $(j, ID_2, \mu_{ID_2}, u_{ID_2})$ from $H$-list for $ID_1$ and $ID_2$, respectively.
  - randomly select $r \leftarrow \mathbb{Z}_q^*$.
  - if $i \notin \{\ell_1, \ell_2\}$: search $h_1$-list for the tuple $((ID_i, ID_j, \bot, w), h)$: if no such tuple exists, randomly choose $h \leftarrow \mathbb{Z}_q^*$ and add $(ID_i, ID_j, \bot, w, h)$ to $h_1$-list.
  - if otherwise (i.e., $i \notin \{\ell_1, \ell_2\}$ or $j \notin \{\ell_1, \ell_2\}$): either compute $k = \hat{e}(\text{pk}_{KGC}^{\mu_{ID_1}}, u_{ID_1})$ if $i \notin \{\ell_1, \ell_2\}$ or compute $k = \hat{e}(\text{pk}_{KGC}^{\mu_{ID_2}}, u_{ID_2})$ if $j \notin \{\ell_1, \ell_2\}$. Note that if $i \notin \{\ell_1, \ell_2\}$ and $j \notin \{\ell_1, \ell_2\}$, $\mathcal{B}$ can randomly set $k = \hat{e}(\text{pk}_{KGC}^{\mu_{ID_1}}, u_{ID_1})$ or $k = \hat{e}(\text{pk}_{KGC}^{\mu_{ID_2}}, u_{ID_2})$.
  - search $h_1$-list for the tuple $((ID_i, ID_j, \bot, w), h)$. If no such tuple exists, randomly choose $h \leftarrow \mathbb{Z}_q^*$ and add $(ID_i, ID_j, \bot, w, h)$ to $h_1$-list.
  - compute $c_1 = g^r$ and $c_2 = g^\theta$, where $\hat{h}$ is retrieved from $h_2$-list (i.e., $h = \hat{h}(c_1)$).
Theorem 2. The proposed scheme is IKGA-CS secure if the underlying signature scheme Sig is EU-CMA secure under the hard GBDH assumption.

Proof. The proof is similar to the proof of Theorem 1 except for the Challenge phase. Thus, we describe only the proof for the Challenge phase.

Challenge. At the end of Phase 1, A outputs a challenged tuple \((w_0, w_1, DO, DU)\) and B executes the following steps:

- obtain \((i, DO, ⊥, u_{DO})\) and \((j, DU, ⊥, u_{DU})\) by calling \(H(DO)\) and \(H(DU)\), respectively. If \(\{i, j\} ≠ \{\ell_1, \ell_2\}\), abort the game.
- search \(h_{11}\)-list for the tuple \(((i, DU), (j, DO), w_0, w_1, h)\). If no such tuple is found, randomly choose \(h \xleftarrow{\$} \mathbb{Z}_q^*\) and add \(((i, DU), (j, w_0, h), h)\) to \(h_{11}\)-list.
- return \(td_w = h\).

Theorem 3. The proposed scheme is MCKA-ICA and IKGA-ICA secure if it is MCKA-CS and IKGA-CS secure, respectively.

Proof. This proof is intuitive. Since malicious ICA cannot collude with the KGA, it cannot obtain any secret key information about the user, even if it has the ability to generate a potentially malicious ICA key pair. However, malicious CS has the ability to obtain the secret key information of any user except for the secret key information of the challenged identities by querying the secret key oracles. Therefore, malicious ICA can be viewed as a weaker variant of malicious CS. Consequently, Theorems 1 and 2 entail Theorem 3. Note that because \(B\) need not reply to the certificate query for \(A\), the signature scheme Sig need not be EU-CMA secure.

VII. THEORETICAL COMPARISON AND PERFORMANCE EVALUATION

Theorem 4. The proposed scheme is MCKA-KGC secure under the hard GBDH assumption.

Proof. Suppose that some PPT algorithm \(A\) can break the MCKA-KGC security of the proposed scheme. If so, then the following proof demonstrates that some other algorithm \(B\) can use \(A\) to solve the GBDH problem.

Before the beginning of the game, \(B\) is given a GBDH instance \((g, g^a, g^b, g^c)\), where \(a, b, c \in \mathbb{Z}_q^*\) are random choices.

Initialization. \(B\) first generates the system parameter \(pp = \{1^\lambda, G_1, G_T, \hat{e}, q, g, H, h_1, h_2\}\) according to the scheme. Subsequently, it sets \(Q_{key} := 1\) to count the number of \(A\) queries to \(O_{kGC}\) and initializes an empty list \(IDList\) to store the corresponding identity of the secret key returned by this oracle. \(B\) also runs \((pk_{ICA}, sk_{ICA}) \leftarrow ICA\text{-Setup}(pp)\) and \((pk_{KGC}, sk_{KGC}) \leftarrow KGC\text{-Setup}(pp)\) and chooses two arbitrary indices \(\ell_1, \ell_2 ≤ Max_{Q_{key}}\), where \(Max_{Q_{key}}\) is the maximum number of queries that can query to the \(O_{kGC}\) oracle. In addition, \(B\) sets the initials for three additional lists (\(H\)-list, \(h_1\)-list, and \(h_2\)-list) and randomly chooses a bit \(b \xleftarrow{\$} \{0, 1\}\). Finally, \(B\) returns \((pp, pk_{ICA}, pk_{KGC}, sk_{KGC})\) to \(A\).
TABLE II
FEATURES OF COMPARED SCHEMES

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Type</th>
<th>Certificateless</th>
<th>Implicit Authentication</th>
<th>No Key Escrow</th>
<th>No Key Distribution</th>
<th>No Secure Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMZKL18</td>
<td>CLAEEKS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LHSYS19</td>
<td>IBAAKS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CWZH19</td>
<td>PAEEKS</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LLZ19</td>
<td>CBAEEKS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PSE20</td>
<td>CLAEEKS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>QCH20</td>
<td>PAEEKS</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LLW21</td>
<td>CBAEEKS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ours</td>
<td>ICA-IBSE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Channel between KGC and DO/DU. Certificates in ICA-IBSE are used to generate partial secret keys only once; by contrast, certificates in PAEEKS are continually used for authentication, and those in CBAEEKS are continually used for encrypting keywords and generating trapdoors.

TABLE III
COMPUTATIONAL COST OF COMPARED SCHEMES

<table>
<thead>
<tr>
<th>Schemes</th>
<th>KeywordEnc</th>
<th>TrapdoorGen</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNZK18</td>
<td>5T_{em} + T_{htp} + 2T_{h} + 3T_{pm}</td>
<td>T_{bp} + 3T_{em} + T_{htp} + 2T_{h} + 2T_{pm}</td>
<td>2T_{bp} + 2T_{em} + T_{htp} + 2T_{h} + 2T_{pm}</td>
</tr>
<tr>
<td>LHSYS19</td>
<td>2T_{bp} + 3T_{em} + 2T_{htp}</td>
<td>T_{bp} + 2T_{em} + 2T_{htp}</td>
<td>2T_{bp} + 2T_{em} + 2T_{htp}</td>
</tr>
<tr>
<td>CWZH19</td>
<td>5T_{pm} + 2T_{pa} + T_{h}</td>
<td>5T_{pm} + 2T_{pa} + T_{h}</td>
<td>4T_{pm} + 5T_{pa}</td>
</tr>
<tr>
<td>LLZ19</td>
<td>2T_{bp} + 3T_{em} + T_{htp} + 4T_{h} + T_{pm}</td>
<td>T_{bp} + 3T_{em} + 2T_{htp} + 3T_{h} + T_{pm}</td>
<td>T_{bp} + T_{h}</td>
</tr>
<tr>
<td>PSE20</td>
<td>T_{bp} + 3T_{em} + 2T_{htp}</td>
<td>T_{bp} + T_{em} + 2T_{h} + 2T_{pm}</td>
<td>T_{bp} + 2T_{em} + T_{htp} + T_{h}</td>
</tr>
<tr>
<td>QCHLZ20</td>
<td>T_{bp} + 3T_{em} + 2T_{htp} + T_{h}</td>
<td>T_{bp} + 2T_{em} + 2T_{htp} + 2T_{h}</td>
<td>T_{bp} + T_{h}</td>
</tr>
<tr>
<td>LLW21</td>
<td>5T_{em} + 4T_{h}</td>
<td>2T_{em} + 2T_{htp} + 2T_{h}</td>
<td>T_{em} + 2T_{h}</td>
</tr>
<tr>
<td>Ours</td>
<td>T_{bp} + 2T_{em} + T_{htp} + 2T_{h}</td>
<td>T_{bp} + T_{em} + T_{htp}</td>
<td>T_{em} + T_{h}</td>
</tr>
</tbody>
</table>

Fig. 2. Comparison of the efficiency of our scheme with LHSYS19, QCHLZ20, and LLW21 schemes.

TABLE IV
COMMUNICATION COST OF COMPARED SCHEMES

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Public Key</th>
<th>Secret Key</th>
<th>Cipher text</th>
<th>Trapdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMZKL18</td>
<td>[G]_{1} + [ID]</td>
<td>[G]_{1}</td>
<td>[G]<em>{1} [G]</em>{2}</td>
<td>[G]<em>{1} [G]</em>{2}</td>
</tr>
<tr>
<td>LHSYS19</td>
<td>[ID]</td>
<td>[G]_{1}</td>
<td>[G]<em>{1} + [G]</em>{2} [2G]_{1}</td>
<td></td>
</tr>
<tr>
<td>CWZH19</td>
<td>[G]_{1}</td>
<td>[G]_{1}</td>
<td>[G]<em>{1} + [G]</em>{2}</td>
<td></td>
</tr>
<tr>
<td>LLZ19</td>
<td>2[G]_{1}</td>
<td>2[G]_{1}</td>
<td>2[G]<em>{1} + 2[G]</em>{2}</td>
<td></td>
</tr>
<tr>
<td>PSE20</td>
<td>[G]_{1} + [ID]</td>
<td>[G]<em>{1} + [G]</em>{2}</td>
<td>[G]<em>{1} + [G]</em>{2}</td>
<td></td>
</tr>
<tr>
<td>QCHLZ20</td>
<td>[G]_{1}</td>
<td>2[G]_{1}</td>
<td>[G]<em>{1} + [G]</em>{2}</td>
<td></td>
</tr>
<tr>
<td>LLW21</td>
<td>3[G]_{1} + [ID]</td>
<td>3[G]<em>{1} + 2[G]</em>{2}</td>
<td>3[G]<em>{1} + [G]</em>{2} + [G]_{3}</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>[ID]</td>
<td>[G]_{1}</td>
<td>[G]<em>{1} + [G]</em>{2}</td>
<td></td>
</tr>
</tbody>
</table>

TABLE V
EXPERIMENTAL PLATFORM

<table>
<thead>
<tr>
<th>Description</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>AMD Ryzen 5-2600 3.4GHz</td>
</tr>
<tr>
<td>CPU processor number</td>
<td>6</td>
</tr>
<tr>
<td>Operation system</td>
<td>Ubuntu 18.04</td>
</tr>
<tr>
<td>Linux kernel version</td>
<td>5.3.0-59-generic</td>
</tr>
<tr>
<td>Random access memory</td>
<td>16.3GB</td>
</tr>
<tr>
<td>Solid state disk</td>
<td>232.9GB</td>
</tr>
</tbody>
</table>

Phases 1. In this phase, $A$ is allowed to query the following oracles adaptively at polynomially many instances:
- $H$-oracle: On the $i$th nonrepeated query $ID_i$, $B$ first searches $H$-list for the entry $(i, ID_i, \mu_{ID_i}, u_{ID_i})$. If no such entry exists, then $B$ randomly chooses $\mu_{ID_i} \leftarrow \mathbb{Z}_q^*$, computes $u_{ID_i} = g^{\mu_{ID_i}}$, adds $(i, ID_i, \mu_{ID_i}, u_{ID_i})$ to $H$-
- $h_1$-oracle: On the query $(ID_i, ID_j, k, w)$, $B$ first searches $h_1$-list for the entry $((ID_i, ID_j, k, w), h)$ and returns $h$. Notably, we assume that $((ID_i, ID_j, k, w), h)$ is identical to $((ID_j, ID_i, k, w), h)$ for $h_1$-list. If no such entry exists, $B$ executes the following steps:
  - retrieve $u_{ID_i}$ and $u_{ID_j}$ by calling $H(ID_i)$ and...
\( H(ID_j) \), respectively.
- check if \( O_{\text{DBDH}}(pk_{KGC}, u_{ID_j}, u_{ID_j}, k) = 1 \).
- if \( \{i, j\} = \{\ell_1, \ell_2\} \) and the DBDH oracle returns 1, return \( k \) as the answer to the DBDH problem and abort.
- randomly choose \( h \leftarrow Z_q^* \), add \((ID_1, ID_j, k, w), h)\) to \( h_1\)-list, and return \( h \) to \( A \).
- \( h_2\)-oracle: On the query \((h, c_1)\), \( B \) searches \( h_2\)-list for the entry \(((h, c_1), h)\) and returns \( h \). If no such entry exists, then \( B \) randomly chooses \( \tilde{h} \leftarrow Z_q^* \), adds \(((h, c_1), \tilde{h})\) to \( h_2\)-list, and returns \( h \) to \( A \).
- \( O_{\text{ikKGC}}() \): When \( A \) issues a issue key query, \( B \) first samples \( ID \leftarrow TD \) such that \( ID \) does not exist in \( h\)-list. Subsequently, \( B \) executes the following steps under the following conditions:
  - if \( Q_{\text{key}} \not\in \{\ell_1, \ell_2\} \): \( B \) obtains \( u_{ID} \) by querying \( H(ID) \).
  - if \( Q_{\text{key}} = \ell_1 \): \( B \) sets \( u_{ID} = g^a \) and adds \((ID, \perp, u_{ID})\) to \( h\)-list.
  - if \( Q_{\text{key}} = \ell_2 \): \( B \) sets \( u_{ID} = g^b \) and adds \((ID, \perp, u_{ID})\) to \( h\)-list.

In addition, \( B \) randomly chooses \( u_{ID,1} \leftarrow Z_q^* \), computes \( u_{ID,1} = g^{h_{1,1}} \), and computes \( u_{ID,2} \in G \) as \( u_{ID,2} = u_B \cdot u_{ID,1} \) and \( \sigma_{\text{Sig}} \leftarrow \text{Sig.Sign}(sk_{ICA}, u_{ID,2}) \). Finally, \( B \) returns \( M_1 = (u_{ID,2}, \sigma_{\text{Sig}}) \) to \( A \), sets \( IDList[Q_{\text{key}}] = ID \), and updates \( Q_{\text{key}} = Q_{\text{key}} + 1 \).
- \( O_{\text{ct}}() \): When \( A \) queries for a ciphertext with \((i, j, w)\), where \( i, j \in \{Q_{\text{key}}\} \), \( B \) executes the following steps:
  - retrieve \((IDList[i], \mu_{IDList[i]}, u_{IDList[i]})\) and \((IDList[j], \mu_{IDList[j]}, u_{IDList[j]})\) from \( h\)-list for \( IDList[i] \) and \( IDList[j] \), respectively.
  - randomly choose \( r \leftarrow Z_q^* \).
  - if \( \{i, j\} = \{\ell_1, \ell_2\} \): search \( h\)-list for the tuple \(((IDList[i], IDList[j]), \perp, w)\). If no such tuple exists, randomly choose \( h \leftarrow Z_q^* \) and add \(((IDList[i], IDList[j]), h)\) to \( h\)-list.
  - if otherwise (i.e., \( i \not\in \{\ell_1, \ell_2\} \) or \( j \not\in \{\ell_1, \ell_2\} \)), either compute \( k = \hat{e}(pk_{KGC}, u_{IDList[i]}) \) if \( i \notin \{\ell_1, \ell_2\} \) or compute \( k = \hat{e}(u_{IDList[i]}, pk_{KGC}) \) if \( j \notin \{\ell_1, \ell_2\} \). Note that if \( i \notin \{\ell_1, \ell_2\} \) and \( j \notin \{\ell_1, \ell_2\} \), \( B \) can randomly set \( k = \hat{e}(pk_{KGC}, u_{IDList[i]}) \) or \( k = \hat{e}(u_{IDList[i]}, pk_{KGC}) \).
  - search \( h\)-list for the tuple \(((IDList[i], IDList[j]), k, w)\). If no such tuple exists, randomly choose \( h \leftarrow Z_q^* \) and add \(((IDList[i], IDList[j]), h)\) to \( h\)-list.
  - compute \( c_1 = g^a \) and \( c_2 = g^b \), where \( h \) is retrieved from \( h\)-list (i.e., \( h_2(h, c_1) \)).
  - return \( ct_w = (c_1, c_2) \).

**Challenge.** At the end of Phase 1, \( A \) outputs the challenged tuple \((w_0 = w_{0,1}, \ldots, w_{0,n}), (w_1 = w_{1,1}, \ldots, w_{1,n})\), \( \alpha, \beta \), and \( B \) executes the following steps:
- obtain \((IDList[\alpha], \perp, u_{IDList[\alpha]})\) and \((IDList[\beta], \perp, u_{IDList[\beta]})\) by calling \( H(IDList[\alpha]) \) and \( H(IDList[\beta]) \), respectively. If \( \{\alpha, \beta\} \neq \{\ell_1, \ell_2\} \), abort the game.
- for \( i = 1, \ldots, n \), perform the following steps. First, randomly choose \( r_i \leftarrow Z_q^* \). Second, search \( h\)-list for the tuple \(((IDList[\alpha], IDList[\beta]), \perp, w_{hi})\), \( h_1 \). If no such tuple is found, randomly choose \( h_{1} \leftarrow Z_q^* \) and add \(((IDList[\alpha], IDList[\beta]), \perp, w_{hi}) \) to \( h\)-list. Third, compute \( ct^*_i = (c^*_i, c^*_j) \), where \( c^*_i = g^{r_i}, c^*_j = g^{r_i h_i} \), where \( h_i \) is retrieved from \( h\)-list (i.e., \( h_i = h_2(h, c^*_i) \)).
- return the challenged ciphertext \( ct^* = (ct^*_1, \ldots, ct^*_n) \).

**Phase 2.** In this phase, \( A \) can keep the query oracles identical to those in Phase 1.

**Guess.** Finally, \( A \) outputs \( b' \in \{0, 1\} \) as its guess. \( B \) searches \( h\)-list for \( k^* \) such that \( O_{\text{DBDH}}(g^a, g^b, g^{k^*}, (k^*)^{−1}) = 1 \) and returns \( k^* \) as the answer.

**Analysis.** Because \( B \) follows the proposed scheme, except that the hash functions are modeled by random oracles, its simulation is identical to that of the real scheme. Because \( \ell_1 \) and \( \ell_2 \) are independent of \( A \)’s perspective, the probability that \( B \) does not abort the game \( \{\alpha, \beta\} = \{IDList[\ell_1], IDList[\ell_2]\} \) in **Challenge** is \( \frac{1}{2} \). Furthermore, because \( h_1 \) is modeled as a random oracle, the adversary’s advantage is negligible, unless \((IDList[\alpha], IDList[\beta], k^*, w_{hi}, h)\) appears in \( h\)-list.

**TABLE VI**

<table>
<thead>
<tr>
<th>Bit-length (bit)</th>
<th>Running time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z_q^* )</td>
<td>( G_1 )</td>
</tr>
<tr>
<td>160</td>
<td>512</td>
</tr>
</tbody>
</table>
such that $(k^*)^{-1} = e(g, g)^{\text{hbrc}(x^{-1})} = e(g, g)^{\text{abc}}$. If this tuple appears in $h_1$-list, then $B$ is necessarily able to solve the GBDH problem. Therefore, if there exists such an $A$ that can break the MCKA-KGC-secure scheme with a nonnegligible advantage $\epsilon$, then there exists some $B$ that can break the GBDH problem with a nonnegligible advantage $\epsilon' \geq \epsilon$. \qed

**Theorem 5.** The proposed scheme is IKGA-KGC secure under the hard GBDH assumption.

**Proof.** The proof is similar to the proof of Theorem 4, except for the Challenge phase. Therefore, only the proof for the Challenge phase is presented.

**Challenge.** At the end of Phase 1, $A$ outputs the challenged tuple $(u_0, w_1, \alpha, \beta)$ and $B$ executes the following steps:

- obtain $(\text{IDList}[\alpha], \bot, u_{\text{IDList}[\alpha]})$ and $(\text{IDList}[\beta], \bot, u_{\text{IDList}[\beta]})$ by calling $H(\text{IDList}[\alpha])$ and $H(\text{IDList}[\beta])$, respectively. If $\{\alpha, \beta\} \neq \{\epsilon_1, \epsilon_2\}$, $B$ aborts the game.
- search $h_1$-list for the tuple $((\text{IDList}[\alpha], \text{IDList}[\beta], \bot, w_0, h)$. If no such tuple is found, randomly choose $h \in \mathbb{Z}_q^*$ and add $((\text{IDList}[\alpha], \text{IDList}[\beta], \bot, w_0, h)$ to $h_1$-list.
- return the challenged trapdoor $td^* = h$.

The result of the cost of each operation and space required by each element is listed in Table VI. In addition, the evaluation results of the concrete implementations are presented in Fig. 2 which shows that while our scheme is slower than QCHLZ20 scheme in performing test, overall comparisons, our scheme can effectively encrypt keywords, generate trapdoors and perform tests. In addition, although LLW21 scheme is a pairing-free scheme and is faster than our scheme in theoretical comparison, it needs to use a lot of hash functions, resulting in the need to constantly switch between different types and big type in MIRACL library and char type in C language back and forth in practice, thus increasing execution time.

VIII. Conclusion

In this paper, we present a novel ICA-IBSE scheme that masters the trade-off between efficiency (in terms of low storage requirement) and convenience (which IBAEKS achieves by avoiding the key escrow problem). A concrete framework is presented, and security proofs are provided, which demonstrate that the ICA-IBSE scheme can resist MCKAs and IKGAs under random oracles. Moreover, we experimentally verify that our scheme not only reduces storage requirements but is also practical relative to its state-of-the-art counterparts.

**References**


https://github.com/brainhub/SHA3IUF

https://crypto.stanford.edu/pbc/

https://github.com/miracle/MIRACL