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 attacks, 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 attacks launched by a malicious cloud server (CS) as part of a so-called insider keyword guessing attacks (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].
Our contributions are summarized as follows:

- To master the trade-off between efficiency and security, on the basis of Chow et al. \[29\] and Emura et al. \[30\], 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 as well as 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 \[24\], \[25\], \[30\], \[31\], \[32\], \[33\], \[34\], \[35\]. 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.

<table>
<thead>
<tr>
<th>Notations</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>Security parameter</td>
</tr>
<tr>
<td>$pp$</td>
<td>Public parameter</td>
</tr>
<tr>
<td>$ID, W, M$</td>
<td>Identity space, Keyword space, Message space</td>
</tr>
<tr>
<td>$q$</td>
<td>A big prime number</td>
</tr>
<tr>
<td>${a, b}$</td>
<td>Set ${a, a + 1, \cdots, b}$ (omit $a$ if $a = 1$)</td>
</tr>
<tr>
<td>$\mathbb{Z}_q$</td>
<td>Integers in $[1, q - 1]$ which are relatively prime to $q$</td>
</tr>
<tr>
<td>$\mathbb{G}_1, \mathbb{G}_T$</td>
<td>Bilinear groups with order $q$</td>
</tr>
<tr>
<td>$\mathbb{G}_{ec}$</td>
<td>Elliptic curve group with order $q$</td>
</tr>
<tr>
<td>$g$</td>
<td>A generator of $\mathbb{G}_1$</td>
</tr>
<tr>
<td>$\hat{e}$</td>
<td>Bilinear pairing</td>
</tr>
<tr>
<td>$H, h_1, h_2$</td>
<td>Cryptographic hash functions</td>
</tr>
<tr>
<td>IKGA</td>
<td>Insider keyword guessing attacks</td>
</tr>
<tr>
<td>MCKA</td>
<td>Multichosen keyword attacks</td>
</tr>
<tr>
<td>ICA, KGC</td>
<td>Identity-certifying authority, Key generation center</td>
</tr>
<tr>
<td>CS, DO, DU</td>
<td>Cloud server, Data owner, Data user</td>
</tr>
<tr>
<td>$ID$, $DO$, $DU$</td>
<td>The identity of ID, DO, and DU, respectively</td>
</tr>
<tr>
<td>$pk_{ICA}$, $sk_{ICA}$</td>
<td>The public key and secret key of ICA</td>
</tr>
<tr>
<td>$pk_{KGC}$, $sk_{KGC}$</td>
<td>The public key and secret key of KGC</td>
</tr>
<tr>
<td>$sk_{ID}$</td>
<td>The secret key of identity ID</td>
</tr>
<tr>
<td>$cert_{ID}, tf_{ID}$</td>
<td>The certificate and trapdoor of identity ID</td>
</tr>
<tr>
<td>$w$</td>
<td>A keyword</td>
</tr>
<tr>
<td>$ct_w$</td>
<td>A ciphertext associated with keyword $w$</td>
</tr>
<tr>
<td>$td_w$</td>
<td>A trapdoor associated with keyword $w$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>A null symbol</td>
</tr>
<tr>
<td>$A, B$</td>
<td>The polynomial time adversary and challenger</td>
</tr>
<tr>
<td>$A^{\mathcal{C}_1}(x)$</td>
<td>$A$ with input $x$ can access the black-box oracle $O$</td>
</tr>
<tr>
<td>$y = F(x)$</td>
<td>Storing the output of an algorithm $F$ with input $x$ to the variable $y$</td>
</tr>
<tr>
<td>$s \leftarrow S$</td>
<td>Sampling an element $s$ from set $S$ uniformly at random</td>
</tr>
<tr>
<td>$</td>
<td>K</td>
</tr>
<tr>
<td>$\negl(\lambda)$</td>
<td>An arbitrary function $f$ is negligible in $\lambda$, where $f(\lambda) = o(\lambda^{-c})$ for every fixed constant $c$</td>
</tr>
<tr>
<td>$\text{Sig}$</td>
<td>Digital signature scheme</td>
</tr>
<tr>
<td>$vk_{\text{Sig}}, sk_{\text{Sig}}$</td>
<td>Verification key and Signing key of Sig</td>
</tr>
<tr>
<td>$m, \sigma_{\text{Sig}}$</td>
<td>Message, Signature</td>
</tr>
<tr>
<td>$Q_{\text{key}}$</td>
<td>The amount of instances obtained by querying Issue Key KGC Oracle</td>
</tr>
<tr>
<td>$IDList$</td>
<td>Storing the corresponding identity of the secret key that is returned by Issue Key KGC Oracle</td>
</tr>
<tr>
<td>$(*)$</td>
<td>Any element in a tuple</td>
</tr>
</tbody>
</table>
B. Digital Signature

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

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

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

$\text{Sig.Verify}(vk_{\text{Sig}}, m, \sigma_{\text{Sig}})$: The verification key $vk_{\text{Sig}}$, a message $m \in \mathcal{M}$, and a signature $\sigma_{\text{Sig}}$ are taken as the inputs, and 1 is the output if the signature is valid; otherwise, 0 is the output.

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

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 $\hat{\cdot} : G_1 \times G_1 \rightarrow G_T$ be a bilinear pairing (a map). We consider that the tuple $(g, g, G_1, G_T, \hat{\cdot})$ is a symmetric bilinear group if the following properties are satisfied:

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

D. Gap Bilinear Diffie–Hellman Assumption [39], [40]

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

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

where $O_{\text{BDH}}$ denotes the maximum number of queries by $A$ to $O_{\text{BDH}}$, and $O_{\text{BDH}}$ denotes a decision bilinear Diffie–Hellman oracle that takes $(g^a, g^b, g^c, T)$ as its input and outputs 1 if $\hat{\cdot}(g^a, g^b, g^c) = T$ and 0 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 identities.
- 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 PAES schemes, because DO and DU know the secret information (i.e., keywords), 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
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 key 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 as inputs, and outputs a certificate certID and a trapdoor information tfID, which are sent to the user 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 M₁ ← User-Obtain-Key(pp, pkKGC, ID, certID, tfID) and submits it to the KGC. Subsequently, the KGC returns a second-round message M₂ ← KGC-Issue-Key(pp, pkKGC, skKGC, pkICA) to the user. At the end of the protocol, the user can locally output a secret key skID or ⊥.

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

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

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

Definition 1 (Correctness and Consistency of ICA-IBSE). For all security parameters λ ∈ N, all DOs DO ∈ ID, all DUs DU ∈ TD, and all keywords w, w′ ∈ W, ICA-IBSE is defined to be correct if, when w = w′, we have Pr[Test(pp, ctw, tdw) = 1] = 1 and ICA-IBSE is defined to be consistent if, when w ≠ w′, we have Pr[Test(pp, ctw, tdw) = 0] = 1 − negl(λ), where pp ← Setup(1^λ); (pkICA, skICA) ← ICA-Setup(pp); (pkKGC, skKGC) ← KGC-Setup(pp); (certi, id) ← ICA-Cert(pp, pkICA, skICA, i); and sk_i ← (User-Obtain-Key(pp, pkKGC, i, certi, td_i), KGC-Issue-Key(pp, pkKGC, skKGC, pkICA)), for i ∈ {DO, DU}, ctw = Encrypt(pp, skDO, DU, w), and tdw ← Trapdoor(pp, DO, skDU, w′).

B. Security Models

To model the different aspects of the attacks described in Section [41], we revise the security model in [25] and [38] to account for multichosen keyword attacks (MCKA) and IKGA in the ICA-IBSE framework. Here, MCKA, recently introduced by Qin et al. [25], ensures that no adversary can obtain any information on keywords from two tuples of encrypted keywords; IKGA ensures 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 attacks from CS (i.e., MCKA-CS and IKGA-CS), but further define additionally security games to model...
the attacks from ICA and KGC (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 KGC 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 also allowed to query user’s partial secret key (i.e., $M_2$) 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., $M_1$) 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 $B$ for the adversary $A$:

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

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

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

Trapdoor Oracle $O_{td}$: For any keyword $w \in \mathcal{W}$, $B$ outputs $td_w \leftarrow$ Trapdoor$(pp, DO, sk_{DU}, w)$ to $A$.

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

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

Trapdoor KGC Oracle $O_{td}$: For any keyword $w \in \mathcal{W}$, $DO$ index $doi$, and $DU$ index $dai$, $B$ first checks whether $doi \in [Q_{key}]$ and $dai \in [Q_{key}]$. If not, $B$ forces $A$ to output a random bit $b' \in \{0, 1\}$. Otherwise, $B$ retrieves the $doi$-th identity’s secret key $ID$List$[doi]$ and $dai$-th identity’s secret key $sk_{\mathcal{ID}List[doi]}$ and subsequently outputs $td_w \leftarrow$ Trapdoor$(pp, ID$List$[doi], sk_{\mathcal{ID}List[doi]}, w)$ 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 the challenged identities $DO, DU \in \mathcal{ID}$. Furthermore, the adversary cannot issue queries to oracles $O_{ct}$ on the challenged keywords $w_{b,i} \in \mathcal{W}$ and cannot issue queries to oracles $O_{td}$ on the challenged keywords $w_{b} \in \mathcal{W}$ for the challenged identities $DO, DU \in \mathcal{ID}$, where $b \in \{0, 1\}$ and $i \in \{1, \ldots, n\}$. We consider that ICA-IBSE is MCKA-CS secure if the advantage

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

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

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

is negligible for any $A$.
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 Issue Key KGC Oracle $O_{ik,kgc}$, and use IDList 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 $IDList[\alpha]$ and $IDList[\beta]$ from $IDList$ as the DO and DU, respectively. Furthermore, the adversary cannot issue queries to $O_{ck,kgc}$ on challenged keywords $w_{b,i} \in W$ and to querying queries to $O_{ck,kgc}$ on challenged keywords $w_{b} \in W$ for the challenged indices $(\alpha, \beta)$, where $b \in \{0,1\}$ and $i \in \{1, \ldots , n\}$.

We consider that ICA-IBSE is MCKA-KGC secure if the advantage

$$Adv_{MCKA-KGC}^{ICA-IBSE}(\lambda) := |Pr[b = b'] - 1/2|$$

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

$$Adv_{IKGA-KGC}^{ICA-IBSE}(\lambda) := |Pr[b = b'] - 1/2|$$

is negligible for any $\mathcal{A}$.

**MCKA-KGC / IKGA-KGC Game**

\[
\begin{align*}
pp & \leftarrow \text{Setup}(1^{\lambda}); \text{IDList} = \emptyset; Q_{key} := 1; b \leftarrow \{0,1\}; \\
(pp,\text{sk},sk) & \leftarrow \text{ICA-Setup}(pp); (pk,kgc,sk) \leftarrow \text{KGC-Setup}(pp); \\
& (w_0, (w_{0,1, \ldots , w_{0,n}}), w_1, (w_{1,1, \ldots , w_{1,n}}), \alpha, \beta) \leftarrow \\
& A^{\text{sk}(\text{sk})} \times A^{\text{sk}(\text{sk})}; \\
& ct_1^* = (ct_1^*, \ldots , ct_n^*), \\
& \text{where } ct_i^* = \text{Encrypt}(pp, sk, ID\text{List}[i], w_{a,i}), \\
& \text{for } i = 1, \ldots , n; \\
& y' \leftarrow A^{\text{sk}(\text{sk})} \times A^{\text{sk}(\text{sk})}; \\
& t_{d}^* \leftarrow \text{Trapdoor}(pp, ID\text{List}[i], sk, ID\text{List}[i], w_{a,i}); \\
& (w_0, w_1, \alpha, \beta) \leftarrow A^{\text{sk}(\text{sk})} \times A^{\text{sk}(\text{sk})}; \\
& (pk, kgc, sk, pp, sk) \leftarrow A^{\text{sk}(\text{sk})} \times A^{\text{sk}(\text{sk})} \\
& t_{d}^* \leftarrow \text{Trapdoor}(pp, ID\text{List}[i], sk, ID\text{List}[i], w_{a,i}); \\
& y' \leftarrow A^{\text{sk}(\text{sk})} \times A^{\text{sk}(\text{sk})} \times A^{\text{sk}(\text{sk})}. 
\end{align*}
\]

**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 $\text{Sig} : (\text{Sig.KeyGen}, \text{Sig.KeyGen}, \text{Sig.Verify})$ 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 $\hat{e} : G_1 \times G_1 \rightarrow G_T$; 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 $pp := \{1^\lambda, G_1, G_T, \hat{e}, q, g, H, h_1, h_2\}$.

**ICA-Setup$(pp)$**: The ICA runs $(\nu_{\text{sk}}, \nu_{\text{sk}}) \leftarrow \text{Sig.KeyGen}(1^{\lambda})$. It then outputs $pk_{\text{ICA}} := \nu_{\text{sk}}; sk_{\text{ICA}} := \nu_{\text{sk}}$.

**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 $u_{ID,2} := u_{ID,1} \cdot \nu_{\text{sk}}$ and $\sigma_{\text{Sig}} \leftarrow \text{Sig.Sign}(sk_{\text{ID},2})$. Finally, it outputs $cert_{ID} := (u_{ID,2}; \sigma_{\text{Sig}}); f_{ID} := y_{ID,1}$.

**User-Obtain-Key$(pp, pk_{\text{KGC}}, ID, cert_{ID}, f_{ID})$**: The user and the KGC run the following steps:

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

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

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

**Encrypt$(pp, sk_{DO}, DU, w)$**: The DO randomly selects $r \leftarrow Z_q^*$ and computes $c_1 = g^r, c_2 = g^{h_2(h_1(DO,DU,k,w); e_1)}$, where $k = e(sk_{DO}, H(DU))$. Subsequently, it outputs $ct_{w} := (c_1, c_2)$.

**Trapdoor$(pp, DO, sk_{DO}, w)$**: The DU outputs $td_{w} := h_1(DO,DU, k, w)$, where $k = e(H(DO), sk_{DU})$.

**Test$(pp, ct_{w}, td_{w})$**: The CS checks whether $c_2 = c_1^{h_2(t_{d}, e_1)}$.

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 $i = \{DO, DU\}$, we have

$$sk_i = y_{i,2} \cdot Y^{-y_{i,1}}$$

$$= u_{i,2}^r \cdot Y^{-y_{i,1}}$$

$$= u_{i,2}^r \cdot u_{i,1}^{-1} \cdot Y^{-y_{i,1}}$$

$$= H(i)^r \cdot Y^{-y_{i,1}} \cdot Y^{-y_{i,1}}$$

$$= H(i)^r.$$
2) Considering the Test algorithm for a ciphertext \( ct_w = (c_1, c_2) \) and a trapdoor \( td_{w'} \), we have
\[
g^{rh_2(h_1(\delta, DU, k, w), c_1)} = c_2 = g^{h_2(td_{w'}, c_1)} = g^{h_2(h_1(DU, k, w'), c_1)}
\]
In addition, we have
\[
k = \epsilon(sk_{DO}, H(DU)) = \epsilon(H(DO)^x, H(DU)) = \epsilon(H(DO), H(DU)^x) = \epsilon(H(DO), sk_{DU}) = k'
\]
Because \( k = k' \), when \( w = w' \), we have \( h_1(DU, k, w) = h_1(DU, k', w') \) and \( c_2 \neq c_1 \); therefore, correctness is satisfied. Conversely, when \( w \neq w' \), because the probability that \( h_1(DU, k, w) = h_1(DU, k', w') \) is negligibly low, we have \( c_2 \neq c_1 \); 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 \( \Sigma \) is EU-CMA secure under the hard GBDH assumption.

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

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 \( pp = \{\lambda, G_1, \mathcal{G}, \epsilon, q, g, H, h, h_1, h_2\} \) according to the scheme. Subsequently, \( B \) chooses \( \epsilon_1, \epsilon_2 \leq q_H \) 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. \( B \) also runs \((pk_{ICA}, sk_{ICA}) = \text{ICA-Setup}(pp)\) and sets \( pk_{KG} = g^a \). Furthermore, \( B \) initializes four lists (cert-list, H-list, H-list, and h2-list) and randomly chooses a bit \( \epsilon \in \{0, 1\} \). Finally, \( B \) returns \((pp, pk_{ICA}, pk_{KG})\) to \( A \).

**Phase 1.** In this phase, \( A \) is allowed to query the following oracles adaptively at polynomially many instances.

- **\( h_1 \)-oracle:** On the \( i \)-th query \( ID_i \in TD \), \( B \) first searches the \( H \)-list for the entry \((i, ID_i, u_{ID_i}, u_{ID_i})\). If no such entry exists, then \( B \) executes the following steps under the following conditions:
  - If \( i \notin \{\epsilon_1, \epsilon_2\} \): \( B \) randomly chooses \( u_{ID_i} \in \mathbb{Z}_q^* \) and computes \( u_{ID_i} = g^{h_1(ID_i)} \) and adds \((i, ID_i, u_{ID_i}, u_{ID_i})\) to the H-list, and returns \( u_{ID_i} \) to \( A \).
  - If \( i = \epsilon_1 \): \( B \) sets \( u_{ID_i} = g^h \) and adds \((i, ID_i, u_{ID_i}, u_{ID_i})\) to the H-list, and returns \( u_{ID_i} \) to \( A \).
  - If \( i = \epsilon_2 \): \( B \) sets \( u_{ID_i} = g^h \) and adds \((i, ID_i, u_{ID_i}, u_{ID_i})\) to the H-list, and returns \( u_{ID_i} \) to \( A \).

- **\( h_2 \)-oracle:** On the query \((h, c_1, c_2)\), \( B \) searches \( h_2 \)-list for the entry \((h, c_1, c_2)\) and returns \( h \). Note that in this case, \((h, c_1, c_2) \neq (h, c_1, c_2) \) and \((h, c_1, c_2) \neq (h, c_1, c_2) \) for \( h_1 \)-list. If no such entry exists, \( B \) runs the following steps:
  - retrieve \( u_{ID_1} \) and \( u_{ID_1} \) by calling \( H(ID_1) \) and \( H(ID_1) \), respectively.
  - check if \( O_{DBDH}(pk_{KG}, u_{ID_1}, u_{ID_1}, k) = 1 \).
  - if \( \{i, j\} = \{\epsilon_1, \epsilon_2\} \) and the GBDH oracle returns \( k \), return \( k \) as the answer to the GBDH problem and abort.
  - randomly choose \( h \in \mathbb{Z}_q^* \), add \((ID_i, ID_j, k, w, h)\) to the \( h_1 \)-list, and return \( h \) to \( A \).

- **\( \Sigma_{cert}() \):** when \( A \) queries for a certificate corresponding to \( ID \), \( B \) first goes through the list for a tuple \((*, ID, *, *)\). If no such tuple is found, \( B \) calls \( H(ID) \) first and obtains \( (i, ID, u_{ID}, u_{ID}) \). Subsequently, \( B \) samples \( y_{ID,1} \in \mathbb{Z}_q \) and computes \( u_{ID,1} = g^{y_{ID,1} + \mu_{ID}} \) and \( u_{ID,2} = g^{y_{ID,2} + \mu_{ID}} \), and \( \sigma_{SIG} = \Sigma_{SIG}(sk_{ICA}, u_{ID,2}, ID) \). \( B \) also sets \( cert_{ID} = (u_{ID,2}, \sigma_{SIG}) \) and adds \((cert_{ID}, tf_{ID}, ID)\) to the cert-list. Finally, \( B \) returns \((cert_{ID}, tf_{ID})\) to \( A \).

- **\( \sigma_{sk}() \):** when \( A \) queries for a secret key with a first-round message \( M_1 \), \( B \) parses \( M_1 = (u_{ID,2}, \sigma_{SIG}) \). Then, \( B \) aborts the game and outputs a random element of \( \mathcal{G} \).

- **\( \Sigma_{cert}() \):** when \( A \) queries for a ciphertext \((ID_i, ID_j, w)\), \( B \) executes the following steps:
  - retrieves \((ID_i, u_{ID,1}, u_{ID}) \) and \((j, ID_j, u_{ID,1}, u_{ID}) \) from the list for \( ID_i \) and \( ID_j \), respectively.
  - randomly select \( r \in \mathbb{Z}_q^* \).
  - if \( \{i, j\} = \{\epsilon_1, \epsilon_2\} \): search \( h_1 \)-list for the tuple \((ID_i, ID_j, w, h)\); if no such tuple exists, randomly choose \( h \in \mathbb{Z}_q \) and add \((ID_i, ID_j, w, h)\) to the \( h_1 \)-list.
  - otherwise (i.e., \( i \in \{\epsilon_1, \epsilon_2\} \) or \( j \notin \{\epsilon_1, \epsilon_2\} \)): either compute \( k = \epsilon(pk_{KG}, u_{ID,1}) \) if \( i \notin \{\epsilon_1, \epsilon_2\} \) or compute \( k = \epsilon(u_{ID,1}, pk_{KG}) \) if \( j \notin \{\epsilon_1, \epsilon_2\} \). Note that if \( i \notin \{\epsilon_1, \epsilon_2\} \) and \( j \notin \{\epsilon_1, \epsilon_2\} \), \( B \) can randomly set \( k = \epsilon(pk_{KG}, u_{ID,1}) \) or \( k = \epsilon(u_{ID,1}, pk_{KG}) \).
  - search \( h_1 \)-list for the tuple \((ID_i, ID_j, k, w)\). If no such tuple exists, randomly choose \( h \in \mathbb{Z}_q \) and add \((ID_i, ID_j, k, w, h)\) to \( h_1 \)-list.
  - compute \( c_1 = g^h \) and \( c_2 = g^h \), where \( h \) is retrieved from \( h_2 \)-list (i.e., \( h = h_2(h, c_1) \)).
- return $ct_w = (c_1, c_2)$.

- $O_{\text{td}}()$: when $A$ queries for a trapdoor with $(ID_i, ID_j, w)$, $B$ executes the following steps:
  - retrieve $(i, ID_i, \mu ID_i, u ID_i)$ and $(j, ID_j, \mu ID_j, u ID_j)$ from $H$-list for $ID_i$ and $ID_j$, respectively.
  - randomly select $r \leftarrow \mathbb{Z}_q^*$.
  - if $\{i, j\} = \{\ell_1, \ell_2\}$: search $h$-list for the tuple $((ID_i, ID_j, \mu, w), h)$. If no such tuple exists, randomly choose $h \leftarrow \mathbb{Z}_q^*$ and add $((ID_i, ID_j, \mu, w), h)$ to $h$-list.
  - otherwise (i.e., $i \notin \{\ell_1, \ell_2\}$ or $j \notin \{\ell_1, \ell_2\}$), either compute $k = \hat{e}(pk_{ID_i, ID_j})$ if $i \notin \{\ell_1, \ell_2\}$ or compute $k = \hat{e}(u_{ID_j}, pk_{ID_i})$ 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 $k = \hat{e}(pk_{ID_i, ID_j})$ or $k = \hat{e}(u_{ID_j}, pk_{ID_i})$.
  - search $h$-list for the tuple $((ID_i, ID_j, \mu, w), h)$. If no such tuple exists, randomly choose $h \leftarrow \mathbb{Z}_q^*$ and add $((ID_i, ID_j, \mu, w), h)$ to $h$-list.
  - return $td_w = h$.

**Challenge.** At the end of **Phase 1**, $A$ outputs the challenged tuple $(\tilde{w}_0 = \{w_{0,1}, \ldots, w_{0,n}\}, \tilde{w}_1 = \{w_{1,1}, \ldots, w_{1,n}\}, DO, DU)$ and $B$ executes the following steps:

- obtain $(i, DO, \mu, u_{DO})$ and $(j, DU, \mu, u_{DU})$ by calling $H(DO)$ and $H(DU)$, respectively. If $\{i, j\} \neq \{\ell_1, \ell_2\}$, abort the game.
- for $i = 1, \ldots, n$, execute the following steps. First, randomly choose $r_i \leftarrow \mathbb{Z}_q^*$. Second, search $h$-list for the tuple $((DO, DU, \mu, w_{ib}), h_i)$; if no such entry is found, randomly choose $h_i \leftarrow \mathbb{Z}_q^*$ and add $((DO, DU, \mu, w_{ib}), h_i)$ to $h$-list. Third, compute $ct^*_i = (c^*_{1,i}, c^*_{2,i})$, where $c^*_{1,i} = g^{r_i h_i}$, $c^*_{2,i} = g^{r_i h_i}$, where $h_i$ is retrieved from $h$-list (i.e., $h_i = h_2(h_i, h_1(i))$).
- return challenged ciphertext $ct^* = (ct^*_{1,1}, \ldots, ct^*_{n,n})$.

**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 $O_{\text{DBDH}}(g^b, g^b, g^b, k^*) = 1$ and returns $k^*$ as answer.

**Analysis.** Because $B$ follows the proposed scheme, with the exception 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 (i.e., $\{i, j\} \notin \{\ell_1, \ell_2\}$ in querying $h$-oracle and $\{DO, DU\} = \{ID_1, ID_2\}$) in **Challenge** is $\frac{3}{\ell_1}$, where $q_H$ is the maximum number of queries that could be made to the $H$ oracle for different inputs. Furthermore, because $h_1$ is modeled as a random oracle, $A$’s advantage is negligible unless $((DO, DU, \mu, w_{ib}, h))$ appears on $h$-list such that $k^* = \hat{e}(g, g)^{abc}$. If this tuple appears on $h$-list, then $B$ is necessarily able to solve the GBDH problem. Therefore, if there exists some $A$ that can break the MCKA-CS-secure scheme with a negligible advantage $\epsilon$, then there exists some $B$ that can break the GBDH problem with a negligible advantage $\epsilon' \geq \epsilon \cdot \frac{3}{q_H}$.

**Theorem 2.** The proposed scheme is IKGA-CS secure if the underlying signature scheme $\text{Sig}$ 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 $(i, DO, \mu, u_{DO})$ and $(j, DU, \mu, u_{DU})$ by calling $H(DO)$ and $H(DU)$, respectively. If $\{i, j\} \neq \{\ell_1, \ell_2\}$, abort the game.
- search $h$-list for the tuple $((DO, DU, \mu, w_i), h)$. If no such tuple is found, randomly choose $h \leftarrow \mathbb{Z}_q^*$ and add $((DO, DU, \mu, w_i), h)$ to $h$-list.
- return challenged trapdoor $td^* = h$.

**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 collide with the KGC, 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, Theorem 1 and Theorem 2 entail Theorem 3. Note that because $B$ does not need to reply to the certificate query for $A$, the signature scheme $\text{Sig}$ does not need to be EU-CMA secure.

**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 $pp = \{\lambda, G_1, \mathbb{G}_T, \hat{e}, q, q_H, h_1, h_2\}$ according to the scheme. Subsequently, it sets $Q_{\text{key}} := 1$ to count the number of $A$ queries to $O_{\text{KGC}}$ and initial of an empty list $IDList$ to store the corresponding identity of the secret key returned by this oracle. $B$ also runs $(pk_{ICA, sk_{ICA}}) \leftarrow \text{ICA-Setup}(pp)$ and $(pk_{KGC, sk_{KGC}}) \leftarrow \text{KGC-Setup}(pp)$ and chooses two arbitrary indices $\ell_1, \ell_2 \leq \text{Max}_{\text{Q}_{\text{key}}}$, where $\text{Max}_{\text{Q}_{\text{key}}}$ is the maximum number of queries that can query to the $O_{\text{KGC}}$ oracle. In addition, $B$ sets the initial for three additional lists (H-list, $h_1$-list, and $h_2$-list) and randomly chooses a bit $b \leftarrow \{0, 1\}$. Finally, $B$ returns $(pp, pk_{ICA}, pk_{KGC}, sk_{KGC})$ to $A$.

**Phase 1.** In this phase, $A$ is allowed to query the following oracles adaptively at polynomially many instances.
Fig. 2. Comparison of the efficiency of our scheme with LHS’19, QCH’20, and LLW21 schemes.

- **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 $H$-list, and returns $u_{ID_i}$ to $A$.

- **$h_1$-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 $O_{DBDH}(pk_{KGC}, u_{ID_i}, u_{ID_j}, k) = 1$.  

### TABLE II

<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>HMZ’18</td>
<td>CLAEKS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LHS’19</td>
<td>IBAEKS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CWZ’19</td>
<td>PAEKS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LLZ19</td>
<td>CBAEKS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PSE20</td>
<td>CLAEKS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>QCH’20</td>
<td>PAEKS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>LLW21</td>
<td>CBAEKS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Ours: ICA-IBSE

1. Channel between KGC and DO/DU.
2. Certificates in ICA-IBSE are used to generate partial secret keys only once; by contrast, certificates in PAEKS are continually used for authentication, and those in CBAEKS 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>HMZ’18</td>
<td>$5T_{cem} + T_{htp} + 2T_h + 3T_{pm}$</td>
<td>$T_{bp} + 3T_{cem} + T_{htp} + 2T_h + 2T_{pm}$</td>
<td>$2T_{bp} + 2T_{cem} + T_h + 2T_{pm}$</td>
</tr>
<tr>
<td>LHS’19</td>
<td>$2T_{bp} + 3T_{cem} + 2T_{htp}$</td>
<td>$T_{bp} + 2T_{cem} + 2T_{htp}$</td>
<td>$2T_{bp} + 2T_{cem} + T_{pm}$</td>
</tr>
<tr>
<td>CWZ’19</td>
<td>$5T_{sm} + 2T_{pa} + 2T_h$</td>
<td>$T_{bp} + 3T_{cem} + T_{htp} + 3T_h + T_{pm}$</td>
<td>$4T_{sm} + 5T_{pa}$</td>
</tr>
<tr>
<td>LLZ19</td>
<td>$2T_{bp} + 3T_{cem} + T_{htp} + 4T_h + T_{pm}$</td>
<td>$T_{bp} + T_{cem} + T_{htp} + T_h$</td>
<td>$T_{bp} + T_h$</td>
</tr>
<tr>
<td>PSE20</td>
<td>$T_{bp} + 3T_{cem} + T_{htp} + T_h$</td>
<td>$2T_{cem} + T_{htp}$</td>
<td>$T_{cem} + T_h$</td>
</tr>
<tr>
<td>QCH’20</td>
<td>$T_{bp} + 3T_{cem} + T_{htp} + 2T_h + 2T_{pm}$</td>
<td>$2T_{pm} + 2T_{cem} + 2T_{htp}$</td>
<td>$2T_{pm} + 2T_{cem} + T_{htp}$</td>
</tr>
<tr>
<td>LLW21</td>
<td>$5T_{sm} + 4T_h$</td>
<td>$T_{bp} + 2T_{cem} + T_{htp} + 2T_h$</td>
<td>$T_{cem} + T_{htp}$</td>
</tr>
</tbody>
</table>

Ours: $T_{bp} + 2T_{cem} + T_{htp} + 2T_h$ | $T_{bp} + T_{htp} + T_h$ | $T_{cem} + T_h$ |

### TABLE IV

Communication Cost of Compared Schemes

<table>
<thead>
<tr>
<th>Schemes</th>
<th>Public Key</th>
<th>Secret Key</th>
<th>Ciphertext</th>
<th>Trapdoor</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMZ’18</td>
<td>$2[G_1] +</td>
<td>ID</td>
<td>]$</td>
<td>$2[G_1] + [G_T]$</td>
</tr>
<tr>
<td>LHS’19</td>
<td>$</td>
<td>ID</td>
<td>$</td>
<td>$2[G_1] + [G_T]$</td>
</tr>
<tr>
<td>CWZ’19</td>
<td>$</td>
<td>G_{w_1}</td>
<td>$</td>
<td>$3[G_{w_1}]$</td>
</tr>
<tr>
<td>LLZ19</td>
<td>$</td>
<td>G_2</td>
<td>$</td>
<td>$3[G_{w_1}]$</td>
</tr>
<tr>
<td>PSE20</td>
<td>$</td>
<td>G_1</td>
<td>+</td>
<td>ID</td>
</tr>
<tr>
<td>QCH’20</td>
<td>$</td>
<td>G_1</td>
<td>$</td>
<td>$</td>
</tr>
<tr>
<td>LLW21</td>
<td>$3[G_{w_1}]$</td>
<td>$2[G_{w_1}] +</td>
<td>Z_q</td>
<td>$</td>
</tr>
</tbody>
</table>

Ours: $|ID|$ | $|G_1|$ | $2[G_1]$ | $|A|$ |

### TABLE V

Experimental Platform

- **Description**
  - CPU: AMD Ryzen 5-2600 3.4GHz
  - CPU processor number: 6
  - Operation system: Ubuntu 18.04
  - Linux kernel version: 5.3.0-59-generic
  - Random access memory: 16.3GB
  - Solid state disk: 232.9GB
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 \(h \leftarrow Z_q^*\), add \((ID_i, ID_j, k, w, h)\) to \(h_1\)-list, and return \(h\) to \(\mathcal{A}\).

- \(h_2\)-oracle: On the query \((h, c_1)\), \(B\) 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 \((\tilde{h}, c_1, \tilde{h})\) to \(h_2\)-list, and returns \(h\) to \(\mathcal{A}\).

- \(O_{rk\text{-}KGC}()\): When \(\mathcal{A}\) issues a issue key query, \(B\) first samples \(ID \leftarrow \mathbb{ID}\) such that \(ID\) does not exist in \(\mathbb{ID}\)-list. Subsequently, \(B\) executes the following steps under the following conditions:

- if \(Q_{key} \notin \{\ell_1, \ell_2\}\): \(B\) obtains \(u_{ID}\) by querying \(H(ID)\).

- if \(Q_{key} = \ell_1\): \(B\) sets \(u_{ID} = g^{\beta_{\mathbb{ID},1}}\) and \(u_{\mathbb{ID},2} = u_{\mathbb{ID}}\) and \(\sigma_{\mathbb{ID}} \leftarrow \text{Sig} \cdot \text{Sign} (sk_{\mathbb{ID}}, k_{\mathbb{ID},2})\). Finally, \(B\) returns \(M_1 = (u_{\mathbb{ID},2}, \sigma_{\mathbb{ID}})\) to \(\mathcal{A}\), sets \(IDList[Q_{key}] = ID\) and updates \(Q_{key} = Q_{key} + 1\).

- \(O_{\text{et-KGC}}()\): when \(\mathcal{A}\) queries for a ciphertext with \((i, j, w)\), where \(i, j \in [Q_{key}]\), \(B\) executes the following steps:

- retrieve \((IDList[i], IDList[j], w, IDList[w])\) and \((IDList[j], IDList[w], u_{IDList[w]})\) from \(H\)-list for \(IDList[i]\) and \(IDList[j]\), respectively.

- randomly choose \(r \leftarrow Z_q^*\).

- if \(i, j \in \{\ell_1, \ell_2\}\): search \(h_1\)-list for the tuple \((IDList[i], IDList[j], w, h)\). If no such tuple exists, randomly choose \(h \leftarrow Z_q^*\) and add \((IDList[i], IDList[j], w, h)\) to \(h_1\)-list.

- otherwise (i.e., \(i \notin \{\ell_1, \ell_2\}\) or \(j \notin \{\ell_1, \ell_2\}\)) either compute \(k = \hat{e}(pk_{KGC}^{IDList[i]}, u_{IDList[j]})\) if \(i \notin \{\ell_1, \ell_2\}\) or compute \(k = \hat{e}(u_{IDList[i]}, pk_{KGC}^{IDList[j]})\) 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 \(k = \hat{e}(pk_{KGC}^{IDList[i]}, u_{IDList[j]}\) or \(k = \hat{e}(u_{IDList[i]}, pk_{KGC}^{IDList[j]}\).

- search \(h_1\)-list for the tuple \((IDList[i], IDList[j], k, w, h)\). If no such tuple exists, randomly choose \(h \leftarrow Z_q^*\) and add \((IDList[i], IDList[j], k, w, h)\) to \(h_1\)-list.

- compute \(c_1 = g^r\) and \(c_2 = g^{hr}\), where \(h\) is retrieved from \(h_2\)-list (i.e., \(h_2(h, c_1)\)).

- return \(ct_w = (c_1, c_2)\).

- \(O_{\text{et-KGC}}()\): when \(\mathcal{A}\) queries for a trapdoor with \((i, j, w)\), where \(i, j \in [Q_{key}]\), \(B\) executes the following steps:

- retrieve \((IDList[i], IDList[j], u_{IDList[w]})\) and \((IDList[j], IDList[w], u_{IDList[w]})\) from \(H\)-list for \(IDList[i]\) and \(IDList[j]\), respectively.

- otherwise (i.e., \(i \notin \{\ell_1, \ell_2\}\) or \(j \notin \{\ell_1, \ell_2\}\)) either compute \(k = \hat{e}(pk_{KGC}^{IDList[i]}, u_{IDList[j]})\) if \(i \notin \{\ell_1, \ell_2\}\) or compute \(k = \hat{e}(u_{IDList[i]}, pk_{KGC}^{IDList[j]})\) 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 \(k = \hat{e}(pk_{KGC}^{IDList[i]}, u_{IDList[j]})\) or \(k = \hat{e}(u_{IDList[i]}, pk_{KGC}^{IDList[j]})\).

- search \(h_1\)-list for the tuple \((IDList[i], IDList[j], k, w, h)\). If no such tuple exists, randomly choose \(h \leftarrow Z_q^*\) and add \((IDList[i], IDList[j], k, w, h)\) to \(h_1\)-list.

Challenge. At the end of Phase 1, \(\mathcal{A}\) outputs the challenged tuple \((w_0 = \{w_0, \ldots, w_0\}, w_i = \{w_1, \ldots, w_n\}, \alpha, \beta)\), 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 \(h_1\)-list for the tuple \((IDList[\alpha], IDList[\beta], \perp, w_i, h_i)\). If no such tuple is found, randomly choose \(h_i \leftarrow Z_q^*\) and add \((IDList[\alpha], IDList[\beta], \perp, w_i, h_i)\) to \(h_1\)-list. Third, compute \(ct_i = (c_i^1, c_i^2)\), where \(c_i^1 = g^{r_i}, c_i^2 = g^{hr_i}\), and \(h_i\) is retrieved from \(h_2\)-list (i.e., \(h_i = h_2(h_i, c_i^1)\)).

- return the challenged ciphertext \(ct = (ct_1, \ldots, ct_n)\).

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

Guess. Finally, \(\mathcal{A}\) outputs \(b' \in \{0, 1\}\) as its guess. \(B\) searches \(h_1\)-list for \(k^*\) such that \(O_{\text{DBDH}}(g^a, g^{b'}, g^c, (k^*)^{-1}g) = 1\) and \((k^*)^{-1}g\) 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 \(\mathcal{A}\)’s perspective, the probability that \(B\) does not abort the game \((\{\alpha, \beta\} = \{IDList[\alpha], IDList[\beta]\})\) in Challenge is \(\frac{2}{2^{\|\text{IDList}\|}}\). Furthermore, because \(h_1\) is modeled as a random oracle, the adversary’s advantage is negligible, unless \((IDList[\alpha], IDList[\beta], k^*, w_i, h_i)\) appears in \(h_1\)-list such that \((k^*)^{-1}g = \hat{e}(g, g)^{\text{hash}(x^{-1})} = \hat{e}(g, g)^{ab_c}\). If this tuple appears in \(h_1\)-list, then \(B\) is necessarily able to solve the

<table>
<thead>
<tr>
<th>Bit-length (bit)</th>
<th>Running time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>Z_q^*</td>
</tr>
<tr>
<td>160</td>
<td>512</td>
</tr>
</tbody>
</table>

TABLE VI

Bit-length of Elements and Running Time of Operations (80-bit Security)
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 $\epsilon' \geq \epsilon^{2^{\frac{2}{n_{\max(Q_{G})}}}}$.

**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 $(w_0, w_1, \alpha, \beta)$ 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 $\{\alpha, \beta\} \neq \{\ell_1, \ell_2\}$, $B$ aborts the game.
- Search $h_1$-list for the tuple $((\text{IDList}[\alpha], \text{IDList}[\beta], \bot, w_b), h)$. If no such tuple is found, randomly choose $h \leftarrow Z_q$ and add $((\text{IDList}[\alpha], \text{IDList}[\beta], \bot, w_b), h)$ to $h_1$-list.
- Return the challenged trapdoor $td^* = h$.

**VII. Theoretical Comparison and Performance Evaluation**

In this section, we detail the theoretical comparison of our scheme with other state-of-the-art schemes, specifically the PAEKS schemes: CWZ'19 [24] and QCH'20 [25], the IBAEKS scheme: LHS'19 [30], the CBAEKS schemes: LLZ19 [32] and LLW21 [31], and the CLAEKS schemes: HMZKL19 [35] and PSE20 [33]. The features of these schemes are listed in Table II. We also evaluate the performance of our proposed scheme against that of the LHS'19 [30], QCH'20 [25], and LLW21 [31] schemes.

**A. Theoretical Comparison**

We compare the schemes with respect to their communication cost and computational cost. The comparison results are presented in Table III and IV. For communication cost, we use $|Z_q^*|$, $|G_1|$, $|G_T|$, and $|G_{ec}|$ to denote the bit lengths of element of the $Z_q^*$, bilinear group $G_1$, bilinear target group $G_T$, and elliptic curve group $G_{ec}$, respectively. In addition, we use $|ID|$ and $|h|$ to denote the bit lengths of a user’s identity and output of the hash function, respectively. Note that in the CBAEKS schemes [31], [32], the size of the public key is the sum of the sizes of the public key and certificate. For computational cost, we use the symbols “KeywordEnc” and “TrapdoorGen” to denote the cost of encryption and trapdoor generation per keyword, respectively. We also use the symbol “Test” to denote the cost of performing a test of whether a ciphertext is matched with a trapdoor. In this theoretical comparison, we consider eight time-consuming operations that are primarily used in these schemes, namely hash-to-point function $(T_{\text{htp}})$, hash $(T_h)$, bilinear pairing $(T_{bp})$, modular exponential over bilinear group $G_1 (T_{em})$, point multiplication over the bilinear group $G_1 (T_{pm})$, modular inverse over $Z_q^*$ $(T_{mi})$, point addition over the elliptic curve group $G_{ec} (T_{pa})$, and scalar multiplication over the elliptic curve group $G_{ec} (T_{sm})$.

**B. Performance Evaluation**

In order to make a more specific comparison, we first experiment to compare the time cost of each operation and the space required by each element, where the time cost of each operation is obtained by the average of 1000 times. Furthermore, to evaluate the performance of our scheme, we fully implement our proposed scheme, LHS'19 [30], QCH'20 [25], and LLW21 [31] schemes. The source codes of comparison and implementations are available at https://github.com/zyliu-crypto/ICA-IBSE. We conduct the experiment in the environment described in Table V. Specifically, we use the SHA3-256 library [4] for the general cryptographic hash function, and we use the PBC library [4] and MIRACL library [5] for operations over bilinear groups ($T_{bp}$, $T_{em}$, and $T_{pm}$) and elliptic curve group ($T_{mi}$, $T_{pa}$, and $T_{sm}$), respectively. For achieving the same security level (i.e., 80-bit), we adopt Type-A pairing with a 160-bit group order, 512-bit group element for $G_1$, and 1024-bit group element for $G_T$ for bilinear pairing, and adopt the security parameter secp160r1 recommended by Standards for Efficient Cryptography Group [2] for elliptic curve group $G_{ec}$.

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 QCH'20 scheme [25] in performing test, overall comparisons, our scheme can effectively encrypt keywords, generate trapdoors and perform tests. In addition, although LLW21 scheme [31] 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 endpoint type 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), convenience, and security. A concrete framework is presented, and security proofs are provided, which demonstrate that the ICA-IBSE scheme can resist MCKA and IKGA under random oracles. Moreover, we experimentally verify that our scheme not only reduces storage requirements but is also practicable relative to its state-of-the-art counterparts.

**References**


https://github.com/brainhub/SHA3IUF
https://crypto.stanford.edu/pbc/
https://github.com/miracl/MIRACL