5-2019

Designated-server identity-based authenticated encryption with keyword search for encrypted emails

Hongbo LI

Qiong HUANG

Jian SHEN

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

Willy SUSILO

Citation

LI, Hongbo; HUANG, Qiong; SHEN, Jian; YANG, Guomin; and SUSILO, Willy. Designated-server identity-based authenticated encryption with keyword search for encrypted emails. (2019). *Information Sciences*. 481, 330-343.

Available at: https://ink.library.smu.edu.sg/sis_research/7291

Information Sciences 481 (2019) 330–343

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Designated-server identity-based authenticated encryption with keyword search for encrypted emails

Hongbo Li[a], Qiong Huang[a,*], Jian Shen[b], Guomin Yang[c], Willy Susilo[c]

[a] College of Mathematics and Informatics, South China Agricultural University, China
[b] School of Computer and Software, Nanjing University of Information Science and Technology, China
[c] School of Computing and Information Technology, University of Wollongong, Wollongong, Australia

ARTICLE INFO

Article history:
Received 20 August 2018
Revised 10 November 2018
Accepted 1 January 2019
Available online 2 January 2019

Keywords:
Keyword search
Encrypted email system
Public key encryption
Identity based encryption
Inside keyword guessing attacks

ABSTRACT

In encrypted email system, how to search over encrypted cloud emails without decryption is an important and practical problem. Public key encryption with keyword search (PEKS) is an efficient solution to it. However, PEKS suffers from the complex key management problem in the public key infrastructure. Its variant in the identity-based setting addresses the drawback, however, almost all the schemes does not resist against offline keyword guessing attacks (KGA) by inside adversaries. In this work we introduce the notion of *designated-server identity-based authenticated encryption with keyword search* (dIBAEKS), in which the email sender authenticates the message while encrypting so that no adversary including the server can launch offline KGA. Furthermore, we strengthen the security requirement so that only the designated server has the capability to search over encrypted emails for receivers. We formally define dIBAEKS and its security models, and propose two dIBAEKS constructions using Type-I and Type-III bilinear pairing, respectively. We compare our schemes with some related IBEKS schemes in the literature, and do experiments to demonstrate its efficiency. Although they are slightly less computationally efficient than but still comparable with the related schemes, our schemes provide stronger security guarantee and better protect users' privacy.

1. Introduction

Cloud storage has received widespread attentions due to its low-cost and almost unlimited data storage space [26]. Users are allowed to store their data in the cloud and to access their data anytime and anywhere, which saves the local data storage. Nowadays, in order to reduce running costs many companies would choose to outsource their email service to an *honest-but-curious* cloud service provider (CSP). It provides the email service as promised, but may try its best to learn users' emails to discover important information. Due to the high sensitivity of some business emails, these emails would be encrypted before sending. Generally, users may have lots of emails, and it is not necessary to download and decrypt all of them.

Considering the scenario shown in Fig. 1. Suppose that Alice sends encrypted emails via an untrusted cloud email server. To save local storage space, Bob will store the encrypted emails in the cloud instead of downloading all of them. Among all the encrypted emails, suppose that Bob wants to retrieve encrypted emails corresponding to some keyword $w =$ '*contract*'.

* Corresponding author.
E-mail address: qhuang@scau.edu.cn (Q. Huang).

https://doi.org/10.1016/j.ins.2019.01.004
0020-0255/

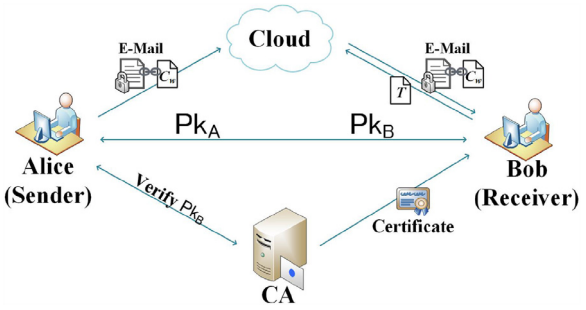

Fig. 1. Encrypted email system with cloud email server.

It is desirable for Bob to download only this email without leaking information about the keyword w it searches and other privacy. However, cryptographic encryption schemes usually hide the feature/structure of the original data, hence, it is hard for the cloud server to search over all the encrypted emails using traditional data search mechanisms. Thus we need a specific encryption mechanism which supports efficient search over encrypted data. The *symmetric searchable encryption* (SSE) [27]is not suitable for this scenario, because to use SSE, Bob has to negotiate with Alice a shared key before sending any encrypted emails, which is rather inconvenient for applications like emails.

PEKS was proposed to address this problem [4]. Different from SSE schemes [12,13,27], in PEKS anyone could be the data owner/sender and share (encrypted) data with the data receiver R, and R can authorize someone to search over encrypted data by giving them a trapdoor associated with a specified keyword w. In the aforementioned scenario, to realize the searching functionality, Alice retrieves keywords $\{w\}$ from emails $\{F\}$ to be sent to Bob, and encrypts both the keywords and the emails using Bob's public key. She then sends both $\{C_w\}$ and $\{C_F\}$ to the cloud email server, and the server would notify Bob that there is an email for him. To search over the encrypted emails for a keyword w, Bob generates and sends a corresponding trapdoor T to the email server, which then searches over $\{C_w\}$ and returns the emails $\{C_F\}$ associated with the matching keyword ciphertexts. After downloading these emails, Bob decrypts them to get the desired email contents.

Although PEKS solves the problem of searching over shared encrypted data, there are still some privacy issues. For example, it was pointed out in [7] that some PEKS schemes suffer from the risk of offline keyword guessing attacks (KGA). A malicious adversary tries candidate keywords one-by-one and checks if the given keyword ciphertext matches the candidate. Due to the small space of real-life keywords, the attack is feasible. Therefore, if the cloud email server becomes malicious, it may recover private information from users' emails by initiating offline KGA. How to construct PEKS secure against offline KGA is a hard problem. Recently, Huang et al. introduced a new primitive named *public-key authenticated encryption with keyword search* (PAEKS), which is a novel and effective method to counter this kind of attacks [16,17]. Roughly speaking, data sender in PAEKS authenticates the keyword while encrypting. However, there are two drawbacks in PAEKS scheme. One is that an outside adversary could obtain the *search pattern* of users [19] once it breaks into the cloud server. Informally, search pattern indicates which searching queries contain the same keyword in the search history. Adversaries may reveal information about the plaintext from the searching frequency. The other is that PAEKS works in the *public key infrastructure* (PKI) and suffers from issues of complex certificate management and heavy cost of maintenance.

To hide search pattern, Baek et al. proposed to have a designated tester to search over the encrypted data [2]. Anyone, except the one who owns the tester's private key, cannot search even if it has the trapdoor. Rhee et al. proposed two generic constructions of *designated tester public-key encryption with keyword search* (dPEKS) based on anonymous identity-based encryption (IBE) [21]. Emura et al. proposed another two generic constructions of dPEKS based on anonymous IBE, tag-based encryption and one-time signature [9]. However, these dPEKS schemes cannot resist inside offline KGA.

It is pointed in Boneh and Franklin [5] that the identity-based encryption (IBE) can be used in the encrypted email system in order to reduce the heavy cost of key management. The receiver's identity, e.g. email address, can be directly used as its public key. Abdalla et al. initiated the study of *identity-based encryption with keyword search* (IBEKS), which integrates the keyword search functionality in IBE [1]. To use the encrypted email system, a sender encrypts an email using IBE and its keywords using IBEKS, and uploads both the encrypted email and encrypted keywords to the cloud email server. To retrieve emails of a certain topic, the receiver delegates the server to search over the encrypted keywords by giving a trapdoor w.r.t. a specific keyword about the topic. After searching, the server returns encrypted emails associated with the matching (encrypted) keywords to the receiver as the search result. However, to our best knowledge, there is no IBEKS construction resisting the inside offline KGA, in the meantime, hiding search pattern from outside adversaries.

Most of the PEKS and IBKES schemes mentioned above are constructed based on *symmetric* (Type-I) bilinear pairing. However, it is known that Type-III *asymmetric* bilinear pairing performs more efficiently with high security parameters [14].

1.1. Our contributions

We propose a new notion called *designated-server identity-based authenticated encryption with keyword search* (dIBAEKS) to solve the aforementioned problems. As shown in Fig. 2, there are four entities in dIBAEKS, a private key generator (PKG),

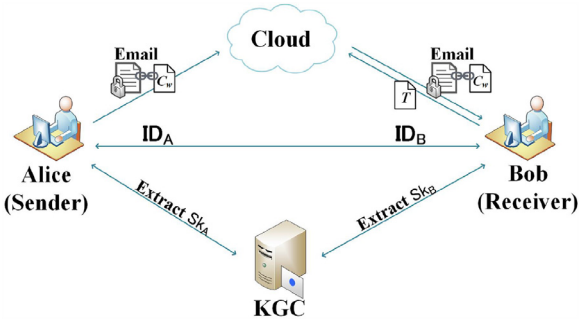

Fig. 2. System model of dIBAEKS.

a cloud email server, a sender (Alice) and a receiver (Bob). Users' identity information are used as their public keys, and PKG is in charge of generating user secret keys based on their identities. The cloud email server stores the encrypted emails and encrypted keywords sent from Alice to Bob. To search over the emails sent from Alice to him, Bob gives to the cloud email server a searching trapdoor T w.r.t a candidate keyword w, with which the server then searches over the ciphertexts using its own secret key and returns the matching encrypted emails containing w without knowing what the keyword w is. Bob then decrypts the returned emails using its secret key and read the contents. In dIBAEKS, no adversary can forge a valid encrypted keyword sent from Alice to Bob unless it knows Alice's secret key. Furthermore, no adversary is able to run the keyword search even if it obtains Bob's trapdoor unless it knows the cloud email server's secret key.

We formally define dIBAEKS and present its security models. Then we propose a concrete dIBAEKS scheme based on *symmetric* (Type-I) bilinear pairing and prove it to be secure against inside offline KGA. Our scheme is shown to satisfy the security that only the designated server can search (a.k.a *designated testability*, see Section 3 for the definition), but this property is of CPA type, i.e., the adversary is not allowed to query any test result. In order to strengthen the designated testability to achieve the CCA type security, we show how to modify the scheme to allow the adversary to issue test queries. In order to further improve the efficiency, we also show how to modify the scheme to work in Type-III asymmetric bilinear pairing setting. The new scheme, denoted by dIBAEKS-3, is much more efficient than dIBAEKS.

To show the advantage of dIBAEKS, we compare it with some related schemes in the literature in terms of security, computational complexity, and communication overhead, and demonstrate the computational efficiency of our scheme via experiments. Although our scheme has a slightly less computationally efficient than but still comparable with the compared schemes, it provides stronger security guarantee and better protects users' privacy.

1.2. Related works

Public key encryption with keyword search (PEKS) was introduced in [4], initiating the study of SE in public key setting. To protect user privacy, Baek et al. pointed out a secure channel is necessary in PEKS, and proposed a secure-channel-free PEKS scheme denoted by dPEKS, where only the designated tester can do the test [2]. Byun et al. proposed a new attack method against PEKS schemes called *offline* KGA [7]. Yau et al. proved neither schemes in [2,4] is secure against the offline KGA [32]. Fang et al. enhanced the security of dPEKS and proposed another dPEKS scheme secure against outsider offline KGA, yet it cannot resist the inside adversaries' offline KGA [10]. Fang et al. in 2013 proposed a dPEKS scheme secure against the outside KGA without requiring random oracles [11]. Other dPEKS schemes are also proposed to enhance the security of dPEKS scheme and to resist the offline KGA, e.g. [15,22–25,31]. Although outside adversaries can be restricted by these schemes, inside adversaries still can succeed in offline KGA. Recently, Huang et al. proposed to add authorization in ciphertext generation which makes an inside adversary lose the ability to generate valid ciphertexts, hence solved the inside offline KGA problem [16,17]. To reduce the heavy cost of public key certification, Abdalla et al. proposed the identity-based encryption with keyword search (IBEKS) [1]. Recently, to improve the efficiency, Tomida et al. proposed another concrete IBEKS scheme, denoted by ACIBEKS [28], and Wu et al. proposed another IBEKS scheme with designated tester [30], denoted by EdIBEKS. However, both ACIBEKS and EdIBEKS schemes cannot resist offline KGA launched by the server. Zhang et al. proposed an IBEKS scheme, denoted by ZXMZ-IBEKS, which can resist outside offline KGA based on the lattice assumption [33]. Although ZXMZ-IBEKS owns the security against quantum attacks, it cannot prevent the inside offline KGA.

2. Preliminaries

2.1. Bilinear pairing

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of the prime order p. A map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map if it satisfies: (1) $\forall g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and $x, y \in \mathbb{Z}$, $\hat{e}(g_1^x, g_2^y) = \hat{e}(g_1, g_2)^{xy}$; (2) if g_1 generates $\mathbb{G}_1$ and g_2 generates $\mathbb{G}_2$, $\hat{e}(g_1, g_1)$ generates $\mathbb{G}_T$; (3)

$\hat{e}(g_1, g_2)$ is efficiently computable for any $g_1, \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ [5]. Bilinear pairings can be classified into the following three types.

- Type-I: $\mathbb{G}_1 = \mathbb{G}_2$;
- Type-II: $\mathbb{G}_1 \neq \mathbb{G}_2$, but an efficiently computable homomorphism $\bar{h} : \mathbb{G}_2 \to \mathbb{G}_1$ exists;
- Type-III: $\mathbb{G}_1 \neq \mathbb{G}_2$, and no efficiently computable homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ exists [14].

2.2. Decisional bilinear Diffie–Hellman assumption

Denote by $\mathsf{Y} = (g, g^x, g^y, g^z \in \mathbb{G}_1, \hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T, Z \in \mathbb{G}_T)$. Let β be a bit such that $\beta = 0$ if $Z = \hat{e}(g, g)^{xyz}$, and $\beta = 1$ if Z is randomly selected from $\mathbb{G}_T$. The decisional bilinear Diffie–Hellman (DBDH) problem is to determine the value of β [6,18].

Definition 1 (DBDH assumption [6,18])**.** The DBDH assumption holds if the probability to solve DBDH problem is negligible for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$, i.e.

$$|\Pr[0 \leftarrow \mathcal{A}(\mathsf{Y})|\beta = 0] - \Pr[0 \leftarrow \mathcal{A}(\mathsf{Y})|\beta = 1]|$$

is negligible.

2.3. Decisional bilinear Diffie–Hellman assumption in Type-III pairing

Denote by $\mathsf{Y}_3 = (g_1, g_1^x, g_1^y \in \mathbb{G}_1, g_2, g_2^y, g_2^z \in \mathbb{G}_2, \hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T, Z \in \mathbb{G}_T)$. Let β be a bit such that $\beta = 0$ if $Z = \hat{e}(g_1, g_2)^{xyz}$, and $\beta = 1$ if Z is randomly selected from $\mathbb{G}_T$. The decisional bilinear Diffie–Hellman in Type-III pairing (DBDH-3) problem is to determine the value of β [8].

Definition 2 (DBDH-3 assumption [8])**.** The DBDH-3 assumption holds if the probability to solve DBDH-3 problem is negligible for any probabilistic polynomial-time (PPT) algorithm $\mathcal{A}$, i.e.

$$|\Pr[0 \leftarrow \mathcal{A}(\mathsf{Y}_3)|\beta = 0] - \Pr[0 \leftarrow \mathcal{A}(\mathsf{Y}_3)|\beta = 1]|$$

is negligible.

3. Definitions and system model

3.1. Definition

A dIBAEKS scheme consists of the following (probabilistic) polynomial-time (PPT) algorithms.

- $(pp, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda)$: Given a security parameter λ, it outputs the public parameter pp and a master secret key msk.
- $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr}) \leftarrow \mathsf{KGen}_{svr}(pp)$: Given pp, it returns the server's public/secret key pair $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr})$.
- $\mathsf{Sk}_{ID_i} \leftarrow \mathsf{KGen}_{usr}(pp, \mathsf{msk}, ID_i)$: Given pp, msk and the identity ID_i of user i, it outputs the secret key Sk_{ID_i} of user i.
- $C_{w,s,r} \leftarrow \mathsf{PEKS}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_s}, ID_s, ID_r)$: Given pp, a keyword w, Pk_{svr}, Sk_{ID_s}, ID_s of a sender, and ID_r of the a receiver, it outputs a ciphertext $C_{w,\,s,\,r}$.
- $T_{w,s,r} \leftarrow \mathsf{Trapdoor}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_r}, ID_s, ID_r)$: Given pp, w, Pk_{svr}, Sk_{ID_r} of a receiver, and ID_s, ID_r, it outputs a trapdoor $T_{w,\,s,\,r}$.
- $\beta \leftarrow \mathsf{Test}(pp, \mathsf{Sk}_{svr}, ID_s, ID_r, C_{w,s,r}, T_{w,s,r})$: Given pp, Sk_{svr}, ID_s, ID_r, $C_{w,\,s,\,r}$ and $T_{w,\,s,\,r}$, it outputs a bit β, which is 1 if $C_{w,\,s,\,r}$ and $T_{w,\,s,\,r}$ contain the same keyword, and 0 otherwise.

3.2. Security models

Below we define the semantic security of dIBAEKS against inside offline KGA via three games between an adversary $\mathcal{A}$ and a challenger $\mathcal{B}$.

Game I: Ciphertext indistinguishability

In this game, the semi-trusted server is assumed to be the adversary $\mathcal{A}$. It should fulfill its obligation as required, but try to obtain information about users' data. Ciphertext indistinguishability aims to prevent the adversary from distinguishing a given ciphertext is the encryption of which of the two keywords (selected by the adversary itself) if $\mathcal{A}$ does not have knowledge of the sender and receiver's secret keys. In other words, ciphertext indistinguishability guarantees that the server cannot search over the ciphertexts if it is not authorized by the user.

1. Setup: $\mathcal{B}$ generates the system parameter pp, the PKG's key pair (pp, msk) and the server's key pair $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr})$. It invokes $\mathcal{A}$ on input pp and $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr})$.
2. Phase 1: $\mathcal{A}$ is allowed to adaptively issue queries for polynomially many times as below.
   - Extract Oracle: Given the identity ID_i of user i, it returns the user's secret key Sk_{ID_i} to $\mathcal{A}$.
   - Trapdoor Oracle: Given a keyword w, ID_s of a sender and ID_r of a receiver, it computes and returns to $\mathcal{A}$ the corresponding trapdoor $T_{w,\,s,\,r}$.

- Ciphertext Oracle: Given a keyword w, ID_s of a sender and ID_r of a receiver, it computes and returns to $\mathcal{A}$ the corresponding ciphertext $C_{w,s,r}$.

3. Challenge: $\mathcal{A}$ submits to $\mathcal{B}$ ID_s^* of a sender, ID_r^* of a receiver and two challenge keywords (w_0^*, w_1^*). $\mathcal{B}$ randomly selects a bit $\beta \in \{0, 1\}$, computes $C_{w_\beta^*,s^*,r^*} \leftarrow \mathsf{PEKS}(pp, w_\beta^*, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_s^*}, ID_s^*, ID_r^*)$, and returns $C_{w_\beta^*,s^*,r^*}$ to $\mathcal{A}$.
4. Phase 2: $\mathcal{A}$ continues to issue queries to the oracles as in Phase 1.
5. Guess: $\mathcal{A}$ outputs a bit $\beta' \in \{0, 1\}$, and wins the game if (1) $\beta' = \beta$, (2) ID_s^*, ID_r^* have not been queried for secret keys, and (3) $\langle w_0^*, ID_s^*, ID_r^*\rangle$, $\langle w_1^*, ID_s^*, ID_r^*\rangle$ have not been submitted to Trapdoor Oracle nor Ciphertext Oracle.

$\mathcal{A}$'s advantage of successfully distinguishing the ciphertexts of dIBAEKS is defined as

$$Adv_{\mathcal{A}}^{\mathcal{C}}(\lambda) = \left|\Pr[\beta' = \beta] - 1/2\right|.$$

Definition 3. A dIBAEKS scheme satisfies ciphertext indistinguishability if for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{\mathcal{C}}(\lambda) \leq \mathrm{negl}(\lambda)$.

Game II: Trapdoor indistinguishability

Same as Game I, the semi-trusted cloud server is also assumed to be the adversary $\mathcal{A}$ in this game. The difference is that now trapdoor indistinguishability aims to prevent $\mathcal{A}$ from learning information about the keyword from simply a given trapdoor if $\mathcal{A}$ does not know the sender and receiver's secret keys. It indicates that even the server cannot generates valid ciphertexts with respect to the sender and the receiver.

1. Setup: Same as in Game I.
2. Phase 1: Same as in Game I.
3. Challenge: $\mathcal{A}$ submits to $\mathcal{B}$ ID_s^* of a sender, ID_r^* of a receiver and two challenge keywords (w_0^*, w_1^*), $\mathcal{B}$ randomly selects a bit $\beta \in \{0, 1\}$, computes $T_{w_\beta^*,s^*,r^*} \leftarrow \mathsf{Trapdoor}(pp, w_\beta^*, ID_s^*, \mathsf{Sk}_{ID_r^*})$, and returns $T_{w_\beta^*,s^*,r^*}$ to $\mathcal{A}$.
4. Phase 2: Same as in Game I.
5. Guess: $\mathcal{A}$ outputs a bit $\beta' \in \{0, 1\}$, and wins the game if (1) $\beta' = \beta$, (2) ID_s^*, ID_r^* have not been queried for secret keys, and (3) $\langle w_0^*, ID_s^*, ID_r^*\rangle$, $\langle w_1^*, ID_s^*, ID_r^*\rangle$ have not been submitted to Trapdoor Oracle nor Ciphertext Oracle.

$\mathcal{A}$'s advantage of successfully distinguishing the trapdoors of dIBAEKS is defined as

$$Adv_{\mathcal{A}}^{\mathcal{T}}(\lambda) = \left|\Pr[\beta' = \beta] - 1/2\right|.$$

Definition 4. A dIBAEKS scheme satisfies trapdoor indistinguishability if for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{\mathcal{T}}(\lambda) \leq \mathrm{negl}(\lambda)$.

Remark. If a dIBAEKS scheme Π satisfies ciphertext indistinguishability, any PPT adversary $\mathcal{A}$ without the knowledge of secret keys of the sender ID_s and the receiver ID_r, is unable to forge a valid trapdoor of either of the challenge keywords w.r.t. ID_s and ID_r. To understand it, assume that $\mathcal{A}$ could forge a valid trapdoor of either w_0^* or w_1^* without knowledge of secret keys of ID_s and ID_r. Then $\mathcal{A}$ easily wins Game I by running the Test algorithm taking as input the challenge ciphertext and the forged trapdoor, which contradicts the ciphertext indistinguishability of Π.

Similarly, we can show that if a dIBAEKS scheme satisfies trapdoor indistinguishability, any PPT adversary without the knowledge of secret keys of ID_s and ID_r, is unable to forge a valid ciphertext of either of the challenge keywords w.r.t. ID_s and ID_r.

Game III: Designated testability

In this game, $\mathcal{A}$ is assumed to be an outside adversary which may obtain users' ciphertexts by breaking into the cloud, and obtain trapdoors by monitoring the communication channel between users and the cloud. However, $\mathcal{A}$ does not have the server's secret key. Designated testability aims to ensure that only the designated server could search over the ciphertexts.

1. Setup: $\mathcal{B}$ generates the system parameter pp, PKG's secret key msk and the server's key pair $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr})$. It invokes $\mathcal{A}$ on input pp and Pk_{svr}.
2. Phase 1: $\mathcal{A}$ could adaptively issue queries to the oracle below for polynomially many times.
   - Extract Oracle: Given the identity ID_i of user i, it returns the secret key Sk_{ID_i}.
3. Challenge: $\mathcal{A}$ submits ID_s^* of a sender, ID_r^* of a receiver and two challenge keywords (w_0^*, w_1^*) to $\mathcal{B}$. $\mathcal{B}$ randomly selects a bit $\beta \in \{0, 1\}$, computes $C_{w_\beta^*,s^*,r^*} \leftarrow \mathsf{PEKS}(pp, w_\beta^*, ID_r^*, \mathsf{Sk}_{ID_s^*})$, and returns $C_{w_\beta^*,s^*,r^*}$ to $\mathcal{A}$.
4. Phase 2: $\mathcal{A}$ continues to issuing queries as in Phase 1.
5. Guess: $\mathcal{A}$ outputs a bit $\beta' \in \{0, 1\}$, and wins the game if $\beta' = \beta$.

Different from **Game I** and **Game II**, there is no restriction on the two challenge keywords and identities, which means that $\mathcal{A}$ can obtain secret keys of the challenge identities and trapdoors of the challenge keywords. $\mathcal{A}$'s advantage of successfully breaking the Designated Testability is defined as

$$Adv_{\mathcal{A}}^{\mathcal{D}}(\lambda) = \left|\Pr[\beta' = \beta] - 1/2\right|.$$

Definition 5. A dIBAEKS scheme satisfies designated testability if for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{\mathcal{D}}(\lambda) \leq \mathrm{negl}(\lambda)$.

Notice that the designated testability defined above is actually of CPA type, as the adversary is not allowed to access to **Test** Oracle, which on input a ciphertext and a trapdoor, as well as the sender and the receiver's identities, outputs a bit

indicating whether the ciphertext contains the same keyword as the trapdoor. Below we define a strengthened version of the designated testability, called *designated testability with test query*, which is of CCA type.

Game IV: Designated testability with Test Oracle

The game is almost the same as Game III, except that $\mathcal{A}$ is additionally allowed to issue queries to the Test Oracle as below in both Phase 1 and Phase 2, and $(ID_s^*, ID_r^*, C_{w_\beta^*, s^*, r^*}, \cdot)$ cannot appear in the Test Oracle in Phase 2.

- Test Oracle: Given ID_s of a sender, ID_r of a receiver, $C_{w,\,s,\,r}$ and $T_{w,\,s,\,r}$, it returns 1 if $C_{w,\,s,\,r}$ and $T_{w,\,s,\,r}$ are both valid and contain the same keyword, and 0 otherwise.

$\mathcal{A}$'s advantage of breaking the Designated Testability with Test Query is defined as

$$Adv_{\mathcal{A}}^{DT}(\lambda) = \left|\Pr[\beta' = \beta] - 1/2\right|.$$

Definition 6. A dIBAEKS scheme satisfies designated testability with Test Oracle if for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{DT}(\lambda) \leq \mathrm{negl}(\lambda)$.

Remark. The reason of restricting $\mathcal{A}$ from issuing test queries on input $(ID_s^*, ID_r^*, C_{w_\beta^*, s^*, r^*}, \cdot)$ for any trapdoor is that $\mathcal{A}$ is allowed to know any user's secret key, and thus can compute the trapdoor of any keyword w.r.t. any sender and any receiver. Due to the trapdoor indistinguishability, we cannot tell a given trapdoor is related to which keyword.

4. Our dIBAEKS scheme

4.1. Construction

Now we propose the construction of dIBAEKS, which makes use of the advantage of bilinear pairing.

- $(pp, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda)$: Given a security parameter λ, it randomly selects the master secret key $\mathsf{msk} = \alpha \in \mathbb{Z}_p$ and computes $\mathsf{mpk} = g^\alpha$. It sets the $pp = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, h, H, H_1, \mathsf{mpk})$, where $\mathbb{G}_1, \mathbb{G}_T$ are cyclic groups of prime order p, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ is a bilinear pairing, g, h are generators of $\mathbb{G}_1$, and $H : \mathbb{G}_T \times \{0,1\}^* \to \mathbb{G}_1, H_1 : \{0,1\}^* \to \mathbb{G}_1$ are cryptographic hash functions. It outputs (pp, msk).
- $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr}) \leftarrow \mathsf{KGen}_{svr}(pp)$: Return the server's public/secret key pair $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr}) = (g^t, t)$, where t is randomly selected from $\mathbb{Z}_p$.
- $\mathsf{Sk}_{ID} \leftarrow \mathsf{KGen}_{usr}(pp, \mathsf{msk}, ID)$: Return the secret key $\mathsf{Sk}_{ID} = H_1(ID)^\alpha$.
- $C_{w,s,r} \leftarrow \mathsf{PEKS}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_s}, ID_s, ID_r)$: It randomly selects $s \in \mathbb{Z}_p$, computes and returns a ciphertext $C_{w,s,r} = (C_1, C_2, C_3)$, where

$$C_1 = \hat{e}(H(k,w), \mathsf{Pk}_{svr}^s),\ C_2 = g^s,\ C_3 = h^s,\ \text{and}\ \ k = \hat{e}(\mathsf{Sk}_{ID_s}, H_1(ID_r)).$$

- $T_{w,s,r} \leftarrow \mathsf{Trapdoor}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_r}, ID_s, ID_r)$: It randomly selects $r \in \mathbb{Z}_p$, computes and returns a trapdoor $T_{w,s,r} = (T_1, T_2)$, where

$$T_1 = H(k,w) \cdot h^r,\ T_2 = g^r,\ \text{and}\ \ k = \hat{e}(H_1(ID_s), \mathsf{Sk}_{ID_r}).$$

- $1/0 \leftarrow \mathsf{Test}(pp, \mathsf{Sk}_{svr}, ID_s, ID_r, C_{w,s,r}, T_{w,s,r})$: Parse the ciphertext $C_{w,\,s,\,r}$ as (C_1, C_2, C_3) and the trapdoor $T_{w,\,s,\,r}$ as (T_1, T_2). It returns 1 if

$$C_1 \cdot \hat{e}(T_2^{\mathsf{Sk}_{svr}}, C_3) = \hat{e}(T_1^{\mathsf{Sk}_{svr}}, C_2),$$

and 0 otherwise.

4.2. Security analysis

We formally analyze the security of our dIBAEKS scheme as below.

Theorem 1. *The proposed dIBAEKS scheme satisfies ciphertext indistinguishability if DBDH assumption holds.*

Proof. Suppose $\mathcal{A}$ is a PPT adversary trying to break the ciphertext indistinguishability. We build an algorithm $\mathcal{B}$ to solve DBDH problem. Given a problem instance, e.g. $Y = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, g^x, g^y, g^z, Z)$, $\mathcal{B}$ works as below.

1. Setup : $\mathcal{B}$ randomly selects h from $\mathbb{G}_1$ and $t \in \mathbb{Z}_p$, and sets $pp = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, h, \mathsf{mpk} = g^z)$ and $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr}) = (g^t, t)$. It invokes $\mathcal{A}$ on input pp and $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr})$.
2. Phase 1 : $\mathcal{A}$ adaptively issues queries to the following oracles which are simulated by $\mathcal{B}$. We assume that $\mathcal{A}$ does not repeat queries to the same oracle, and that $\mathcal{A}$ would not use an identity ID in any computation before issuing it to random oracle H_1; otherwise, the value of $H_1(ID)$ is random to $\mathcal{A}$ and the probability that $\mathcal{A}$ guesses the correct value of $H_1(ID)$ is negligible.
   - Hash Oracle $\mathcal{O}_H$: Given an element $k \in \mathbb{G}_T$ and a keyword w, it selects a random element from $\mathbb{G}_T$, and returns it as the output of $H(k, w)$.

- Hash Oracle $\mathcal{O}_{H_1}$: The oracle maintains an initially empty list $L_{H_1} = \{\langle \cdot, \cdot, \cdot \rangle\}$. Assume $\mathcal{A}$ issues queries to $\mathcal{O}_{H_1}$ for at most q_H times. it randomly chooses $i, j \in \{1, \cdots, q\}$, and guesses that the i-th query and the j-th query issued by $\mathcal{A}$ to $\mathcal{O}_{H_1}$ are the challenge sender and receiver's identities (ID_s^*, ID_r^*), respectively. Given an identity ID, it returns $\mathcal{A}$ based on the following cases.
  - If this is the i-th query, e.g. $ID = ID_s^*$, it returns $H_1(ID) = g^x$, and adds $\langle ID, g^x, \perp \rangle$ into L_{H_1}.
  - If this is the j-th query, e.g. $ID = ID_r^*$, it returns $H_1(ID) = g^y$, and adds $\langle ID, g^y, \perp \rangle$ into L_{H_1}.
  - Otherwise, it randomly selects $v \in \mathbb{Z}_p$, returns $H_1(ID) = g^v$, and adds $\langle ID, g^v, v \rangle$ into L_{H_1}.
- Extract Oracle $\mathcal{O}_E$: Taking ID as input, if $ID = ID_s^*$ or $ID = ID_r^*$, it outputs a random bit β' and aborts. Otherwise, it retrieves the tuple $\langle ID, H_1(ID), v \rangle$ from L_{H_1}, and returns the secret key $\mathsf{Sk}_{ID} = (g^z)^v$ to $\mathcal{A}$.
- Ciphertext Oracle $\mathcal{O}_C$: Given (w, ID_s, ID_r), it randomly selects $s \in \mathbb{Z}_p$, and computes the ciphertext $C_{w,s,r} = (C_1, C_2, C_3)$ based on the following cases.
  - If $(ID_s, ID_r) = (ID_s^*, ID_r^*)$ or $(ID_s, ID_r) = (ID_r^*, ID_s^*)$, it computes

    $$C_1 = \hat{e}(H(Z, w), \mathsf{Pk}_{svr})^s,\ C_2 = g^s,\ C_3 = h^s.$$

  - Otherwise, at least one of ID_s and ID_r is not equal to ID_s^* nor ID_r^*. W.l.o.g., we assume that $ID_s \notin \{ID_s^*, ID_r^*\}$. It retrieves the tuple $\langle ID_s, H_1(ID_s), v_s \rangle$ from L_{H_1}, computes $k = \hat{e}(g^z, H_1(ID_r))^{v_s}$ and returns

    $$C_1 = \hat{e}(H(k, w), \mathsf{Pk}_{svr})^s,\ C_2 = g^s,\ C_3 = h^s.$$

- Trapdoor Oracle $\mathcal{O}_T$: Given (w, ID_s, ID_r), it randomly selects $r \in \mathbb{Z}_p$, and computes the trapdoor $T_{w,s,r} = (T_1, T_2)$ based on the following cases.
  - If $(ID_s, ID_r) = (ID_s^*, ID_r^*)$ or $(ID_s, ID_r) = (ID_r^*, ID_s^*)$, it computes

    $$T_1 = H(Z, w) \cdot h^r,\ T_2 = g^r.$$

  - Otherwise, at least one of ID_s and ID_r is not equal to ID_s^* nor ID_r^*. W.l.o.g., we assume that $ID_s \notin \{ID_s^*, ID_r^*\}$. It retrieves the tuple $\langle ID_s, H_1(ID_s), v_s \rangle$ from L_{H_1}, computes $k' = \hat{e}(g^z, H_1(ID_r))^{v_s}$ and returns

    $$T_1 = H(k', w) \cdot h^r,\ T_2 = g^r.$$

3. Challenge : $\mathcal{A}$ submits two challenge keywords w_0^*, w_1^*, ID_s^* of a sender and ID_r^* of a receiver. $\mathcal{B}$ randomly selects a bit $\hat{\beta} \in \{0, 1\}$, an element $s \in \mathbb{Z}_p$, and returns the ciphertext $C_{w_{\hat{\beta}}^*, s^*, r^*} = (C_1^*, C_2^*, C_3^*)$, where

$$C_1^* = \hat{e}(H(Z, w_{\hat{\beta}}^*), \mathsf{Pk}_{svr})^s,\ C_2^* = g^s,\ C_3^* = h^s.$$

4. Phase 2 : Same as in Phase 1.
5. Guess : $\mathcal{A}$ outputs a bit $\hat{\beta}'$. $\mathcal{B}$ outputs $\beta' = 0$, if $\hat{\beta}' = \hat{\beta}$, and 1 otherwise.

$\mathcal{B}$ would abort if its guess of the challenge identities is not correct. Denote this event by $\overline{\mathsf{F}}$. If $\mathcal{B}$ aborts, the random bit output by $\mathcal{B}$ is equal to β with probability 1/2. As $\mathcal{B}$ makes its guess at random, the probability that $\overline{\mathsf{F}}$ does not happen is $1/q_H(q_H - 1)$, i.e. $\Pr[\overline{\mathsf{F}}] = 1/q_H(q_H - 1)$.

Suppose that $\mathcal{B}$ does not abort. If $Z = \hat{e}(g, g)^{xyz}$, e.g. $\beta = 0$, the simulation provided by $\mathcal{B}$ is identical to $\mathcal{A}$'s view in a real attack, and $\mathcal{A}$ would win the game with probability $Adv_{\mathcal{A}}^{\mathcal{C}}(\lambda) + 1/2$. If Z is randomly selected from $\mathbb{G}_T$, then $k = H(Z, w_{\hat{\beta}})$ will be a random element of $\mathbb{G}_1$, and thus the challenge ciphertext completely hides the bit $\hat{\beta}$. Although in this case the C_1 component of a ciphertext (and the T_1 component of a trapdoor) w.r.t. ID_s^* and ID_r^* in the simulation above is not exactly the same as a real ciphertext, $\mathcal{B}$ uses the same way to generate the ciphertext C_{w,s^*,r^*} and the trapdoor T_{w,s^*,r^*} so that C_{w,s^*,r^*} and T_{w,s^*,r^*} will make the Test algorithm output 1 if the keywords are the same. Hence, $\mathcal{A}$ would win the game with probability at most 1/2. Therefore, the advantage of $\mathcal{B}$ in solving the DBDH problem is

$$\begin{aligned}
Adv_{\mathcal{B}}^{DBDH}(\lambda) &= \left|\Pr[\beta' = \beta|\mathsf{F}] \cdot \Pr[\mathsf{F}] + \Pr[\beta' = \beta|\overline{\mathsf{F}}] \cdot \Pr[\overline{\mathsf{F}}] - 1/2\right| \\
&= \Big|\frac{1}{2} \cdot (1 - \Pr[\overline{\mathsf{F}}]) + (\Pr[\beta' = 0|\overline{\mathsf{F}} \wedge \beta = 0] \cdot \Pr[\beta = 0] \\
&\quad + \Pr[\beta' = 1|\overline{\mathsf{F}} \wedge \beta = 1] \cdot \Pr[\beta = 1]) \cdot \Pr[\overline{\mathsf{F}}] - \frac{1}{2}\Big| \\
&\geq \Big|\frac{1}{2} \cdot (1 - \Pr[\overline{\mathsf{F}}]) + \Pr[\overline{\mathsf{F}}] \cdot ((Adv_{\mathcal{A}}^{\mathcal{C}}(\lambda) + \frac{1}{2}) \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2}) - \frac{1}{2}\Big| \\
&= \frac{1}{2}\Pr[\overline{\mathsf{F}}] \cdot Adv_{\mathcal{A}}^{\mathcal{C}}(\lambda) \\
&= \frac{1}{2q_H(q_H - 1)} \cdot Adv_{\mathcal{A}}^{\mathcal{C}}(\lambda).
\end{aligned}$$

If $Adv_{\mathcal{A}}^{\mathcal{C}}(\lambda)$ is non-negligible, so is $Adv_{\mathcal{B}}^{DBDH}(\lambda)$. □

Theorem 2. *The proposed dIBAEKS scheme satisfies trapdoor indistinguishability if DBDH assumption holds.*

The proof is similar with that of Theorem 1. The difference is that in the proof of Theorem 2, $\mathcal{B}$ generates the challenge trapdoor as $T_{w^*_{\hat{\beta}},s^*,r^*} = (T_1^*, T_2^*)$, where

$$T_1^* = H(Z, w^*_{\hat{\beta}}) \cdot h^r, \ T_2^* = g^r,$$

and $r \in \mathbb{Z}_p$ is randomly chosen by $\mathcal{B}$. We omit the detailed proof here for simplicity.

Theorem 3. *Our dIBAEKS scheme satisfies designated testability if DBDH assumption holds.*

Proof. Suppose $\mathcal{A}$ is an adversary trying to break the designated testability of our dIBAEKS scheme. We build an algorithm $\mathcal{B}$ to solve DBDH problem. Given a problem instance, e.g. $Y = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, g^x, g^y, g^z, Z)$, $\mathcal{B}$ works as below.

1. Setup : $\mathcal{B}$ randomly selects $\alpha, \gamma \in \mathbb{Z}_p$ and sets $pp = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, \ h = g^\gamma, \mathsf{mpk} = g^\alpha)$ and $\mathsf{Pk}_{svr} = g^x$. It invokes $\mathcal{A}$ on input pp and Pk_{svr}.
2. Phase 1 : $\mathcal{B}$ answers queries of $\mathcal{A}$ as below, assuming that $\mathcal{A}$ does not repeat its queries.
   - Hash Query H: $\mathcal{B}$ maintains a lit $L_H = \{\langle \cdot, \cdot, \cdot \rangle\}$, which is initially empty. Given an element $k \in \mathbb{G}_T$ and a keyword w, $\mathcal{B}$ randomly selects $v_{k,w} \in \mathbb{Z}_p$, returns
   $$H(k, w) = g^y \cdot g^{v_{k,w}},$$
   and adds $\langle (k, w), H(k, w), v_{k,w} \rangle$ into L_H.
   - Hash Query H_1: Given ID, $\mathcal{B}$ selects a random element from $\mathbb{G}_1$, and returns it to $\mathcal{A}$ as the output of $H_1(ID)$.
   - Exact Query: Given ID, $\mathcal{B}$ returns $\mathsf{Sk}_{ID} = H_1(ID)^\alpha$.
3. Challenge : $\mathcal{A}$ submits to $\mathcal{B}$ two challenge keywords (w_0^*, w_1^*), ID_s^* of a sender and ID_r^* of a receiver. $\mathcal{B}$ randomly selects a bit $\hat{\beta} \in \{0, 1\}$, and retrieves the tuple $\langle (k^*, w^*_{\hat{\beta}}), H(k^*, w^*_{\hat{\beta}}), v_{k^*,w^*_{\hat{\beta}}} \rangle$ from L_H, where $k^* = \hat{e}(H_1(ID_s^*), H_1(ID_r^*))^\alpha$. If there is no such a tuple, $\mathcal{B}$ generates it as in answering $\mathcal{A}$'s H queries. It then computes and returns the challenge ciphertext $C_{w^*_{\hat{\beta}},s^*,r^*} = (C_1^*, C_2^*, C_3^*)$, where
   $$C_1^* = Z \cdot \hat{e}(g^z, g^x)^{v_{k,w^*_{\hat{\beta}}}}, \ C_2^* = g^z, \ C_3^* = (g^z)^\gamma = h^z.$$
4. Phase 2 : Same as in Phase 1.
5. Guess : $\mathcal{A}$ outputs a bit $\hat{\beta}'$. $\mathcal{B}$ outputs $\beta' = 0$, if $\hat{\beta}' = \hat{\beta}$, and 1 otherwise.

If $Z = \hat{e}(g, g)^{xyz}$, we have that $C_1^* = \hat{e}(H(k, w^*_{\hat{\beta}}), \mathsf{Pk}_{svr})^z$, and $C_{w^*_{\hat{\beta}},s^*,r^*}$ is a well distributed challenge ciphertext. Hence, the view of $\mathcal{A}$ is identical to that in a real attack, and $\mathcal{A}$ would win the game with the probability of $Adv^D_{\mathcal{A}}(\lambda)$. If Z is random, so is C_1^*. The challenge ciphertext completely hides the bit $\hat{\beta}$, and $\mathcal{A}$ wins the game only with probability 1/2. We then have that the probability that $\mathcal{B}$ solves the DBDH problem is

$$\begin{aligned} Adv^{DBDH}_{\mathcal{B}}(\lambda) &= \left|\Pr[\beta' = 1|\beta = 1] \cdot \Pr[\beta = 1] + \Pr[\beta' = 0|\beta = 0] \cdot \Pr[\beta = 0] - 1/2\right| \\ &= \left|1/2 \cdot 1/2 + 1/2 \cdot Adv^D_{\mathcal{A}}(\lambda) - 1/2\right| \\ &= 1/2 Adv^D_{\mathcal{A}}(\lambda). \end{aligned}$$

If $Adv^D_{\mathcal{A}}(\lambda)$ is non-negligible, so is $Adv^{DBDH}_{\mathcal{B}}(\lambda)$. □

4.3. How to achieve designated testability with Test Oracle

In the above subsection, we proved the designated testability of our proposed scheme, which is of CPA type. However, as both the ciphertext and the trapdoor in our scheme are malleable, it is not hard to see that if $\mathcal{A}$ is allowed to access the Test Oracle (c.f. Def. 6), it will break the designated testability trivially. To solve this issue, we modify the scheme to achieve the *designated testability with Test Oracle*. The new scheme dIBAEKS′ is described as below.

- $(pp, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda)$: it is the same as that in the original scheme, except that here it selects two additional cryptographic hash function $H_2 : \mathbb{G}_T \times \mathbb{G}_1^2 \rightarrow \mathbb{G}_1$ and $H_3 : \mathbb{G}_1^2 \rightarrow \mathbb{G}_1$.
- $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr}) \leftarrow \mathsf{KGen}_{svr}(pp)$: Same as in the original scheme.
- $\mathsf{Sk}_{ID} \leftarrow \mathsf{KGen}_{usr}(pp, \mathsf{msk}, ID)$: Same as in the original scheme.
- $C_{w,s,r} \leftarrow \mathsf{PEKS}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_s}, ID_s, ID_r)$: Randomly select $s \in \mathbb{Z}_p$, compute and return $C_{w,s,r} = (C_1, C_2, C_3, C_4)$, where
  $$C_1 = \hat{e}(H(k, w), \mathsf{Pk}_{svr})^s, \ C_2 = g^s, \ C_3 = h^s,$$
  $$C_4 = H_2(C_1, C_2, C_3)^s, \ k = \hat{e}(\mathsf{Sk}_{ID_s}, H_1(ID_r)).$$
- $T_{w,s,r} \leftarrow \mathsf{Trapdoor}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_r}, ID_s, ID_r)$: Randomly select $r \in \mathbb{Z}_p$, compute and return $T_{w,s,r} = (T_1, T_2, T_3)$, where
  $$T_1 = H(k, w) \cdot h^r, \ T_2 = g^r, \ T_3 = H_3(T_1, T_2)^r, \ k = \hat{e}(H_1(ID_s), \mathsf{Sk}_{ID_r}).$$

- $1/0 \leftarrow \mathsf{Test}(pp, \mathsf{Sk}_{svr}, ID_s, ID_r, C_{w,s,r}, T_{w,s,r})$: Return 1 if the following equations hold:

$$\hat{e}(C_2, h) = \hat{e}(g, C_3), \tag{1}$$

$$\hat{e}(C_2, H_2(C_1, C_2, C_3)) = \hat{e}(g, C_4), \tag{2}$$

$$\hat{e}(T_2, H_3(T_1, T_2)) = \hat{e}(g, T_3), \tag{3}$$

$$C_1 \cdot \hat{e}(T_2, C_3)^{\mathsf{Sk}_{svr}} = \hat{e}(T_1, C_2)^{\mathsf{Sk}_{svr}}, \tag{4}$$

and 0 otherwise.

Remark. The Test algorithm requires eight pairing evaluations, which is costly. To reduce the computational complexity, Eqs. (1)–(3) can be combined into one single equation as follows,

$$\hat{e}(C_2, h^{\mu_1} \cdot H_2(C_1, C_2, C_3)^{\mu_2}) \cdot \hat{e}(T_2, H_3(T_1, T_2)^{\mu_3}) = \hat{e}(g, C_3^{\mu_1} \cdot C_4^{\mu_2} \cdot T_3^{\mu_3}),$$

where μ_1, μ_2, μ_3 are randomly selected from $\mathbb{Z}_p$. The new equation requires three pairing evaluations only. With overwhelming probability, if Eq. (4) holds, Eqs. (1)–(3) hold as well.

The main purpose of introducing C_4 in the ciphertext and T_3 in the trapdoor is to prevent an adversary from modifying the ciphertext and the trapdoor. Below we analyze the security of the new scheme dIBAEKS′.

Theorem 4. *The new scheme dIBAEKS′ satisfies ciphertext indistinguishability if DBDH assumption holds.*

Theorem 5. *The new scheme dIBAEKS′ satisfies trapdoor indistinguishability if DBDH assumption holds.*

Theorem 6. *The new scheme dIBAEKS′ satisfies designated testability with Test Oracle if DBDH assumption holds.*

The proofs of Theorems 4 and 5 are the same as those of Theorems 1 and 2, respectively, and thus we omit them for simplicity. Below we provide the proof of Theorem 6.

Proof. Suppose $\mathcal{A}$ is an adversary trying to break the designated testability of our dIBAEKS scheme. We build an algorithm $\mathcal{B}$ to solve DBDH problem. Given a problem instance, e.g. $Y = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, g^x, g^y, g^z, Z)$, $\mathcal{B}$ works as below.

1. Setup : Same as that in the proof of Theorem 3.
2. Phase 1 : Same as that in the proof of Theorem 3, except that the additional Hash Oracles and Test Oracle are answered by $\mathcal{B}$ as follows.
   - Hash Oracle H_2: it maintain an initially empty list $L_{H_2} = \{\langle \cdot, \cdot, \cdot \rangle\}$. Given (C_1, C_2, C_3), $\mathcal{B}$ randomly selects $\delta \in \mathbb{Z}_p$, returns $H_2(C_1, C_2, C_3) = (g^x)^\delta$, and adds the tuple $\langle (C_1, C_2, C_3), H_2(C_1, C_2, C_3), \delta \rangle$ into L_{H_2}.
   - Hash Oracle H_3: it maintain an initially empty list $L_{H_3} = \{\langle \cdot, \cdot, \cdot \rangle\}$. Given (T_1, T_2), $\mathcal{B}$ randomly selects $\xi \in \mathbb{Z}_p$, returns $H_3(T_1, T_2) = (g^x)^\xi$, and adds the tuple $\langle (T_1, T_2), H_2(T_1, T_2), \xi \rangle$ into L_{H_3}.
   - Test Oracle: Given $(ID_s, ID_r, C_{w,s,r}, T_{w,s,r})$, where $C_{w,s,r} = (C_1, C_2, C_3, C_4)$ and $T_{w,s,r} = (T_1, T_2, T_3)$, it returns 0 if either of Eqs. (1)–(3) does not hold. Otherwise, it retrieves the tuple $\langle (C_1, C_2, C_3), H_2(C_1, C_2, C_3), \delta \rangle$ from L_{H_2} and the tuple $\langle (T_1, T_2), H_3(T_1, T_2), \xi \rangle$ from L_{H_3}. If there are no such tuples, it generates them as in answering H_2 and H_3 queries. It returns 1 if the following equation holds:

$$C_1 \cdot \hat{e}(T_3^{\frac{1}{\xi}}, C_3) = \hat{e}(T_1, C_4^{\frac{1}{\delta}}),$$

   and 0 otherwise. Notice that if either $C_{w,s,r}$ or $T_{w,s,r}$ is not well-formed, it would not pass the checks above.
3. Challenge : At some point, $\mathcal{A}$ submits to $\mathcal{B}$ ID_s^* of a sender, ID_r^* of a receiver, and two challenge keywords (w_0^*, w_1^*). $\mathcal{B}$ randomly selects a bit $\hat{\beta} \in \{0, 1\}$, and retrieves the tuple $\langle (k^*, w_{\hat{\beta}}^*), H(k^*, w_{\hat{\beta}}^*), v_{k^*, w_{\hat{\beta}}^*} \rangle$ from L_H, where $k^* = \hat{e}(H_1(ID_s^*), H_1(ID_r^*))^\alpha$. If there is no such a tuple, $\mathcal{B}$ generates it as in answering $\mathcal{A}$'s H queries. It randomly selects $\delta^* \in \mathbb{Z}_p$, returns the challenge ciphertext $C_{w_{\hat{\beta}}^*, s^*, r^*} = (C_1^*, C_2^*, C_3^*, C_4^*)$, where

$$C_1^* = \hat{e}(H(k^*, w_{\hat{\beta}}^*), \mathsf{Pk}_{svr})^z = Z \cdot \hat{e}(g^z, g^x)^{h_{k, w_{\hat{\beta}}^*}},$$

$$C_2^* = g^z,\ C_3^* = (g^z)^{r'},\ C_4^* = (g^z)^{\delta^*},$$

   and adds the tuple $\langle (C_1^*, C_2^*, C_3^*), g^{\delta^*}, \delta^* \rangle$ into L_{H_2}.
4. Phase 2 : Same as in Phase 1 except that the inputs $(ID_s^*, ID_r^*, C_{w_{\hat{\beta}}^*, s^*, r^*}, \cdot)$ cannot appear in the Test Oracle.
5. Guess : $\mathcal{A}$ outputs a bit $\hat{\beta}'$. $\mathcal{B}$ outputs $\beta' = 0$, if $\hat{\beta}' = \hat{\beta}$, and 1 otherwise.

By a similar probability analysis, we have that if $\mathcal{A}$ breaks the designated testability with test query with non-negligible advantage, $\mathcal{B}$ breaks the DBDH assumption with non-negligible advantage as well.

□

Table 1
Comparison of security.

Schemes	OKGA/w	OKGA/o	IKGA	DT	ID-based
PAEKS [16]	√	√	√	×	×
ACIBEKS [28]	√	×	×	×	√
EdIBEKS [30]	√	√	×	√	√
dIBAEKS	√	√	√	√	√
dIBAEKS-3	√	√	√	√	√

OKGA/w: security against outside keyword guessing attacks with secure channel. OKGA/o: security against outside keyword guessing attacks without secure channel. IKGA: security against inside keyword guessing attacks. DT: designated testability.

5. dIBAEKS scheme based on type-III pairing

As pointed out by Chatterjee et al. [8], the symmetric bilinear pairing runs slower than asymmetric bilinear pairing. To further improve the computational efficiency in practice, we modify our dIBAEKS scheme to work in the Type-III bilinear pairing setting, and propose another scheme, denoted by dBIAEKS-3.

- $(pp, \mathsf{msk}) \leftarrow \mathsf{Setup}(\lambda)$: Given a security parameter λ, the algorithm randomly selects a master secret key $\mathsf{msk} = \alpha \in \mathbb{Z}_p$ and computes $\mathsf{mpk} = g^{\alpha} \in \mathbb{G}_1$. It sets $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, p, g \in \mathbb{G}_1, h \in \mathbb{G}_2, H, H_1, H_2\ \mathsf{mpk})$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of prime order p, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a Type-III bilinear pairing, g is a generator of $\mathbb{G}_1$, h is a generator of $\mathbb{G}_2$, and $H : \mathbb{G}_T \times \{0,1\}^* \rightarrow \mathbb{G}_1, H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$, $H_2 : \{0,1\}^* \rightarrow \mathbb{G}_2$ are cryptographic hash functions. The algorithm outputs (pp, msk).
- $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr}) \leftarrow \mathsf{KGen}_{svr}(pp)$: Return the server's public/secret key pair $(\mathsf{Pk}_{svr}, \mathsf{Sk}_{svr}) = (h^t, t)$, where t is randomly selected from $\mathbb{Z}_p$.
- $\mathsf{Sk}_{ID} \leftarrow \mathsf{KGen}_{usr}(pp, \mathsf{msk}, ID)$: Return the secret key $\mathsf{Sk}_{ID} = (\mathsf{Sk}_{ID}^{(1)}, \mathsf{Sk}_{ID}^{(2)}) = (H_1(ID)^{\alpha}, H_2(ID)^{\alpha})$.
- $C_{w,s,r} \leftarrow \mathsf{PEKS}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_s}, ID_s, ID_r)$: It randomly selects $s \in \mathbb{Z}_p$, computes and returns a ciphertext $C_{w,s,r} = (C_1, C_2, C_3)$, where

$$C_1 = \hat{e}(H(k,w)^s, \mathsf{Pk}_{svr}),\ C_2 = g^s,\ C_3 = h^s,\ \text{and}\ \ k = \hat{e}(\mathsf{Sk}_{ID_s}^{(1)}, H_2(ID_r)).$$

- $T_{w,s,r} \leftarrow \mathsf{Trapdoor}(pp, w, \mathsf{Pk}_{svr}, \mathsf{Sk}_{ID_r}, ID_s, ID_r)$: It randomly selects $r \in \mathbb{Z}_p$, computes and returns a trapdoor $T_{w,s,r} = (T_1, T_2)$, where

$$T_1 = H(k,w) \cdot g^r,\ T_2 = h^r,\ \text{and}\ \ k = \hat{e}(H_1(ID_s), \mathsf{Sk}_{ID_r}^{(2)}).$$

- $1/0 \leftarrow \mathsf{Test}(pp, \mathsf{Sk}_{svr}, ID_s, ID_r, C_{w,s,r}, T_{w,s,r})$: Parse the ciphertext $C_{w,\,s,\,r}$ as (C_1, C_2, C_3) and the trapdoor $T_{w,\,s,\,r}$ as (T_1, T_2). It returns 1 if

$$C_1 \cdot \hat{e}(C_2^{\mathsf{Sk}_{svr}}, T_2) = \hat{e}(T_1^{\mathsf{Sk}_{svr}}, C_3),$$

and 0 otherwise.

Theorem 7. *The proposed dIBAEKS-3 scheme satisfies ciphertext indistinguishability if DBDH-3 assumption holds.*

Theorem 8. *The proposed dIBAEKS-3 scheme satisfies trapdoor indistinguishability if DBDH-3 assumption holds.*

Theorem 9. *The proposed dIBAEKS-3 scheme satisfies designated testability if DBDH-3 assumption holds.*

Security proofs of dIBAEKS-3 scheme are quite similar with those of dIBAEKS scheme in Section 4, hence we omit them here. The difference is that the security of the dIBAEKS-3 scheme is now based on DBDH-3 assumption instead of DBDH assumption.

6. Analysis and comparison

We compare our dIBAEKS and dIBAEKS-3 schemes with related schemes in the literature, e.g. PAEKS [16], ACIBEKS [28] and EdIBEKS [30], in terms of security, computational overhead and communication overhead. Table 1 shows the security comparison between these schemes and our dIBAEKS schemes. In the PKI setting, PAEKS is secure against outside and inside offline KGA, but it does not satisfy the designated testability and suffers from the complex certificate management problem. In identity based encryption setting, ACIBEKS is only secure against outside KGA attacks, and needs a secure channel. EdIBEKS can resist outside offline KGA attacks, but is vulnerable to inside offline KGA attacks. Our dIBAEKSs are secure against inside offline KGA attacks and satisfy the designated testability.

Computational complexity comparison of the five schemes is given in Table 2. We denote by E the evaluation of a modular exponentiation on a $\mathbb{G}_1$ element in the Type-I symmetric bilinear pairing setting, and denote by E_1 and E_2 the evaluation

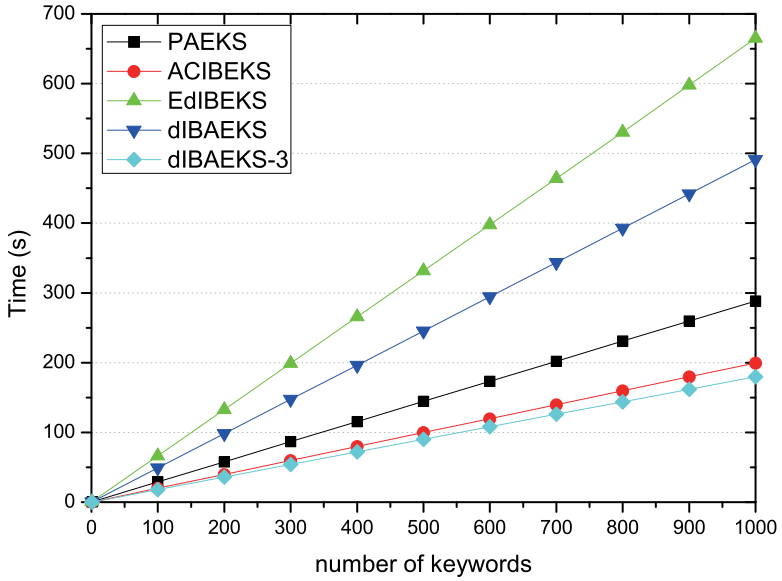

Fig. 3. Running time of PEKS algorithm.

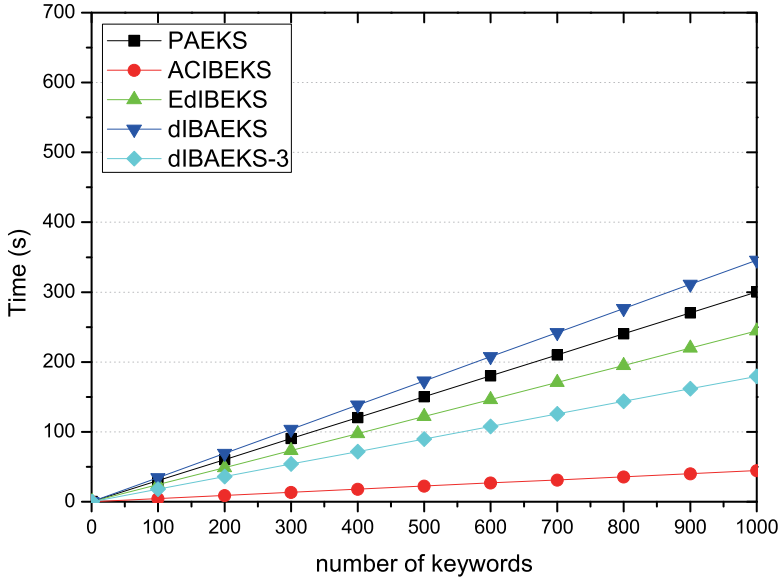

Fig. 4. Running time of Trapdoor algorithm.

of a modular exponentiation on a $\mathbb{G}_1$ element and that on a $\mathbb{G}_2$ element, respectively. Commonly, in both settings, we denote the evaluation of a hash function by H, a bilinear pairing by P, a multiplication by M and a division by D, respectively. We compare the running overhead of PEKS, Trapdoor and Test algorithms of schemes in Table 2. Overall, the computational cost of our dIBAEKS schemes are slightly larger than but still comparable with PAEKS, ACIEKS and EdIBEKS.

Communication overhead of the four schemes is given in Table 3. Denote the length of an element in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and set $\mathbb{Z}_p$ by L_1, L_2, L_T, and L_p, respectively. We unify the lengths of the collision-resistant hash functions in different schemes and denote the length by h. In identity-based encryption, because users' public keys are their identities, which could be of arbitrary length, so we omit the lengths of public keys in identity-based schemes. According to Table 3, the communication overhead of our IBAEKS scheme is almost the same as EdIBEKS, and is slightly larger than PAEKS and ACIEKS.

We implemented the four schemes using C language in a VMware virtual machine (VMware Workstation 14 Pro v14.1.3) [29] with 2GB memory. The host machine is a workstation with a 14-core 3.10 GHz Intel i9-7940X CPU, 64GB memory and Windows 10 Professional OS. We used PBC library [20] to implement the bilinear pairing, and chose Type-A curve (with

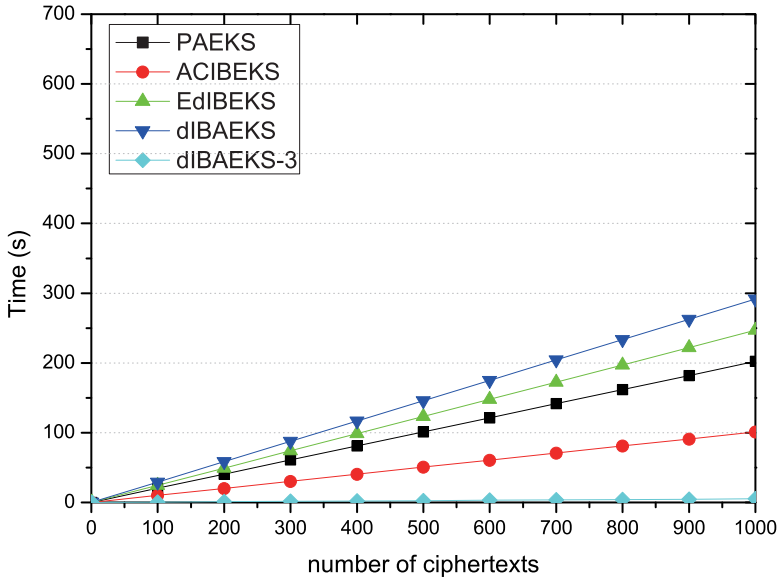

Fig. 5. Running time of Test algorithm.

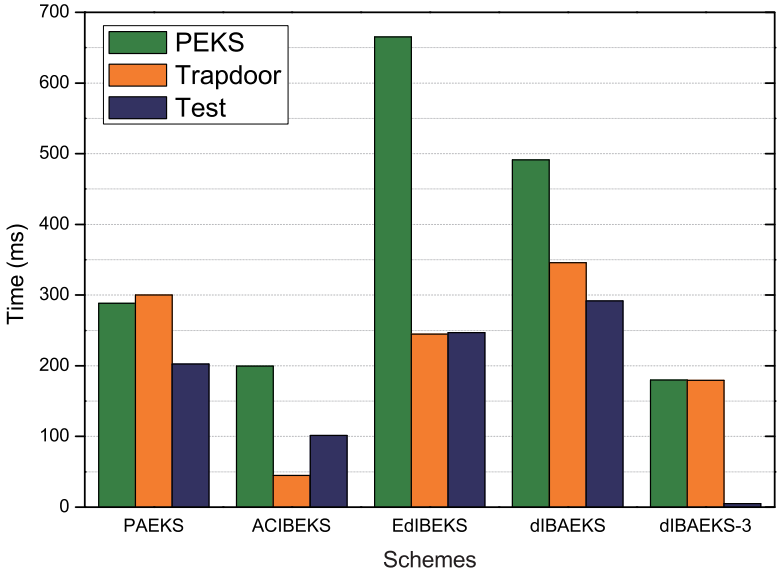

Fig. 6. Average running time of each algorithm.

qbits = 2048) of the PBC in the symmetric setting and Type-F curve (with 451-bit base field) in the asymmetric setting [3,20].

Figs. 3 and 4 show the running time of PEKS and Trapdoor algorithms of the schemes, respectively. The *x*-axis is the number of keywords to be encrypted, and the *y*-axis is the time needed for the PEKS algorithm to encrypt all the keywords (resp. Trapdoor algorithm to generate trapdoors for all the keywords).

Fig. 5 shows the running time of Test algorithm of the schemes. The *x*-axis is the number of ciphertexts to be tested, and the *y*-axis is the running time needed for Test to finish all the tests. Fig. 6 shows the average running time of each algorithm of the schemes. Overall, our dIBAEKS is slightly less computationally efficient than but still comparable with the three related schemes, but it provides stronger security guarantee and better protects users' privacy. Furthermore, our dIBAEKS-3 scheme owns the fastest Test algorithm among the schemes, due to the employment of Type-III bilinear pairing, which means that users need the least time to search over encrypted data in practice.

Table 2
Comparison of computational complexity.

Schemes	PEKS	Trapdoor	Test
PAEKS [16]	3E+H+M	E+H+P	2P+M
ACIBEKS [28]	3E+H+P	E+H	H+P
EdIBEKS [30]	3E+H+P+M	2E+H+D	2E+H+2P+M
dIBAEKS	3E+H+2P	2E+H+P+M	2E+2P+M
dIBAEKS-3	$2E_1+1E_2+H+2P$	$1E_1+1E_2+H+P+M$	$2E_1+2P+M$

Table 3
Comparison of communicational overhead.

Schemes	Pk/ID	Sk_{ID}	C	T_w
PAEKS [16]	$1L_1$	$1L_p$	$2L_1$	$1L_T$
ACIBEKS [28]	*	$1L_1$	$1L_1+h$	$1L_1$
EdIBEKS [30]	*	$1L_1$	$2L_1+h$	$2L_1$
dIBAEKS	*	$1L_1$	$2L_1+1L_T$	$2L_1$
dIBAEKS-3	*	$1L_1+1L_2$	$1L_1+1L_2+1L_T$	$1L_1+1L_2$

*: The length is not fixed.

7. Conclusion and future works

We introduced the notion of designated-server identity-based authenticated encryption with keyword search, and proposed a concrete dIBAEKS scheme. We proved it to be secure against inside offline KGA and achieve designated testability based on a simple number-theoretic assumption. The scheme could be slightly modified to satisfy the CCA-type designated testability. We also showed how to modify the scheme to work in asymmetric bilinear pairing setting to improve the efficiency. dIBAEKS can be well applied to the encrypted email system to protect users' privacy. However, our dIBAEKS and dIBAEKS-3 schemes have the property that a trapdoor can only be used to search over ciphertexts sent from a specific sender, as both ciphertexts and trapdoors are binded with the identity of the sender (and that of the receiver). We consider to construct a more flexible dIBAEKS scheme in which a trapdoor can be used to search over multiple users' encrypted data in the future work.

Acknowledgments

We are grateful to the anonymous reviewers for their invaluable comments, which help us to improve the quality of the paper. This work is supported by National Natural Science Foundation of China (Nos. 61872152, 61472146), Guangdong Natural Science Funds for Distinguished Young Scholar (No. 2014A030306021), Guangdong Program for Special Support of Top-notch Young Professionals (No. 2015TQ01X796), Pearl River Nova Program of Guangzhou (No. 201610010037), and the Graduate Student Overseas Study Program of South China Agricultural University (No. 2018LHPY025).

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, H. Shi, Searchable encryption revisited: consistency properties, relation to anonymous IBE, and extensions, in: Crypto, 3621, Springer, 2005, pp. 205–222.
[2] J. Baek, R. Safavi-Naini, W. Susilo, Public key encryption with keyword search revisited, in: International Conference on Computational Science and Its Applications, Springer, 2008, pp. 1249–1259.
[3] R. Barbulescu, S. Duquesne, Updating key size estimations for pairings, J. Cryptol. (2018), doi:10.1007/s00145-018-9280-5.
[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano, Public key encryption with keyword search, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 506–522, doi:10.1007/978-3-540-24676-3_30.
[5] D. Boneh, M. Franklin, Identity-based encryption from the weil pairing, in: Annual International Cryptology Conference, Springer, 2001, pp. 213–229.
[6] X. Boyen, The uber-assumption family., in: Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1–3, 2008. Proceedings, 2008, pp. 39–56.
[7] J.W. Byun, H.S. Rhee, H.-A. Park, D.H. Lee, Off-line keyword guessing attacks on recent keyword search schemes over encrypted data, in: Workshop on Secure Data Management, Springer, 2006, pp. 75–83, doi:10.1007/11844662_6.
[8] S. Chatterjee, A. Menezes, On cryptographic protocols employing asymmetric pairings – the role of ψ revisited, Discrete Appl. Math. 159 (13) (2011) 1311–1322.
[9] K. Emura, A. Miyaji, M.S. Rahman, K. Omote, Generic constructions of secure-channel free searchable encryption with adaptive security, Secur. Commun. Netw. 8 (8) (2015) 1547–1560.
[10] L. Fang, W. Susilo, C. Ge, J. Wang, A secure channel free public key encryption with keyword search scheme without random oracle, in: Cryptology and Network Security, International Conference, CANS 2009, Kanazawa, Japan, December 12–14, 2009. Proceedings, 2009, pp. 248–258.
[11] L. Fang, W. Susilo, C. Ge, J. Wang, Public key encryption with keyword search secure against keyword guessing attacks without random oracle, Inf. Sci. 238 (2013) 221–241.
[12] Z. Fu, X. Wu, C. Guan, X. Sun, K. Ren, Toward efficient multi-Keyword fuzzy search over encrypted outsourced data with accuracy improvement, IEEE Trans. Inf. Foren. Secur. 11 (12) (2016) 2706–2716, doi:10.1109/TIFS.2016.2596138.
[13] Z. Fu, X. Wu, Q. Wang, K. Ren, Enabling central keyword-based semantic extension search over encrypted outsourced data, IEEE Trans. Inf. Forens. Secur. 12 (12) (2017) 2986–2997, doi:10.1109/TIFS.2017.2730365.
[14] S.D. Galbraith, K.G. Paterson, N.P. Smart, Pairings for cryptographers, Discrete Appl. Math. 156 (16) (2008) 3113–3121.
[15] L. Guo, W.-C. Yau, Efficient secure-channel free public key encryption with keyword search for EMRS in cloud storage, J. Med. Syst. 39 (2) (2015) 11.

[16] Q. Huang, H. Li, An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks, Inf. Sci. 403–404 (2017) 1–14.
[17] Q. Huang, H. Li, An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks, 2018, (Cryptology ePrint Archive: Report 2018/007). http://eprint.iacr.org/.
[18] A. Joux, A one round protocol for tripartite diffieÅ!'chellman, J. Cryptol. 17 (4) (2006) 385–393.
[19] C. Liu, L. Zhu, M. Wang, Y.-a. Tan, Search pattern leakage in searchable encryption: attacks and new construction, Inf. Sci. 265 (2014) 176–188, doi:10.1016/j.ins.2013.11.021.
[20] B. Lynn, et al., Pairing-Based Cryptography Library, 2013 https://crypto.stanford.edu/pbc/.
[21] H.S. Rhee, J.H. Park, D.H. Lee, Generic construction of designated tester public-key encryption with keyword search, Inf. Sci. 205 (2012) 93–109.
[22] H.S. Rhee, J.H. Park, W. Susilo, D.H. Lee, Improved searchable public key encryption with designated tester, in: International Symposium on Information, Computer, and Communications Security, 2009, pp. 376–379.
[23] H.S. Rhee, J.H. Park, W. Susilo, D.H. Lee, Trapdoor security in a searchable public-key encryption scheme with a designated tester, J. Syst. Softw. 83 (5) (2010) 763–771.
[24] H.S. Rhee, W. Susilo, H.-J. Kim, Secure searchable public key encryption scheme against keyword guessing attacks, IEICE Electronics Express 6 (5) (2009) 237–243, doi:10.1587/elex.6.237.
[25] Z.-Y. Shao, B. Yang, On security against the server in designated tester public key encryption with keyword search, Inf. Process. Lett. 115 (12) (2015) 957–961.
[26] J. Shen, T. Zhou, X. Chen, J. Li, W. Susilo, Anonymous and traceable group data sharing in cloud computing, IEEE Trans. Inf. Forens. Secur. PP (99) (2017) 1, doi:10.1109/TIFS.2017.2774439.
[27] D.X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000, 2000, pp. 44–55, doi:10.1109/SECPRI.2000.848445.
[28] K. Tomida, M. Mohri, Y. Shiraishi, Keyword searchable encryption with access control from a certain identity-based encryption, in: Future Information Technology, Springer, 2014, pp. 113–118.
[29] VMware, Vmware, 2017, (http://www.vmware.com).
[30] T.-Y. Wu, T.-T. Tsai, Y.-M. Tseng, Efficient searchable id-based encryption with a designated server, Ann. Telecommun. 69 (7) (2014) 391–402.
[31] Y. Yang, M. Ma, Conjunctive keyword search with designated tester and timing enabled proxy re-encryption function for e-health clouds, IEEE Trans. Inf. Forens. Secur. 11 (4) (2016) 746–759, doi:10.1109/TIFS.2015.2509912.
[32] W.-C. Yau, S.-H. Heng, B.-M. Goi, Off-line keyword guessing attacks on recent public key encryption with keyword search schemes, in: International Conference on Autonomic and Trusted Computing, Springer, 2008, pp. 100–105, doi:10.1007/978-3-540-69295-9_10.
[33] X. Zhang, C. Xu, L. Mu, J. Zhao, Identity-based encryption with keyword search from lattice assumption, China Commun. 15 (4) (2018) 164–178.