An Efficient Public-Key Searchable Encryption Scheme Secure against Inside Keyword Guessing Attacks

Qiong Huang[a,b], Hongbo Li[a]

[a]*College of Mathematics and Informatics, South China Agricultural University, Guangzhou 510642, China*
[b]*Nanjing University of Information Science and Technology, Nanjing 210044, China*

Abstract

How to efficiently search over encrypted data is an important and interesting problem in the cloud era. To solve it, Boneh *et al.* introduced the notion of public key encryption with keyword search (PEKS), in 2004. However, in almost all the PEKS schemes an inside adversary may recover the keyword from a given trapdoor by exhaustively guessing the keywords offline. How to resist the inside keyword guessing attack in PEKS remains a hard problem.

In this paper we propose introduce the notion of *Public-key Authenticated Encryption with Keyword Search* (PAEKS) to solve the problem, in which the data sender not only encrypts a keyword, but also authenticates it, so that a verifier would be convinced that the encrypted keyword can only be generated by the sender. We propose a concrete and efficient construction of PAEKS, and prove its security based on simple and static assumptions in the random oracle model under the given security models. Experimental results show that our scheme enjoys a comparable efficiency with Boneh *et al.*'s scheme.

Keywords: Public Key Authenticated Encryption with Keyword Search, Searchable Encryption, Keyword Guessing Attack, Random Oracle Model

1. Introduction

With the rapid development of cloud computing technology, a large amount of data now has been stored onto the cloud. Since the data owner loses its control of the data, several security and privacy issues arise in cloud storage service, among which data privacy is a very sensitive problem. Encryption is an effective way to protect the data from being leaked. However, traditional search mechanisms do not work for encrypted data. How to efficiently search over encrypted data thus becomes an important and interesting problem, and has attracted many researchers' attention [7, 23, 42, 38, 31, 16, 34, 21, 17, 15, 32, 40].

In 2004, Boneh *et al.* introduced the notion of Public Key Encryption with Keyword Search (PEKS), integrating keyword search functionality into public key encryption. They proposed the first PEKS scheme (denoted by BDOP-PEKS hereafter) [7] and showed that it is secure based on

Email addresses: qhuang@scau.edu.cn (Qiong Huang), hongbo@stu.scau.edu.cn (Hongbo Li)

the bilinear Diffie-Hellman assumption in the random oracle model. The framework of PEKS is illustrated in Figure 1.

There are three parties involved, a data sender called Alice, a data receiver called Bob and a cloud server. Alice has a bunch of sensitive documents $\{F_i\}$ to share with his friend Bob. First, Alice extracts keywords $\{w_{i,j}\}$ from each document F_i, and encrypts the keywords using the PEKS scheme. Besides, Alice encrypts each document with a (possibly another) encryption scheme. Let the ciphertexts be $\{C_{w_{i,j}}\}$ and $\{C_i\}$, respectively. Alice uploads all the ciphertexts onto the cloud server. To search over the encrypted documents whether there is any one containing some keyword w, Bob computes a trapdoor T_w for w using his secret key, and gives it to the cloud server (via a secure channel). With T_w, the server runs the Test algorithm to test each encrypted keyword $C_{w_{i,j}}$ weather it contains the same keyword as T_w or not. If $C_{w_{i,j}}$ matches T_w, the associated document contains w. When the search finishes, the server returns the search result to Bob. Notice that during the search, the server does not know the content of the documents, nor the keyword.

1.1. Keyword Guessing Attack

Ideally, the keyword space is assumed to be at least super-polynomially large. However, in real applications it is usually not that large. Keywords are often chosen from a low-entropy keywords space. Therefore, it may be feasible for the adversary to guess what keywords a document contains by launching the *keyword guessing attack* (KGA) [23, 42]. Roughly speaking, in a KGA attack the adversary tries each possible keyword, encrypts it, and tests the ciphertext with the given trapdoor. If the test succeeds, the adversary knows which keyword is encapsulated in the given trapdoor. Because people usually choose keywords which are used frequently and easy to memorize, the server is able to find out the underling keyword. Assuming in an encrypted mail system, user Alice sends an encrypted email attached with the PEKS ciphertext of a keyword to another user Bob. An adversary can apply the KGA attack on the PEKS ciphertext and may reveal the keyword of the email if the trapdoor sent from Bob to the email server is available. In this way, the adversary may know the theme of the email and thus the user's privacy is leaked. Such an attack is usually launched by the cloud server or any other role inside the cloud service management. Therefore, it is also called *Inside Keyword Guessing Attack* (IKGA).

To resist the KGA attacks, researchers have proposed several ideas and introduced different notions. Roughly speaking, the KGA works for two reasons possibly. First, the adversary could get the trapdoor. Second, it can do the test freely. Therefore, in order to prevent an adversary from launching a KGA attack, one can either protect the trapdoor from being leaked to an outside attacker, for example, setting up a secure channel between the receiver and the server so that only the server can get the trapdoor; or restrict the unauthorized adversary from doing the test, i.e. *designated-tester* PEKS [34, 21] (i.e. only the designated server can do the test), PEKS with *authorization* [38, 31] (i.e. only the authorized one can do the test). However, neither method can prevent an inside adversary from launching the KGA attack. Hence, how to build a (public-key) searchable encryption scheme which is secure against inside keyword guessing attack is still an open problem.

1.2. Related works

Recently, many PEKS schemes and variants have been proposed in the literature. Roughly speaking, these works can be classified into the following types: 1) multi-users access control in PEKS [37, 31, 38], 2) fuzzy keyword search in PEKS [10, 12, 27], 3) flexible keyword search in PEKS [29, 18, 24], and 4) trapdoor privacy in PEKS [42, 23, 1, 28, 35]. Works on trapdoor privacy mainly focus on preventing the adversary from revealing the keywords from given ciphertexts. However, to the best of our knowledge, almost all of them can not resist the inside keyword guessing attack. The only scheme [39] known to counterattack the IKGA attack makes use of two servers and assumes that the two servers do not collude.

There are several ways to leak information about the keyword. For example, the search result, e.g. the number of records satisfying the search query, leaks certain information about the keyword in the trapdoor, which seems to be unavoidable if the server is curious about the keyword. Besides, the trapdoor itself might reveal certain information about the keyword [1]. In existing PEKS schemes, e.g. [7, 4], keywords may be revealed due to the leakage of search patterns [28]. The server is often assumed to be honest-but-curious. It would honestly do its job but try to find more information about the keywords (as well as the document contents). The server has the capability to monitor the communication channels and obtain all encrypted data, indices and trapdoors. Once the server reveals users' search pattern, the search frequency of a keyword would be exposed to the server, leading to privacy leakage.

In 2006 and 2008 Jin *et al.* [23] and Yau *et al.* [42] studied the off-line keyword guessing attack on some existing PEKS schemes, and showed that those schemes are susceptible to attacks by inside adversaries. Recently Fang *et al.* [14] and Guo *et al.* [20] proposed public key encryption with keyword search schemes secure against outside keyword guessing attack. However, an inside adversary would still have the chance to successfully launch a keyword guessing attack. Xu *et al.* [41] proposed encryption schemes with fuzzy keyword search secure against outside keyword guessing attack. Wang *et al.* [39] suggested to use two cloud servers in order to resist the inside keyword guessing attack, as long as the two servers do not collude.

1.3. Our Contributions

In this paper we study the problem of how to resist inside keyword guessing attack in PEKS and try to solve it. Based on the observation that in a KGA attack the server is able to encrypt each keyword candidate and test its ciphertext with the given trapdoor, we suggest a method to prevent the server from doing so. Roughly, we make the following contributions in the paper.

1. We introduce the notion of *Public-key Authenticated Encryption with Keyword Search* (PAEKS), in which the data sender not only encrypts the keyword but also authenticates it, so that the server cannot encrypt a keyword itself and thus cannot launch the inside keyword guessing attack successfully.
2. We present the security model of PAEKS, and propose a concrete construction. Security of the scheme is proved in the given security model based on simple and static assumptions, for example Decisional Bilinear Diffie-Hellman assumption and a simple variant of Decision Linear assumption, with the help of random oracles.

3. We compare our scheme with some other related PEKS schemes in terms of both computation and communication efficiency. We also do some experiments to demonstrate the efficiency of our scheme. Experimental results show that its efficiency is comparable with that of Boneh *et al.*'s scheme.

1.4. Paper Organization

In the next section we give a brief description of some preliminaries. We review the BDOP-PEKS scheme and show its security weakness under the inside keyword guessing attack in Section 3. In Section 4 we define the new notion of PAEKS, and give its security model here. We then propose our concrete construction of PAEKS in Section 5 and prove its security in random oracle model. In Section 6 we compare our scheme with Boneh *et al.*'s scheme and show the experiment results. Finally, we conclude the paper in Section 7.

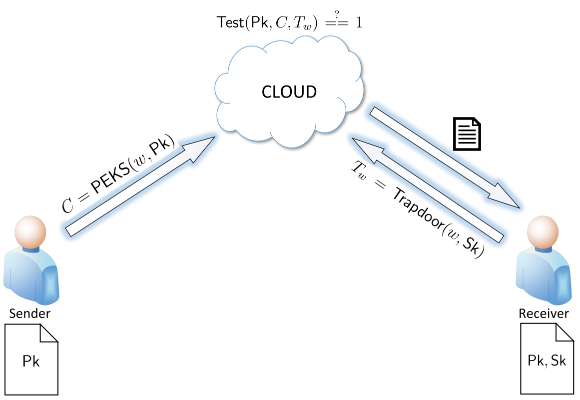

Figure 1: PEKS System Framework

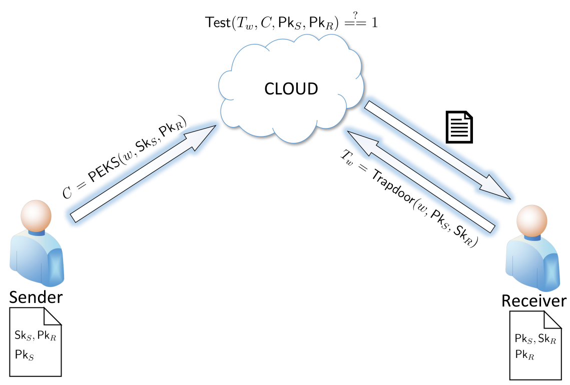

Figure 2: PAEKS System Framework

2. Preliminaries

2.1. Bilinear Pairing

Bilinear pairing [8] plays an important role in the construction of many cryptographic schemes, including our PAEKS scheme. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ be a bilinear pairing, mapping from groups $\mathbb{G}_1$ and $\mathbb{G}_1$ to $\mathbb{G}_T$, where $\mathbb{G}_1$ and $\mathbb{G}_T$ are cyclic groups of the same prime order p. It has the following properties.

- **Bilinearity.** For any $g, h \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}$, $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$.
- **Non-degeneracy.** For any generator $g \in \mathbb{G}_1$, $\hat{e}(g, g) \in \mathbb{G}_T$ is a generator of $\mathbb{G}_T$.
- **Computability.** For any $g, h \in \mathbb{G}_1$, there is an efficient algorithm to compute $\hat{e}(g, h)$.

2.2. Decisional Bilinear Diffie-Hellman Assumption

Decisional Bilinear Diffie-Hellman (DBDH) Problem [25, 9] is stated as below.

> Given a generator $g \in \mathbb{G}_1$ and elements $g^x, g^y, g^z \in \mathbb{G}_1$ where x, y, z are randomly chosen from $\mathbb{Z}_p$, distinguish $\hat{e}(g,g)^{xyz} \in \mathbb{G}_T$ from a random element of $\mathbb{G}_T$.

Definition 1 (DBDH Assumption). The DBDH assumption [25, 9] assumes that for any probabilistic polynomial-time algorithm $\mathcal{A}$, the following holds:

$$|\Pr[\mathcal{A}(g, g^x, g^y, g^z, \hat{e}(g,g)^{xyz}) = 1] - \Pr[\mathcal{A}(g, g^x, g^y, g^z, \hat{e}(g,g)^{r}) = 1]| \leq \text{negl}(\lambda),$$

where the probability is taken over the random choices of $g \in \mathbb{G}_1$, $x, y, z, r \in \mathbb{Z}_p$ and the random coins tossed by $\mathcal{A}$.

2.3. Decision Linear Assumption and a Variant

The well-known Decision Linear (DLIN) problem [6] states as below.

> Given $g, g^x, g^y, g^{xr}, g^{sy} \in \mathbb{G}_1$, distinguish g^{r+s} from g^z, where x, y, z, r, s are randomly chosen from $\mathbb{Z}_p$.

Our public-key authenticated encryption with keyword search makes use of a variant of DLIN. We call it *modified Decision Linear* (mDLIN) problem, and state it as below.

> Given $g, g^x, g^y, g^{xr}, g^{s/y} \in \mathbb{G}_1$, distinguish g^{r+s} from g^z, where z is randomly chosen from $\mathbb{Z}_p$.

The mDLIN problem differs from the DLIN problem only in the fifth component of the input, which is g^{sy} in DLIN problem and is $g^{s/y}$ in mDLIN problem.

It is not hard to find a reduction from the InvDDH problem (which is conjectured to be equivalent to DDH problem) [5] to mDLIN problem in groups without pairings [1]. The converse is believed to be false. Therefore, we believe that the mDLIN problem is intractable, and make the following assumption.

Definition 2 (mDLIN Assumption). The mDLIN assumption assumes that for any probabilistic polynomial-time algorithm $\mathcal{A}$, the following holds:

$$|\Pr[\mathcal{A}(g, g^x, g^y, g^{xr}, g^{s/y}, g^{r+s}) = 1] - \Pr[\mathcal{A}(g, g^x, g^y, g^{xr}, g^{s/y}, g^{z}) = 1]| \leq \text{negl}(\lambda),$$

where the probability is taken over the random choices of $g \in \mathbb{G}_1$, $x, y, z, r, s \in \mathbb{Z}_p$ and the random coins tossed by $\mathcal{A}$.

[1] InvDDH problem is easy in bilinear groups.

3. Public Key Encryption with Keyword Search

3.1. Definition

A public-key encryption scheme with keyword search (PEKS) consists of the following five (probabilistic) polynomial-time algorithms [7].

- $\mathsf{Setup}(\lambda)$: It takes as input the security parameter λ and outputs a global system parameter Param.
- $\mathsf{KeyGen}(\mathsf{Param})$: It takes as input a system parameter Param and outputs a public/secret key pair $(\mathsf{Pk}, \mathsf{Sk})$. The algorithm is run by the data receiver.
- $\mathsf{PEKS}(w, \mathsf{Pk})$: It takes as input a keyword w and the receiver's public key Pk, and outputs a ciphertext C of w. The algorithm is run by the data sender.
- $\mathsf{Trapdoor}(w, \mathsf{Sk})$: It takes as input a keyword w and the secret key Sk, and outputs a corresponding trapdoor T_w. The algorithm is run by the data receiver.
- $\mathsf{Test}(\mathsf{Pk}, C, T_w)$: It takes as input the receiver's public key Pk, a ciphertext C and a trapdoor T_w, and outputs 1 indicating that C and T_w contain the same keyword, and 0 otherwise. The algorithm is run by the cloud server.

Boneh *et al.* proposed the first bilinear pairing based PEKS scheme in 2004 [7]. Hereafter we denote it by BDOP-PEKS. They proved that the BDOP-PEKS scheme is secure against chosen keyword attacks in the random oracle model assuming Bilinear Diffie-Hellman (BDH) problem is intractable. The BDOP-PEKS scheme is described below.

- $\mathsf{Setup}(\lambda)$: The algorithm initializes the global system parameter $\mathsf{Param} = \{p, g, \mathbb{G}_1, \mathbb{G}_T, \hat{e}, H_1, H_2\}$, where $\mathbb{G}_1, \mathbb{G}_T$ are cyclic groups of prime order p, g is the generator of $\mathbb{G}_1$, and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ is a bilinear pairing, $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_T \to \{0,1\}^{\log p}$ are cryptographic hash functions.
- $\mathsf{KeyGen}(\mathsf{Param})$: The algorithm randomly selects $x \leftarrow \mathbb{Z}_p$ and sets $\mathsf{Pk} := (g, h = g^x)$ and $\mathsf{Sk} := x$.
- $\mathsf{PEKS}(w, \mathsf{Pk})$: The algorithm randomly selects $r \leftarrow \mathbb{Z}_p$, and computes the PEKS ciphertext of keyword w as
$$C := (C_1, C_2) = (g^r, H_2(t)),$$
where $t = \hat{e}(H_1(w), h^r)$.
- $\mathsf{Trapdoor}(w, \mathsf{Sk})$: Output $T_w := H_1(w)^{\mathsf{Sk}} = H_1(w)^x$.
- $\mathsf{Test}(\mathsf{Pk}, C, T_w)$: Output 1 if $H_2(\hat{e}(T_w, C_1)) = C_2$, and 0 otherwise.

3.2. *Analysis of BDOP-PEKS against IKGA*

Now we are going to show that Boneh *et al.*'s scheme is vulnerable to the inside keyword guessing attack. Assume the cloud server is honest-but-curious. To recover the keyword contained in a trapdoor T_w, the cloud server chooses a keyword candidate w' from the keyword space, and checks whether w' equals to the keyword w contained in T_w. The keyword guessing attack works as follows [23, 42].

1. Assume that the data receiver wants to search some keyword over the encrypted documents. It computes a trapdoor T_w according to the keyword, and gives it to the server for searching.
2. The server begins an exhaustive search. It chooses a new keyword candidate w' from the keyword space, runs the PEKS algorithm to encrypt w'. Let the ciphertext be C'. It then runs the Test algorithm on input C' and T_w (as well as the receiver's public key). If the algorithm outputs 0, the server choose another keyword candidate and repeat the above again. Otherwise, the server knows that the receiver wanted to search for documents containing the keyword w'.

Due to the fact that the keyword space in real applications is usually not that big, the server would be able to finish the keyword guessing attack in a reasonably short time.

4. Public-key Authenticated Encryption with Keyword Search

4.1. *Definition*

Since the introduction of PEKS, many researches have been working on its security, for example, [35, 4, 13]. However, to the best of our knowledge, none of them could resist keyword guessing attack from inside adversaries (in the single-server setting) [33, 34, 28, 2, 11]. In this section, we introduce the notion of *Public-key Authenticated Encryption with Keyword Search* (PAEKS), which aims to resist inside keyword guessing attack. Our notion differs from PEKS and its variants mainly in that the server in PAEKS could no more encrypt a keyword by itself. Instead, the encryption algorithm requires the sender to authenticate the keyword while encrypting it. To achieve the goal, the PEKS encryption algorithm requires the sender's secret key as part of the input. Therefore, the server could not launch again the keyword guessing attack over the ciphertexts shared between the sender and the receiver, because it could not compute *authenticated* ciphertexts on behalf of the sender.

Same as PEKS, there are three parties in PAEKS as well, a data sender, a data receiver, and a server. Algorithms run by these parties are almost the same as those in PEKS, except that the PEKS encryption algorithm now requires the sender to put its secret key into the input, and that the trapdoor generation algorithm and test algorithm also need the sender's public key as part of the input. Formally, we consider the following definition.

Definition 3 (PAEKS). A Public-key Authenticated Encryption with Keyword Search (P-AEKS) scheme consists of the following (probabilistic) polynomial-time algorithms.

- $\mathsf{Setup}(\lambda)$: The global parameter generation algorithm takes the security parameter λ as input, and outputs global system parameter Param.

- $\mathsf{KeyGen}_S(\mathsf{Param})$: The sender's key pair generation algorithm takes the global system parameter Param as input, and outputs a public/secret key pair $(\mathsf{Pk}_S, \mathsf{Sk}_S)$ of the sender.

- $\mathsf{KeyGen}_R(\mathsf{Param})$: The receiver's key pair generation algorithm takes the global system parameter Param as input, and outputs a public/secret key pair $(\mathsf{Pk}_R, \mathsf{Sk}_R)$ of the receiver.

- $\mathsf{PEKS}(w, \mathsf{Sk}_S, \mathsf{Pk}_R)$: The keyword encryption algorithm takes a keyword w, the receiver's public key Pk_R and the sender's secret key Sk_S as input, and outputs a PEKS ciphertext C of the keyword w.

- $\mathsf{Trapdoor}(w, \mathsf{Pk}_S, \mathsf{Sk}_R)$: The trapdoor generation algorithm takes a keyword w, the sender's public key Pk_S and the receiver's secret key Sk_R as input, and outputs a trapdoor T_w.

- $\mathsf{Test}(T_w, C, \mathsf{Pk}_S, \mathsf{Pk}_R)$: The test algorithm takes a trapdoor T_w, a PEKS ciphertext C, the sender's public key PK_S and the receiver's public key Pk_R, and outputs 1 if C and T_w contain the same keyword, and 0 otherwise.

Correctness requires that for any honestly generated key pairs $(\mathsf{Pk}_S, \mathsf{Sk}_S)$ and $(\mathsf{Pk}_R, \mathsf{Sk}_R)$, for any keyword w,

$$\mathsf{Test}(T_w, C, \mathsf{Pk}_S, \mathsf{Pk}_R) = 1$$

holds with probability 1, where $C \leftarrow \mathsf{PEKS}(w, \mathsf{Sk}_S, \mathsf{Pk}_R)$ and $T_w \leftarrow \mathsf{Trapdoor}(w, \mathsf{Pk}_S, \mathsf{Sk}_R)$.

The global system parameter generation algorithm is run in a trusted way so that everyone in the system would trust the parameters. The sender and the receiver run their own key generation algorithm once. To encrypt a keyword w, the sender runs the PEKS algorithm to generate the corresponding ciphertext C, which is then uploaded to the server (along with the encrypted documents containing w). To search over the ciphertexts shared by the sender (to the receiver), the receiver runs the trapdoor generation algorithm to generate a trapdoor for some keyword w, and gives it to the server via a secure channel. Given the trapdoor, the server runs the Test algorithm to search over those ciphertexts shared between the sender and the receiver, and returns the search result to the receiver. Figure 2 (page 4) shows the system framework of PAEKS.

Remark. The notion of PAEKS prevents a third-party from generating a valid ciphertext, and provides both confidentiality and integrity of the plaintext. In this sense it could be viewed as a public-key variant of *authenticated encryption* [26]. On the other hand, PAEKS is closely related to the notion of *Signcrpytion* [43] which guarantees confidentiality and integrity of the plaintext simultaneously as well. However, as we will show below, PAEKS imposes different requirements on security than signcryption.

4.2. Security Models

Similar with PEKS, security of PAEKS requires that there is no probabilistic polynomial-time adversary which could distinguish trapdoors or ciphertexts. Formally, we consider the following games, which are played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Game 1: Trapdoor Privacy

1. Given a security parameter λ, the challenger $\mathcal{C}$ generates the global system parameter Param, and prepares the challenge sender's public key Pk_S and the challenge receiver's public key Pk_R. It then invokes the adversary $\mathcal{A}$ on input $(\mathsf{Param}, \mathsf{Pk}_S, \mathsf{Pk}_R)$.
2. The adversary is allowed to adaptively issue queries to the following oracles for polynomially many times.
   - **Trapdoor Oracle** $\mathcal{O}_T$: Given a sender's public key $\tilde{\mathsf{Pk}}_S$ and a keyword w, the oracle computes the corresponding trapdoor T_w with respect to $\tilde{\mathsf{Pk}}_S$ and Sk_R, and returns T_w to $\mathcal{A}$.
   - **Ciphertext Oracle** $\mathcal{O}_C$: Given a receiver's public key $\tilde{\mathsf{Pk}}_R$ and a keyword w, the oracle computes the corresponding ciphertext C with respect to Sk_S and $\tilde{\mathsf{Pk}}_R$, and returns C to $\mathcal{A}$.
3. At some point, $\mathcal{A}$ chooses two keywords (w_0^*, w_1^*) such that (Pk_S, w_0^*), (Pk_S, w_1^*) have not been queried for trapdoors and (Pk_R, w_0^*), (Pk_R, w_1^*) have not been queried for ciphertexts, and submits them to $\mathcal{C}$ as the challenge keywords. The challenger $\mathcal{C}$ randomly chooses a bit $b \in \{0, 1\}$, computes $T_{w_b^*} \leftarrow \mathsf{Trapdoor}(w_b^*, \mathsf{Pk}_S, \mathsf{Sk}_R)$ and returns it to $\mathcal{A}$.
4. The adversary continues to issuing queries to $\mathcal{O}_T$ and $\mathcal{O}_C$ as above, with the restriction that neither (Pk_S, w_0^*) nor (Pk_S, w_1^*) could be submitted to $\mathcal{O}_T$ and neither (Pk_R, w_0^*) nor (Pk_R, w_1^*) could be submitted to $\mathcal{O}_C$.
5. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. It wins the game if and only if $b' = b$.

We define $\mathcal{A}$'s advantage of successfully distinguishing the trapdoors of PAEKS as

$$\mathrm{Adv}_{\mathcal{A}}^{T}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

Game 2: Ciphertext Indistinguishability

1. The challenger $\mathcal{C}$ generates Param and prepares challenge $\mathsf{Pk}_S, \mathsf{Pk}_R$ as in Game 1. It then invokes the adversary $\mathcal{A}$ on input $(\mathsf{Param}, \mathsf{Pk}_S, \mathsf{Pk}_R)$.
2. The adversary issues queries to oracles $\mathcal{O}_T$ and $\mathcal{O}_C$ as in Game 1.
3. At some point, $\mathcal{A}$ chooses two keywords (w_0^*, w_1^*) such that (Pk_S, w_0^*), (Pk_S, w_1^*) have not been queried for trapdoors and (Pk_R, w_0^*), (Pk_R, w_1^*) have not been queried for ciphertexts, and submits them to $\mathcal{C}$ as the challenge keywords. The challenger $\mathcal{C}$ randomly chooses a bit $b \in \{0, 1\}$, computes $C_b^* \leftarrow \mathsf{PEKS}(w_b^*, \mathsf{Sk}_S, \mathsf{Pk}_R)$ and returns it to $\mathcal{A}$.
4. The adversary continues to issue queries to $\mathcal{O}_T$ and $\mathcal{O}_C$ as above, with the restriction that neither (Pk_S, w_0^*) nor (Pk_S, w_1^*) could be submitted to $\mathcal{O}_T$ and neither (Pk_R, w_0^*) nor (Pk_R, w_1^*) could be submitted to $\mathcal{O}_C$.
5. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. It wins the game if and only if $b' = b$.

We define $\mathcal{A}$'s advantage of successfully distinguishing the ciphertexts of PAEKS as

$$\mathrm{Adv}_{\mathcal{A}}^{C}(\lambda) = \left| \Pr[b' = b] - \frac{1}{2} \right|.$$

Definition 4. A PAEKS scheme is *semantically secure against inside keyword guessing attack* if for any probabilistic polynomial-time adversary $\mathcal{A}$, both $\mathrm{Adv}_{\mathcal{A}}^{T}(\lambda)$ and $\mathrm{Adv}_{\mathcal{A}}^{C}(\lambda)$ are negligible in the security parameter λ.

Remark. Our security model has a major difference with other PEKS schemes. In our model, the adversary is given access to not only the trapdoor generation oracle, but also the PEKS ciphertext generation oracle. This is because in PAEKS any third party other than the sender and the receiver, is not able to (authenticatively) encrypt a keyword with respect to the sender and the receiver. Furthermore, the security models above are considered in the *multi-user* setting and the *chosen-key* model [22], in which there are multiple data senders and multiple data receivers, and the adversary is allowed to choose any (rogue) public key to use even without knowing the corresponding secret key.

5. Our PAEKS Scheme

5.1. Construction

Below we are going to present our construction of PAEKS. The scheme is based on the DBDH assumption and mDLIN assumption, and makes use of a bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, as well as a collision resistant hash function $H : \{0,1\}^* \to \mathbb{G}_1$. Our PAEKS scheme works as follows.

- $\mathsf{Setup}(\lambda)$: Select a bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, where $\mathbb{G}_1, \mathbb{G}_T$ are cyclic groups of prime order p. Choose two random generators g, h of $\mathbb{G}_1$. Choose a cryptographic hash function $H : \{0,1\}^* \to \mathbb{G}_1$. Return $\mathsf{Param} = (\mathbb{G}_1, \mathbb{G}_T, p, g, h, \hat{e}, H)$.
- $\mathsf{KeyGen}_S(\mathsf{Param})$: Randomly select $y \leftarrow \mathbb{Z}_p$ and set $\mathsf{Pk}_S := g^y$ and $\mathsf{Sk}_S := y$. Return $(\mathsf{Pk}_S, \mathsf{Sk}_S)$.
- $\mathsf{KeyGen}_R(\mathsf{Param})$: Randomly select $x \leftarrow \mathbb{Z}_p$ and set $\mathsf{Pk}_R := g^x$ and $\mathsf{Sk}_R := x$. Return $(\mathsf{Pk}_R, \mathsf{Sk}_R)$.
- $\mathsf{PEKS}(w, \mathsf{Sk}_S, \mathsf{Pk}_R)$: Randomly select a number $r \leftarrow \mathbb{Z}_p$, and compute

  $$C_1 := H(\mathsf{Pk}_S, \mathsf{Pk}_R, w)^{\mathsf{Sk}_S} \cdot h^r, \quad C_2 := {\mathsf{Pk}_R}^r.$$

  Output the ciphertext $C := (C_1, C_2)$.
- $\mathsf{Trapdoor}(w, \mathsf{Pk}_S, \mathsf{Sk}_R)$: Output the trapdoor $T_w := \hat{e}(H(\mathsf{Pk}_S, \mathsf{Pk}_R, w)^{\mathsf{Sk}_R}, \mathsf{Pk}_S)$.
- $\mathsf{Test}(T_w, C, \mathsf{Pk}_S, \mathsf{Pk}_R)$: Output 1 if

  $$T_w \cdot \hat{e}(C_2, h) = \hat{e}(C_1, \mathsf{Pk}_R),$$

  and 0 otherwise.

Correctness. Let the receiver's key pair be $(\mathsf{Pk}_R, \mathsf{Sk}_R) = (g^x, x)$ and the sender's key pair be $(\mathsf{Pk}_S, \mathsf{Sk}_S) = (g^y, y)$. Let w be the keyword contained in C and w' be that in T_w. Then we have the followings.

$$\begin{aligned}
C_1 &= H(g^y, g^x, w)^{\mathsf{Sk}_S} \cdot h^r = H(g^y, g^x, w)^y \cdot h^r \\
C_2 &= \mathsf{Pk}_R{}^r = g^{xr} \\
T_w &= \hat{e}(H(g^y, g^x, w')^{\mathsf{Sk}_R}, \mathsf{Pk}_S) = \hat{e}(H(g^y, g^x, w')^x, g^y) \\
T_w \cdot \hat{e}(C_2, h) &= \hat{e}(H(g^y, g^x, w')^x, g^y) \cdot \hat{e}(g^{xr}, h) \\
&= \hat{e}(H(g^y, g^x, w')^y, g^x) \cdot \hat{e}(h^r, g^x) \\
e(C_1, \mathsf{Pk}_R) &= \hat{e}(H(g^y, g^x, w)^y \cdot h^r, g^x) \\
&= \hat{e}(H(g^y, g^x, w)^y, g^x) \cdot \hat{e}(h^r, g^x)
\end{aligned}$$

We take into account the following two cases.

1. If the keywords w and w' are the same, i.e., $w = w'$, we have $H(g^y, g^x, w) = H(g^y, g^x, w')$ and thus the equation
$$T_w \cdot \hat{e}(C_2, h) = \hat{e}(C_1, \mathsf{Pk}_R)$$
holds.
2. If the keywords w and w' are different, i.e., $w \neq w'$, we have $H(g^y, g^x, w) \neq H(g^y, g^x, w')$ due to the collision resistance of hash function H, and thus
$$T_w \cdot \hat{e}(C_2, h) \neq \hat{e}(C_1, \mathsf{Pk}_R).$$

Therefore, our PAEKS scheme is correct.

5.2. Security Proof

In this section we provide a security proof of our PAEKS scheme. Formally, we have the following theorem.

Theorem 1. *Our PAEKS scheme is semantically secure against inside keyword guessing attack in the random oracle model, assuming DBDH problem and mDLIN problem are intractable.*

The theorem simply follows from the lemmas below.

Lemma 1. *For any PPT adversary $\mathcal{A}$ against the trapdoor privacy of our PAEKS scheme, its advantage $\mathrm{Adv}_{\mathcal{A}}^{T}(\lambda)$ is negligible if DBDH assumption holds.*

PROOF. Assume that there is a PPT adversary $\mathcal{A}$ which breaks the trapdoor privacy of our PAEKS scheme with a non-negligible advantage ϵ_T, we then use it to construct another PPT algorithm $\mathcal{B}$ to solve the DBDH problem. The reduction is shown in Figure 3.

The algorithm $\mathcal{B}$ takes as input a DBDH problem instance, e.g. $(\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, g^x, g^y, g^z, Z)$, where x, y, z are randomly chosen from $\mathbb{Z}_p$, and Z is either equal to $\hat{e}(g, g)^{xyz}$ or a random element

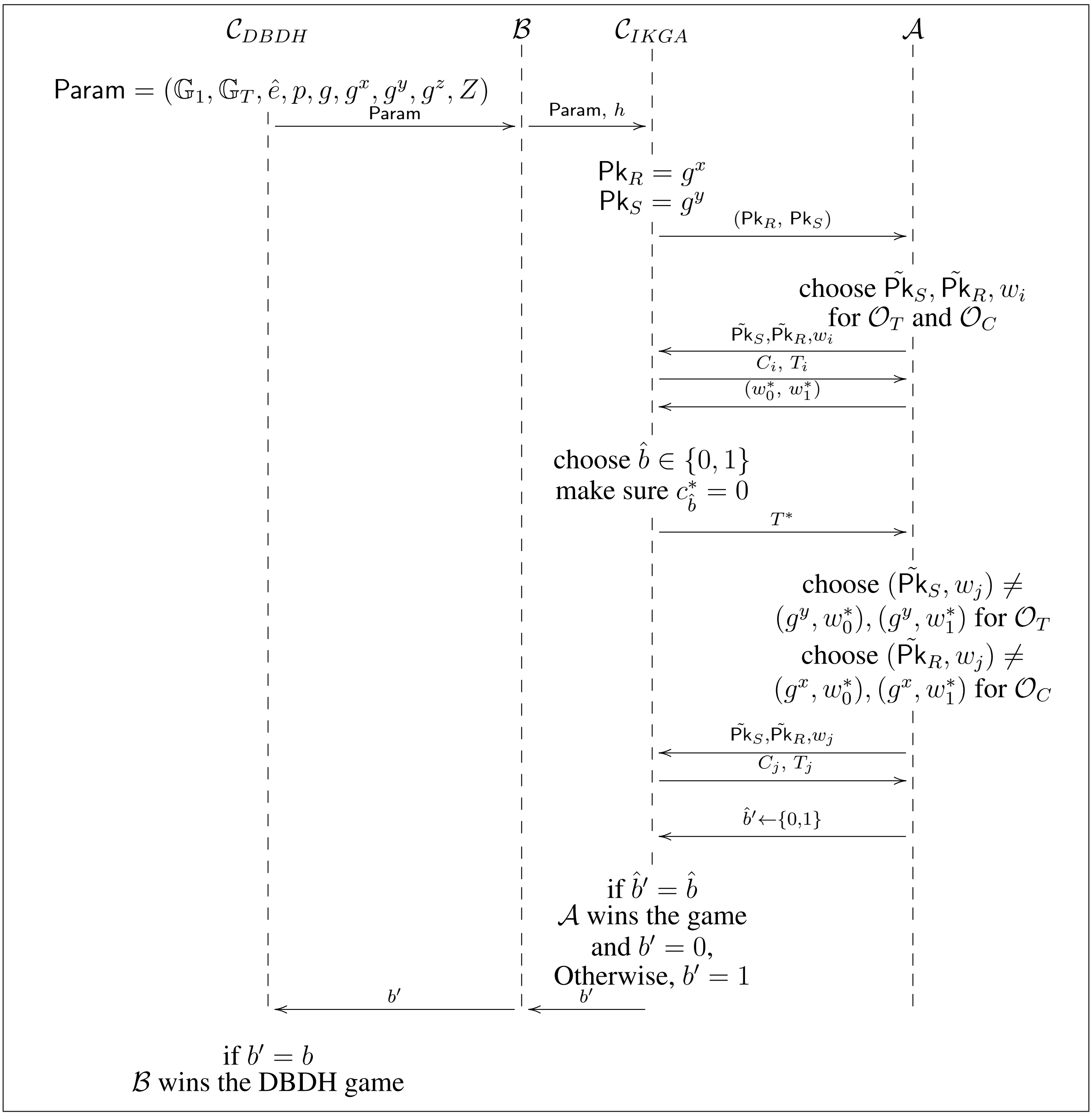

Figure 3: Trapdoor Privacy Reduction

of $\mathbb{G}_T$. Let b be a bit such that $b = 0$ if $Z = \hat{e}(g, g)^{xyz}$, and $b = 1$ if Z is random. $\mathcal{B}$ randomly selects $h \in \mathbb{G}_1$, and sets the public parameter as $\mathsf{Param} = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, h)$, the challenged receiver's public key as $\mathsf{Pk}_R = g^x$ and, the challenged sender's public key as $\mathsf{Pk}_S = g^y$, which implies $\mathsf{Sk}_R = x$ and $\mathsf{Sk}_S = y$. It then runs $\mathcal{A}$ on input $(\mathsf{Param}, \mathsf{Pk}_S, \mathsf{Pk}_R)$, and begins to answer the adversary's queries. We make the following assumptions for simplicity.

1. The adversary issues at most q_H, q_T, q_C queries to the hash oracle $\mathcal{O}_H$, the trapdoor oracle $\mathcal{O}_T$ and the ciphertext oracle $\mathcal{O}_C$, respectively.
2. The adversary does not repeat a query to an oracle.
3. The adversary would not issue a query $(\tilde{\mathsf{Pk}}_S, w)$ to $\mathcal{O}_T$ nor issue $(\tilde{\mathsf{Pk}}_R, w)$ to $\mathcal{O}_C$ before issuing $(\tilde{\mathsf{Pk}}_S, \mathsf{Pk}_R, w)$ or $(\mathsf{Pk}_S, \tilde{\mathsf{Pk}}_R, w)$ to $\mathcal{O}_H$.

The oracles are simulated by $\mathcal{B}$ as follows.

- **Hash Oracle** $\mathcal{O}_H$. Given a tuple $(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i)$, $\mathcal{B}$ randomly selects $a_i \leftarrow \mathbb{Z}_p$, and tosses a biased coin c_i such that $\Pr[c_i = 0] = \delta$, which will be determined later. $\mathcal{B}$ sets

$$h_i = g^z \cdot g^{a_i}$$

  if $c_i = 0$, and sets

$$h_i = g^{a_i}$$

  otherwise. It then adds the tuple $\langle(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i), h_i, a_i, c_i\rangle$ into a list L_H (which is initially empty), and returns $H(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i) = h_i$ as the hash value of $(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i)$ to $\mathcal{A}$.

- **Trapdoor Oracle** $\mathcal{O}_T$. Given $(\tilde{\mathsf{Pk}}_S, w_i)$, $\mathcal{B}$ retrieves $\langle(\tilde{\mathsf{Pk}}_S, \mathsf{Pk}_R, w_i), h_i, a_i, c_i\rangle$ from list L_H. If $c_i = 0$, it aborts and outputs a random bit b' as its guess of b. Otherwise, it computes the trapdoor T_i as

$$T_{w_i} = \hat{e}(\mathsf{Pk}_R, \tilde{\mathsf{Pk}}_S)^{a_i} = \hat{e}(H(\tilde{\mathsf{Pk}}_S, \mathsf{Pk}_R, w_i), \tilde{\mathsf{Pk}}_S)^{\mathsf{Sk}_R}.$$

  It is easy to see that T_{w_i} is a correct trapdoor. $\mathcal{B}$ then returns T_{w_i} to the adversary.

- **Ciphertext Oracle** $\mathcal{O}_C$. Given $(\tilde{\mathsf{Pk}}_R, w_i)$, $\mathcal{B}$ randomly selects $r_i \leftarrow \mathbb{Z}_p$ and retrieves the tuple $\langle(\mathsf{Pk}_S, \tilde{\mathsf{Pk}}_R, w_i), h_i, a_i, c_i\rangle$ from list L_H. If $c_i = 0$, it aborts and outputs a random bit b' as its guess of b. Otherwise, it computes the ciphertext as

$$C_{w_i} = (C_{w_i,1}, C_{w_i,2}) = ((\mathsf{Pk}_S)^{a_i} \cdot h^{r_i}, (\tilde{\mathsf{Pk}}_R)^{r_i}).$$

  Obviously C_{w_i} is a well distributed ciphertext. $\mathcal{B}$ returns it to $\mathcal{A}$.

At some point the adversary submits two keywords w_0^*, w_1^*, where (g^y, w_0^*), (g^y, w_1^*) have not been queried to oracle $\mathcal{O}_T$ and (g^x, w_0^*), (g^x, w_1^*) have not been queried to oracle oracle $\mathcal{O}_C$. $\mathcal{B}$ retrieves the tuples $\langle(g^y, g^x, w_0^*), h_0^*, a_0^*, c_0^*\rangle$ and $\langle(g^y, g^x, w_1^*), h_1^*, a_1^*, c_1^*\rangle$ from the list L_H, and computes the challenge trapdoor T^* as follows.

- In case $c_0^* = c_1^* = 1$, $\mathcal{B}$ aborts and outputs a random bit b' as its guess of b.

- In case $c_0^* = 0$ or $c_1^* = 0$, let $\hat{b}$ be the bit such that $c_{\hat{b}}^* = 0$. $\mathcal{B}$ computes the trapdoor $T^* = Z \cdot \hat{e}(g^x, g^y)^{a_{\hat{b}}^*}$. If $Z = \hat{e}(g,g)^{xyz}$, then $T^* = \hat{e}(g,g)^{xy(z+a_{\hat{b}}^*)} = \hat{e}(h_{\hat{b}'}, g^{xy})$. If Z is a random element of $\mathbb{G}_T$, so is T^*.

$\mathcal{B}$ returns T^* to the adversary, which then continues to issuing queries to the oracles, with the restriction that it could not issue (g^y, w_0^*), (g^y, w_1^*) to $\mathcal{O}_T$ and could not issue (g^x, w_0^*), (g^x, w_1^*) to $\mathcal{O}_C$. Finally, $\mathcal{A}$ outputs a bit $\hat{b}'$. If $\hat{b}' = \hat{b}$, $\mathcal{B}$ outputs $b' = 0$; otherwise, it outputs $b' = 1$.

Denote by abt the event that $\mathcal{B}$ aborts during the game. There are two cases in which $\mathcal{B}$ aborts, as follows.

1. $c_i = 0$ in the simulation of $\mathcal{O}_T$ and $\mathcal{O}_C$. Denote it by abt_1. Due to that each c_i is selected randomly and independently, the probability that abt_1 does not happen is

$$\Pr[\overline{\mathsf{abt}_1}] = (1-\delta)^{q_T+q_C}.$$

2. $c_0^* = c_1^* = 1$ in the generation of the challenge trapdoor. Denote it by abt_2. The probability that abt_2 does not happen is

$$\Pr[\overline{\mathsf{abt}_2}] = 1-(1-\delta)^2.$$

Therefore, the probability that $\mathcal{B}$ does not abort in the game is bounded by

$$\Pr[\overline{\mathsf{abt}}] = \Pr[\overline{\mathsf{abt}_1}] \cdot \Pr[\overline{\mathsf{abt}_2}] = (1-\delta)^{q_T+q_C} \cdot (1-(1-\delta)^2).$$

When $\delta = 1 - \sqrt{\frac{q_T+q_C}{q_T+q_C+2}}$, the probability $\Pr[\overline{\mathsf{abt}}]$ takes the maximum value

$$\Pr[\overline{\mathsf{abt}}] = \left(\frac{q_T+q_C}{q_T+q_C+2}\right)^{(q_T+q_C)/2} \cdot \frac{2}{q_T+q_C+2},$$

which is approximately equal to $\frac{2}{(q_T+q_C)e}$. and thus non-negligible.

It is readily seen that if $\mathcal{B}$ does not abort in the game, the view of $\mathcal{A}$ is identically distributed as in a real attack. Conditioned on that $\mathcal{B}$ does not abort, if $\mathcal{A}$ succeeds in breaking the trapdoor privacy of our scheme, $\mathcal{B}$ also succeeds in telling Z is equal to $\hat{e}(g,g)^{xyz}$ or a random element of $\mathbb{G}_T$. Therefore, the probability that $\mathcal{B}$ succeeds in guessing the bit b (and thus solves the DBDH problem) is

$$\begin{aligned}
\Pr[b' = b] &= \Pr[b' = b \wedge \mathsf{abt}] + \Pr[b' = b \wedge \overline{\mathsf{abt}}] \\
&= \Pr[b' = b | \mathsf{abt}]\Pr[\mathsf{abt}] + \Pr[b' = b | \overline{\mathsf{abt}}]\Pr[\overline{\mathsf{abt}}] \\
&= \frac{1}{2}(1 - \Pr[\overline{\mathsf{abt}}]) + (\epsilon_T + \frac{1}{2}) \cdot \Pr[\overline{\mathsf{abt}}] \\
&= \frac{1}{2} + \epsilon_T \cdot \Pr[\overline{\mathsf{abt}}].
\end{aligned}$$

If ϵ_T is non-negligible, so is $|\Pr[b' = b] - 1/2|$.

This completes the proof. □

Lemma 2. *For any PPT adversary against the ciphertext indistinguishability of our PAEKS scheme, its advantage* $\mathrm{Adv}_{\mathcal{A}}^{C}(\lambda)$ *is negligible if mDLIN assumption holds.*

PROOF. Assume that there is a PPT adversary $\mathcal{A}$ which breaks the ciphertext indistinguishability of our PAEKS scheme with a non-negligible advantage ϵ_C, we use it to construct another PPT algorithm $\mathcal{B}$ to solve the mDLIN problem. The reduction is shown in Figure 4.

The algorithm $\mathcal{B}$ takes as input an mDLIN problem instance, e.g. $(\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, g^x, g^y, g^{rx}, g^{s/y}, Z)$ where x, y, r, s are randomly chosen from $\mathbb{Z}_p$, and Z is either equal to g^{r+s} or a random element of $\mathbb{G}_1$. Let b be a bit such that $b = 0$ if $Z = g^{r+s}$, and $b = 1$ if Z is random. $\mathcal{B}$ randomly selects $t \leftarrow \mathbb{Z}_q$, and sets the public parameter as $\mathsf{Param} = (\mathbb{G}_1, \mathbb{G}_T, \hat{e}, p, g, h = g^t)$, the challenged receiver's public key as $\mathsf{Pk}_R = g^x$, and the challenged sender's public key as $\mathsf{Pk}_S = g^y$, which implies $\mathsf{Sk}_R = x$ and $\mathsf{Sk}_S = y$. It then runs $\mathcal{A}$ on input $(\mathsf{Param}, \mathsf{Pk}_R, \mathsf{Pk}_S)$, and begins to answer the adversary's queries. Here we make the same assumptions as in Lemma 1. The oracles are then simulated by $\mathcal{B}$ as follows.

- **Hash Oracle** $\mathcal{O}_H$. Given a tuple $(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i)$, $\mathcal{B}$ randomly selects $a_i^* \leftarrow \mathbb{Z}_p$, computes $a_i = a_i^* \cdot t$, and tosses a biased coin c_i such that $\Pr[c_i = 0] = \delta$, which will be determined later. $\mathcal{B}$ sets
$$h_i = g^{s/y} \cdot g^{a_i}$$
if $c_i = 0$, and sets
$$h_i = g^{a_i}$$
otherwise. It then adds the tuple $\langle(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i), h_i, a_i, c_i\rangle$ into a list L_H (which is initially empty), and returns $H(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i) = h_i$ as the hash value of $(\tilde{\mathsf{Pk}}_S, \tilde{\mathsf{Pk}}_R, w_i)$ to $\mathcal{A}$.

- **Trapdoor Oracle** $\mathcal{O}_T$. The trapdoor oracle answers the adversary's queries in the same way as in the proof of Lemma 1.

- **Ciphertext Oracle** $\mathcal{O}_C$. The ciphertext oracle answers the adversary's queries in the same way as in the proof of Lemma 1.

At some point the adversary submits two keywords w_0^*, w_1^*, where (g^y, w_0^*), (g^y, w_1^*) have not been queried to oracle $\mathcal{O}_T$ and (g^x, w_0^*), (g^x, w_1^*) have not been queried to oracle oracle $\mathcal{O}_C$. $\mathcal{B}$ retrieves the tuples $\langle(g^y, g^x, w_0^*), h_0^*, a_0^*, c_0^*\rangle$ and $\langle(g^y, g^x, w_1^*), h_1^*, a_1^*, c_1^*\rangle$ from the list L_H, and computes the challenge ciphertext C^* as follows.

- In case $c_0^* = c_1^* = 1$, $\mathcal{B}$ aborts and outputs a random bit b' as its guess of b.

- In case $c_0^* = 0$ or $c_1^* = 0$, let $\hat{b}$ be the bit such that $c_{\hat{b}}^* = 0$, and we have $h_{\hat{b}}^* = g^{s/y} \cdot g^{a_{\hat{b}}^*} = g^{(s+y\cdot a_{\hat{b}}^*)/y}$. $\mathcal{B}$ computes the ciphertext
$$C^* = (C_1^*, C_2^*) = (Z \cdot g^{a_{\hat{b}}^*} \cdot g^{y\cdot a_{\hat{b}}^*}, (g^{xr} \cdot (g^x)^{a_{\hat{b}}^*})^{\frac{1}{t}}).$$
If $Z = g^{r+s}$, then
$$C_1^* = g^{r+s} \cdot g^{a_{\hat{b}}^*} \cdot g^{y\cdot a_{\hat{b}}^*} = g^{(s+y\cdot a_{\hat{b}}^*)} g^{(r+a_{\hat{b}}^*)} = h_{\hat{b}}^{*y} \cdot h^{(r+a_{\hat{b}}^*)/t}, \quad C_2^* = g^{x(r+a_{\hat{b}}^*)/t},$$
where $(r + a_{\hat{b}}^*)/t$ is a random number in $\mathcal{A}$'s view. If Z is a random element of $\mathbb{G}_1$, so is C_1^*. Besides, C_2^* is also random in $\mathcal{A}$'s view because of the randomness of $a_{\hat{b}}^*$.

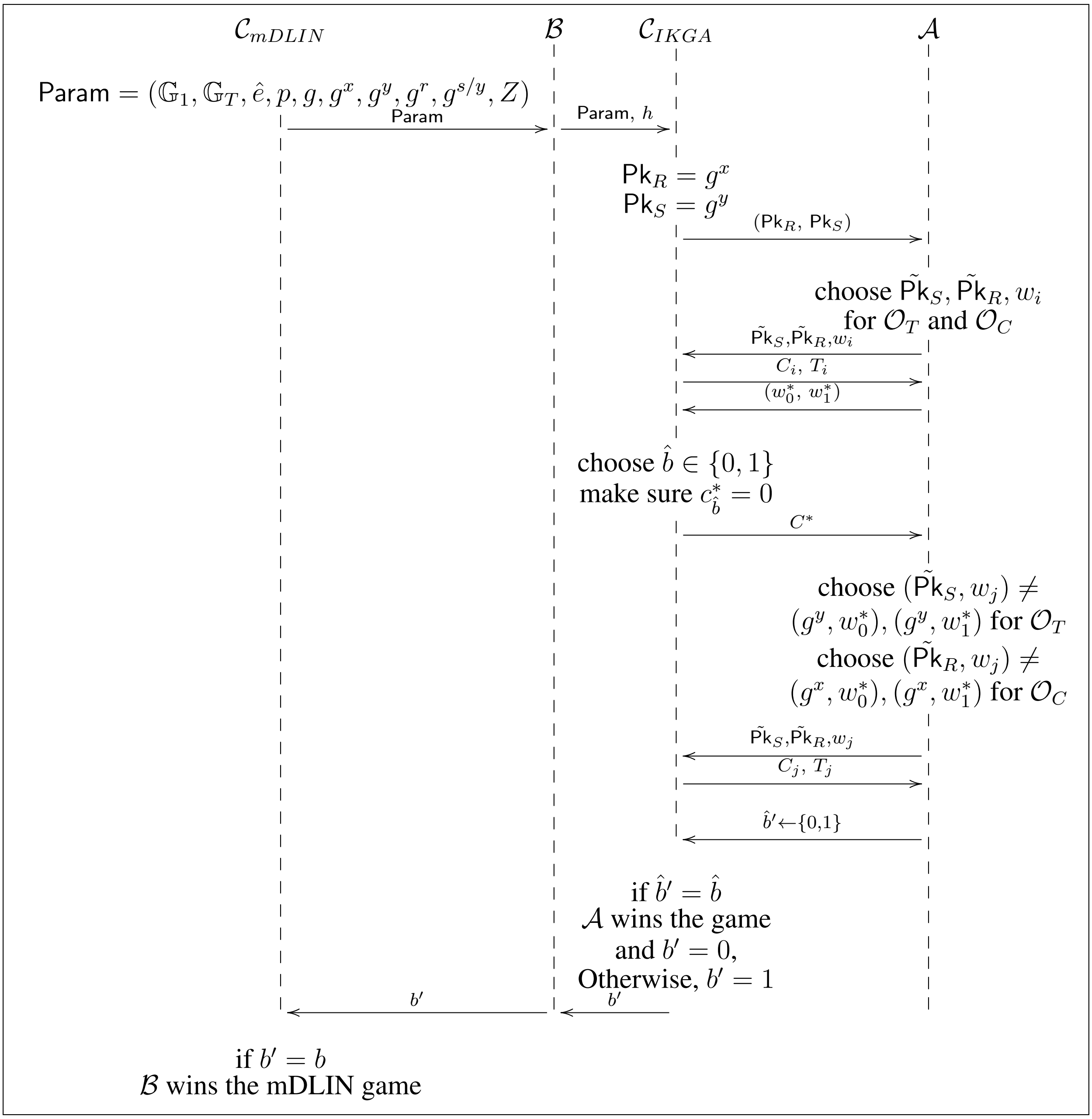

Figure 4: Ciphertext Indistinguishability Reduction

$\mathcal{B}$ returns C^* to the adversary, which then continues to issuing queries to the oracles with the restriction that it could not issue (g^y, w_0^*), (g^y, w_1^*) to $\mathcal{O}_T$ and could not issue (g^x, w_0^*), (g^x, w_1^*) to $\mathcal{O}_C$. Finally, $\mathcal{A}$ outputs a bit $\hat{b}'$. If $\hat{b}' = \hat{b}$, $\mathcal{B}$ outputs $b' = 0$; otherwise, it outputs $b' = 1$.

Denote by abt the event that $\mathcal{B}$ aborts during the game. The probability that $\mathcal{B}$ does not abort in the game is the same as that in the proof of Lemma 1. Therefore, when $\delta = 1 - \sqrt{\frac{q_T+q_C}{q_T+q_C+2}}$, the probability $\Pr[\overline{\mathsf{abt}}]$ takes the maximum value

$$\Pr[\overline{\mathsf{abt}}] = \left(\frac{q_T + q_C}{q_T + q_C + 2}\right)^{(q_T+q_C)/2} \cdot \frac{2}{q_T + q_C + 2},$$

which is approximately equal to $\frac{2}{(q_T+q_C)e}$, and thus non-negligible.

It is readily seen that if $\mathcal{B}$ does not abort in the game, the view of $\mathcal{A}$ is identically distributed as in a real attack. Conditioned on that $\mathcal{B}$ does not abort, if $\mathcal{A}$ succeeds in breaking the semantic security of our scheme, $\mathcal{B}$ also succeeds in telling Z is equal to g^{r+s} or a random element of $\mathbb{G}_1$. Therefore, we have that the probability that $\mathcal{B}$ succeeds in guessing the bit b is

$$\begin{aligned}\Pr[b' = b] &= \Pr[b' = b \wedge \mathsf{abt}] + \Pr[b' = b \wedge \overline{\mathsf{abt}}] \\ &= \Pr[b' = b|\mathsf{abt}]\Pr[\mathsf{abt}] + \Pr[b' = b|\overline{\mathsf{abt}}]\Pr[\overline{\mathsf{abt}}] \\ &= \frac{1}{2}(1 - \Pr[\overline{\mathsf{abt}}]) + (\epsilon_C + \frac{1}{2}) \cdot \Pr[\overline{\mathsf{abt}}] \\ &= \frac{1}{2} + \epsilon_C \cdot \Pr[\overline{\mathsf{abt}}].\end{aligned}$$

If ϵ_C is non-negligible, so is $|\Pr[b' = b] - 1/2|$.

This completes the proof. □

6. Experiments and Efficiency Comparison

In this section we compare our scheme with some other related PEKS schemes. The comparison in terms of computation efficiency is given in Table 1, where we use symbols E, H, and P to denote the evaluation of a modular exponentiation, a collision-resistant hash function and a bilinear pairing, respectively. As shown in Table 1, the computation costs of the PEKS, the Trapdoor and Test algorithms of our PAEKS scheme are comparable with those of the schemes in [7], [24], [3] and [4].

The comparison of communication cost is shown in Table 2, where the symbols $|\mathbb{G}_1|$, $|\mathbb{G}_T|$, and $|\mathbb{Z}_p|$ denote the length of an element in group $\mathbb{G}_1$, $\mathbb{G}_T$ and $\mathbb{Z}_p$, respectively. Length of the security parameter is denoted by n. We compare the schemes in terms of the sizes of public key, ciphertext and trapdoor. All the schemes have secret keys of almost the same length. As shown in Table 2, our PAEKS scheme has a low communication cost comparable with [7], [3], [4], and [19].

We implemented our PAEKS scheme and BDOP-PEKS scheme [7] on a laptop with 2.30GHz Intel i7 CPU, 8GB memory, and Ubuntu 15.04 64-bit operating system and used PBC library [30]. We chose Type-A pairing in the PBC library, which makes use of the curve $y^2 = x^3 + x$ over the field F_q for prime $q \equiv 3 \bmod 4$ [30]. [2] We did experiments to compare the computation efficiency of the two schemes. We ran each algorithm for different times and record the time consumed by it. To compare the efficiency of each algorithm of BDOP-PEKS scheme and our PAEKS scheme in more details, we tested the running time of PEKS, Trapdoor and Test algorithms of the two schemes, respectively. The results are shown in Figure 5 to Figure 7. Figure 5 tells that the PEKS algorithms in PAEKS and BDOP-PEKS schemes enjoy similar efficiency. Figure 6 shows that the Trapdoor algorithms of the two schemes require comparable running time. The Test algorithm in our PAEKS scheme consumes time nearly twice that of the same algorithm in BDOP-PEKS scheme, as shown in Figure 7. However, in both schemes the running time of Test algorithm is much less than PEKS and Trapdoor algorithms.

(*Optimized Keyword Search*). Our PAEKS scheme has a feature that the trapdoor generation is deterministic and is valid with respect to the specified sender and receiver. The server is able to pre-process all the keyword ciphertexts shared between the sender and the receiver, i.e. computing $T' = \hat{e}(C_1, \mathsf{Pk}_R)/\hat{e}(C_2, g)$ for each ciphertext, then the search over encrypted keywords becomes extremely fast as the server can simply compare the trapdoor T_w received from the data receiver with each T', and return the files associated to the matched encrypted keyword.

Table 1: Computation Efficiency Comparison

Scheme	PEKS	Trapdoor	Test
[7]	2E+2H+P	E+H	H+P
[24]	2E+H+P	2E+H	E+P
[3]	3E+4H+P	E+H	H+P
[4]	3E+2H+2P	E+H	H+P
[19]	6E+2H+P	2E+H	E+H+P
[13]	6E+H+3P	2E+H	E+H+P
[35]	2E+2H+9P	H+P	E+H+P
[34]	2E+2H+P	3E+2H	E+H+P
[21]	2E+2H+3P	2E+H	E+H+P
[36]	9E+3H+3P	2E	5E+H+4P
PAEKS	3E+H	E+H+P	2P

7. Conclusion and Future Work

In this paper we proposed another method to resist inside keyword guessing attacks against public key encryption with keyword search. We introduced the notion of public-key authenticated encryption, which differs from PEKS mainly in that we now require the data sender to use its

[2] In our experiment we used the prime $q = 878071079966331252243778198475404981580688319941420821102865339926647563088022295707862517942266222142315585876958231745927771336731748132492512999822 4791$.

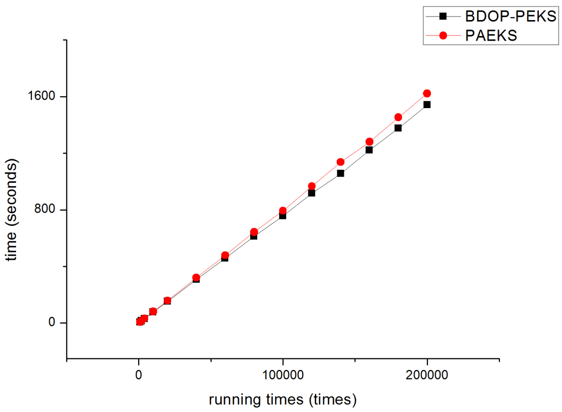

Figure 5: Running Time of PEKS Algorithm

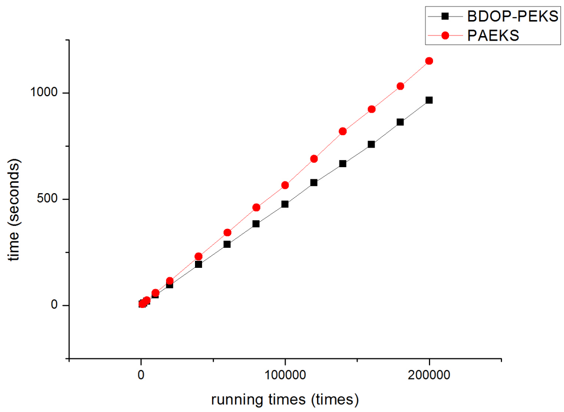

Figure 6: Running Time of Trapdoor Algorithm

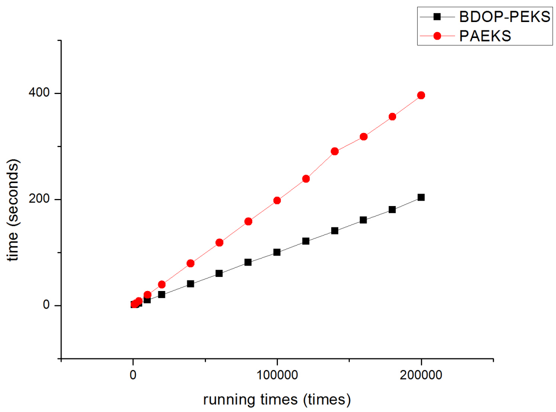

Figure 7: Running Time of Test Algorithm

Table 2: Communication Efficiency Comparison

Scheme	$\vert\mathsf{Pk}\vert$	$\vert C\vert$	$\vert T_w\vert$
[7]	$\vert\mathbb{G}_1\vert$	$\vert\mathbb{G}_1\vert + n$	$\vert\mathbb{G}_1\vert$
[24]	$2\vert\mathbb{G}_1\vert$	$2\vert\mathbb{G}_1\vert + \vert\mathbb{G}_T\vert$	$\vert\mathbb{G}_1\vert + \vert\mathbb{Z}_p\vert + n$
[3]	$\vert\mathbb{G}_1\vert$	$\vert\mathbb{G}_1\vert + 3n$	$\vert\mathbb{G}_1\vert$
[4]	$2\vert\mathbb{G}_1\vert$	$\vert\mathbb{G}_1\vert + n$	$\vert\mathbb{G}_1\vert$
[19]	$\vert\mathbb{G}_1\vert$	$2\vert\mathbb{G}_1\vert + n$	$\vert\mathbb{G}_1\vert$
[13]	$2\vert\mathbb{G}_1\vert$	$2\vert\mathbb{G}_1\vert + 2\vert\mathbb{G}_T\vert$	$\vert\mathbb{G}_1\vert + \vert\mathbb{Z}_p\vert$
[35]	$2\vert\mathbb{G}_1\vert$	$\vert\mathbb{G}_1\vert + n$	$2\vert\mathbb{G}_1\vert$
[34]	$2\vert\mathbb{G}_1\vert$	$\vert\mathbb{G}_1\vert + n$	$2\vert\mathbb{G}_1\vert$
[21]	$3\vert\mathbb{G}_1\vert$	$\vert\mathbb{G}_1\vert + n$	$2\vert\mathbb{G}_1\vert$
[36]	$2\vert\mathbb{G}_1\vert$	$5\vert\mathbb{G}_1\vert + 3\vert\mathbb{G}_T\vert$	$3\vert\mathbb{G}_1\vert$
PAEKS	$\vert\mathbb{G}_1\vert$	$2\vert\mathbb{G}_1\vert$	$\vert\mathbb{G}_T\vert$

secret key to *authenticate* the data while encrypting it. The server would not be able to encrypt again a random keyword chosen by itself on behalf of the sender and the receiver, and thus could not launch the inside keyword guessing attack. We present a concrete construction of PAEKS and proved it to be semantically secure in the random oracle model based on reasonable assumptions. The scheme is efficient in the sense that it has almost the same computation efficiency as Boneh et al.'s PEKS scheme.

Our construction of PAEKS relies on the mDLIN assumption, which was newly proposed. In the future work we consider to construct PAEKS schemes based on standard and well-accepted assumptions, i.e. DLIN. Besides, the security of our scheme needs the random oracle model, which may not be reserved when the random oracles are replaced with real-life hash functions. Therefore, it would be meaningful to construct a scheme in the standard model. Furthermore, our scheme considers single keyword. How to build a PAEKS scheme supporting multi keywords is also an interesting problem.

Acknowledgement

This work is supported by 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), the National Natural Science Foundation of China (No. 61472146), and the CICAEET fund and the PAPD fund (No. KJR1615).

References

[1] Arriaga, A., Tang, Q., 2013. Trapdoor privacy in asymmetric searchable encryption schemes. IACR Cryptology ePrint Archive 2013, 330.

[2] Arriaga, A., Tang, Q., Ryan, P., 2014. Trapdoor Privacy in Asymmetric Searchable Encryption Schemes. Springer International Publishing.

[3] Baek, J., Safavi-Naini, R., Susilo, W., 2006. On the Integration of Public Key Data Encryption and Public Key Encryption with Keyword Search. Springer Berlin Heidelberg.

[4] Baek, J., Safavi-Naini, R., Susilo, W., 2008. Public key encryption with keyword search revisited. In: International conference on Computational Science and Its Applications. Springer, pp. 1249–1259.

[5] Bao, F., Deng, R. H., Zhu, H., 2003. Variations of diffie-hellman problem. In: International Conference on Information and Communications Security. Springer, pp. 301–312.

[6] Boneh, D., Boyen, X., Shacham, H., 2004. Short group signatures. In: Annual International Cryptology Conference. Springer, pp. 41–55.

[7] Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G., 2004. Public key encryption with keyword search. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp. 506–522.

[8] Boneh, D., Franklin, M., 2001. Identity-based encryption from the weil pairing. In: Annual International Cryptology Conference. Springer, pp. 213–229.

[9] Boyen, X., 2008. The uber-assumption family. In: Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings. pp. 39–56.

[10] Cao, N., Wang, C., Li, M., Ren, K., Lou, W., 2014. Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Transactions on parallel and distributed systems 25 (1), 222–233.

[11] Chen, Y.-C., 2015. Speks: Secure server-designation public key encryption with keyword search against keyword guessing attacks. Comput. J. 58, 922–933.

[12] Chuah, M., Hu, W., 2011. Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data. In: 2011 31st International Conference on Distributed Computing Systems Workshops. IEEE, pp. 273–281.

[13] Fang, L., Susilo, W., Ge, C., Wang, J., 2009. 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. pp. 248–258.

[14] Fang, L., Susilo, W., Ge, C., Wang, J., 2013. Public key encryption with keyword search secure against keyword guessing attacks without random oracle. Inf. Sci. 238, 221–241.

[15] Fu, Z., Ren, K., Shu, J., Sun, X., Huang, F., 2016. Enabling personalized search over encrypted outsourced data with efficiency improvement. IEEE Trans. Parallel Distrib. Syst. 27 (9), 2546–2559.

[16] Fu, Z., Sun, X., Liu, Q., ZHOU, L., SHU, J., 2015. Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing. IEICE Transactions on Communications 98 (1), 190–200.

[17] Fu, Z., Wu, X., Guan, C., Sun, X., Ren, K., 2016. Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. IEEE Transactions on Information Forensics and Security 11 (12), 2706–2716.

[18] Golle, P., Staddon, J., Waters, B., 2004. Secure conjunctive keyword search over encrypted data. In: Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings. pp. 31–45.

[19] Gu, C., Zhu, Y., Pan, H., 2007. Efficient public key encryption with keyword search schemes from pairings. In: Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5, 2007, Revised Selected Papers. pp. 372–383.

[20] Guo, L., Yau, W.-C., 2015. Efficient secure-channel free public key encryption with keyword search for emrs in cloud storage. J. Medical Systems 39, 11.

[21] Hu, C., Liu, P., 2011. A secure searchable public key encryption scheme with a designated tester against keyword guessing attacks and its extension. In: International Conference on Computer Science, Environment, Ecoinformatics, and Education. Springer, pp. 131–136.

[22] Huang, Q., Yang, G., Wong, D. S., Susilo, W., 2008. Efficient optimistic fair exchange secure in the multi-user setting and chosen-key model without random oracles. In: Proceedings of RSA Conference, Cryptographers' Track, CT-RSA 2008. Vol. 4964 of Lecture Notes in Computer Science. Springer, pp. 106–120.

[23] Jin, W. B., Rhee, H. S., Park, H. A., Dong, H. L., 2006. Off-Line Keyword Guessing Attacks on Recent Keyword Search Schemes over Encrypted Data. Springer Berlin Heidelberg.

[24] Jin Park, D., Kim, K., Lee, P. J., 2004. Public key encryption with conjunctive field keyword search. In: International Conference on Information Security Applications. pp. 73–86.
[25] Joux, A., 2006. A one round protocol for tripartite diffiechellman. Journal of Cryptology 17 (4), 385–393.
[26] Krawczyk, H., 2001. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In: Proceedings of Advances in Cryptology - CRYPTO 2001. Vol. 2139 of Lecture Notes in Computer Science. Springer, pp. 310–331.
[27] Li, H., Liu, D., Dai, Y.-S., Luan, T. H., Shen, X., 2015. Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage. IEEE Trans. Emerging Topics Comput. 3, 127–138.
[28] Liu, C., Zhu, L., Wang, M., Tan, Y. A., 2014. Search pattern leakage in searchable encryption: Attacks and new construction. Information Sciences 265 (5), 176–188.
[29] Lv, Z., Hong, C., Zhang, M., Feng, D., 2014. Expressive and secure searchable encryption in the public key setting (full version). IACR Cryptology ePrint Archive 2014, 614.
[30] Lynn, B., *et al*, 2013. Pairing-based cryptography library. `https://crypto.stanford.edu/pbc/`.
[31] Ma, S., Huang, Q., Zhang, M., Yang, B., 2015. Efficient public key encryption with equality test supporting flexible authorization. IEEE Trans. Information Forensics and Security 10, 458–470.
[32] Ren, Y., Shen, J., Wang, J., Han, J., Lee, S., 2015. Mutual verifiable provable data auditing in public cloud storage. Journal of Internet Technology 16 (2), 318.
[33] Rhee, H. S., Park, J. H., Susilo, W., Lee, D. H., 2009. Improved searchable public key encryption with designated tester. In: International Symposium on Information, Computer, and Communications Security. pp. 376–379.
[34] Rhee, H. S., Park, J. H., Susilo, W., Lee, D. H., 2010. Trapdoor security in a searchable public-key encryption scheme with a designated tester. Journal of Systems and Software 83 (5), 763–771.
[35] Rhee, H. S., Susilo, W., Kim, H. J., 2009. Secure searchable public key encryption scheme against keyword guessing attacks. Ieice Electronics Express 6 (5), 237–243.
[36] Shao, Z.-Y., Yang, B., 2015. On security against the server in designated tester public key encryption with keyword search. Information Processing Letters 115 (12), 957–961.
[37] Sun, W., Yu, S., Lou, W., Hou, Y. T., Li, H., 2016. Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. IEEE Trans. Parallel Distrib. Syst. 27, 1187–1198.
[38] Tang, Q., 2012. Public key encryption schemes supporting equality test with authorisation of different granularity. IJACT 2, 304–321.
[39] Wang, C. H., Tai-Yuan, T. U., 2014. Keyword search encryption scheme resistant against keyword-guessing attack by the untrusted server. Journal of Shanghai Jiaotong University 19 (4), 440–442.
[40] Xia, Z., Wang, X., Sun, X., Wang, Q., 2016. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Transactions on Parallel and Distributed Systems 27 (2), 340–352.
[41] Xu, P., Jin, H., Wu, Q., Wang, W., 2010. Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack. IEEE Trans. Computers 62, 2266–2277.
[42] Yau, W. C., Heng, S. H., Goi, B. M., 2008. Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In: Autonomic and Trusted Computing, International Conference, ATC 2008, Oslo, Norway, June 23-25, 2008, Proceedings. pp. 100–105.
[43] Zheng, Y., 1997. Digital signcryption or how to achieve cost(signature & encryption) $<<$ cost(signature) + cost(encryption). In: Proceedings of Advances in Cryptology - CRYPTO 1997. Vol. 1294 of Lecture Notes in Computer Science. Springer, pp. 165–179.