New Framework for Secure Server-Designation Public Key Encryption with Keyword Search

Xi-Jun Lin *,Lin Sun †and Haipeng Qu ‡

April 1, 2016

Abstract: Recently, a new framework, called secure server-designation public key encryption with keyword search (SPEKS), was introduced to improve the security of dPEKS (which suffers from the on-line keyword guessing attack) by defining a new security model 'original ciphertext indistinguishability'. In this paper, we note that off-line keyword guessing attack can be launched by a malicious server to find the keyword used for generating the trapdoor, which was not considered in the related work. SPEKS can suffer from this kind of attack. Moreover, the security model defined for TD-IND in SPEKS is incomplete. Owing to the shown weaknesses, the existing security models are enhanced for trapdoor indistinguishability by defining two new security models. Finally, we propose a new framework.

Key words: Computer Networks; Information Security; Distributed Systems

1 Introduction

Nowadays, cloud storage is an important technique for sharing data, especially large media data, between senders and receivers. More and more users enjoy the services provided by cloud storage systems, such as iCloud, SkyDrive and Dropbox. Encryption is an effective and practical approach to protect the sensitive data stored in the servers.

Public key encryption with keyword search(PEKS) [1, 3, 8–13, 17, 18, 20] is a notion of dealing with obtaining encrypted data from the servers with regard to cloud storages. PEKS enables a sender stores encrypted sensitive data with searchable ciphertexts into a server. A receiver who wants to obtain the sender's sensitive data should provide a keyword trapdoor to the server. After testing the validness of the keyword trapdoor with searchable ciphertexts, the server will send the encrypted sensitive data to the receiver.

The notion of PKES was first introduced by Boneh et al. [3] They proposed the first PEKS scheme, and the notion of *searchable ciphertext indistinguishability* (SC-IND) for the security of PKES. However, secure channel is required in their scheme.

Baek et al. [2] pointed out that PEKS suffers from an attack that without a secure channel an attacker has the ability to link the matching trapdoor and ciphertext. Hence,

*X. J. Lin is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China. email: linxj77@163.com

†L. Sun is with the College of Liberal Arts, Qingdao University. Qingdao 266071, P.R.China.

‡H. Qu is with the Department of Computer Science and Technology, Ocean University of China. Qingdao 266100, P.R.China.

they redefined the notion of SC-IND. The refined SC-IND guarantees that a malicious server that has not obtained the trapdoors for given keywords cannot tell which searchable ciphertext encrypts which keyword, and the outsider (without the server's private key) cannot make any decisions about the searchable ciphertexts even though the outsider gets all the trapdoors for the keywords that it holds.

Byun et al. [5] pointed out that PEKS can suffer from off-line keyword guessing attack: An attacker can find the keyword used for generating the trapdoor, since keywords have much lower entropy than passwords.

The notion of *trapdoor indistinguishability* (TD-IND) was defined by Rhee et al. [14] to address off-line keyword guessing attack. TD-IND guarantees that without the server's private key the trapdoor does not reveal any information with respect to the keyword. Rhee et al. also proposed a new PEKS scheme which is an approach to withstand off-line keyword guessing attack. However, their dPEKS can suffer from online keyword guessing attack, which was pointed out by Yau et al. [19].

Recently, Chen [6] defined the security model for *original ciphertext indistinguishability* (OC-IND), which guarantees that the original ciphertext is indistinguishable for an outsider. They also proposed a new framework, called secure server-designation public key encryption with keyword search (SPEKS), for withstanding online keyword guessing attack.

Our Contribution

We note that although some solutions for realizing secure PEKS are elegant, there remains an important issue which was not addressed in the related work. In fact, the existing security model for trapdoor indistinguishability [6, 14] only guarantees that *without the server's private key the trapdoor does not reveal any information with respect to the keyword*. That is, only outsider was considered. However, the off-line keyword guessing attack can be launched by a malicious server [1] to find the keyword used for generating the trapdoor, which was not considered in the related work. Hence, the existing security model for TD-IND is incomplete.

Motivated by this, our contribution in this paper is three folds:

- We first point out that the SPEKS has the weakness mentioned above. Moreover, the security model defined for TD-IND in SPEKS is incomplete. That is, the trapdoor in fact is distinguishable.
- We enhance the existing security model for trapdoor indistinguishability to remedy this weakness by defining two new security models (see Section 4.2.3). In our enhanced security models, the security guarantees that the trapdoor does not reveal any information on any keyword for an outsider (including the sender) or the malicious server.
- Finally, we define a new framework for addressing this weakness, which followed by a concrete scheme.

[1] We stress here that a malicious server means an honest-but-curious server in the related work and this paper.

2 Preliminaries

2.1 Bilinear Pairings

Let G_1, G_2 and G_T be multiplicative cyclic groups of prime order p with the security parameter λ. Let g_1, g_2 be generators of G_1, G_2 respectively. The bilinear map $e : G_1 \times G_2 \rightarrow G_T$ is defined as follows:

1. Bilinearity: $\forall u \in G_1, \forall v \in G_2$ and $\forall a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $\exists g_1, g_2$ such that $e(g_1, g_2)$ has order p, that is, $e(g_1, g_2)$ is a generator of G_T.

Note that the group operations in G_1, G_2, G_T and e are all efficiently computable, while no efficiently computable isomorphism can be found between G_1 and G_2 [12, 15].

2.2 Decisional Bilinear Diffie-Hellman (BDH) Assumption

Let (p, G_1, G_2, G_T, e) be that defined in Section 2.1. Let g_1, g_2 be generators of G_1, G_2 respectively. The assumption is that $e(g_1, g_2)^{abc}$ and $T \in G_T$ are indistinguishable for any probabilistic polynomial time algorithm $\mathcal{A}$ if the challenge instances $D = ((p, G_1, G_2, G_T, e), g_1, g_1^a, g_1^c, g_2, g_2^b)$ and T are given. The advantage of $\mathcal{A}$ is defined as

$$Adv_{\mathcal{A}}^{DBDH}(\lambda) = |Pr[\mathcal{A}(D, e(g_1, g_2)^{abc}) = 1] - Pr[\mathcal{A}(D, T) = 1]|.$$

Note that the probability is taken over the random choice of $T \in G_T$ and $a, b, c \in \mathbb{Z}_p$.

2.3 Symmetric eXternal Diffie-Hellman (SXDH) Assumption

Let (p, G_1, G_2, G_T, e) be that defined in Section 2.1. Let g_1, g_2 be generators of G_1, G_2 respectively. The assumption is that g_1^{ab} from $\eta \in G_1$ are indistinguishable for any probabilistic polynomial time algorithm $\mathcal{A}$ if the challenge instances $D = ((p, G_1, G_2, G_T, e), g_1, g_2, g_1^a, g_1^b)$ and η are given. The advantage of $\mathcal{A}$ is defined as

$$Adv_{\mathcal{A}}(\lambda) = |Pr[\mathcal{A}(D, g_1^{ab}) = 1] - Pr[B(D, \eta) = 1]|$$

where the probability is taken over the random choice of $\eta \in G_1$ and $a, b \in \mathbb{Z}_p$.

2.4 Definition of Traditional Public Key Encryption

In traditional public key encryption (TE), there are two participants, a receiver and a sender. The definition of the TE scheme is as follows.

- TE.Setup(k): This algorithm takes a security parameter k as input, and outputs the system parameter $\mathcal{SP}$.
- TE.KeyGen($\mathcal{SP}$): This algorithm takes the system parameter $\mathcal{SP}$ as input, and outputs the public/private key pair R_{pub}, R_{priv}.
- TE.Encrypt($\mathcal{SP}, M, R_{pub}$): This algorithm takes the system parameter $\mathcal{SP}$, a message M and the public key R_{pub} as inputs, and outputs C as the ciphertext to the receiver.

- TE.Decrypt($\mathcal{SP}, R_{priv}, C$): This algorithm takes the system parameter $\mathcal{SP}$, the private key R_{priv} and a ciphertext C as inputs, and retrieves the message M.

Ciphertext indistinguishability against adaptive chosen ciphertext attack (IND-CCA2). Let $\mathcal{C}$ be a challenger and $\mathcal{A}$ be a probabilistic polynomial time adversary. The security game for IND-CCA2 works between $\mathcal{A}$ and $\mathcal{C}$ as follows.

1. *Setup*: TE.Setup and TE.KeyGen are run by $\mathcal{C}$, and then the system parameters $\mathcal{SP}$ and a public key R_{pub} are returned to $\mathcal{A}$. Note that R_{priv} is kept secretly.
2. *Phase 1*: $\mathcal{A}$ queries Decrypt oracle to $\mathcal{C}$. A message M which is retrieved from a ciphertext C for $\mathcal{A}$'s queries will be returned.
3. *Challenge*: $\mathcal{A}$ selects two messages M_0^*, M_1^* for challenge. C picks $b \in \{0,1\}$ randomly and outputs a challenge ciphertext $C^* = \text{TE.Encrypt}(\mathcal{SP}, M_b^*, R_{pub})$ to $\mathcal{A}$.
4. *Phase 2*: $\mathcal{A}$ can make Decrypt oracle as long as $C \neq C^*$.
5. *Guess*: $\mathcal{A}$ outputs $b' \in \{0,1\}$. If $b = b'$, then $\mathcal{A}$ wins the game.

The advantage of $\mathcal{A}$ to break a TE scheme is denoted by $Adv_{\mathcal{A},TE}^{IND-CCA2}(k) = |Pr[b = b'] - \frac{1}{2}|$.

Definition 1 *If $Adv_{\mathcal{A},TE}^{IND-CCA2}(k)$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$, we say that the TE scheme meets IND-CCA2.*

3 SPEKS and Its Weaknesses

SPEKS [6] is based on identity-based encryption (IBE) and TE. Now, we first recall the definition of IBE.

Definition of IBE. In IBE, there is a trust agency (TA) which takes its master private key and a user's identity ID to generate the user's private key. The formal definition of IBE scheme is as follows.

- IBE.Setup(k): This algorithm is run by the TA. It takes a security parameter k as input, the master public/private key pair P_{pub}, P_{priv} and the system parameter $\mathcal{SP}$ are outputted. Note that the master public key P_{pub} and an identity space $\mathcal{ID}$ are included in $\mathcal{SP}$.
- IBE.Extract($\mathcal{SP}, ID, P_{priv}$): This algorithm takes the system parameter $\mathcal{SP}$, an identity ID and the master private key P_{priv} as inputs, generates the user's private key U_{priv}.
- IBE.Encrypt($\mathcal{SP}, M, ID$): This algorithm takes the system parameter $\mathcal{SP}$, a message M and the receiver's ID as inputs, outputs C as the ciphertext.
- IBE.Decrypt($\mathcal{SP}, U_{priv}, C$): This algorithm takes the system parameter $\mathcal{SP}$, the receiver's private key U_{priv} and a ciphertext C as inputs, retrieves the message M.

3.1 SPEKS

The SPEKS scheme proposed in [6] is recalled as follows.

- *GlobalSetup(k)*: This algorithm runs TE.Setup and IBE.Setup to return $\mathcal{SP}$ as the system parameter, where the identity space $\mathcal{ID} = \mathcal{W}$.
- *dKeyGen($\mathcal{SP}$)*: This algorithm takes the system parameter $\mathcal{SP}$ to run TE.KeyGen to generate R_{pub}, R_{priv}. Finally, it outputs the server's public/private key pair $S_{pub} = R_{pub}, S_{priv} = R_{priv}$.
- *rKeyGen($\mathcal{SP}$)*: This algorithm takes the system parameter $\mathcal{SP}$ as input. It runs IBE.Setup to output the master key pair P_{pub}, P_{priv} of IBE. Finally, it outputs the receiver's public/private key pair $A_{pub} = P_{pub}, A_{priv} = P_{priv}$.
- *dPEKS($\mathcal{SP}, W_i, A_{pub}, S_{pub}$)*: It takes the system parameter $\mathcal{SP}$, a keyword W_i, the receiver's public key A_{pub} and the server's public key S_{pub} as inputs, and then outputs the searchable ciphertext C_i = TE.Encrypt($\mathcal{SP}, C', S_{pub}$), where C' = IBE.Encrypt ($\mathcal{SP}, A_{pub}, 1, W_i$).

  Finally, $\tilde{M}||C_1, \cdots, C_n$ will be sent to the server by the sender. Note that $\tilde{M}$ is the ciphertext of the real message M by using the receiver's public key.
- *Trapdoor($\mathcal{SP}, W', A_{priv}, S_{pub}$)*: The receiver runs this algorithm, which takes the system parameter $\mathcal{SP}$, a keyword W', the receiver's private key A_{priv} and the server's public key S_{pub} as inputs, to output the keyword trapdoor T = TE.Encrypt ($\mathcal{SP}$, IBE.Extract($\mathcal{SP}, W', A_{priv}$), S_{pub}).
- *Test($\mathcal{SP}, C_i, T, S_{priv}$)*: The server runs this algorithm, which takes the system parameter $\mathcal{SP}$, a searchable ciphertext C_i, a keyword trapdoor T and the server's private key as inputs, to generate T' = TE.Decrypt ($\mathcal{SP}, S_{priv}, T$) and C_i' = TE.Decrypt ($\mathcal{SP}, S_{priv}, C_i$). Finally, it checks whether or not 1 = IBE.Decrypt($\mathcal{SP}, T', C_i'$) holds or not. If it is the case, it outputs 1, otherwise 0.
- *Encrypt($\mathcal{SP}, \tilde{M}, A_{pub}$)*: The server computes and sends $\tilde{C}$ = TE.Encrypt($\mathcal{SP}$, $\tilde{M}$, A_{pub}) to the receiver, where $\tilde{M}$ is the matching message.
- *Decrypt($\mathcal{SP}, \tilde{C}, A_{priv}$)*: Upon receiving $\tilde{C}$, the receiver retrieves $\tilde{M}$ by computing $\tilde{M}$ = TE.Decrypt ($\mathcal{SP}, A_{priv}, \tilde{C}$). Then, the receiver recover the real message M by using his private key.

3.2 The Weaknesses in SPEKS

Here, we point out two weaknesses in SPEKS: (1) Given a trapdoor a malicious server can find the keyword used for generating the trapdoor. That is, the trapdoor is distinguishable when the server is malicious, which is not considered in the related work. (2) The security model for TD-IND defined in SPEKS is incomplete, the trapdoor in fact is distinguishable [2].

Note that SPEKS is a generic transformation from IBE and TE. Hence, we assume that without loss of generality Boneh-Franklin's IBE [4] is used in SPEKS.

[2] the details of the security model is referred to the original paper [6]

Definition of Boneh-Franklin's IBE. Boneh-Franklin's IBE consists of following algorithms:

- Setup(k): This algorithm takes a security parameter k as input, and performs as follows:
  - Generate a large random prime p and two cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ of order p.
  - Let bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.
  - Pick a generator $P \in \mathbb{G}_1$ randomly.
  - Pick a random number $s \in \mathbb{Z}_p$ and compute $P_{pub} = sP$.
  - Pick two secure hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1, H_2 : \mathbb{G}_2 \to \{0,1\}^n$.

  The system parameter $\mathcal{SP}$ is $(p, P, \mathbb{G}_1, \mathbb{G}_2, e, H_1, H_2)$. The master public key P_{pub} is also published. The master private key is $P_{priv} = s$.
- Extract($\mathcal{SP}, ID, P_{priv}$): This algorithm outputs the user's private key $U_{priv} = sH_1(ID)$.
- Encrypt($\mathcal{SP}, P_{pub}, M, ID$): This algorithm picks $r \in_R \mathbb{Z}_p^*$ and computes $C_1 = rP, C_2 = M \oplus H_2(e(H_1(ID), rP_{pub}))$, where $M \in \{0,1\}^n$ is the plaintext. The ciphertext is $C = (C_1, C_2)$.
- Decrypt($\mathcal{SP}, U_{priv}, C$): Let $C = (U, V)$. This algorithm outputs the plaintext $M = V \oplus H_2(U_{priv}, U)$.

In the case of Boneh-Franklin's IBE is used, we show the weaknesses as follows:

- *Weakness* 1: The trapdoor T outputted from *Trapdoor* algorithm is $T = TE.Encrypt(\mathcal{SP}, T', S_{pub})$, where $T' = sH_1(W')$.

  After receiving T from the receiver, the malicious server first decrypts T with its private key S_{priv} to recover T', and then performs the following steps to guess the keyword W':
  - Step 1: Guess an appropriate keyword W, and computes $H_1(W)$.
  - Step 2: Check whether or not $e(P, T') = e(P_{pub}, H_1(W))$. If it is the case, W is a valid keyword. Otherwise, go to Step 1.
- *Weakness* 2: In the security model for TD-IND defined in [6]. The adversary can perform as follows.
  - Submit two keywords W_0^*, W_1^* in the *Challenge* phase. As the result, $\mathcal{C}$ returns the challenge trapdoor T^* = TE.Encrypt ($\mathcal{SP}, S_{pub}$, IBE.Extract ($\mathcal{SP}, A_{priv}, W_b^*$)).
  - Compute C = TE.Encrypt ($\mathcal{SP}, S_{pub}$, IBE.Encrypt ($\mathcal{SP}, A_{pub}, W_1^*, 1$)). And send (C, T^*) to $\mathcal{C}$ for *Test-query* in Phase 2. As a result, $\mathcal{C}$ will return 1 or 0.
  - If 1 is returned, output $b' = 1$ as the answer for the challenge; otherwise, output $b' = 0$.

  Clearly, if $W_b^* = W_1^*$, *Test* oracle will return 1; otherwise, return 0.

4 Our Proposed New Framework

In this section, we present our new framework. In this new framework, three entities are involved: the sender, the receiver and the server.

It should be noted that the ciphertext is generated with the receiver's public key in SPEKS. That is, the sender generates the ciphertext for the specific receiver. In this case, it is practical to consider that the receiver should be aware of the identity of the sender. On the other hand, for considering the receiver's privacy in some scenarios, only the authorized senders having the ability to generate the ciphertexts for the specific receiver is more practical in real life.

From above, the idea behind our framework is as follows: The receiver authorizes a sender by sending him a secret token, only with which the sender can generate valid searchable ciphertexts for the receiver. *We stress here that the communication of token is once for all.* Moreover, the receiver's private key establishes a link between his trapdoors and all authorized senders' searchable ciphertexts. As a result, the receiver can search all authorized senders' searchable ciphertexts with only one trapdoor. [3]

Compared with SPEKS, there is an additional algorithm: *Token*. The receiver generates the token with his private key by running algorithm *Token*, and then sends it to the sender secretly. More in details, the workflow of our proposal is presented as follows.

1. To authorize a sender to generate valid encrypted data for him, the receiver should generate a token with his private key, and then send it to the sender secretly.
2. With the token and the keywords, the sender generates the searchable ciphertexts for the receiver.
3. To search for a specific keyword, the receiver issues the server a trapdoor, generated based on his private key and the keyword.
4. With the trapdoor from the receiver, the server searches encrypted data of all authorized senders (by the receiver) to decide whether there exist searchable ciphertexts which contain the same keyword as that in the trapdoor.

4.1 The Definition

Our framework consists of the following algorithms. Here, Let U be the sender and V be the receiver.

- *GlobalSetup*(λ): This algorithm takes λ (the security parameter) as input, outputs the system parameter $\mathcal{SP}$, where a keyword space $\mathcal{W}$ is included.
- *dKeyGen*($\mathcal{SP}$): The server runs this algorithm, which takes $\mathcal{SP}$ as input, to output the server's public/private key pair, (S_{pub}, S_{priv}).
- *rKeyGen*($\mathcal{SP}$): The receiver V runs this algorithm, which takes $\mathcal{SP}$ as input, to output his public/private key pair, (V_{pub}, V_{priv}).
- *Token*($\mathcal{SP}, V_{priv}$): This algorithm takes $\mathcal{SP}$ and the receiver V's private key V_{priv} as inputs, run by the receiver V, generates a token TK.

[3] The similar approach was used in [16].

- *dPEKS*($\mathcal{SP}, W_i, TK, S_{pub}$): The sender U runs this algorithm, which takes $\mathcal{SP}$, a keyword $W_i \in \mathcal{W}$, a token TK and the server's public key S_{pub} as inputs, to generate C_i as the searchable ciphertext.

  Finally, U will send $\{E_{V_{pub}}[M]||C_1, \cdots, C_n\}$ to the server [4], and $C_1, \cdots, C_n$ are searchable ciphertexts for keywords $W_1, \cdots, W_n$.

- *Trapdoor*($\mathcal{SP}, W', V_{priv}, S_{pub}$): This algorithm takes the system parameter $\mathcal{SP}$, a keyword $W' \in \mathcal{W}$, the receiver V's private key V_{priv} and the server's public key S_{pub} as inputs, run by the receiver V, outputs the trapdoor T.

- *Test*($\mathcal{SP}, C_i, T, S_{priv}$): The server runs this algorithm, which takes $\mathcal{SP}$, the searchable ciphertext C_i, the keyword trapdoor T and the server's private key S_{priv} as inputs, to output 1 if $W' = W_i$, or 0 otherwise. If 1 is outputted, algorithm *Encrypt* will be performed by the server.

- *Encrypt*($\mathcal{SP}, V_{pub}, \tilde{M}$): The server runs this algorithm, which takes $\mathcal{SP}$, the receiver V's public key V_{pub} and $\tilde{M} = E_{V_{pub}}[M]$ as inputs, to generate $\tilde{C}$ as the ciphertext of $\tilde{M}$.

- *Decrypt*($\mathcal{SP}, V_{priv}, \tilde{C}$): The receiver V runs this algorithm, which takes $\mathcal{SP}$, the receiver V's private key V_{priv} and the ciphertext $\tilde{C}$ as inputs, to retrieve the real message M.

4.2 Security Models

Before presenting the security models, we first give the following queries can be made by the adversary $\mathcal{A}$.

- *Trapdoor-query*: $\mathcal{A}$ submits a keyword for this query. A trapdoor with respect to this keyword will be returned.

- *Test-query*: $\mathcal{A}$ submits a trapdoor with a searchable ciphertext for this query. 1 or 0 will be returned.

- *Decrypt-query*: $\mathcal{A}$ submits a ciphertext for this query. The corresponding message will be returned.

- *dPEKS-query*: $\mathcal{A}$ submits a keyword for this query. A searchable ciphertext of this keyword with respect to the token will be returned.

4.2.1 Security Models for SC-IND

In these security models, the adversary $\mathcal{A}$ acts as a malicious server or a receiver. SC-IND guarantees that the adversary cannot distinguish a searchable ciphertext generated from one of two keywords W_0^*, W_1^* which are chosen by the adversary. The security model for SC-IND is the following games between $\mathcal{A}_1$ (or $\mathcal{A}_2$) and a challenger $\mathcal{C}$.

Game 1 (SC-IND1) *Let $\mathcal{A}_1$ be the malicious server. The game is performed as follows.*

[4] M is the message, E is a secure standard public key encryption

1. Setup: $\mathcal{A}_1$ *personally computes* (S_{pub}, S_{priv}) *as the server's public/private key, and then sends* S_{pub} *to* $\mathcal{C}$. $\mathcal{C}$ *computes* V*'s public/private key pair* (V_{pub}, V_{priv})*, and then sends* V_{pub} *to* $\mathcal{A}_1$.

2. Phase 1: $\mathcal{A}_1$ *can make Trapdoor-query and dPEKS-query.*

3. Challenge: $\mathcal{A}_1$ *selects two keywords* W_0^*, W_1^* *for challenge.* $\mathcal{C}$ *picks* $\beta \in_R \{0,1\}$ *and outputs* $C^* = dPEKS(\mathcal{SP}, W_\beta^*, TK, S_{pub})$ *as the challenge searchable ciphertext, which will be given to* $\mathcal{A}_1$.

4. Phase 2: $\mathcal{A}_1$ *can keep making queries as done in Phase* 1.

5. Guess: $\mathcal{A}_1$ *outputs* $\beta' \in \{0,1\}$. *If* $\beta = \beta'$, $\mathcal{A}_1$ *wins this game.*

The restriction is that *Trapdoor-query* has never been queried for W_0^*, W_1^*.

Game 2 (SC-IND2) *Let* $\mathcal{A}_2$ *be the receiver. The game is performed as follows.*

1. Setup: $\mathcal{A}_2$ *personally outputs* (V_{pub}, V_{priv}) *as the receiver's public/private key, and sends* V_{pub} *to* $\mathcal{C}$. $\mathcal{C}$ *computes the server's public/private key* (S_{pub}, S_{priv})*, and then sends* S_{pub} *to* $\mathcal{A}_2$. TK *is sent to* $\mathcal{C}$ *by* $\mathcal{A}_2$.

2. Phase 1: $\mathcal{A}_2$ *can make* (C_i, T) *for Test-query.*

3. Challenge: $\mathcal{A}_2$ *selects two keywords* W_0^*, W_1^* *for challenge.* $\mathcal{C}$ *picks* $\beta \in_R \{0,1\}$ *and outputs* $C^* = dPEKS(\mathcal{SP}, W_\beta^*, TK, S_{pub})$ *as the challenge searchable ciphertext, which will be given to* $\mathcal{A}_2$.

4. Phase 2: $\mathcal{A}_2$ *can keep making queries as done in Phase* 1.

5. Guess: $\mathcal{A}_2$ *outputs* $\beta' \in \{0,1\}$. *If* $\beta = \beta'$, $\mathcal{A}_2$ *wins this game.*

The restrictions are as follows:

- In Phase 1, $\mathcal{A}_2$ is not allowed to submit (C_i, T^*) for *Test-query*, where T^* is a trapdoor on W_0^*, W_1^*.
- In Phase 2, $\mathcal{A}_2$ is not allowed to submit (C_i, T) for *Test-query*, where T is a trapdoor on W_0^*, W_1^* or $C_i = C^*$.

The advantage of $\mathcal{A}_1$ or $\mathcal{A}_2$ to break a scheme in our framework is denoted by $Adv_{\mathcal{A}_1 or \mathcal{A}_2}^{SC-IND}(k) = |Pr[b = b'] - \frac{1}{2}|$.

Definition 2 *A scheme in our framework is said to meet SC-IND against the adaptive chosen keyword attack if* $Adv_{\mathcal{A}_1 or \mathcal{A}_2}^{SC-IND}(k)$ *is negligible for any probabilistic polynomial time adversary* $\mathcal{A}_i (i = 1, 2)$.

4.2.2 Security Model for OC-IND

In this security model, the adversary $\mathcal{A}_3$ acts as an outsider. OC-IND guarantees that the adversary cannot distinguish an original ciphertext generated from one of two messages $\tilde{M}_0^*, \tilde{M}_1^*$ which are chosen by the adversary. The security model for OC-IND is the following game between $\mathcal{A}_3$ and $\mathcal{C}$.

Game 3 (OC-IND) *: Let $\mathcal{A}_3$ be an outsider. The game is performed as follows.*

1. Setup*: $\mathcal{C}$ computes (S_{pub}, S_{priv}) as the server's public/private key and the receiver V's public/private key pair (V_{pub}, V_{priv}), and then sends S_{pub}, V_{pub} to $\mathcal{A}_3$.*
2. Phase 1*: $\mathcal{A}_3$ can query $\tilde{C}$ for Decrypt-query. A message $\tilde{M}$ will be returned to $\mathcal{A}_3$.*
3. Challenge*: $\mathcal{A}_3$ selects two messages $\tilde{M}_0^*, \tilde{M}_1^*$ for challenge. Finally, $\mathcal{C}$ picks $\beta \in_R \{0,1\}$, and then computes and outputs $\tilde{C}^* = Encrypt(\mathcal{SP}, V_{pub}, \tilde{M}_\beta^*)$ to $\mathcal{A}_3$ as the challenge ciphertext.*
4. Phase 2*: $\mathcal{A}_3$ can keep making queries as done in Phase* 1.
5. Guess*: $\mathcal{A}_3$ outputs $\beta' \in \{0,1\}$. If $\beta = \beta'$, $\mathcal{A}_3$ wins this game.*

The restriction is that $\tilde{C}^*$ is not allowed for *Decrypt-query* in Phase 2.

The advantage of $\mathcal{A}_3$ to break a scheme in our framework is defined by $Adv_{\mathcal{A}_3}^{OC-IND}(k) = |Pr[b = b'] - \frac{1}{2}|$.

Definition 3 *A scheme in our framework is said to meet OC-IND against the adaptive chosen message attack if $Adv_{\mathcal{A}_3}^{OC-IND}(k)$ is negligible for any PPT adversary $\mathcal{A}_3$.*

4.2.3 Security Models for TD-IND

Here, we enhance the existing security model for TD-IND by considering not only an outside attacker (including the sender) but also the malicious server. In the enhanced security model of TD-IND, the adversary $\mathcal{A}$ acts as the sender or the malicious server. It guarantees that the adversary cannot distinguish a trapdoor generated from one of two keywords W_0^*, W_1^* which are chosen by the adversary. The enhanced security models for TD-IND is the following games between $\mathcal{A}_4$ (or $\mathcal{A}_5$) and a challenger $\mathcal{C}$.

Game 4 (TD-IND1) *: Let $\mathcal{A}_4$ be the sender. The game is performed as follows.*

1. Setup*: $\mathcal{C}$ computes (S_{pub}, S_{priv}) as the server's public/private key and the receiver V's public/private key pair (V_{pub}, V_{priv}), and then sends S_{pub}, V_{pub} to $\mathcal{A}_4$.*
2. Phase 1*: $\mathcal{A}_4$ can submit for Trapdoor-query and Test-query.*
3. Challenge*: $\mathcal{A}_4$ chooses two keywords W_0^*, W_1^* for challenge. $\mathcal{C}$ picks $\beta \in_R \{0,1\}$ and computes $T^* = Trapdoor(\mathcal{SP}, W_\beta^*, V_{priv}, S_{pub})$ as the challenge trapdoor, which will be given to $\mathcal{A}_4$.*
4. Phase 2*: $\mathcal{A}_4$ can keep making queries as done in Phase* 1.

5. Guess: $\mathcal{A}_4$ *outputs* $\beta' \in \{0,1\}$. *If* $\beta = \beta'$, $\mathcal{A}_4$ *wins this game.*

The restrictions are as follows:

- W_0^*, W_1^* are not allowed for *Trapdoor-query.*
- (C_i, T^*) is not allowed for *Test-query*, where T^* is a trapdoor from W_0^*, W_1^*.

Game 5 (TD-IND2) *: Let* $\mathcal{A}_5$ *be the malicious server. The game is performed as follows.*

1. Setup: $\mathcal{A}_5$ *personally computes* (S_{pub}, S_{priv}) *as the server's public/private key, and then sends* S_{pub} *to* $\mathcal{C}$. $\mathcal{C}$ *computes* (V_{pub}, V_{priv}) *as the receiver* V*'s public/private key, and then sends* V_{pub} *to* $\mathcal{A}_5$.
2. Phase 1: $\mathcal{A}_5$ *can make Trapdoor-query and dPEKS-query.*
3. Challenge: $\mathcal{A}_5$ *chooses two keywords* W_0^*, W_1^* *for challenge.* $\mathcal{C}$ *picks* $\beta \in_R \{0,1\}$ *and computes* $T^* = Trapdoor(\mathcal{SP}, W_\beta^*, V_{priv}, S_{pub})$ *as the challenge trapdoor, which will be given to* $\mathcal{A}_5$.
4. Phase 2: $\mathcal{A}_5$ *can keep making queries as done in Phase* 1.
5. Guess: $\mathcal{A}_5$ *outputs* $\beta' \in \{0,1\}$. *If* $\beta = \beta'$, $\mathcal{A}_5$ *wins this game.*

The restriction is that W_0^*, W_1^* are not allowed for *Trapdoor-query.*

The advantage of $\mathcal{A}_4$ or $\mathcal{A}_5$ to break a scheme in our framework is defined by $Adv^{TD-IND}_{\mathcal{A}_4 or \mathcal{A}_5}(k) = |Pr[b = b'] - \frac{1}{2}|$.

Definition 4 *A scheme in our framework is said to meet TD-IND against the adaptive chosen keyword attack if* $Adv^{TD-IND}_{\mathcal{A}_4 or \mathcal{A}_5}(k)$ *is negligible for any probabilistic polynomial time adversary* $\mathcal{A}_i (i = 4, 5)$.

5 The Concrete Scheme

5.1 The Construction

Here, we propose a concrete scheme as follows.

- *GlobalSetup(*λ*)*: This algorithm first picks a large prime p randomly, and then generates the groups G_1, G_2, G_T and the bilinear pairing $e : G_1 \times G_2 \to G_T$. Then, secure hash functions $H_1 : \mathcal{W} \to G_1$ and $H_2 : \{0,1\}^* \to G_2$ are selected. Finally, it outputs $\mathcal{SP}$ as the system parameter. Note that $(e, p, G_1, G_2, G_T, H_1, H_2) \in \mathcal{SP}$.
- *dKeyGen(*$\mathcal{SP}$*)*: This algorithm takes $\mathcal{SP}$ as input, and runs TE.KeyGen to compute (R_{pub}, R_{priv}). Finally, it outputs the server's public/private key pair $S_{pub} = R_{pub}, S_{priv} = R_{priv}$.
- *rKeyGen(*$\mathcal{SP}$*)*: This algorithm takes $\mathcal{SP}$ as input, and runs TE.KeyGen to compute (P_{pub}, P_{priv}). Moreover, it picks $z \in_R \mathbb{Z}_p^*$. Finally, it outputs the receiver V's public/private key pair $V_{pub} = P_{pub}, V_{priv} = (P_{priv}, z)$.

- *Token*($\mathcal{SP}, V_{priv}$): This algorithm takes $\mathcal{SP}$ and the receiver V's private key V_{priv}, where $z \in V_{priv}$. Then, it outputs $TK = H_2(V)^z$ as the token. [5]

- *dPEKS*($\mathcal{SP}, W_i, TK, S_{pub}$): This algorithm takes $\mathcal{SP}$, a keyword W_i, the token TK and the server's public key S_{pub}. Then, it performs as follows:

  1. Pick $k \in_R \mathbb{Z}_p^*$.
  2. Compute $C = (c_1, c_2) = (e(H_1(W_i), TK)^k, H_2(V)^k)$.
  3. Output the searchable ciphertext $C_i = \text{TE.Encrypt}\ (\mathcal{SP}, C, S_{pub})$.

  Eventually, the sender sends $\tilde{M}$ with n searchable ciphertexts $C_1, \cdots, C_n$ to the server, where $\tilde{M} = \text{TE.Encrypt}\ (\mathcal{SP}, M, P_{pub})$ for the real message M.

- *Trapdoor*($\mathcal{SP}, W', V_{priv}, S_{pub}$): This algorithm takes $\mathcal{SP}$, a keyword W', the receiver V's private key V_{priv} and the server's public key S_{pub}, where $z \in V_{priv}$. Then, it performs as follows:

  1. Compute $\Theta = H_1(W')^z$.
  2. Output the trapdoor $T = \text{TE.Encrypt}(\mathcal{SP}, \Theta, S_{pub})$.

- *Test*($\mathcal{SP}, C_i, T, S_{priv}$): This algorithm takes $\mathcal{SP}$, a searchable ciphertext C_i, a keyword trapdoor T and the server's private key S_{priv} as inputs, performs the following steps:

  1. Compute $\Theta = \text{TE.Decrypt}\ (\mathcal{SP}, S_{priv}, T)$
  2. Compute $C = (c_1, c_2) = \text{TE.Decrypt}(\mathcal{SP}, S_{priv}, C_i)$
  3. Check whether the equation $c_1 = e(\Theta, c_2)$ holds or not. If it is the case, 1 is outputted, otherwise 0.

- *Encrypt*($\mathcal{SP}, V_{pub}, \tilde{M}$): This algorithm takes $\mathcal{SP}$, the receiver V's public key V_{pub} and a message $\tilde{M}$ as inputs, outputs $\tilde{C} = \text{TE.Encrypt}(\mathcal{SP}, \tilde{M}, P_{pub})$, where $P_{pub} \in V_{pub}$ (note that $\tilde{M} = TE.Encrypt(\mathcal{SP}, M, P_{pub})$).

- *Decrypt*($\mathcal{SP}, V_{priv}, \tilde{C}$): This algorithm takes $\mathcal{SP}$, the receiver V's private key V_{priv} and a ciphertext $\tilde{C}$ as inputs, retrieves M by computing $M = \text{TE.Decrypt}\ (\mathcal{SP}, P_{priv}, \text{TE.Decrypt}\ (\mathcal{SP}, P_{priv}, \tilde{C}))$, where $P_{priv} \in V_{priv}$.

5.2 Correctness

Here, we show the correctness of our proposal.

First, we know $c_1 = e(H_1(W_i), TK)^k$ and $c_2 = H_2(V)^k$, where $TK = H_2(V)^z$. Given a trapdoor, we have that $T = \text{TE.Encrypt}(\mathcal{SP}, \Theta, S_{pub})$, where $\Theta = H_1(W')^z$. Clearly, if $W_i = W'$, then $\Theta = H_1(W_i)^z$ and $c_1 = e(\Theta, c_2)$, which will make *Test* algorithm output 1 with probability 1. On the other hand, if $W_i \neq W'$, then $\Theta \neq H_1(W_i)^z$ with overwhelming probability. Hence, $c_1 \neq e(\Theta, c_2)$ with overwhelming probability, which will make *Test* algorithm output 0.

Moreover, we have that $\tilde{C} = \text{TE.Encrypt}(\mathcal{SP}, \tilde{M}, P_{pub})$ and $\tilde{M} = TE.Encrypt(\mathcal{SP}, M, P_{pub})$. From the consistency of TE, we have that $M = \text{TE.Decrypt}\ (\mathcal{SP}, P_{priv}, \text{TE.Decrypt}\ (\mathcal{SP}, P_{priv}, \tilde{C}))$ in Decrypt holds.

[5] To send the token to the sender via public channel, the receiver can encrypt the token with the sender's public key, and then the sender recovers it with his private key. For simplicity, we omit the details of this step.

5.3 Security

Here, we prove the security of our proposal. We stress here that the restriction in each proof should adhere to that in the corresponding game defined in Section 4.2.

Theorem 1 *Given a proposed scheme. Suppose that TE is IND-CCA secure and H_1, H_2 are random oracles. If $\mathcal{A}_1$ can break SC-IND1 with non-negligible probability, then there exists an algorithm $\mathcal{B}$ solving the BDH problem with non-negligible probability.*

Proof. To prove this theorem, we constructs an algorithm $\mathcal{B}$ by running $\mathcal{A}_1$ as a subroutine to solve the BDH problem.

1. *Setup*: $\mathcal{B}$ receives a challenge instance $(p, G_1, G_2, G_T, e, g_1, g_1^a, g_1^c, g_2, g_2^b, T)$ of BDH problem from $\mathcal{C}$ ($\mathcal{B}$'s task is to decide whether or not $e(g_1, g_2)^{abc} = T$). $\mathcal{B}$ generates the receiver V's public/private key pairs (V_{pub}, V_{priv}).

   Lists H_1-list and H_2-list, which are initial empty, are maintained by $\mathcal{B}$ to answer random oracle queries.

   $\mathcal{B}$ sends the parameters $\mathcal{SP}$ with V_{pub} to $\mathcal{A}_1$, where $(p, G_1, G_2, G_T, e) \in \mathcal{SP}$. $\mathcal{A}_1$ personally generates the server's public/private key pair (S_{pub}, S_{priv}), and then sends S_{pub} to $\mathcal{B}$.

   $\mathcal{B}$ picks a random number j as the challenge keyword index candidate.

2. *Phase* 1: $\mathcal{A}_1$ is allowed to make the following queries to $\mathcal{B}$.

   - *H_1-query*: $\mathcal{A}_1$ submits $W_i \in \mathcal{W}$ to $\mathcal{B}$. $\mathcal{B}$ responds as follows:
     (a) If $i = j$, return g_1^c as the answer.
     (b) If W_i has been queried previously, $\mathcal{B}$ searches the H_1-list to find r_1 and then returns $g_1^{r_1}$ as the answer; otherwise, $r_1 \in \mathbb{Z}_p^*$ is picked randomly and a new item $[W_i, r_1]$ is stored into H_1-list.
   - *H_2-query*: $\mathcal{A}_1$ submits α to $\mathcal{B}$. $\mathcal{B}$ responds with a random number $g_2^{r_2}$ to $\mathcal{A}_1$: If α has been queried previously, $\mathcal{B}$ searches the H_2-list to find r_2; otherwise, $r_2 \in \mathbb{Z}_p^*$ is picked randomly and a new item $[\alpha, r_2]$ is stored into H_2-list.
   - *Trapdoor-query*: $\mathcal{A}_1$ submits $W_i \in \mathcal{W}$ for this query. $\mathcal{B}$ performs as follows:
     (a) If $i = j$, abort with failure.
     (b) Compute $\Theta = g_1^{ar_1}$ and output $T = \text{TE.Encrypt}(\mathcal{SP}, \Theta, S_{pub})$ as the answer, where r_1 is from H_1-list such that $g_1^{r_1} = H_1(W_i)$.
   - *dPEKS-query*: $\mathcal{A}_1$ submits $W_i \in \mathcal{W}$ for this query. $\mathcal{B}$ performs as follows:
     (a) If $i = j$, abort with failure.
     (b) Set $c_1 = e(g_1^{ar_1}, g_2^{r_2})^k$ and $c_2 = g_2^{kr_2}$, where $k \in \mathbb{Z}_p^*$ is picked randomly, r_1 (resp. r_2) is from H_1-list (resp. H_2-list) such that $g_1^{r_1} = H_1(W_i)$ (resp. $g_2^{r_2} = H_2(V)$).
     (c) Outputs $C_i = \text{TE.Encrypt}(\mathcal{SP}, C, S_{pub})$ as the answer, where $C = (c_1, c_2)$.

3. *Challenge*: $\mathcal{A}_1$ chooses two keywords W_0^*, W_1^* for challenge. $\mathcal{B}$ picks $\beta \in_R \{0, 1\}$ and performs as follows:

(a) If W_β^* has been queried for H_1-*query* but it is not the j-th different keyword, then abort with failure.

(b) Compute $C^* = (c_1, c_2) = (T^{r_2}, g_2^{br_2})$, where r_2 is from H_2-list such that $g_2^{r_2} = H_2(V)$.

(c) Output a searchable ciphertext C_β^* = TE.Encrypt $(\mathcal{SP}, C^*, S_{pub})$ as challenge.

4. *Phase* 2: $\mathcal{A}_1$ can keep making queries as done in Phase 1.

5. *Guess*: $\mathcal{A}_1$ outputs $\beta' \in \{0, 1\}$. If $\beta = \beta'$, $\mathcal{B}$ outputs *yes* to $\mathcal{C}$ for the challenge; otherwise, $\mathcal{B}$ outputs *no*.

Clearly, since H_1, H_2 are random oracles, $\mathcal{A}_1$ wins the game implies that $\mathcal{B}$ can decide whether or not $T = e(g_1, g_2)^{abc}$ correctly with non-negligible probability. □

Theorem 2 *Given a proposed scheme. Suppose that H_1 and H_2 are random oracles. If $\mathcal{A}_2$ can break SC-IND2 with non-negligible probability, then there is an algorithm $\mathcal{B}$ which can break the IND-CCA2 security of TE with non-negligible probability.*

Proof. To prove this theorem, we constructs an algorithm $\mathcal{B}$ by running $\mathcal{A}_2$ as a subroutine to break the IND-CCA2 security of TE. $\mathcal{B}$ receives pk^* from $\mathcal{C}$ as challenge.

1. *Setup*: Let pk^* be the server's public key S_{pub} (note that $\mathcal{B}$ does not know the server's private key S_{priv}). $\mathcal{B}$ generates the system parameter $\mathcal{SP}$. S_{pub} and $\mathcal{SP}$ are sent to $\mathcal{A}_2$.

   Lists H_1-list and H_2-list, which are initial empty, are maintained by $\mathcal{B}$ to answer random oracle queries.

   $\mathcal{A}_2$ personally generates the receiver V's public/private key pair (V_{pub}, V_{priv}) and the token TK. V_{pub} and TK are sent to $\mathcal{B}$.

2. *Phase* 1: $\mathcal{A}_2$ is allowed to make the following queries to $\mathcal{B}$.

   - H_1-*query*: $\mathcal{A}_2$ submits $W \in \mathcal{W}$ for this query. $\mathcal{B}$ responds with h_1 as follows: If W has been queried previously, $\mathcal{B}$ searches the H_1-list to find h_1; otherwise, $h_1 \in G_1$ is picked randomly and a new item $[W, h_1]$ is stored into H_1-list.
   - H_2-*query*: $\mathcal{A}_2$ submits α for this query. $\mathcal{B}$ responds with h_2 as follows: If α has been queried previously, $\mathcal{B}$ searches the H_2-list to find h_2; otherwise, $h_2 \in G_2$ is picked randomly and a new item $[\alpha, h_2]$ is stored into H_2-list.
   - *Test*-query: $\mathcal{A}_2$ submits (C_i, T) for this query. $\mathcal{B}$ performs the following steps:
     (a) Send T to $\mathcal{C}$ for Decrypt oracle, and then $\mathcal{C}$ returns Θ as the answer, where Θ = TE.Decrypt $(\mathcal{SP}, S_{priv}, T)$.
     (b) Send C_i to $\mathcal{C}$ for Decrypt oracle, and then $\mathcal{C}$ returns $C = (c_1, c_2)$ as the answer, where C = TE.Decrypt$(\mathcal{SP}, S_{priv}, C_i)$.
     (c) Check whether the equation $c_1 = e(\Theta, c_2)$ holds or not. If it is the case, 1 is outputted; otherwise 0.

3. *Challenge*: $\mathcal{A}_2$ picks two keywords W_0^*, W_1^* for challenge. For each $b \in \{0, 1\}$, $\mathcal{B}$ computes C_b^* since it knows TK.

Then, $\mathcal{B}$ sends C_0^*, C_1^* to $\mathcal{C}$ as the challenge messages. $\mathcal{C}$ picks $\beta \in_R \{0,1\}$ and responds with $C^* = \text{TE.Encrypt}(\mathcal{SP}, C_\beta^*, S_{pub})$ as the challenge ciphertext. As a result, $\mathcal{B}$ outputs C^* to $\mathcal{A}_2$ as the challenge searchable ciphertext.

4. *Phase* 2: $\mathcal{A}_2$ can keep making queries as done in Phase 1.

5. *Guess*: $\mathcal{A}_2$ outputs $\beta' \in \{0,1\}$. $\mathcal{B}$ directly sends β' to $\mathcal{C}$ for the challenge.

Note that C^* is the ciphertext of C_β^*. Moreover, C_β^* is computed from W_β^*. Clearly, if $\beta' = \beta$, then $\mathcal{B}$ can answer $\mathcal{C}$'s challenge correctly with non-negligible probability. □

Theorem 3 *Given a proposed scheme. Suppose that H_1 and H_2 are random oracles. If $\mathcal{A}_3$ can break OC-IND with non-negligible probability, then there is an algorithm $\mathcal{B}$ which can break the IND-CCA2 security of TE with non-negligible probability.*

Proof. To prove this theorem, we constructs an algorithm $\mathcal{B}$ by running $\mathcal{A}_3$ as a subroutine to break the IND-CCA2 security of TE. $\mathcal{B}$ receives the challenge public key pk^* from $\mathcal{C}$.

1. *Setup*: $\mathcal{B}$ generates $\mathcal{SP}$, the server's public/private key (S_{pub}, S_{priv}) and the receiver V's public key V_{pub}, where $P_{pub} \in V_{pub}$ and $P_{pub} = pk^*$ (note that $\mathcal{B}$ does not know the receiver V's partial private key P_{priv}). $\mathcal{B}$ sends V_{pub}, S_{pub} to $\mathcal{A}_3$.

   Lists H_1-list and H_2-list, which are initial empty, are maintained by $\mathcal{B}$ to answer random oracle queries.

2. *Phase* 1: $\mathcal{A}_3$ is allowed to make the following queries to $\mathcal{B}$.

   - As for H_1-query and H_2-query, $\mathcal{B}$ performs as in the proof of Theorem 2.
   - *Decrypt*-query: $\mathcal{A}_3$ submits $\tilde{C}$ for Decrypt-query. $\mathcal{B}$ directly submits $\tilde{C}$ to $\mathcal{C}$ for Decrypt oracle. $\mathcal{C}$ returns the corresponding message $\tilde{M}$ to $\mathcal{B}$, and then $\mathcal{B}$ directly returns $\tilde{M}$ to $\mathcal{A}_3$ as the answer.

3. *Challenge*: $\mathcal{A}_3$ chooses two challenge messages $\tilde{M}_0^*, \tilde{M}_1^*$ for challenge. $\mathcal{B}$ sends $\tilde{M}_0^*, \tilde{M}_1^*$ to $\mathcal{C}$ as the challenge messages. As a response, $\mathcal{C}$ picks $\beta \in_R \{0,1\}$ and computes the corresponding ciphertext $\tilde{C}^*$ to $\mathcal{B}$. $\mathcal{B}$ directly returns $\tilde{C}^*$ to $\mathcal{A}_3$ as the challenge ciphertext.

4. *Phase* 2: $\mathcal{A}_3$ can keep making queries as done in Phase 1.

5. *Guess*: $\mathcal{A}_3$ outputs $\beta' \in \{0,1\}$. $\mathcal{B}$ directly sends β' to $\mathcal{C}$ for the challenge.

Clearly, if $\beta' = \beta$, then $\mathcal{B}$ can break the IND-CCA2 security of TE with non-negligible probability. □

Theorem 4 *Given a proposed scheme. Suppose that H_1 and H_2 are random oracles. If $\mathcal{A}_4$ can break TD-IND1 with non-negligible probability, then there is an algorithm $\mathcal{B}$ which can break the IND-CCA2 security of TE with non-negligible probability.*

Proof. To prove this theorem, we constructs an algorithm $\mathcal{B}$ by running $\mathcal{A}_4$ as a subroutine to break the IND-CCA2 security of TE. $\mathcal{B}$ receives the challenge public key pk^* from $\mathcal{C}$.

1. *Setup*: Let the server's public key $S_{pub} = pk^*$ (note that the server's private key S_{priv} is unknown to $\mathcal{B}$). $\mathcal{B}$ generates the system parameter $\mathcal{SP}$, the receiver V's public/private key pair (V_{pub}, V_{priv}) and the token TK.

   Lists H_1-list and H_2-list, which are initial empty, are maintained by $\mathcal{B}$ to answer random oracle queries.

   $\mathcal{B}$ sends $\mathcal{SP}, TK, V_{pub}, S_{pub}$ to $\mathcal{A}_4$.

2. *Phase* 1: $\mathcal{A}_4$ is allowed to make the following queries to $\mathcal{B}$.

   - As for H_1-*query* and H_2-*query*, $\mathcal{B}$ performs as in the proof of Theorem 2.
   - *Trapdoor*-query: $\mathcal{A}_4$ submits $W \in \mathcal{W}$ for Trapdoor-query. $\mathcal{B}$ performs as follows:
     (a) Compute $\Theta = \text{Trapdoor}(\mathcal{SP}, W, V_{priv}, S_{pub})$.
     (b) Output $T = \text{TE.Encrypt}\ (\mathcal{SP}, \Theta, S_{pub})$ to $\mathcal{A}_4$ as the answer.
   - *Test*-query: $\mathcal{A}_4$ submits (C_i, T) for *Test-query*. $\mathcal{B}$ performs the following steps:
     (a) Send T to $\mathcal{C}$ for Decrypt oracle, and then $\mathcal{C}$ returns Θ as the answer, where $\Theta = \text{TE.Decrypt}\ (\mathcal{SP}, S_{priv}, T)$.
     (b) Send C_i to $\mathcal{C}$ for Decrypt oracle, and then $\mathcal{C}$ returns $C = (c_1, c_2)$ as the answer, where $C = \text{TE.Decrypt}(\mathcal{SP}, S_{priv}, C_i)$.
     (c) Check whether the equation $c_1 = e(\Theta, c_2)$ holds or not. If it is the case, 1 is outputted, otherwise 0.

3. *Challenge*: $\mathcal{A}_4$ picks two keywords W_0^*, W_1^* for challenge. For each $b \in \{0,1\}$, $\mathcal{B}$ computes $\Theta_b^* = \text{Trapdoor}(\mathcal{SP}, W_b^*, V_{priv}, S_{pub})$. Then, $\mathcal{B}$ sends Θ_0^*, Θ_1^* to $\mathcal{C}$ as the challenge messages. $\mathcal{C}$ picks $\beta \in_R \{0,1\}$ and generates C^* by encrypting Θ_β^*. $\mathcal{C}$ returns C^* to $\mathcal{B}$ as the challenge ciphertext. As a result, $\mathcal{B}$ outputs the challenge trapdoor $T^* = C^*$ to $\mathcal{A}_4$.

4. *Phase* 2: $\mathcal{A}_4$ can keep making queries as done in Phase 1.

5. *Guess*: $\mathcal{A}_4$ outputs $\beta' \in \{0,1\}$. $\mathcal{B}$ directly sends β' to $\mathcal{C}$ for the challenge.

Clearly, if $\beta' = \beta$, then $\mathcal{B}$ can break the IND-CCA2 security of TE with non-negligible probability. □

Theorem 5 *Given a proposed scheme. Suppose that TE is IND-CCA2 secure and H_1, H_2 are random oracles. If $\mathcal{A}_5$ can break TD-IND2 with non-negligible probability, then there exists an algorithm $\mathcal{B}$ solving the SXDH problem with non-negligible probability.*

Proof. To prove this theorem, we constructs an algorithm $\mathcal{B}$ by running $\mathcal{A}_5$ as a subroutine to solve the SXDH problem. $\mathcal{B}$ receives a challenge instance $\mathcal{D} = (p, e, G_1, G_2, G_T, g_1, g_2, g_1^a, g_1^b, \eta)$ of SXDH problem from the challenger $\mathcal{C}$. $\mathcal{B}$'s task is to decide whether or not $g_1^{ab} = \eta$.

1. *Setup*: $\mathcal{B}$ generates $\mathcal{SP}$, where $\mathcal{D} \in \mathcal{SP}$. Then, $\mathcal{B}$ computes (V_{pub}, V_{priv}) as the receiver V's public/private key. Lists H_1-list and H_2-list, which are initial empty, are maintained by $\mathcal{B}$ to answer random oracle queries.

   $\mathcal{B}$ sends V_{pub} to $\mathcal{A}_5$. $\mathcal{A}_5$ personally computes the server's public/private key (S_{pub}, S_{priv}), and then sends S_{pub} to $\mathcal{B}$.

   $\mathcal{B}$ picks a random number j as the challenge keyword index candidate.

2. *Phase* 1: $\mathcal{A}_5$ is allowed to make the following queries to $\mathcal{B}$.

   - H_1-*query*: $\mathcal{A}_5$ submits $W_i \in \mathcal{W}$ to $\mathcal{B}$. $\mathcal{B}$ responds as follows:
     (a) If $i = j$, return g_1^b as the answer.
     (b) If W_i has been queried previously, $\mathcal{B}$ searches the H_1-list to find r_1 and then returns $g_1^{r_1}$ as the answer; otherwise, $r_1 \in \mathbb{Z}_p^*$ is picked randomly and a new item $[W_i, r_1]$ is stored into H_1-list.
   - H_2-*query*: $\mathcal{A}_5$ submits α to $\mathcal{B}$. $\mathcal{B}$ responds with a random number $g_2^{r_2}$ to $\mathcal{A}_5$: If α has been queried previously, $\mathcal{B}$ searches the H_2-list to find r_2; otherwise, $r_2 \in \mathbb{Z}_p^*$ is picked randomly and a new item $[\alpha, r_2]$ is stored into H_2-list.
   - *Trapdoor*-query: $\mathcal{A}_5$ submits $W_i \in \mathcal{W}$ for this query. $\mathcal{B}$ responds as follows:
     (a) If $i = j$, abort with failure.
     (b) Compute $\Theta = g_1^{ar_1}$, where r_1 is from H_1-list such that $g_1^{r_1} = H_1(W_i)$. Then, it outputs $T = \text{TE.Encrypt}(\mathcal{SP}, \Theta, S_{pub})$ as the answer.
   - *dPEKS-query*: Upon receiving W_i for this query. $\mathcal{B}$ performs as follows:
     (a) Pick $k \in_R \mathbb{Z}_p^*$.
     (b) If $i = j$, set $c_1 = e(\eta, g_2^{r_2})^k$; otherwise, set $c_1 = e(g_1^{ar_1}, g_2^{r_2})^k$, where r_1 (resp. r_2) is from H_1-list (resp. H_2-list) such that $g_1^{r_1} = H_1(W_i)$ (resp. $g_2^{r_2} = H_2(V)$).
     (c) Compute $c_2 = g_2^{kr_2}$ and let $C = (c_1, c_2)$.
     (d) Output $\text{TE.Encrypt}(\mathcal{SP}, C, S_{pub})$ as the searchable ciphertext.

3. *Challenge*: $\mathcal{A}_5$ chooses two keywords W_0^*, W_1^* for challenge. $\mathcal{B}$ picks $\beta \in_R \{0, 1\}$ and performs as follows:

   (a) If W_β^* has been queried for H_1-*query* but it is not the j-th different keyword, then abort with failure.
   (b) Set $\Theta^* = \eta$.
   (c) Compute $T^* = \text{TE.Encrypt}\ (\mathcal{SP}, \Theta^*, S_{pub})$.
   (d) Output T^* as the challenge trapdoor.

4. *Phase* 2: $\mathcal{A}_5$ can keep making queries as done in Phase 1.

5. *Guess*: $\mathcal{A}_5$ outputs $\beta' \in \{0, 1\}$. If $\beta = \beta'$, then $\mathcal{B}$ outputs *yes* to $\mathcal{C}$ as the answer to the challenge; otherwise, $\mathcal{B}$ outputs *no*.

Clearly, if $\eta = g^{ab}$, then $\Theta^* = H_1(W_\beta^*)^a$ is a valid trapdoor; otherwise, it is only a random number since H_1 is a random oracle. Hence, $\mathcal{A}_5$ wins the game implies that $\mathcal{B}$ can decide whether or not $\eta = g^{ab}$ correctly with non-negligible probability. □

5.4 Comparisons

Tab. 1 shows the security comparisons among literature schemes.

Compared with SPEKS, our proposal requires one more algorithm *Token*. However, it is trade-off between efficiency and security. On the other hand, SPEKS is builded from IBE and TE, while ours is more simpler since only TE is required.

Table 1: Security comparisons among literature schemes

Scheme	Secure Channel	SC-IND1	SC-IND2	OC-IND	TD-IND1	TD-IND2
BCOP [3]	Required	Secure	Insecure	Insecure	Insecure	Insecure
BSS [2]	Required	Secure	Secure	Insecure	Insecure	Insecure
RPSL [14]	Required	Secure	Secure	Insecure	Secure	Insecure
SPEKS [6]	Free	Secure	Secure	Secure	Secure	Insecure
Ours	Free	Secure	Secure	Secure	Secure	Secure

6 Conclusion

In this paper, we note that off-line keyword guessing attack can be launched by a malicious server to find the keyword used for generating the trapdoor, which was not considered in the related work. SPEKS can suffer from this kind of attack. Moreover, the security model defined for TD-IND in SPEKS is incomplete. Owing to the shown weaknesses, we enhanced the existing security models for trapdoor indistinguishability by defining two new security models. We also proposed a new framework.

Acknowledgment

This work was supported by Shandong Special Project of Education Enrollment Examination (No.ZK1437B005), Chinese Ministry of Education, Humanities and Social Sciences Research Project (No. 14YJCZH136), Shandong "Twelfth Five Year" Language Application Research Project (No.3032), The First Characteristic of Elite Schools Construction Project of Qingdao University (No.05091304), Innovative Teaching Laboratory Research Project in 2014 of Qingdao University (No.10).

References

[1] Abdalla, M. et al.: 'Searchable encryption revisited: consistency properties, relation to anonymous ibe, and extensions'. J. Cryptol., 21, 2008. pp. 350-391.

[2] Baek, J., Safavi-Naini, R., Susilo, W.: 'Public Key Encryption with Keyword Search Revisited'. ICCSA'08. 5072(2008). pp.1249-1259.

[3] Boneh, D., Crescenzo, G.D., Ostrovsky,R., Persiano,G.: 'Public key encryption with keyword search'. EUROCRYPT'04. 3027(2004). pp. 506C522.

[4] Boneh, D., Franklin, M.: 'Identity-Based Encryption from the Weil Pairing'. CRYPTO'01. 2139(2001). pp.213-229.

[5] Byun, J.W., Rhee, H.S., Park, H.A., Lee, D.H.:'Off-line Keyword Guessing Attacks on Recent Keyword Search Schemes Over Encrypted Data'. SDM'06. 4165(2006). pp. 75-83.

[6] Chen, Y.C.: 'SPEKS: Secure Server-Designation Public Key Encryption with Keyword Search against Keyword Guessing Attacks'. The Computer Journal, Vol. 58 No. 4, 2015. pp.922-933.

[7] Chen, R., Mu, Y., Yang, G., Guo, F., Wang, X.: 'A new general framework for secure public key encryption with keyword search'. in ACISP, 2015, pp. 59-76.

[8] Chen, R., Mu, Y., Yang, G., Guo, F., Wang, X.: 'Dual-server public-key encryption with keyword search for secure cloud storage'. IEEE Transactions on Information Forensics and Security, vol. 11, no. 4, pp. 789-798, 2016.

[9] Hu, C., Liu, P.: 'An enhanced searchable public key encryption scheme with a designated tester and its extensions'. J.Comput., 7, 2012. pp.716-723.

[10] Hwang, Y.H., Lee, P.J.: 'Public Key Encryption with Conjunctive Keyword Search and its Extension to a Multi-user System'. Proc. Pairing'07. 4575(2007). pp. 2-22.

[11] Jeong, I.R., Kwon, J.O., Hong, D., Lee, D.H.: 'Constructing PEKS schemes secure against keyword guessing attacks is possible?' Comput. Commun., 32, 2009. pp.394-396.

[12] Nigel, P. Smart and Frederik Vercauteren. On computable isomorphisms in efficient asymmetric pairing-based systems. Discrete Applied Mathematics, 155(4), 2007. pp.538-547.

[13] Park, D.J., Kim, K., Lee, P.J.: 'Public Key Encryption with Conjunctive Field Keyword Search'. WISA'04. 3325(2004). pp. 73-86.

[14] Rhee, H.S., Park, J.H., Susilo,W., Lee, D.H.:'Trapdoor security in a searchable public-key encryption scheme with a designated tester'. J. Syst. Softw., 83, 2010. pp.763-771.

[15] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. Discrete Applied Mathematics, 156(16), 2008. pp.3113-3121.

[16] Tang, Q.: 'Nothing is for free: security in searching shared and encrypted data'. IEEE Transactions on Information Forensics and Security, 9(11), 2014, pp.1943-1952.

[17] Xu,P., Wu,Q., Wang,W., Susilo, W., Domingo-Ferrer, J., Jin, H.: 'Generating Searchable Public-Key Ciphertexts with Hidden Structures for Fast Keyword Search'. IEEE Transactions on Information Forensics and Security, 10(9), pp. 1993-2006, 2015.

[18] Xu,P., Jin, H., Wu, Q., Wang, W.: 'Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack,' IEEE Trans. Computers, vol. 62, no. 11, pp. 2266-2277, 2013.

[19] Yau,W.C., Phan, R.C., Heng, S.H., Goi, B.M.:'Keyword Guessing Attacks on Secure Searchable Public Key Encryption Schemes with a Designated Tester'. Int. J. Comput. Math., 90, 2013. pp.2581-2587.

[20] Zhang, B., Zhang, F.: 'An efficient public key encryption with conjunctive-subset keyword search'. J. Netw. Comput. Appl., 34, 2011. pp.262-267.