2017 IEEE Trustcom/BigDataSE/ICESS

Hidden-Token Searchable Public-Key Encryption

Cong Zuo[†], Jun Shao[†*], Zhe Liu[‡], Yun Ling[†] and Guiyi Wei[†]
[†]School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, China
Email: zuocong10@gmail.com, chn.junshao@gmail.com, yling@zjgsu.edu.cn, weigy@zjgsu.edu.cn
[‡]APSIA, Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg
Email: sduliuzhe@gmail.com

***Abstract*—In this paper, we propose a variant of searchable public-key encryption named hidden-token searchable public-key encryption with two new security properties: token anonymity and one-token-per-trapdoor. With the former security notion, the client can obtain the search token from the data owner without revealing any information about the underlying keyword. Meanwhile, the client cannot derive more than one token from one trapdoor generated by the data owner according to the latter security notion. Furthermore, we present a concrete hidden-token searchable public-key encryption scheme together with the security proofs in the random oracle model.**

***Index Terms*—Searchable Encryption, Public-Key, Hidden-Token, Random Oracle Model**

I. Introduction

In recent years, the development and deployment of Internet of Things (IoT) have gained substantial attention in the industry and research community since it enables people to collect data everywhere including environment, infrastructures and businesses. However, it is not easy to store this huge amount of information locally. The popular solution nowadays is to outsource these data to a third party (e.g. the cloud). While these data are the assets of the one who implemented the devices of IoT, they should be encrypted before outsource, which at the same time jeopardizes the usefulness of these data, such as searchability.

To solve this dilemma between usefulness and confidentiality of data, many research efforts have been proposed, e.g. Oblivious RAM (ORAM) [1], [2], Private Information Retrieval (PIR) [3], [4] and Searchable Encryption (SE) [5], [6], [7], [8]. Among these, searchable encryption, which enables searchability on ciphertexts, is the most promising technique. For instance, the SE scheme in [8] allows many data sources to generate encrypted index of the data and the data owner to delegate the searchability by giving search trapdoors to the clients. By using this SE scheme, the devices of IoT can encrypt the collected data, and upload the resulting ciphertexts with the index generated by using this SE scheme to the cloud directly. Meanwhile, the data owner can generate the search trapdoor according to the query from the client, who can later use the search token generated from the trapdoor to ask the cloud do the search on the index. The high level description of the above framework is given in Fig. 1.

[*]Corresponding author.

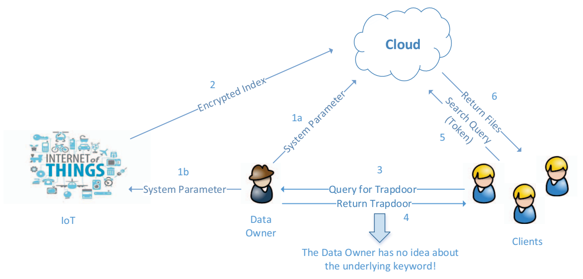

Fig. 1. Architecture of our framework

In most of the existing SE schemes, the data owner is usually considered as trusted, while the cloud and the client are considered as malicious. They focus on how to protect the confidentiality of the trapdoor and the corresponding keyword, while the privacy of the client is always ignored. However, in some situations, the way how the client deals with the data could be considered as an asset. For example, the information of stock market and how to deal with the information are both valuable. Hence, the keyword corresponding to the trapdoor should be also protected. Once the client privacy (keyword hidden) is considered, the benefit of the data owner could be hurt. For example, without knowing the underlying keyword of the trapdoor, the client could trick the data owner to generate one trapdoor corresponding to many tokens. It is not good for the data owner especially when the payment of data service is based on how many tokens the client obtains.

To address the above challenges, in this paper, we propose a variant of searchable encryption, named hidden-token searchable encryption. One of the important security properties of hidden-token searchable encryption is to protect the client privacy, i.e., the data owner cannot deduce the keyword corresponding to the queried trapdoor. Besides, we also guarantee the benefit of the data owner when the client privacy is under protection, i.e., one trapdoor can derive only one token. The contributions of this paper can be summarized as follows.

- To protect the client privacy while guaranteeing the benefit of the data owner, we propose the concept of hidden-token searchable encryption, including its definitions and security models.
- We present the first concrete hidden-token searchable encryption scheme along with its security analysis.

A. *Related Works*

Generally speaking, there are two kinds of searchable encryption, one is searchable symmetric-key encryption, the other is searchable public-key encryption.

Song et al. [5] were the first one using symmetric-key encryption to address keyword search on encrypted data. Later, many works have been done in this area [5], [9], [10]. The term of *Searchable Symmetric-Key Encryption* was first introduced by Curtmola et al. [6]. After that, Kamara et al. [11] proposed a dynamic searchable encryption which extends the scheme in [6]. To make the searchable symmetric-key encryption more expressive, in 2013, Cash et al. [12] proposed a practical highly-scalable searchable symmetric encryption which is quite efficient and expressive. Later, many schemes have been proposed based on this scheme. In 2013, Jarecki et al. [13] proposed searchable symmetric encryption with private information retrieval. In 2014, Cash et al. [14] proposed dynamic searchable encryption which extends [12]. To make the scheme in [12] more expressive, in 2016, Zuo et al. [7] proposed a trusted boolean search on cloud using searchable symmetric encryption.

In the line of research of searchable symmetric-key encryption, the most basic setting is where the data owner is the one who performing the search on the encrypted database which is stored in a third party (e.g. the cloud). The work of Curtmola et al. [6] was the first scheme to extend the two-party model (the data owner and the server) of SSE to the multi-client setting. However, this scheme did not allow the interaction between the data owner and the client in each query which led to the inefficient implementation. To circumvent this obstacle, in 2010, Chase et al. [15] introduced a SSE with controlled disclosure which allowed such interaction. Later, in 2011, to protect the privacy of the query, De Cristofaro et al. [16] proposed a privacy-preserving sharing of sensitive information scheme which extended the multi-client SSE to the OSPIR setting. Later, [13] introduced a more expressive and scalable SSE with OSPIR by using the technique of [12].

Most of the above schemes do not consider the protection of the client privacy, only the scheme in [16], [13] considered the multi-client with private information retrieval which called OSPIR. However, in OSPIR, the data owner still knows some information about the keyword, e.g., the attribute.

The concept of *Searchable Public-Key Encryption* was first introduced by Boneh et al. [8]. Later, many schemes [17], [18], [19] improved the security of the scheme. In [18], Rhee et al. first gave the security notion named trapdoor indistinguishability against an active attacker who is able to get trapdoors for any keyword of his choice. This security of a trapdoor guarantees that the trapdoor does not reveal any information on any keyword without the data owners secret key. However, this security is different from the one we study in this paper. In particular, we aim to guarantee that the data owner cannot know the keyword from trapdoors, while trapdoor indistinguishability cannot guarantee this. In [19], Zhu et al. proposed a searchable public-key encryption scheme which provided predicate privacy. However, the scheme cannot hide the keyword from the data owner who is responsible for issuing the trapdoor either. Moreover, this scheme is quite inefficient in the real world.

The main difference between searchable symmetric-key encryption and searchable public-key encryption is that the encryption key is public or not. Recall the IoT application of searchable encryption, searchable public-key encryption is more suitable. Hence, in this paper, we only consider the (hidden-token) searchable public-key encryption.

B. *Organization*

The remaining paper is organized as follows. In Section II, we give the definition of our hidden-token searchable public-key encryption (HSPE) and the corresponding security models. Besides that, we also give the necessary background for our assumption. After that, we present a concrete HSPE scheme in Section III and its security analysis in Section IV. Finally, we give the conclusion in Section V.

II. Definitions and Security Models

In this section, we present a variant of searchable public-key encryption named hidden-token searchable public-key encryption (HSPE), including its definitions and security models. Compared to the normal searchable public-key encryption, HSPE aims to prevent the search keyword of clients being leaked to the data owner while protecting the data owner's rights. Furthermore, in this section we also review the complexity assumptions that are used to prove the security of our proposed HSPE scheme.

A. *Definition of Hidden-Token Searchable Public-Key Encryption*

As mentioned before, HSPE aims to keep the search keyword of clients secret from the data owner, while clients cannot abuse this property. For easy expression, we separate the trapdoor generation algorithm in the normal searchable public-key encryption into three algorithms: Randomization, Trapdoor Generation and Token Generation algorithms. The details of the definition are as follows.

Definition 1 (Hidden-Token Searchable Public-Key Encryption)**.** *A hidden-token searchable public-key encryption scheme consists of the following algorithms:*

- $(pk, sk) \leftarrow \mathbf{Setup}(1^\lambda)$*: It takes the security parameter* 1^λ *as input, the setup algorithm* $\mathbf{Setup}$ *outputs a public-key* pk *and the corresponding private key* sk*. This algorithm is usually performed by the data owner who owns the data stored in a third party, e.g., a cloud.*
- $(s, kw') \leftarrow \mathbf{Rand}(kw, pk)$*: It takes a keyword* kw *and (part of) the public-key as input, the keyword randomization algorithm* $\mathbf{Rand}$ *outputs a masked keyword* kw' *corresponding to* kw *and a state* s*. This algorithm is usually performed by the client who wants to do the search on the encrypted data.*
- $(\mathtt{trap}_{kw}) \leftarrow \mathbf{Trap}(kw', sk, pk)$*: It takes a masked keyword* kw' *corresponding to* kw *and the key pair* (sk, pk)

as input, the trapdoor generation algorithm $\mathbf{Trap}$ *outputs a trapdoor* $\mathtt{trap}_{kw}$ *corresponding to the masked keyword* kw'*. This algorithm is usually performed by the data owner who holds the private key* sk*.*

- $(\mathtt{token}_{kw}) \leftarrow \mathbf{Token}(\mathtt{trap}_{kw}, s, pk)$*: It takes a trapdoor* $\mathtt{trap}_{kw}$ *and state value* s *corresponding to the masked keyword* kw'*, and (part of) public-key* pk *as input, the token generation algorithm* $\mathbf{Token}$ *outputs a token* $\mathtt{token}_{kw}$ *corresponding to the masked keyword* kw'*. This algorithm is usually performed by the client who wants to do the search on the encrypted data.*
- $(I_{kw}) \leftarrow \mathbf{Index}(kw, pk)$*: It takes a keyword* kw *and (part of) a public-key* pk *as input, the index generation algorithm* $\mathbf{Index}$ *outputs an index* I_{kw} *of the data stored in the third party. This algorithm can be performed by the party who holds the public key* pk*.*
- $(1 \text{ or } 0) \leftarrow \mathbf{Test}(I_{kw_I}, \mathtt{token}_{kw_t}, pk)$*: It takes an index* I_{kw_I} *corresponding to keyword* kw_I*, a token* $\mathtt{token}_{kw_t}$ *corresponding to keyword* kw_t*, and (part of) the public-key* pk *as input, the test algorithm* $\mathbf{Test}$ *outputs* 1 *if* $kw_I = kw_t$ *or* 0 *otherwise.*

Correctness. The correctness of a HSPE scheme requires that $(pk, sk) \leftarrow \mathbf{Setup}(1^\lambda)$, and any pair $(s, kw') \leftarrow \mathbf{Rand}(kw_t, pk)$, the following conditions must hold: if $kw_t = kw_I$, then the value of $\mathbf{Test}(\mathbf{Index}(kw_I, pk), \mathbf{Token}(\mathbf{Trap}(kw', sk, pk), s, pk), pk)$ is 1; otherwise, it is 0.

B. Security Models of Hidden-Token Searchable Public-Key Encryption

Like the normal searchable public-key encryption, the hidden-token searchable public-key encryption should hold *Index Anonymity* security that aims to prevent adversaries knowing keywords of the data from the index. Furthermore, as the underlying applications require, the hidden-token searchable public-key encryption should also hold *Token Anonymity* to prevent the data owner knowing search keywords of clients via the trapdoor generation, and *One-Token-Per-Trapdoor* security to prevent clients abusing the token anonymity security. The details are as follows.

1) Index Anonymity: The index anonymity is defined via the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- **Setup:** The challenger $\mathcal{C}$ takes a security parameter 1^λ and runs $\mathbf{Setup}$ algorithm to get a key pair (pk, sk). It gives the adversary $\mathcal{A}$ the resulting pk while keeping the secret key sk to itself.
- **Phase 1:** In this phase, $\mathcal{A}$ is allowed to query the following oracles adaptively.
  - $\mathcal{O}_{rand}(kw)$: On input a keyword kw, $\mathcal{C}$ returns a corresponding masked keyword kw' and a state s.
  - $\mathcal{O}_{trap}(kw')$: On input a masked keyword kw', where kw' is generated from $\mathcal{O}_{rand}$, $\mathcal{C}$ returns the corresponding trapdoor $\mathtt{trap}_{kw}$.
  - $\mathcal{O}_{token}(\mathtt{trap}_{kw}, s)$: On input a trapdoor $\mathtt{trap}_{kw}$ corresponding to the masked keyword kw' and a state s, where (kw', s) are generated from $\mathbf{Rand}(kw, pk)$, $\mathcal{C}$ returns the corresponding token $\mathtt{token}_{kw}$.
- **Challenge:** $\mathcal{A}$ sends two keywords kw_0^*, kw_1^* of equal length to $\mathcal{C}$. kw_0 and kw_1 are restricted by the following condition: The adversary never queries $\mathcal{O}_{trap}$ with kw' corresponding to kw_0^* or kw_1^*. Note that oracle $\mathcal{O}_{trap}$ never answers the query with kw' that is not from $\mathcal{O}_{rand}$, which enables the challenger $\mathcal{C}$ to check whether kw_0 and kw_1 violate the above restriction. On receiving the valid kw_0^*, kw_1^*, $\mathcal{C}$ encapsulates kw_b^* as the challenge Index $I^*_{kw_b^*}$ to $\mathcal{A}$, where b is chosen randomly from $\{0, 1\}$.
- **Phase 2:** Almost the same as Phase 1, except that the restriction in Challenge phase also validates in this phase.
- **Guess:** $\mathcal{A}$ outputs a guess b' of b, and wins the game if $b = b'$.

The advantage $\mathbf{Adv}_{IA}(1^\lambda)$ is defined as $|\Pr[b = b'] - 1/2|$ for the index anonymity game. The HSPE scheme is said to be $CPA-IA$ secure if all efficient adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}_{IA}(1^\lambda)$ is negligible.

2) Token Anonymity: The token anonymity is also defined by a game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. As mentioned before, the token anonymity is used to prevent the data owner knowing search keywords of clients. In this case, the data owner could be corrupted by the adversary.

- **Setup:** Identical to that in the index anonymity game.
- **Phase 1:** $\mathcal{A}$ is allowed to query the following oracles adaptively.
  - $\mathcal{O}_{sk}(pk)$: On input the public-key pk, $\mathcal{C}$ returns the corresponding private key sk. Note that this oracle is allowed to be queried only once during the whole game.
  - $\mathcal{O}_{rand}(kw)$: On input a keyword kw, $\mathcal{C}$ returns a corresponding masked keyword kw' and a state s.
  - $\mathcal{O}_{trap}(kw')$: On input a masked keyword kw', $\mathcal{C}$ returns the corresponding trapdoor $\mathtt{trap}_{kw}$.
  - $\mathcal{O}_{token}(\mathtt{trap}_{kw}, s)$: On input a trapdoor $\mathtt{trap}_{kw}$ corresponding to the masked keyword kw' and a state s, $\mathcal{C}$ returns the corresponding token $\mathtt{token}_{kw}$.
- **Challenge:** $\mathcal{A}$ sends two keywords kw_0^*, kw_1^* of equal length to $\mathcal{C}$. $\mathcal{C}$ generates the masked keyword $kw_b'^*$ and returns to $\mathcal{A}$ as the challenge masked keyword, where b is chosen randomly from $\{0, 1\}$.
- **Phase 2:** Identical to Phase 1.
- **Guess:** $\mathcal{A}$ outputs a guess b' of b, and wins the game if $b = b'$.

The advantage $\mathbf{Adv}_{TA}(1^\lambda)$ is defined as $|\Pr[b = b'] - 1/2|$ for the token anonymity game. The HSPE scheme is said to be $CPA-TA$ secure if all efficient adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}_{TA}(1^\lambda)$ is negligible.

3) One-Token-Per-Trapdoor: The One-Token-Per-Trapdoor security guarantees that clients can only derive one token from one trapdoor, which prevents clients abusing the token anonymity. Likewise, it is also defined by a game played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The details are as follows.

- **Setup:** Identical to that in the index anonymity game.
- **Phase 1:** $\mathcal{A}$ is allowed to query the following oracles adaptively.
  - $\mathcal{O}_{rand}(kw)$: On input a keyword kw, $\mathcal{C}$ returns a corresponding masked keyword kw' and a state s.
  - $\mathcal{O}_{trap}(kw')$: On input a masked keyword kw', $\mathcal{C}$ returns the corresponding trapdoor $\mathtt{trap}_{kw}$.
  - $\mathcal{O}_{token}(\mathtt{trap}_{kw}, s)$: On input a trapdoor $\mathtt{trap}_{kw}$ corresponding to the masked keyword kw' and a state s, $\mathcal{C}$ returns the corresponding token $\mathtt{token}_{kw}$.
- **Challenge:** $\mathcal{A}$ sends a masked keyword kw'^* to $\mathcal{C}$. $\mathcal{C}$ outputs $\mathbf{Trap}(kw'^*, sk)$ as the challenge trapdoor.
- **Phase 2:** Identical to Phase 1.
- **Output:** $\mathcal{A}$ outputs two tokens $(\mathtt{token}^*_{kw_0^*}, s_0^*)$ and $(\mathtt{token}^*_{kw_1^*}, s_1^*)$. If $(kw'^*, s_b^*) \leftarrow \mathbf{Rand}(kw_b^*, pk)$ and $kw_0^* \neq kw_1^*$, $\mathcal{A}$ wins the game.

The advantage $\mathbf{Adv}_{OT}(1^\lambda)$ is defined as $\Pr[\mathcal{A} \text{ wins}]$ for the one-token-per-trapdoor security game. The HSPE scheme is said to be *OT* secure if all efficient adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}_{OT}(1^\lambda)$ is negligible.

C. Bilinear Map and Complexity Assumptions

In this part, we will briefly review the bilinear map, decisional bilinear Diffie-Hellman assumption and extended discrete logarithm assumption that will be used in our proposal.

1) Bilinear Map: Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of the same big prime order p, and g be a generator of $\mathbb{G}$. let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a pairing, i.e. a map satisfies the following properties:

- Bilinearity. $e(g^a, g^b) = e(g, g)^{ab}$ for any $a, b \in \mathbb{Z}_p^*$.
- Non-degenerate. $e(g, g)$ is a generator of group $\mathbb{G}_T$.
- Computability. e can be computed efficiently.

We denote $\mathtt{BSetup}$ as an algorithm that takes as input the security parameter 1^λ and outputs the parameters for a bilinear map as $(p, g, \mathbb{G}, \mathbb{G}_T, e)$.

2) Decisional Bilinear Diffie-Hellman (DBDH) Assumption: Let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear map, both $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of prime order p. Choose a random generator g of $\mathbb{G}$ and random a, b, c, z from $\mathbb{Z}_p^*$. The decisional Bilinear Diffie-Hellman (DBDH) problem is to distinguish between the tuples of the form $(g, g^a, g^b, g^c, e(g, g)^{abc})$ and $(g, g^a, g^b, g^c, e(g, g)^z)$. An algorithm $\mathcal{A}$ has an advantage ϵ in solving DBDH if

$$|\Pr[\mathcal{A}(g, g^a, g^b, g^c, e(g, g)^{abc}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c, e(g, g)^z) = 1]| \geq \epsilon.$$

The decisional Bilinear Diffie-Hellman (DBDH) assumption is that for any efficient $\mathcal{A}$, ϵ is negligible.

3) Extended Discrete Logarithm (eDL) Assumption: Let $\mathbb{G}$ be a cyclic group of prime order p. Choose a random generator g of $\mathbb{G}$ and random a, b from $\mathbb{Z}_p^*$. The extended discrete logarithm (eDL) problem is to compute a/b from the tuple (g, g^a, g^b). An algorithm $\mathcal{A}$ has an advantage ϵ in solving eDL if

$$\Pr[\mathcal{A}(g, g^a, g^b) = a/b] \geq \epsilon.$$

The extended discrete logarithm (eDL) assumption is that for any efficient $\mathcal{A}$, ϵ is negligible.

III. The Proposed HSPE Scheme

In this section, we present our concrete HSPE scheme which is based on the identity-based encryption scheme due to Boneh and Franklin [20]. The details are as follows.

- **Setup**: On input the security parameter 1^λ, it runs $\mathtt{BSetup}(1^\lambda)$ to obtain $(p, g, \mathbb{G}, \mathbb{G}_T, e)$. After that, it chooses a random sk from $\mathbb{Z}_p^*$ and computes $pk = g^{sk}$. It also chooses a secure cryptographic hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$. The security analysis will consider H as random oracle. At last, it publishes $(p, g, \mathbb{G}, \mathbb{G}_T, e, pk, H)$ as the public-key while keeping sk secret.
- **Rand**: On input a keyword kw, it outputs $kw' = H(kw)^r$ as the masked keyword and $s = r$ as the state, where r is chosen randomly from $\mathbb{Z}_p^*$.
- **Trap**: On input a masked keyword $kw' = H(kw)^r$ and the private key sk, it outputs a trapdoor $\mathtt{trap}_{kw} = (kw')^{sk}$.
- **Token**: On input a trapdoor $\mathtt{trap}_{kw}$ and a state s, it outputs a token $\mathtt{token}_{kw} = \mathtt{trap}_{kw}^{1/s}$.
- **Index**: On input a keyword kw and the public-key $(p, g, \mathbb{G}, \mathbb{G}_T, e, pk, H)$, it outputs an index $I_{kw} = (I_{kw}^{(1)}, I_{kw}^{(2)}) = (g^t, e(H(kw), pk)^t)$, where t is chosen randomly from $\mathbb{Z}_p^*$.
- **Test**: On input a token $\mathtt{token}_{kw_t}$ and an index I_{kw_I}, it outputs 1 if $e(\mathtt{token}_{kw_t}, I_{kw_I}^{(1)}) \overset{?}{=} I_{kw_I}^{(2)}$ holds, or 0 otherwise.

Correctness: We can easily obtain the correctness of our proposal according to the correctness of Boneh-Franklin scheme [20]. Especially, when $kw_t = kw_I$, we have that $e(\mathtt{token}_{kw_t}, I_{kw_I}^{(1)}) = e(\mathtt{trap}_{kw_t}^{1/r}, g^t) = e(((H(kw_t)^r)^s k)^{1/r}, g^t) = e(H(kw_t), pk)^t = e(H(kw_I), pk)^t = I_{kw_I}^{(2)}$, and $e(\mathtt{token}_{kw_t}, I_{kw_I}^{(1)}) \neq I_{kw_I}^{(2)}$ if $kw_t \neq kw_I$.

IV. Security Analysis

In this section, we will prove that our proposal holds the index anonymity, token anonymity and one-token-per-trapdoor security properties in the random oracle model.

Theorem 1 (Index Anonymity)**.** *Our proposal holds the index anonymity based on the DBDH assumption in the random oracle model.*

Proof: Assume that $\mathcal{A}$ is an index anonymity adversary that has advantage ϵ against our proposal, then we can build an algorithm $\mathcal{B}$ solving the DBDH problem with advantage at least $4\epsilon/e^2(2+q)^2$ via interacting $\mathcal{A}$ with the following game, where $\mathcal{A}$ makes at most $q > 0$ hash queries.

- **Setup:** The challenger $\mathcal{C}$ takes a DBDH tuple $(g, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, T)$, where T is either $e(g,g)^{\mathbf{abc}}$ or $e(g,g)^{\mathbf{z}}$. $\mathcal{B}$ sets public-key as $(g, pk = g^{\mathbf{a}}, H)$. Note that $sk = \mathbf{a}$ is unknown to $\mathcal{B}$. H is a random oracle controlled by $\mathcal{B}$ as described below.
  $\mathcal{O}_h$: All hash values in this game are generated from this oracle. To answer the hash queries, algorithm $\mathcal{B}$ should maintain a list named H^{list} of tuples $\{kw_i, Q_i, \alpha_i, coin_i\}$ that is built as below. H^{list} is initially empty, and algorithm $\mathcal{B}$ responds the hash queries kw_i as follows.
  1) If kw_i exists in a tuple $\{kw_i, Q_i, \alpha_i, coin_i\}$ of H^{list} then algorithm $\mathcal{B}$ responds with $H(kw_i) = Q_i \in \mathbb{G}^*$.
  2) Otherwise, $\mathcal{B}$ chooses a random $coin_i \in \{0,1\}$ with $\Pr[coin_i = 0] = \delta$, where δ will be determined later. Algorithm $\mathcal{B}$ chooses a random $\alpha_i \in \mathbb{Z}_p^*$. If $coin_i = 0$ compute $Q_i = g^{\alpha_i} \in \mathbb{G}^*$; otherwise, $Q_i = (g^{\mathbf{b}})^{\alpha_i}$. At last, algorithm $\mathcal{B}$ adds the tuple $\{kw_i, Q_i, \alpha_i, coin_i\}$ into H^{list} and responds to $\mathcal{A}$ with $H(kw_i) = Q_i$.

  Note that either way Q_i is uniformly distributed in $\mathbb{G}^*$ and independent of $\mathcal{A}$'s current view as required.
- **Phase 1:** $\mathcal{B}$ builds the following oracles.
  - $\mathcal{O}_{rand}$: This oracle maintains a list of tuples $\{kw_i, kw_i', s_i\}$, named R^{list}. On input kw_i, algorithm $\mathcal{B}$ chooses a random r_i and sets $kw_i' = Q_i^{r_i}$ and $s_i = r_i$, where Q_i is from $\mathcal{O}_h(kw_i)$. At last, algorithm $\mathcal{B}$ gives (kw_i', s_i) to algorithm $\mathcal{A}$ and stores (kw_i, kw_i', s_i) into R^{list}.
  - $\mathcal{O}_{trap}$: On input kw_i', algorithm $\mathcal{B}$ searches kw_i' in R^{list}. If it does not exist, algorithm $\mathcal{B}$ refuses this query; otherwise, it searches kw_i in H^{list}. If $coin_i = 0$, compute $\mathtt{trap}_{kw_i} = (g^{\mathbf{a}})^{\alpha_i s_i}$, where α_i and s_i are the values corresponding to kw_i in H^{list} and R^{list}, respectively; otherwise, it reports *failure* and aborts.
  - $\mathcal{O}_{token}$: On input a pair $(\mathtt{trap}_i, s_i)$, $\mathcal{B}$ responds $\mathcal{A}$ with $\mathtt{token}_i = \mathtt{trap}_i^{1/s_i}$.
- **Challenge:** $\mathcal{A}$ sends two keywords kw_0^*, kw_1^* with the restrictions specified in the index anonymity game. $\mathcal{B}$ queries $\mathcal{O}_h$ with kw_0^* and kw_1^*. If $coin_{kw_0^*} = 0$ or $coin_{kw_1^*} = 0$, then it reports *failure* and aborts. Otherwise, it computes $I_{kw_b^*}^* = (g^{\mathbf{c}}, T^{\alpha_{kw_b^*}})$, where b is chosen randomly. Note that if $T = e(g,g)^{\mathbf{abc}}$, then $I_{kw_b^*}^*$ is a valid index of keyword kw_b^*.
- **Phase 2:** Almost the same as Phase 1, except that kw_0 and kw_1 should follow the restriction as specified in the index anonymity game.
- **Guess:** $\mathcal{A}$ outputs a guess b' of b.

If algorithm $\mathcal{B}$ does not abort during the simulation and the input tuple is sampled from $e(g,g)^{\mathbf{abc}}$, then algorithm $\mathcal{A}$'s view is identical to its view in a real attack game and therefore $\mathcal{A}$ satisfies $|\Pr[b = b'] - 1/2| \geq \epsilon$.

To complete the proof, it remains to calculate the probability that algorithm $\mathcal{B}$ does not abort during the simulation, while the analysis can be proceeded similar as that in [20]. The probability that $\mathcal{B}$ does not abort in phase 1 or 2 is δ^q which is same as [20]. While in the challenge phase, $\mathcal{B}$ queries $\mathcal{O}_h$ twice. Hence, the probability that $\mathcal{B}$ does not abort during the challenge phase is $(1-\delta)^2$. So, the probability that $\mathcal{B}$ does not abort in this game is $\delta^q(1-\delta)^2$. When $\delta = q/(2+q)$, this value is maximized which is $4/e^2(2+q)^2$. Therefore, the probability that $\mathcal{B}$ does not abort is at least $4/e^2(2+q)^2$, where $\mathcal{A}$ makes at most $q > 0$ hash queries. As a result, $\mathcal{B}$'s advantage is at least $4\epsilon/e^2(2+q)^2$ as required. ■

Theorem 2 (Token Anonymity)**.** *Our proposal holds the token anonymity, and the adversary's advantage is zero.*

Proof: We will show that $\mathcal{A}$'s winning probability is exact 1/2 in the below.

- **Setup:** The challenger $\mathcal{C}$ runs $\mathbf{Setup}(1^\lambda)$ to obtain the key pair (pk, sk), and sends pk to $\mathcal{A}$ while keeping sk secret.
- **Phase 1:** The challenger $\mathcal{C}$ builds the following oracles.
  - $\mathcal{O}_{sk}(pk)$: The challenger $\mathcal{C}$ simply returns sk to the adversary $\mathcal{A}$.
  - $\mathcal{O}_{rand}$: On input kw_i, the challenger $\mathcal{C}$ runs $\mathbf{Rand}(kw_i, pk)$ to obtain (kw_i', s_i) and sends (kw_i', s_i) to $\mathcal{A}$.
  - $\mathcal{O}_{trap}$: On input kw_i', the challenger $\mathcal{C}$ runs $\mathbf{Trap}(kw_i', sk, pk)$ to obtain $\mathtt{trap}_{kw_i}$ and sends $\mathtt{trap}_{kw_i}$ to $\mathcal{A}$.
  - $\mathcal{O}_{token}$: On input a pair $(\mathtt{trap}_i, s_i)$, the challenger $\mathcal{C}$ runs $\mathbf{Token}(\mathtt{trap}_i, s_i)$ to obtain $\mathtt{token}_{kw_i}$ and sends $\mathtt{token}_{kw_i}$ to $\mathcal{A}$.
- **Challenge:** $\mathcal{A}$ sends two keywords kw_0^*, kw_1^* with the restrictions specified in the token anonymity game. $\mathcal{C}$ chooses $coin$ from $\{0,1\}$ randomly. If $coin = 1$, it returns $H(kw_b^*)^r$ to $\mathcal{A}$, where b and r are chosen randomly from $\{0,1\}$ and $\mathbb{Z}_p^*$, respectively. If $coin = 0$, it returns a random R^* from $\mathbb{G}$ to $\mathcal{A}$.
- **Phase 2:** Identical to Phase 1.
- **Guess:** $\mathcal{A}$ outputs a guess b' of b.

It is easy to see that if R^* is responded to $\mathcal{A}$, then $\Pr[b' = b] = 1/2$. On the other hand, r is chosen randomly from $\mathbb{Z}_p^*$, hence $H(kw_b^*)^r$ has the same distribution with R from the view of the adversary and we have that $\Pr[b' = b] = 1/2$ when $H(kw_b^*)^r$ is responded to $\mathcal{A}$. ■

Theorem 3 (One-Token-Per-Trapdoor)**.** *Our proposal holds the one-token-per-trapdoor security based on the eDL assumption in the random oracle model.*

Proof: Assume that $\mathcal{A}$ is a one-token-per-trapdoor adversary that has advantage ϵ against our proposal, then we can build an algorithm $\mathcal{B}$ solving the eDL problem by interacting with $\mathcal{A}$ as follows.

- **Setup:** The challenger $\mathcal{C}$ takes an eDL tuple $\{g, g^{\mathbf{a}}, g^{\mathbf{b}}\}$, and runs $\mathbf{Setup}(1^\lambda)$ to obtain the key pair (pk, sk), and sends pk to $\mathcal{A}$ while keeping sk secret. H is a random oracle controlled by $\mathcal{B}$ as described below.

$\mathcal{O}_h$: All hash values in the game are generated from this oracle. To answer the hash queries, algorithm $\mathcal{B}$ should maintain a list named H^{list} of tuples $(kw_i, Q_i, \alpha_i, coin_i)$ that is built as below. H^{list} is initially empty, and algorithm $\mathcal{B}$ responds the hash queries kw_i as follows.

1) If kw_i exists in a tuple $(kw_i, Q_i, \alpha_i, coin_i)$ of H^{list} then algorithm $\mathcal{B}$ responds with $H(kw_i) = Q_i \in \mathbb{G}^*$.
2) Otherwise, $\mathcal{B}$ chooses a random $coin_i \in \{0,1\}$ with $\Pr[coin_i = 0] = \delta$, where δ will be determined later. Algorithm $\mathcal{B}$ chooses a random $\alpha_i \in \mathbb{Z}_p^*$. If $coin_i = 0$ compute $Q_i = (g^{\mathbf{a}})^{\alpha_i} \in \mathbb{G}^*$; otherwise, $Q_i = (g^{\mathbf{b}})^{\alpha_i}$. At last, algorithm $\mathcal{B}$ adds the tuple $(kw_i, Q_i, \alpha_i, coin_i)$ into H^{list} and responds to $\mathcal{A}$ with $H(kw_i) = Q_i$.

Note that either way Q_i is uniformly distributed in $\mathbb{G}^*$ and independent of $\mathcal{A}$'s current view as required.

- **Phase 1:** $\mathcal{B}$ builds the following oracles.
  - $\mathcal{O}_{rand}$: On input kw_i, $\mathcal{B}$ runs $\mathbf{Rand}(kw_i, pk)$ to obtain (kw_i', s_i) and sends (kw_i', s_i) to $\mathcal{A}$.
  - $\mathcal{O}_{trap}$: On input kw_i', $\mathcal{B}$ runs $\mathbf{Trap}(kw_i', sk, pk)$ to obtain $\mathtt{trap}_{kw_i}$ and sends $\mathtt{trap}_{kw_i}$ to $\mathcal{A}$.
  - $\mathcal{O}_{token}$: On input a tuple $(\mathtt{trap}_i, s_i)$, $\mathcal{B}$ runs $\mathbf{Token}(\mathtt{trap}_i, s_i)$ to obtain $\mathtt{token}_{kw_i}$ and sends $\mathtt{token}_{kw_i}$ to $\mathcal{A}$.
- **Challenge:**$\mathcal{A}$ sends a masked keyword kw'^* to $\mathcal{B}$. $\mathcal{B}$ runs $\mathbf{Trap}(kw'^*, sk, pk)$ to obtain $\mathtt{trap}_{kw^*}$ and sends $\mathtt{trap}_{kw^*}$ to $\mathcal{A}$.
- **Phase 2:** Identical to Phase 1.
- **Output:** $\mathcal{A}$ outputs two pairs $(\mathtt{token}^*_{kw_0^*}, s_0^*)$ and $(\mathtt{token}^*_{kw_1^*}, s_1^*)$ satisfying $(kw'^*, s_b^*) \leftarrow \mathbf{Rand}(kw_b^*, pk)$ and $kw_0^* \neq kw_1^*$, where $b \in \{0,1\}$. We have that $(\mathtt{token}^*_{kw_0^*})^{s_0^*} = (\mathtt{token}^*_{kw_1^*})^{s_1^*} = (kw'^*)^{sk}$.

Once $\mathcal{A}$ outputs $(\mathtt{token}^*_{kw_0^*}, s_0^*)$ and $(\mathtt{token}^*_{kw_1^*}, s_1^*)$, $\mathcal{B}$ searches $(\mathtt{token}^*_{kw_0^*})^{1/sk}$ and $(\mathtt{token}^*_{kw_1^*})^{1/sk}$ in H^{list}, and obtains the corresponding $(\alpha_0^*, coin_0^*)$ and $(\alpha_1^*, coin_1^*)$ from H^{list}. If $coin_0^* = coin_1^*$, $\mathcal{B}$ reports *failure* and aborts; otherwise, $\mathcal{B}$ can solve the eDL problem as follows.

- If $coin_0^* = 0$ and $coin_1^* = 1$, then we have that $\mathbf{a} \cdot \alpha_0^* \cdot s_0^* = \mathbf{b} \cdot \alpha_1^* \cdot s_1^* \Rightarrow \mathbf{a}/\mathbf{b} = \alpha_1^* \cdot s_1^* / (\alpha_0^* \cdot s_0^*)$.
- If $coin_0^* = 1$ and $coin_1^* = 0$, then we have that $\mathbf{b} \cdot \alpha_0^* \cdot s_0^* = \mathbf{a} \cdot \alpha_1^* \cdot s_1^* \Rightarrow \mathbf{a}/\mathbf{b} = \alpha_0^* \cdot s_0^* / (\alpha_1^* \cdot s_1^*)$.

It is easy to see that the probability of $coin_0^* \neq coin_1^*$ is $2\delta \cdot (1-\delta)$ which is maximized at $\delta_{opt} = 1/2$. Using δ_{opt}, the probability that $\mathcal{B}$ solves the eDL problem with 1/2 at least as required.

■

V. Conclusion

In this paper, we studied the problem how to keep the keyword corresponding to the trapdoor secret from the data owner. At the same time, we also proposed the security notion named *one-token-per-trapdoor* that aims to protect the benefit of the data owner. In particular, we present the concept of hidden-token searchable public-key encryption, including the definitions and security models. Furthermore, we gave a concrete hidden-token searchable public-key encryption scheme together with the security proofs in the random oracle model. In the future, we would like to study the new scheme proven-secure in the standard model or with more expressive keyword search.

Acknowledgement

The authors would like to thank the anonymous reviewers for the valuable comments. Cong Zuo and Jun Shao are supported by NSFC No. 61472364, Yun Ling is supported by NSFC No. 61379121, and Guiyi Wei is supported by NSFC No. 61472365.

References

[1] M. T. Goodrich and M. Mitzenmacher, "Privacy-preserving access of outsourced data via oblivious ram simulation," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2011, pp. 576–587.

[2] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Privacy-preserving group data access via stateless oblivious ram simulation," in *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2012, pp. 157–167.

[3] C. Gentry and Z. Ramzan, "Single-database private information retrieval with constant communication rate," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2005, pp. 803–815.

[4] S. Yekhanin, "Private information retrieval," *Communications of the ACM*, vol. 53, no. 4, pp. 68–73, 2010.

[5] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2000, pp. 44–55.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *CCS06*, pp. 79–88, 2006.

[7] C. Zuo, J. Macindoe, S. Yang, R. Steinfeld, and J. K. Liu, "Trusted boolean search on cloud using searchable symmetric encryption," *TrustCom*, vol. 16, pp. 113–120.

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

[9] E.-J. Goh, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[10] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *International Conference on Applied Cryptography and Network Security*. Springer, 2005, pp. 442–455.

[11] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 965–976.

[12] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 353–373.

[13] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 875–888.

[14] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation." in *NDSS*, vol. 14. Citeseer, 2014, pp. 23–26.

[15] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 577–594.

[16] E. De Cristofaro, Y. Lu, and G. Tsudik, “Efficient techniques for privacy-preserving sharing of sensitive information,” in *International Conference on Trust and Trustworthy Computing*. Springer, 2011, pp. 239–253.

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

[18] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, “Trapdoor security in a searchable public-key encryption scheme with a designated tester,” *Journal of Systems and Software*, vol. 83, no. 5, pp. 763–771, 2010.

[19] B. Zhu, B. Zhu, and K. Ren, “Peksrand: Providing predicate privacy in public-key encryption with keyword search,” in *Communications (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[20] D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing,” in *Annual International Cryptology Conference*. Springer, 2001, pp. 213–229.