J. Parallel Distrib. Comput. 111 (2018) 152–161

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Efficient and secure searchable encryption protocol for cloud-based Internet of Things

Libing Wu [a], Biwen Chen [a], Kim-Kwang Raymond Choo [b,c], Debiao He [a,d,*]

[a] State Key Lab of Software Engineering, Computer School, Wuhan University, Wuhan, China
[b] Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249, USA
[c] School of Information Technology & Mathematical Sciences, University of South Australia, Adelaide, SA 5095, Australia
[d] Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Gulin, China

HIGHLIGHTS

- We define a searchable encryption model achieving efficiency and security for Cloud of Things.
- We propose an efficient and secure searchable encryption protocol for Cloud of Things.
- We show that our proposed protocol is provably secure. It satisfies the security requirements: inside KGA resilience forward privacy and file-injection attack resilience.
- Detailed performance analysis and experimental result are given.

ARTICLE INFO

Article history:
Received 29 March 2017
Received in revised form 28 June 2017
Accepted 17 August 2017
Available online 24 August 2017

Keywords:
Internet of Things
Cloud-of-Things
Searchable encryption
Forward privacy
File-injection attack resilience
Insider keyword guessing attack resilience

ABSTRACT

Internet of things (IoT) applications comprising thousands or millions of intelligent devices or things is fast becoming a norm in our inter-connected world, and the significant amount of data generated from IoT applications is often stored in the cloud. However, searching encrypted data (i.e. Searchable Encryption—SE) in the cloud remains an ongoing challenge. Existing SE protocols include searchable symmetric encryption (SSE) and public-key encryption with keyword search (PEKS). Limitations of SSE include complex and expensive key management and distribution, while PEKS suffer from inefficiency and are vulnerable to insider keyword guessing attacks (KGA). Besides, most protocols are insecure against file-injection attacks carried out by a malicious server. Thus, in this paper, we propose an efficient and secure searchable encryption protocol using the trapdoor permutation function (TPF). The protocol is designed for cloud-based IoT (also referred to as Cloud of Things – CoT) deployment, such as Cloud of Battlefield Things and Cloud of Military Things. Compared with other existing SE protocols, our proposed SE protocol incurs lower computation cost at the expense of a slightly higher storage cost (which is less of an issue, considering the decreasing costs of storage). We also prove that our protocol achieves inside KGA resilience, forward privacy, and file-injection attack resilience.

1. Introduction

In an Internet of Things (IoT) architecture, there are many real-world objects (also referred to as devices or things) connected to the Internet. These interconnected objects (e.g. sensors, mobile devices such as Android and iOS devices, wearable devices, and drones or unmanned aerial vehicles) are responsible for sensing, collecting, disseminating and exchanging data in a broad range of context, such as public/homeland security (e.g. smart cities), utility (e.g. smart grids), logistics (e.g. smart supply chains), and intelligent building (e.g. smart homes). The trend of IoT in our modern society is explained in a recent report from Gartner, which estimated that 63 million IoT devices will be attempting to connect to the network each second by 2020 [33].

Cloud computing can also play a supporting role in IoT architecture, as explained by Roopaei, Rad and Choo [28]. The authors used the Cloud of Things-based automated irrigation as a use case to illustrate how integrating cloud and IoT can make energy use more efficient and less costly. This is not surprising, considering that cloud computing can provide affordable and real-time computing and storage capabilities [30].

* Corresponding author at: State Key Lab of Software Engineering, Computer School, Wuhan University, Wuhan, China.
E-mail address: hedebiao@163.com (D. He).

http://dx.doi.org/10.1016/j.jpdc.2017.08.007
0743-7315/

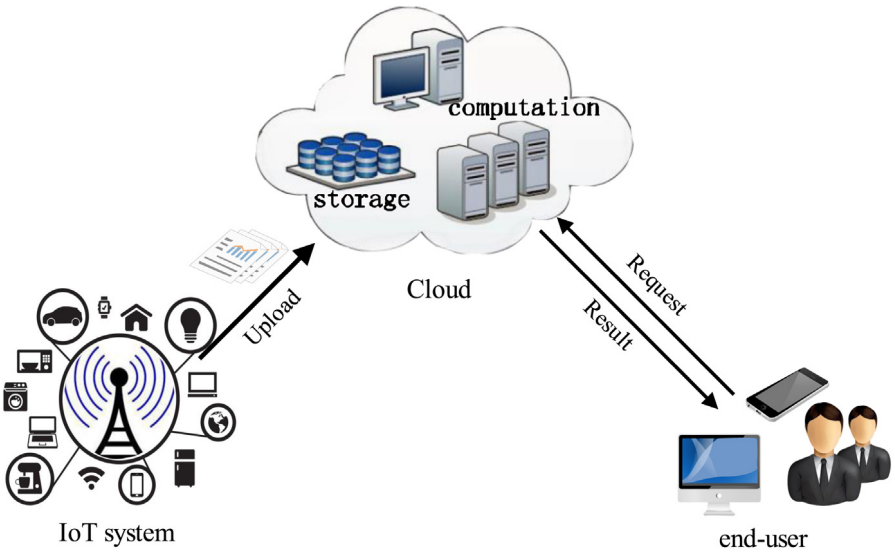

Fig. 1. A typical Cloud of Things (CoT) system.

A typical cloud-based IoT system is shown in Fig. 1. In a CoT environment, data collected by various smart devices is uploaded to the cloud server. Then, the user obtain information of interest from the cloud server via a client device. Since it is not realistic to expect that a cloud server is completely trustworthy (e.g. the server may be compromised and there may exists a corrupted or malicious insider) [14,18], data outsourced to or stored in the cloud should be protected (e.g. using a secure encryption scheme). However, searching on encrypted data is challenging given today's technology. Thus, Searchable Encryption (SE) has emerged as a salient research inquiry. A SE protocol is designed to allow one to search on encrypted data containing specified keywords and to obtain the response from the server based on the keyword trapdoor without the need to decrypt the data. The server is also prevented from learning the content of the user's query.

A number of SE protocols/schemes have been proposed in the literature since the work of Song et al. [29], and these SE protocols can be broadly categorized into searchable symmetric encryption (SSE) protocols (see [6,9,15,19,23,29,31]) and public-key encryption with keyword search (PEKS) protocols (see [1–3,5,12,16,17,24,34–36]). It is known that the SSE protocols generally are more efficient, but suffer from complex and expensive key management distribution [10] limitations (due to the fact that data owner needs to share a key with each user by the secure channel [4]). We refer the interested reader to a recent review of SSE schemes by Poh et al. [26].

PEKS protocols are known for their stronger security and flexibility, but a key limitation with such protocols is the inability to resist inside keyword guessing attacks [7] (e.g. from a malicious server or cloud employee). Specifically, the cloud server can use the user's public key to encrypt some keywords and use the keyword's ciphertext to test the content of a trapdoor. Recently in 2016, Chen et al. [13] proposed a PEKS protocol to resist such an attack, but their protocol is insecure against an external adversary.

We also observe that most SE protocols are vulnerable to file-injection attacks and have weak forward privacy. For example, Zhang et al. [37] demonstrated that in a file-injection attack, an adversary can recovery all of the user's trapdoors using very few injected files. Such an attack is clearly a threat to SE protocols, undermining the privacy of user data. In summary, if one want to deploy existing SE protocols in a CoT environment, the following limitations need to be addressed:

1. Security requirements: As evidenced by the findings reported in [13,37], there is a need to design protocols that are secure against inside keyword guessing attacks and file-injection attacks in addition to achieving other standard security properties.

2. Computational overheads: The amount of data generated by CoT devices in some applications may be significant (i.e. big data issues). Thus, there is a need to design efficient protocols with low computational overheads that are suitable for CoT deployment.

Therefore, in this paper, we construct an efficient and secure SE protocol using the trapdoor permutation function. The protocol uses neither bilinear pairing operation nor map-to-point hash operation. The search time of protocol is only related to the database update times. We then demonstrate the security and evaluate the performance of the proposed protocol.

We will briefly review related literature in the next section, before presenting the relevant background materials in Section 3. We present the proposed protocol in Section 4. In Sections 5 and 6, we analyze the security and evaluate the performance of our protocol, respectively. Finally, we conclude the paper in the last section.

2. Related literature

In this section, we present the related existing SE protocols and roughly categorize them according to their design goals, namely: efficiency, application and security.

Efficiency: The search time is one of the key factors in determining the efficiency of SE protocol. Song's [29] search time is linear to the database size, thus the protocol is inefficient in a big data environment. To mitigate such a limitation, Curtmola et al. [15] presented an index-based SSE construction to achieve sublinear search time. The complexity of the index-based protocol is related to the keyword space. In 2013, Cash et al. [9] presented a highly-scalable SSE, designed to support very large database. From existing literature, a practical SE protocol should minimize the search time, which is not a surprising observation.

However, in most PEKS protocol, complex operations (e.g. bilinear pairing, map-to-point hash) affect efficiency. To reduce the computational complexity, Di [16] proposed a PEKS protocol without bilinear pairing operation. However, the protocol is not practical. Recently, using homomorphic smooth projective hash functions, Chen [13] designed a protocol that does not require the bilinear pairing operation. Thus, their protocol is more efficient. In other words, the design of a SE protocol should avoid having complex operations.

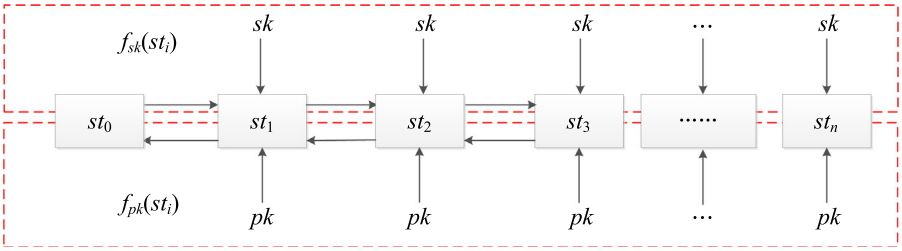

Fig. 2. Multi-Round TPF.

Application: SE protocol should adapt to a variety of application scenarios. Boneh [5] first designed a PEKS protocol for the email system based on public key cryptosystem. To facilitate data sharing, multi-owner SE protocols [32,36] were proposed. Such protocols generally allow multiple data owners to contribute data to multiple receivers. In practice, a typical scenario is for each data receiver to belong to a different access authority (e.g. different service providers). Thus, SE protocols should have flexible access control strategies [11]. Zheng [27] defined an attribute-based keyword searchable encryption protocol, which allows an authorized user to search over the outsourced data. Liang [24] also designed a protocol that offers searchable attribute-based functionality and flexible keyword update service. Thus, the design of SE protocol should consider the application environment.

Security: Security is the most important property in the design of any cryptographic protocols. For example, Goh [19] provided the security definition for SE protocols, and reiterated the importance of security in SE protocols. To prevent privacy leakage, Abdalla et al. [1] suggested that user search pattern and access pattern should be protected. Islam et al. [21] and Cash et al. [8] demonstrated that information leakage can be abused by a passive attacker to reveal the user's sensitive information. Recently, Zhang et al. [37] demonstrated that an adaptive attack can reveal the content of a past query by inserting as few as 10 new files in the dynamic database. Thus, SE protocol should be designed to prevent or reduce information leakage.

Meanwhile, most PEKS protocols are vulnerable to inside KGA [7,22], which is usually launched by a malicious cloud server. In 2013, Peng [35] presented the public-key encryption with Fuzzy keyword Search (PEFKS), designed to withstand inside KGA using two trapdoors. A number of such protocols have been designed for the cloud. However, the dynamic nature of the cloud environment may in itself be a risk. For example, due to the dynamic nature of database on the cloud, the updating operations of the database may result in (accidental) information leakage. This is an area of active research. For example, to protect the security of dynamic databases, a number of SE protocols with forward privacy have been presented in the literature [6,34].

3. Background

A summary of the notations used in this paper is presented in Table 1.

3.1. Preliminaries

Computational Diffie–Hellman (CDH): Given three elements $P, aP, bP \in \mathbb{G}_1$, where $a, b \in \mathbb{Z}_q^*$ are selected randomly. $Adv^{CDH}(\lambda)$ is defined as the advantage of an adversary A in computing abP, without knowing either a or b. We say that $Adv^{CDH}(\lambda) \leq nelg(\lambda)$ if the CDH assumption holds.

Trapdoor Permutation Function (TPF): A TPF f is a special one-way function $f : \mathcal{X} \to \mathcal{Y}$. There also exists a key generation algorithm $\mathbf{Gen}(1^\lambda) \to (pk, sk)$, where

$$f_{pk}(x) = y \in \mathcal{Y} \quad \text{and} \quad f_{sk}(y) = x \in \mathcal{X}.$$

Table 1
Summary of notations.

Notation	Description
λ	A security parameter
$\mathbb{G}_1$	An additive group of prime order q
P	A generator of $\mathbb{G}_1$
$negl(\lambda)$	A negligible probability
Ω	The system parameters
$(PK_{s(r)}, SK_{s(r)})$	The public/private keys of DS(DR).
W	The keyword set
TPF	A trapdoor permutation function
I_w	A set of indexes of files containing the keyword w
ind_i	The index of ith file
c	A counter that records the encryption times
CI_w	The ciphertext of index set I_w
UT_w	The storage location of CI_w
δ_w	The authorization token of keyword w
T_w	The trapdoor of keyword w
ED	The database of ciphertext
EI_w	The pair of (UT_w, CI_w)
SL	The state list of database
$\|S\|$	The size of set S
$\|\|$	Concatenation operator
$\oplus$	The bit-wise XOR operation
$X = \{a_1 a_2 ...\}_2$	The binary form of X

If a TPF f is security, then it should satisfy the following properties:

1. **Computability**: Given pk, $\forall x \in \mathcal{X}$, and $f_{pk}(x) \in \mathcal{Y}$ can be easily computed.
2. **One-Way**: Given $y = f_{pk}(x) \in \mathcal{Y}, f_{sk}(y) = x$ is computationally infeasible if the trapdoor sk is unknown.

For any attacker A, if f is secure, we say the advantage of the adversary in breaking the one-way property is negligible:

$$Adv_{f,A}^{OW}(\lambda) \leq negl(\lambda).$$

Besides, if a TPF f is executed multiple times, then the process can be described as Fig. 2.

Inside Keyword Guessing Attack (Inside KGA) [20] can be described as follows. In such an attack, a malicious cloud server [25] guesses each keyword, obtains the ciphertext of the guessed keyword, and tests the ciphertext with the given trapdoor. If the test succeeds, then the cloud server knows the keyword corresponds to the trapdoor.

File-Injection Attack (FIA) [37] can be described as follows. We assume for simplicity that the keyword set W is represented by set $\{0, 1, \ldots, |W|-1\}$ and the elements in the set $\{0, 1, \ldots, |W|-1\}$ are binary, i.e., the keyword w_0 can be represented by $\log|W|$ bits $\{00..0\}_2$. The attacker generates a set $\mathbf{F}$ of injected files, of size $\log|W|$. Every injected file contains exactly half the keywords of W, and the ith injected file contains those keywords, whose ith most-significant bit is equal to 1. Finally, according to the search results of trapdoor T_w, the attacker can deduce the keyword w that corresponds to T_w. Let $R = \{a_1 a_2 ...\}_2$ denote the search results on the injected files. The attacker sets $r_i = 1$, if and only if, the ith

	F_1	F_2	F_3
w_0	N	N	N
w_1	N	N	Y
w_2	N	Y	N
w_3	N	Y	Y
w_4	Y	N	N
w_5	Y	N	Y
w_6	Y	Y	N
w_7	Y	Y	Y

Y: The file contains the keyword
N: The file does not contain the keyword

Fig. 3. FIA example where $|W| = 8$.

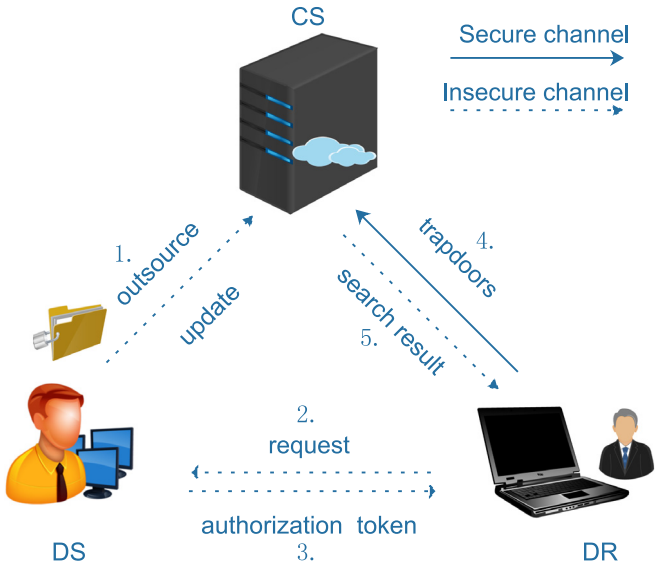

Fig. 5. System model.

injected file is returned. Then, the attacker determines the keyword $R = \{a_1a_2...a_{\log|W|}\}_2$ corresponds to the trapdoor T_w.

Fig. 3 depicts an example of FIA, where $|W| = 8$, and the keyword set W can be expressed as $W = (\{000\}_2, \{001\}_2, \{010\}_2, \{011\}_2, \{100\}_2, \{101\}_2, \{110\}_2, \{111\}_2)$. Meanwhile, the attacker needs to inject $\log 8 = 3$ files, and each injected file contains four keywords. If two files $\{F_1, F_2\}$ are returned in response to the token T_w, then we know that the keyword w is the keyword $\{110\}_2$ (i.e. w_6).

Database's State List: The database's State List (SL) consists of three tuple $\langle w_i, c_i, st_{c_i w_i}\rangle$, where $w_i \in W$ is an index of SL, c_i denotes a counter, and $st_{c_i w_i}$ denotes a string corresponding to current counter c_i. When the data containing w_i is inserted into the database during the updating phase, SL will be automatically updated (i.e. adding c_i with 1 and replacing $st_{c_i w_i}$ with $st_{(c_i+1)w_i}$). Furthermore, the updated process can be executed a number of times.

As shown in Fig. 4, if the files containing keywords $w_1, w_3, w_{|W|}$ are inserted into the database, then SL's initial status will be the updated status.

3.2. System model

The system model of our protocol is described in Fig. 5, which comprises three entities, namely: data sender (DS), data receiver (DR) and cloud server (CS). DS outsources local data to CS, and any authorized DR can query the encrypted data by sending a trapdoor. Each entity has different responsibilities, as described below:

- *DS* is responsible for initializing the system, creating keyword–indexes of files, and encrypting and uploading data.
- *DR* searches data by sending the authorization trapdoor.
- *CS* is responsible for storing the data from DS and responding to query from DR.

In the system model, DO can continuously update the database. Thus, our protocol is more suitable for the CoT environment. DR can search the related keywords which are authorized by DO. The request/authorization method provides a flexible access control policy. Thus, our system model is efficient.

3.3. Formal definition

Our protocol consists of the following algorithms:

- *Setup*(λ): Takes as input security parameter λ, and outputs system global parameter Ω.

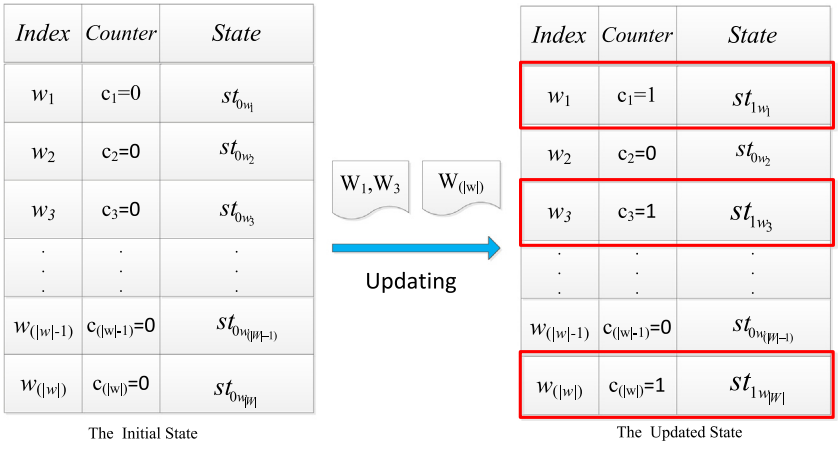

Fig. 4. Database's state list.

- *KeyGen*(Ω): Takes as input system parameter Ω, and outputs public/private key pairs (PK_s, SK_s), (PK_r, SK_r) for DS and DR, respectively.
- *Enc*(PK_r, SK_s, I_{w_i}, w_i): Takes as input DR's public key PK_r, DS's private key SK_s and an index set I_{w_i} of w_i, and outputs ciphertext EI_{w_i} of the index set I_{w_i}.
- *AuthToken*(PK_r, w_i): Takes as input DR's public key PK_r and the authorized keyword w_i, and outputs an authorization token δ_{w_i} of w_i.
- *Trapdoor*(PK_s, SK_r, δ_{w_i}, w_i): Takes as input DS's public key PK_s, DR's private key SK_r and authorization token δ_{w_i} of keyword w_i, and outputs the trapdoor T_{w_i}.
- *Search*(ED, T_{w_i}): Takes as input encrypted database ED and a trapdoor T_{w_i}, and outputs ciphertext EI_{w_i} of index set I_{w_i}.

Security Requirements For a typical deployment (e.g. in CoT or IoT), a SE protocol should satisfy the following security requirements:

- Inside KGA Resilience: A malicious cloud server is not able to deduce keyword corresponding to the user's trapdoor using inside KGA.
- Forward privacy: An attacker is not able to search any newly updated data using previous trapdoors.
- FIA Resilience: An attacker is not able to recover keyword corresponding to the user's trapdoor using FIA.

4. Proposed protocol

In this section, we present our proposed SE protocol, comprising the following algorithms:

Setup(λ) takes as input the security parameter λ, and performs the following:

1. DS chooses an additive group $\mathbb{G}_1$ with a large prime order q, and a generator P.
2. DS selects a TPF $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are two sets of strings with length l. Then, DS executes **Gen**(1^λ) to generate f's public/private key pair (pk, sk).
3. DS selects three resistant-collision hash functions: $h_1 : \{0,1\}^l \to \mathbb{Z}_q$, $h_2 : \mathbb{Z}_q^* \times \{0,1\}^l \to \{0,1\}^*$ and $h_3 : \mathbb{G}_1 \times \mathbb{G}_1 \to \{0,1\}^{(l+\log^q)}$.
4. DS constructs a SL **List**. Then, DS initializes **List**$[w_i].c = 0$ and **List**$[w_i].st_c = st_{0w_i}$, where st_{0w_i} is a random string of length l and $w_i = w_1, \ldots, w_{|W|}$.
5. DS sets the system parameter $\Omega = \{pp, sp\}$. Then, DS publishes parameter $pp = (q, P, \mathbb{G}_1, h_1, h_2, h_3, f, pk)$ and keeps $sp = (sk,$ **List**$)$ secretly.

KeyGen(Ω) takes as input the system parameter Ω, and performs the following:

1. randomly chooses $SK_s = s$, $SK_r = \alpha$ as the respective private keys of DS and DR.
2. computes $PK_s = s \cdot P \in \mathbb{G}_1$, $PK_r = \alpha \cdot P \in \mathbb{G}_1$ as the respective public keys of DS and DR.

Enc(PK_r, SK_s, I_{w_i}, w_i) takes as input the parameters $PK_r = \alpha \cdot P$, $SK_s = s$ and an index set $I_{w_i} = \{ind_1, ind_2, \ldots, ind_n\}$ of w_i, and performs the following:

1. DS computes $K = s \cdot PK_r \in \mathbb{G}_1$.
2. If $n = 1$, then DS selects a random index ind^*, and sets $I^*_{w_i} = \{ind_1, ind_2\}$ and $n = 2$, where $ind_2 = ind^*$. Otherwise, sets $I^*_{w_i} = \{ind_1, ind_2, \ldots, ind_n\}$.
3. DS encrypts $I^*_{w_i}$ as follows:
   - searches **List** to obtain **List**$[w_i]$ and sets $j =$ **List**$[w_i].c$ and $st_{jw_i} =$ **List**$[w_i].st_c$.
   - for each index ind_m, DS computes $st_{(j+m)w_i} = f_{sk}(st_{(j+m-1)w_i})$, $UT_{(j+m)w_i} = h_1(st_{(j+m)w_i})$ and $CI_{(j+m)w_i} = h_2(K, st_{(j+m)w_i}) \oplus ind_m$, where $m = 1, 2, \ldots, n$.
   - updates **List** to make **List**$[w_i].c = (j + n)$ and **List**$[w_i].st_c = st_{(j+n)w_i}$
4. DS sends $EI_{w_i} = (UT_{(j+m)w_i}, CI_{(j+m)w_i})$ to CS, where $m = 1, 2, \ldots, n$.
5. CS stores EI_{w_i} like this $ED[UT_{(j+m)w}] = CI_{(j+m)w_i}$, where $m = 1, 2, \ldots, n$.

AuthToken(PK_r, w_i) takes as input the parameters (PK_r, w_i), and performs the following:

1. DS selects a number $k \in \mathbb{Z}_q^*$ randomly, and computes $AU_{1w_i} = k \cdot P$ and $A_{w_i} = k \cdot PK_r$.
2. DS computes $AU_{2w_i} = h_3(A_{w_i}, AU_{1w_i}) \oplus$ (**List**$[w_i].c \parallel$ **List**$[w_i].st_c$).
3. DS sends the authorization token $\delta_{w_i} = (AU_{1w_i}, AU_{2w_i})$ to DR.

Trapdoor(PK_s, SK_r, δ_{w_i}, w_i) takes as input the parameters $PK_s = sP$, $SK_r = \alpha$ and the authorization token $\delta_{w_i} = (AU_{1w_i}, AU_{2w_i})$ of keyword w_i, and performs the following:

1. DR computes $A^*_{w_i} = \alpha \cdot AU_{1w_i}$.
2. DR computes (**List**$^*[w_i].c \parallel$ **List**$^*[w_i].st_c) = h_3(A^*_{w_i}, AU_{1w_i}) \oplus AU_{2w_i}$, and uses **List**$^*[w_i]$ as the trapdoor of keyword w_i.
3. DR keeps the trapdoor $T_{w_i} = (c, st_{cw_i})$, where $c =$ **List**$^*[w_i].c$, $st_{cw_i} =$ **List**$^*[w_i].st_c$.

Search(ED, T_{w_i}) takes as input the ciphertext database ED, and the trapdoor $T_{w_i} = (c, st_{cw_i})$ of keyword w_i sent by DR through a secure channel. CS performs the following:

1. for each j, CS computes $st_{(j-1)w_i} = f_{pk}(st_{jw_i})$, $UT_{jw_i} = h_1(st_{jw_i})$ and obtains encrypted index $CI_{jw_i} = ED[UT_{jw_i}]$, where $j = c, c-1, c-2, \ldots, 1$.
2. CS returns $EI_{w_i} = \{CI_{1w_i}, CI_{2w_i}, \ldots, CI_{cw_i}\}$ to DR.

5. Security analysis

We now analyze the security of the proposed protocol presented in Section 4.

5.1. Soundness

Definition 1. *(Soundness)*: For every sufficiently large security parameter λ, for $\forall w_0 \neq w_1 \in W$, there is a negligible function $negl(\lambda)$ such that Fig. 6 holds.

The soundness of the protocol is demonstrated when search indexes containing the keyword w_1 will result in the return of an encrypted index EI_{w_1} of w_1, when it is authorized to search. Otherwise, no information will be returned.

Theorem 1. *The construction of proposed protocol presented in Section 4 is sound, in the sense of Definition 1.*

Proof. Assume ind_0, ind_1 are two indexes of files which contain w_0, w_1, respectively. Assume pk, sk are TPF f's public and private keys. Let **List**$[w_0].c = 0$, **List**$[w_0].st_c = st_{0w_0}$, **List**$[w_1].c = 0$, **List**$[w_1].st_c = st_{0w_1}$. In addition, we assume that ($SK_s = s$, $SK_r = \alpha$) ($PK_s = s \cdot P$, $PK_r = \alpha \cdot P$) are the public and private keys of DS and DR, respectively. Then, we have $K = sPK_r$, $st_{1w_0} = f_{sk}(st_{0w_0})$, $st_{1w_1} = f_{sk}(st_{0w_1})$, $EI_{w_0} = (UT_{1w_0}, CI_{1w_0})$ and $EI_{w_1} = (UT_{1w_1}, CI_{1w_1})$, where $UT_{1w_0} = h_1(st_{1w_0})$, $CI_{1w_0} = h_2(K, st_{1w_0}) \oplus$

$$\Pr\left[\begin{array}{l} Setup(\lambda)\rightarrow\Omega;\\ KeyGen(\lambda)\rightarrow(PK_s,SK_s,PK_r,SK_r);\\ \mathrm{Enc}(PK_r,SK_s,ind_{w_0},w_0)\rightarrow(EI_{w_0},List[w_0]),\\ Enc(PK_r,SK_s,ind_{w_1},w_1)\rightarrow(EI_{w_1},List[w_1]);\\ (EI_{w_0},EI_{w_1})\rightarrow ED;\\ AuthToken(PK_r,w_1)\rightarrow\delta_{w_1};\\ Trapdoor(PK_s,SK_r,\delta_{w_1},w_1)\rightarrow T_{w_1};\\ R(\Omega,PK_s,SK_r,PK_r)\longrightarrow T_{w_0};\\ Search(ED,T_{w_1})=EI_{w_1}\text{ and }Search(ED,T_{w_0})=NULL \end{array}\right] = 1-negl(\lambda)$$

Fig. 6. Definition of soundness.

ind_0 and $UT_{1w_1} = h_1(st_{1w_1}), CI_{1w_1} = h_2(K, st_{1w_1}) \oplus ind_1$, respectively. We update **List**$[w_0]$ and **List**$[w_1]$, where **List**$[w_0].c = 1$, **List**$[w_0].st_c = st_{1w_0}$, **List**$[w_1].c = 1$ and **List**$[w_1].st_c = st_{1w_1}$.

An user authorized to search w_1 will be given the authorization token $\delta_{w_1} = (AU_{1w_1}, AU_{2w_1})$, and the latter is generated as follows:

- selects a random number $k \in \mathbb{Z}_q^*$, and computes $AU_{1w_1} = k \cdot P$.
- computes $AU_{2w_1} = h_3(A_{w_1}, AU_{1w_1}) \oplus$ (**List**$[w_1].c \parallel$ **List**$[w_1].st_c$), where $A_{w_1} = k \cdot PK_r$.

The user generates the trapdoor T_{w_1} of w_1 using the authorization token δ_{w_1}. The process is as follows:

- computes $A_{w_1} = \alpha \cdot AU_{1w_1}$.
- computes (**List**$[w_1].c \parallel$ **List**$[w_1].st_c) = h_3(A_{w_1}, AU_{1w_1}) \oplus AU_{2w_1}$.

Then, the user obtains the trapdoor $T_{w_1} = (c_1, st_{c_1w_1})$ of w_1, where $c_1 =$ **List**$[w_1].c$ and $st_{c_1w_1} =$ **List**$[w_1].st_c$. Since the user has not been authorized to search w_0, the trapdoor T_{w_0} of w_0 cannot be computed. We randomly select $(c_0, st_{c_0w_0})$ as the trapdoor T_{w_0} of w_0.

As $st_{c_1w_1} =$ **List**$[w_1].st_c = st_{1w_1}$, the equalities $UT_{1w_1} = h_1(st_{c_1w_1})$ and $CI_{1w_1} = h_2(K, st_{c_1w_1}) \oplus ind_1$ hold with probability 1. However, since $st_{c_0w_0} \neq st_{1w_0}$ and h_1 is a collision-resistant hash function, we have $UT_{1w_0} \neq h_1(st_{c_0w_0})$ and $CI_{1w_0} \neq h_2(K, st_{c_0w_0}) \oplus ind_0$ with probability 1-$negl(\lambda)$.

As shown in Fig. 6, if a user is authorized to search w_1 but is not authorized to search w_0, then $Search(ED, T_{w_1}) = EI_{w_1} = (UT_{1w_1}, CI_{1w_1})$ and $Search(ED, T_{w_0}) = NULL$ will hold with probability 1-$negl(\lambda)$.

5.2. Data privacy

To ensure data privacy, we define the data privacy security of the protocol using a challenge–response game. The attacker A attempts to distinguish between the ciphertexts of two different indexes. The protocol should ensure that $Enc(PK_r, SK_s, I_{w_i}, w_i)$ does not reveal any information of I_{w_i} unless T_{w_i} is available.

Data privacy game

Setup: The challenger C runs $Setup(\lambda)$, $KeyGen(\lambda)$ to generate system parameter Ω and public/private key pairs (PK_r, SK_r), (PK_s, SK_s) of DR and DS, respectively. Then, C sends (PK_r, PK_s) to A.

Phase 1: A makes the following queries.

1. **Enc Query** (I_{w_i}, w_i): A adaptively selects the index–keyword pair (I_{w_i}, w_i) for the *Enc* query. C responds with $EI_{w_i} = Enc(PK_r, SK_s, I_{w_i}, w_i)$.
2. **Authorization Token Query** (w_i): A adaptively selects the keyword w_i for the authorization token query. C responds with $\delta_{w_i} = AuthToken(PK_r, w_i)$.
3. **Trapdoor Query**(δ_{w_i}, w_i): A adaptively queries the trapdoor T_{w_i} for the keyword $w_i \in W$. C responds with $T_{w_i} = Trapdoor(PK_s, SK_r, \delta_{w_i}, w_i)$.

Challenge: A sends two challenged index–keyword pairs $(I_{w_0}, w_0), (I_{w_1}, w_1)$. C picks a random number $b \in \{0, 1\}$ and sends ciphertext $EI_{w_b} =$ **Enc**$(PK_r, SK_s, I_{w_b}, w_b)$ to A.

Phase 2: A issues queries similar to those in Phase 1 adaptively. The only restriction is that (w_0, w_1) cannot be selected for the authorization token query and trapdoor query.

Guess: A outputs $b' \in \{0, 1\}$ and wins the data privacy game if $b' = b$.

Definition 2. For any polynomial-time adversary A and all sufficiently large λ, if the probability winning the data privacy game satisfies

$$Pro_A^{dp}(\lambda) \leq \frac{1}{2} + negl(\lambda),$$

then we say the protocol provides data privacy.

Theorem 2. *If a TPF f is secure in the random oracle model, then our proposed protocol satisfies data privacy property, in the sense of Definition 2.*

Proof. Suppose A makes at most $q_{h_1}, q_{h_2}, q_{h_3}$ hash queries to h_1, h_2, h_3. To simplify the proof process, we assume $I_{w_i} = ind_1, \ldots, ind_n$, where $n > 1$. Suppose pk, sk are the public/private keys of TPF f. Suppose **List**$[w_i]$ is initialized as **List**$[w_i].c = 0$ and **List**$[w_i].st_c = st_{0w_i}$, where $w_i = \{w_1, \ldots, w_{|W|}\}$.

First, C selects the system parameter $\Omega = (pp, sp)$, where $pp = (q, P, \mathbb{G}_1, h_1, h_2, h_3, f, pk)$ and $sp = (sk,$ **List**$)$. Then, C generates two public/private key pairs $(PK_s = sP, SK_s = s)$, $(PK_r = \alpha P, SK_r = \alpha)$ of DS and DR, respectively. Finally, C sends the parameter (pp, PK_s, PK_r) to A.

To store the results of hash queries of (h_1, h_2, h_3), C constructs the respective lists: $list_{h_1}, list_{h_2}, list_{h_3}$. The process is as follows:

h_1-*query*: Upon receiving a string $st_i \in \{0, 1\}^*$, if there exists (st_i, u_i) in $list_{h_1}$, then C returns u_i. Otherwise, C picks a new random value $u_i \in \mathbb{Z}_q^*$ for each new st_i and sets $h_1(st_i) = u_i$. Besides, C inserts the pair (st_i, u_i) into the $list_{h_1}$. The $list_{h_1}$ is initially empty.

h_2-*query*: Upon receiving a pair (K_i, v_i), where $K_i \in \mathbb{G}_1$ and $v_i \in \{0, 1\}^*$, if there exists (K_i, v_i, a_i) in $list_{h_2}$, C returns a_i. Otherwise, C picks a new random value $a_i \in \{0, 1\}^*$ for each new (K_i, v_i) and sets $h_2(K_i, v_i) = a_i$. Besides, C inserts the tuple (K_i, v_i, a_i) into the $list_{h_2}$. The $list_{h_2}$ is initially empty.

h_3-*query*: Upon receiving a pair (U_i, V_i), where $U_i, V_i \in \mathbb{G}_1$, if there exists (U_i, V_i, z_i) in $list_{h_3}$, C returns z_i. Otherwise, C picks a new random value $z_i \in \{0, 1\}^*$ for each new (U_i, V_i) and sets $h_3(U_i, V_i) = z_i$. Besides, C inserts the tuple (U_i, V_i, z_i) into the $list_{h_3}$. The $list_{h_3}$ is initially empty

*Phase*1: A adaptively issues the following queries.

Enc query (I_{w_i}, w_i): Upon receiving (I_{w_i}, w_i), where $I_{w_i} = \{ind_1, \ldots, ind_n\}$, C does as follows:

- searches the state list **List** to obtain $j =$ **List**$[w_i].c$ and $st_{jw_i} =$ **List**$[w_i].st_c$.
- for each index ind_m,where $m = 1, 2, \ldots, n$, C computes $st_{(j+m)w_i} = f_{sk}(st_{(j+m-1)w_i})$, $UT_{(j+m)w_i} = u_{(j+m)w_i}$ and $CI_{(j+m)w_i} = a_{(j+m)w_i} \oplus ind_m$, where $u_{(j+m)w_i} = h_1(st_{(j+m)w_i})$ and $a_{(j+m)w_i} = h_2(K, st_{(j+m)w_i})$ are obtained by making h_1, h_2 queries, respectively.

- updates **List** to make $\mathbf{List}[w_i].c = (j+n)$ and $\mathbf{List}[w_i].st_c = st_{(j+n)w_i}$.
- returns $EI_{w_i} = (UT_{(j+m)w_i}, CI_{(j+m)w_i})$ to A, where $m = 1, 2, \ldots, n$.

AuthToken query (PK_r, w_i): Upon receiving w_i, C performs the following:

- selects a random number $k \in \mathbb{Z}_q^*$, then computes $AU_{1w_i} = k \cdot P$ and $A_{w_i} = k \cdot PK_r$.
- makes a h_3 query to obtain $h_3(A_{w_i}, AU_{1w_i}) = z_i$ and computes $AU_{2w_i} = z_i \oplus (\mathbf{List}[w_i].c \parallel \mathbf{List}[w_i].st_c)$.
- returns the authorization token $\delta_{w_i} = (AU_{1w_i}, AU_{2w_i})$.

Trapdoor query (δ_{w_i}, w_i): Upon receiving (δ_{w_i}, w_i), C directly returns $\mathbf{List}[w_i]$ as the trapdoor $T_{w_i} = (c, st_{cw_i})$ of w_i, where $c = \mathbf{List}[w_i].c$ and $st_{cw_i} = \mathbf{List}[w_i].st_c$.

Challenge: If A decides to terminate at the end of Phase 1, then it outputs two challenged index–keyword pairs $(I_{w_0}, w_0), (I_{w_1}, w_1)$, where $|I_{w_0}| = |I_{w_1}|$. C performs the following:

- picks a random value $b \in \{0, 1\}$ and searches **List** to obtain $j = \mathbf{List}[w_b].c$ and $st_{jw_b} = \mathbf{List}[w_b].st_c$.
- for each ind_m, where $m = 1, 2, \ldots, |I_{w_b}|$, C selects a random string $st^*_{(j+m)w_b}$ instead of computing $st_{(j+m)w_b} = f_{sk}(st_{(j+m-1)w_b})$. Then, C makes the h_1, h_2 queries to obtain $u_{(j+m)w_b} = h_1(st^*_{(j+m)w_b})$ and $a_{(j+m)w_b} = h_2(K, st_{(j+m)w_b})$ before computing $CI_{(j+m)w_b} = a_{(j+m)w_b} \oplus ind_m$.
- sets $\mathbf{List}[w_b].c = (j + |I_{w_b}|)$ and $\mathbf{List}[w_b].st_c = st^*_{(j+|I_{w_b}|)w_b}$
- returns $EI_{w_b} = (u_{(j+m)w_b}, CI_{(j+m)w_b})$ to A, where $m = 1, 2, \ldots, |I_{w_b}|$.

Phase2: A issues queries similar to those in Phase 1. The only restriction is that (w_0, w_1) cannot be selected for the authorization token query and trapdoor query.

Guess. Finally, A outputs its guess $b' \in \{0, 1\}$. If $b' = b$, then C outputs 1; otherwise, C outputs 0.

According to the security definition of TPF, A cannot distinguish between $f_{sk}(st_{(j+m)w_b})$ and a random string $st^*_{(j+m)w_b}$ without having the private key sk of TPF f (i.e., one-way property). Therefore, EI_{w_b} is legitimate in the attacker's view. Since EI_{w_b} is obtained by random oracle h_1, h_2, A's output b' will satisfy $b' = b$ with probability at most $\frac{1}{2}$. In other words, if A can obtain any $st_{(j+m)w_b}$ with probability $Pro_f^{OW}(\lambda)$, then the probability that A can win the data privacy game is $Pro_A^{dp} = \frac{1}{2} + Pro_f^{OW}(\lambda)$. However, since the TPF f is secure, $Pro_f^{OW}(\lambda) \leq negl(\lambda)$ and $Pro_A^{dp} \leq \frac{1}{2} + negl(\lambda)$. Thus, our protocol achieves data privacy.

5.3. Authorization token privacy

To preserve the privacy of DR's query, the authorization token should not leak keyword information. In the authorization token privacy game, the attacker A plays a challenge–response game with the challenger C and attempts to distinguish an authorization token of designated keyword from some other authorization tokens. If A wins the game, then A has obtained some useful information from the authorization token set.

Authorization Token privacy game

Setup: The challenger C runs $Setup(\lambda)$, $KeyGen(\lambda)$ to generate system parameter Ω and public/private key pairs (PK_r, SK_r), (PK_s, SK_s) of DR and DS, respectively. Then, C sends (PK_r, PK_s) to A.

Phase 1: A makes the following queries.

1. *Authorization Token Query*(w_i): A adaptively selects the keyword w_i for authorization token query. C responds with $\delta_{w_i} = AuthToken(PK_r, w_i)$.
2. *Trapdoor Query*(δ_{w_i}, w_i): A adaptively selects the keyword w_i for trapdoor query. C responds with $T_{w_i} = Trapdoor(PK_s, SK_r, \delta_{w_i}, w_i)$.

Challenge: A adaptively outputs a challenge authorization token pair (w_0, w_1). C picks a random $\gamma \in \{0, 1\}$ and returns $\delta_{w_\gamma} = AuthToken(PK_r, w_\gamma)$

Phase 2: A can choose to continue making the above queries, and the only restriction is that (w_0, w_1) cannot make the trapdoor queries.

Guess: A outputs $\gamma' \in \{0, 1\}$ and wins the authorization token privacy game if $\gamma' = \gamma$.

Definition 3. For any polynomial-time adversary and all sufficiently large λ, if the probability winning the authorization token privacy game satisfies

$$Pro_A^{atp}(\lambda) \leq \frac{1}{2} + negl(\lambda),$$

then we say the protocol achieves authorization token privacy.

Theorem 3. *If the CDH assumption holds in the random model, then the proposed protocol achieves authorization token privacy, in the sense of Definition 3.*

Proof. Suppose A makes at most $q_{h_1}, q_{h_2}, q_{h_3}$ hash queries to h_1, h_2, h_3, and A solves the CDH assumption with probability Pro_A^{CDH}. To simplify the proof process, we assume $I_{w_i} = ind_1, \ldots, ind_n$, where $n > 1$. Suppose pk, sk are the public/private keys of TPF f, $\mathbf{List}[w_i]$ is initialized as $\mathbf{List}[w_i].c = 0$, and $\mathbf{List}[w_i].st_c = st_{0w_i}$, where $w_i = \{w_1, \ldots, w_{|W|}\}$.

There exists a simulator S given $P, a \cdot P, b \cdot P$ wishes to output $ab \cdot P$ (i.e. S solves the CDH assumption). S simulates the challenger and interacts with A as follows:

Let pk, sk be the public/private key of a secure TPF f. Then, S selects the system parameter $\Omega = (pp, sp)$, where $pp = (q, P, \mathbb{G}_1, h_1, h_2, h_3, f, pk)$ and $sp = (sk, \mathbf{List})$. S generates the public/private key pair $(PK_s = s \cdot P, SK_s = s)$ of DS, and sets DR's public $PK_r = a \cdot P$. Then, S sends (pp, PK_s, PK_r) to A.

To store the results of hash queries of (h_1, h_2, h_3), respectively, C constructs $list_{h_1}, list_{h_2}, list_{h_3}$. The process is as follows:

h_1-query: Upon receiving a string $st_i \in \{0, 1\}^*$, if there exists (st_i, u_i) in $list_{h_1}$, then C returns u_i. Otherwise, C picks a new random value $u_i \in \mathbb{Z}_q^*$ for each new st_i and sets $h_1(st_i) = u_i$. In addition, S inserts (st_i, u_i) into $list_{h_1}$, which is initially empty.

h_2-query: Upon receiving a pair (K_i, v_i), where $K_i \in \mathbb{G}_1$ and $v_i \in \{0, 1\}^*$, if there exists (K_i, v_i, a_i) in $list_{h_2}$, then C returns a_i. Otherwise, C picks a new random value $a_i \in \{0, 1\}^*$ for each new (K_i, v_i) and sets $h_2(K_i, v_i) = a_i$. In addition, C inserts the tuple (K_i, v_i, a_i) into the (initially empty) $list_{h_2}$.

h_3-query: Upon receiving a pair (U_i, V_i), where $U_i, V_i \in \mathbb{G}_1$, if there exists (U_i, V_i, z_i) in $list_{h_3}$, then C returns z_i. Otherwise, C picks a new random value $z_i \in \{0, 1\}^*$ for each new (U_i, V_i) and sets $h_3(U_i, V_i) = z_i$. In addition, C inserts the tuple (U_i, V_i, z_i) into the $list_{h_3}$. The $list_{h_3}$ is initially empty.

*Phase*1: A adaptively issues the following queries.

Authorization Token query(PK_r, w_i): Upon receiving w_i, C performs the following:

- selects a number $k \in \mathbb{Z}_q^*$ randomly, and computes $AU_{1w_i} = kP$ and $A_{w_i} = kPK_r$.
- makes a h_3 query to obtain $h_3(A_{w_i}, AU_{1w_i}) = z_i$, and computes $AU_{2w_i} = z_i \oplus (\mathbf{List}[w_i].c \parallel \mathbf{List}[w_i].st_c)$
- returns the authorization token $\delta_{w_i} = (AU_{1w_i}, AU_{2w_i})$ of w_i.

Trapdoor query (δ_{w_i}, w_i): Upon receiving (δ_{w_i}, w_i), C returns $\mathbf{List}[w_i]$ as the trapdoor $T_{w_i} = (c, st_{cw_i})$ of w_i, where $c = \mathbf{List}[w_i].c$ and $st_{cw_i} = \mathbf{List}[w_i].st_c$.

Table 2
Computation cost and security properties: A comparative summary.

Protocol	Computation			Security		
	Enc phase	Authorization & Trapdoor phase	Search phase	FP	FIA Resilience	Inside KGA
Boneh et al. [5]	$1T_p + 1T_{mtp} \approx 15.068$ ms	$1T_{mtp} + 1T_{sm} \approx 8.712$ ms	$1T_p + 1T_{mtp} \approx 15.068$ ms	×	×	×
Wang et al. [34]	$4T_{sm} + 1T_p \approx 11.286$ ms	$5T_{sm} + 1T_p \approx 12.272$ ms	$4T_p + 1T_{mtp} + 1T_{sm} \approx 38.08$ ms	√	×	×
Chen et al. [13]	$4T_{exp} + 1T_{mtp} + 2T_{mul} \approx 9.664$ ms	$4T_{exp} + 1T_{mtp} + 2T_{mul} \approx 9.664$ ms	$7T_{exp} + 3T_{mul} \approx 3.391$ ms	×	×	√
Bost [6]	$1T_e + 3T_h \approx 0.194$ ms	$1T_h \approx 0.005$ ms	$1T_d + 2T_h \approx 5.901$ ms	√	×	√
Our Protocol	$1T_e + 2T_h \approx 0.199$ ms	$3T_{sm} + 1T_h \approx 2.963$ ms	$1T_d \approx 5.896$ ms	√	√	√

Challenge. If A decides to terminate at the end of Phase 1, then it outputs a challenged keyword pair (w_0, w_1). C picks a random $\gamma \in \{0, 1\}$ and performs the following:

- sets $AU_{1w_\gamma} = bP$.
- selects a random string η of length $l + \log^q$, and computes $AU_{2w_\gamma} = \eta \oplus (\mathbf{List}[w_\gamma].c \parallel \mathbf{List}[w_\gamma].st_c)$.
- returns the authorization token $\delta_{w_\gamma} = (AU_{1w_\gamma}, AU_{2w_\gamma})$.

Note that this challenge implicitly defines $AU_{2w_\gamma} = h_2(A_{w_\gamma}, AU_{1w_\gamma}) \oplus (\mathbf{List}[w_\gamma].c \parallel \mathbf{List}[w_\gamma].st_c)$, where $A_{w_\gamma} = abP$. With this definition, δ_{w_γ} is valid in A' view.

Phase 2: A issues queries similar to those in Phase 1. The only restriction is that (w_0, w_1) cannot be selected for the trapdoor query.

Guess: Finally, A outputs its guess $\gamma' \in \{0, 1\}$ indicating whether the challenge δ_{w_γ} is the result of $AuthToken(PK_r, w_0)$ or $AuthToken(PK_r, w_1)$. In the real attack, if $h_2(A_{w_\gamma}, AU_{1w_\gamma})$ is not requested by A, then δ_γ is an authorization token of w_0 or w_1 is independent of A' view. Thus, A's output γ' will satisfy $\gamma' = \gamma$ with probability at most $\frac{1}{2}$. In other words, if A issues a query $h_2(A_{w_\gamma}, AU_{1w_\gamma})$, this implies A can solve the CDH problem, $A_{w_\gamma} = abP$. Thus, the probability winning the authorization token privacy game is $Pro_A^{atp} = \frac{1}{2} + Pro_A^{CDH}$. However, if the CDH assumption holds, then $Pro_A^{CDH} \leq negl(\lambda)$ and $Pro_A^{atp} \leq \frac{1}{2} + negl(\lambda)$. Therefore, our protocol achieves authorization token privacy. A are not able to obtain any information from the authorization token.

5.4. Security requirement analysis

Inside KGA Resilience: According to the description of our proposed protocol in Section 4, the encryption process requires as input the parameters PK_r, SK_s and the index set I_{w_i}, where the secret key SK_s of DO is used to generate the ciphertext of the set I_{w_i}. Thus, a malicious cloud server is not able to generate ciphertexts of the keywords and test the given trapdoor without the secret key SK_s. SK_s is key to achieving inside KGA resilience. Thus, we say our protocol is insider KGA resilience.

Forward Privacy: Let us assume a new index $ind^*_{w_i}$ of file containing keyword w_i will be inserted into the cloud server. Let $\mathbf{List}[w_i].c = j$ and $\mathbf{List}[w_i].st_c = st_{jw_i}$. According to the description of our protocol, DS first computes $st_{(j+1)w_i} = f_{sk}(st_{jw_i})$. Then, DS computes $EI_{w_i} = (UT_{(j+1)w_i}, CI_{(j+1)w_i})$, where $UT_{(j+1)w_i} = h_1(st_{(j+1)w_i})$ and $CI_{(j+1)w_i} = h_2(K, st_{(j+1)w_i}) \oplus ind^*_{w_i}$. Finally, CS updates the encryption database $ED[UT_{(j+1)w_i}] = CI_{(j+1)w_i}$. If the user searches the updated database ED using the previous $T_{w_i} = (j, st_{jw_i})$, then CS will compute $UT_{(j-m)w_i} = h_3(st_{(j-m)w_i})$ and $st_{(j-m-1)w_i} = f_{pk}(st_{(j-m)w_i})$ for each $m = 0, 1, \ldots, (j-1)$. CS returns the search result $(UT_{(j-m)w_i}, ED[UT_{(j-m)w_i}])$, where $m = 0, 1, \ldots, (j-1)$. According to the returned result, we know the new index $ind^*_{w_i}$ is not returned. Thus, we say our protocol achieves forward privacy.

FIA Resilience: To prevent the adversary from recovering the content of query based on the indexes of the returned result, our protocol randomly stores the newly inserted file and encrypts the index of inserted file by $CI_{(j+m)w_i} = h_2(K, st_{(j+m)w_i}) \oplus ind_m$. However, since the adversary may inject only one file at a time, it can still know the index of injected file corresponding to the encrypted index. Eventually, the adversary recovers the content of query by the returned encrypted indexes. For example, although Bost's protocol [6] also encrypts the index of file, it still cannot resist such an attack. To address this drawback, we add a padding index ind^* if $|I_{w_i}| = 1$ in our protocol. The adversary is, therefore, unable to determine the mapping between the encrypted indexes and the plaintext index. Therefore, our protocol can resist the file-injected attack.

6. Performance analysis

We will now present a comparative summary of the computation cost and security properties between our protocol and those presented in [5,6,13,34], based on bilinear pairing operation, the map-to-point, exponentiation, keyed hash function, scalar point multiplication, etc.—see Table 2. The notations used in the comparative summary are as follows:

1. T_p: Time cost for a bilinear pairing.
2. T_{mtp}: Time cost for a map-to-point hash function.
3. T_{sm}: Time cost for a scalar point multiplication operation in $\mathbb{G}_1$.
4. T_{exp}: Time cost for an exponentiation operation in $\mathbb{G}_2$.
5. T_{mul}: Time cost for a multiplication operation in $\mathbb{G}_2$.
6. T_e: Time cost for a encryption process of TPF, $f_{sk}(x) = y$.
7. T_d: Time cost for a decryption process of TPF, $f_{pk}(y) = x$.
8. T_h: Time cost for a keyed hash function.

As shown in Table 2, in our protocol, the computation cost of encryption and authorization phase, trapdoor generation phase and search phase are $1T_e + 2T_h$, $3T_{sm} + 1T_h$ and $1T_d$, respectively. Our protocol is the only protocol to achieve inside KGA resilience, forward privacy and FIA resilience.

To achieve a baseline security level for comparison, the 1024-bit RSA algorithm is used as the trapdoor function, and the keyed hash function is instantiated using SHA-1. $\mathbb{G}_1$ with order q is generated by a generator on an elliptic curve $E(F_p)$, where q and p are the 160-bits and 512-bits prime numbers, respectively. To evaluate the efficiency of the five protocols, we perform our experiments on a personal computer with a single Intel core i5 CPU 2.50 GHz, 8.00 GB of RAM, a 250 GB, Lenovo T410 running Windows 7. We run the related operations based on *MIRACL* library each for *100,000* times, and the average runtime is presented in Table 3.

As shown in Fig. 7, the time cost in the protocols of Bost [6] and ours is similar. However, our protocol is more efficient at the encryption phase, due to the fact that we do not require any complex operations, such as pairing and map-to-point. The protocol in [5] is the most inefficient due to the need for a pairing operation and a map-to-point hash function operation per encryption phase.

As presented in Fig. 8, our protocol has a higher computational cost at the authorization and trapdoor phase than that of Bost [6], although it is still significantly lower than the other protocols. As indicated in Fig. 9, the computation cost at the search phase in our protocol is similar to Bost's protocol [6] but is slightly higher than Chen et al.'s [13]. The main reason is that the decryption of RSA

Table 3
Runtime of main operations.

operation	T_p	T_{mtp}	T_{sm}	T_{exp}	T_{mul}	T_e	T_d	T_h
Time (ms)	7.342	7.726	0.986	0.484	0.001	0.189	5.896	0.005

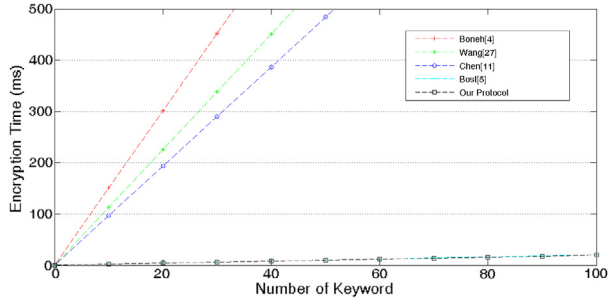

Fig. 7. Computation cost at encryption phase.

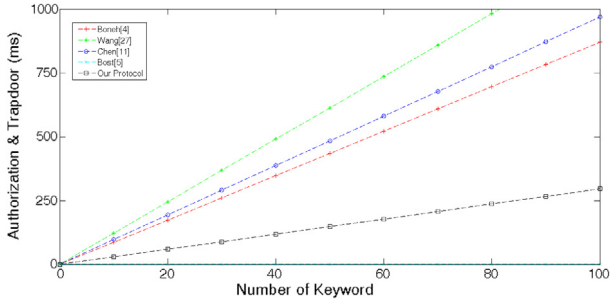

Fig. 8. Computation costs at authorization and trapdoor phase.

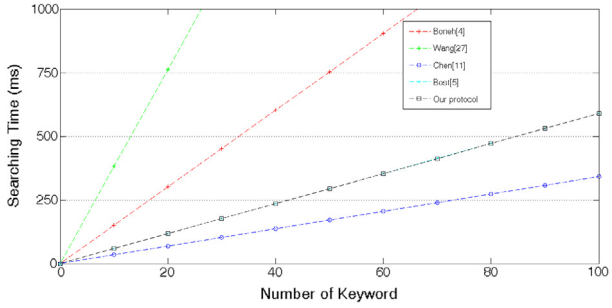

Fig. 9. Computation cost at search phase.

is a time-consuming operation. However, the search time of our protocol is still less than the remaining protocols.

In other words, the performance of our protocol is similar to that of Bost [6] and is more efficient than the remaining three protocols studied in this paper [5,13,34]. In addition, our protocol avoids the key management and distribution problem, and achieves inside KGA resilience, forward privacy and FIA resilience.

7. Conclusion

Cloud of Things is likely to be more popular and possibly resulting in other trends such as Cloud of Battlefield Things and Cloud of Military Things. Therefore, it is important for any organization, public or private, seeking to deploy Cloud of Things and related architecture to be assured of the security of the system and privacy of data outsourced to the cloud.

In this paper, we sought to contribute to one of many Cloud of Things security and privacy challenges. Specifically, we defined a searchable encryption protocol, its security model, and security requirements. We then proved the security of the protocol, as well as demonstrating the utility of the protocol in comparison to four other related protocols in the literature.

Future research includes collaborating with a Cloud of Things provider to implement a prototype of the proposed protocol, with the aims of evaluating and refining the protocol to make it more scalable and applicable for real-world deployment.

Acknowledgments

The work was supported by the National Natural Science Foundation of China (Nos. 61472287, 61501333, 61572379, and 61402339), the National High-tech R&D Program of China (863 Program) (No. 2015AA016004), the open fund of Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201608), and the Natural Science Foundation of Hubei Province of China (Nos. 2015CFA068and 2015CFB257).

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, H. Shi, Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions, in: Annual International Cryptology Conference, Springer, 2005, pp. 205–222.
[2] J. Baek, R. Safavi-Naini, W. Susilo, Public key encryption with keyword search revisited, in: International Conference on Computational Science and Its Applications, Springer, 2008, pp. 1249–1259.
[3] F. Bao, R.H. Deng, X. Ding, Y. Yang, Private query on encrypted data in multi-user settings, in: International Conference on Information Security Practice and Experience, Springer, 2008, pp. 71–85.
[4] S. Bhattacharya, T.F. Keefe, W.-T. Tsai, Covert channel secure hypercube message communication, J. Parallel Distrib. Comput. 26 (2) (1995) 233–247.
[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano, Public key encryption with keyword search, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 506–522.
[6] R. Bost, $o\varphi o\varsigma$: forward secure searchable encryption, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 1143–1154.
[7] J.W. Byun, H.S. Rhee, H.-A. Park, D.H. Lee, Off-line keyword guessing attacks on recent keyword search schemes over encrypted data, in: Workshop on Secure Data Management, Springer, 2006, pp. 75–83.
[8] D. Cash, P. Grubbs, J. Perry, T. Ristenpart, Leakage-abuse attacks against searchable encryption, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 668–679.
[9] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, M. Steiner, Highly-scalable searchable symmetric encryption with support for boolean queries, in: Advances in Cryptology–CRYPTO 2013, Springer, 2013, pp. 353–373.
[10] A. Castiglione, A. De Santis, B. Masucci, Key indistinguishability versus strong key indistinguishability for hierarchical key assignment schemes, IEEE Trans. Dependable Secure Comput. 13 (4) (2016) 451–460.
[11] A. Castiglione, A. De Santis, B. Masucci, F. Palmieri, A. Castiglione, X. Huang, Cryptographic hierarchical access control for dynamic structures, IEEE Trans. Inf. Forensics Secur. 11 (10) (2016) 2349–2364.
[12] R. Chen, Y. Mu, G. Yang, F. Guo, X. Huang, X. Wang, Y. Wang, Server-aided public key encryption with keyword search, IEEE Trans. Inf. Forensics Secur. 11 (12) (2016) 2833–2842.
[13] R. Chen, Y. Mu, G. Yang, F. Guo, X. Wang, Dual-server public-key encryption with keyword search for secure cloud storage, IEEE Trans. Inf. Forensics Secur. 11 (4) (2016) 789–798.
[14] K.-K.R. Choo, Cloud computing: Challenges and future directions, in: Trends & Issues in Crime and Criminal Justice, vol. 400, 2010, pp. 1–6.
[15] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, J. Comput. Secur. 19 (5) (2011) 895–934.
[16] G. Di Crescenzo, V. Saraswat, Public key encryption with searchable keywords based on Jacobi symbols, in: International Conference on Cryptology in India, Springer, 2007, pp. 282–296.

[17] C. Dong, G. Russello, N. Dulay, Shared and searchable encrypted data for untrusted servers, J. Comput. Secur. 19 (3) (2011) 367–397.
[18] C. Esposito, A. Castiglione, B. Martini, K.-K.R. Choo, Cloud manufacturing: security, privacy, and forensic concerns, IEEE Cloud Comput. 3 (4) (2016) 16–22.
[19] E.-J. Goh, Secure indexes, in: IACR Cryptology EPrint Archive, vol. 2003, 2003, pp. 216.
[20] Q. Huang, H. Li, An efficient public-key searchable encryption scheme secure against inside keyword guessing attacks, Inform. Sci. 403 (2017) 1–14.
[21] M.S. Islam, M. Kuzu, M. Kantarcioglu, Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In: NDSS, vol. 20, 2012, p. 12.
[22] I.R. Jeong, J.O. Kwon, D. Hong, D.H. Lee, Constructing PEKS schemes secure against keyword guessing attacks is possible? Comput. Commun. 32 (2) (2009) 394–396.
[23] S. Kamara, C. Papamanthou, T. Roeder, Dynamic searchable symmetric encryption, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security, ACM, 2012, pp. 965–976.
[24] K. Liang, W. Susilo, Searchable attribute-based mechanism with efficient data sharing for secure cloud storage, IEEE Trans. Inf. Forensics Secur. 10 (9) (2015) 1981–1992.
[25] K. Liu, N. Abu-Ghazaleh, K.-D. Kang, Location verification and trust management for resilient geographic routing, J. Parallel Distrib. Comput. 67 (2) (2007) 215–228.
[26] G.S. Poh, J.-J. Chin, W.-C. Yau, K.-K.R. Choo, M.S. Mohamad, Searchable symmetric encryption: designs and challenges, ACM Comput. Surv. 50 (3) (2017) 40.
[27] C. Prabhavathi, P.A. Vardhini, Verifiable Attribute Based Keyword Search over Outsourced Encrypted Data.
[28] M. Roopaei, P. Rad, K.-K.R. Choo, Cloud of things in smart agriculture: intelligent irrigation monitoring by thermal imaging, IEEE Cloud Comput. 4 (1) (2017) 10–15.
[29] D.X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: 2000 IEEE Symposium on Security and Privacy, 2000. S&P 2000. Proceedings, IEEE, 2000, pp. 44–55.
[30] W. Song, B. Wang, Q. Wang, Z. Peng, W. Lou, Y. Cui, A privacy-preserved full-text retrieval algorithm over encrypted data for cloud storage applications, J. Parallel Distrib. Comput. 99 (2017) 14–27.
[31] E. Stefanov, C. Papamanthou, E. Shi, Practical dynamic searchable encryption with small leakage, in: NDSS, vol. 71, 2014, pp. 72–75.
[32] S.-F. Sun, J.K. Liu, A. Sakzad, R. Steinfeld, T.H. Yuen, An efficient non-interactive multi-client searchable encryption with support for boolean queries, in: European Symposium on Research in Computer Security, Springer, 2016, pp. 154–172.
[33] L.-O. Wallin, T. Zimmerman, 2017 Strategic Roadmap for IoT Network Technology. Technical Report; Gartner, 2017.
[34] X.-F. Wang, Y. Mu, R. Chen, X.-S. Zhang, Secure channel free id-based searchable encryption for peer-to-peer group, J. Comput. Sci. Tech. 31 (5) (2016) 1012–1027.
[35] P. Xu, H. Jin, Q. Wu, W. Wang, Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack, IEEE Trans. Comput. 62 (11) (2013) 2266–2277.
[36] Y. Yang, H. Lu, J. Weng, Multi-user private keyword search for cloud computing, in: 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2011, pp. 264–271.
[37] Y. Zhang, J. Katz, C. Papamanthou, All your queries are belong to us: The power of file-injection attacks on searchable encryption, in: IACR Cryptology ePrint Archive, vol. 2016, 2016, p. 172.

Libing Wu was born in 1972. He received the B.S. and M.S. degrees in computer science from Central China Normal University, Wuhan, China, in 1994 and 2001, respectively. He received his Ph.D. degree in computer science from Wuhan University in 2006. He is now a professor in the School of Computer Science, Wuhan University, China. He is a senior member of IEEE and CCF. His areas of research interests include distributed computing, trusted software and wireless sensor networks.

Biwen Chen received his M.S. degree in computer sciences from Hubei University of Technology, Wuhan, China, in 2013. And he is currently pursuing the Ph.D. degrees in computer science from Wuhan University, Wuhan, China. His major research interests include cryptography, data security and privacy in cloud computing, cyber security.

Kim-Kwang Raymond Choo received his Ph.D. in Information Security in 2006 from Queensland University of Technology, Australia. He currently holds the Cloud Technology Endowed Professorship at The University of Texas at San Antonio, and is an adjunct associate professor at University of South Australia. He is the recipient of various awards including ESORICS 2015 Best Paper Award, Winning Team of the Germany's University of Erlangen-Nuremberg (FAU) Digital Forensics Research Challenge 2015, 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and British Computer Society's Wilkes Award in 2008. He is a fellow of the Australian Computer Society, and a senior member of IEEE.

Debiao He received his Ph.D. degree in applied mathematics from School of Mathematics and Statistics, Wuhan University in 2009. He is currently a professor of the State Key Lab of Software Engineering, Computer School, Wuhan University. His main research interests include cryptography and information security, in particular, cryptographic protocols.