Dynamic Searchable Public-Key Ciphertexts with Fast Performance and Practical Security

Peng Xu, Xia Gao, Wei Wang, Willy Susilo, Qianhong Wu, Hai Jin

Abstract

Public-key encryption with keyword search (PEKS) allows a sender to generate keyword-searchable ciphertexts using a receiver's public key and upload them to a server. Upon receiving a keyword-search trapdoor from the receiver, the server finds all matching ciphertexts. Due to the characteristics of public-key encryption, PEKS is inherently suitable for the application of numerous senders. Hence, PEKS is a well-known method to achieve secure keyword search over the encrypted email system. However, we find that without a keyword-search trapdoor, the traditional concept of PEKS still allows the server to have the obvious advantage to distinguish ciphertexts in practice. In other words, the traditional PEKS cannot guarantee the well-recognized semantic security in practice. To solve this problem, this paper defines a new concept called dynamic searchable public-key encryption (DSPE). It can hide the relationships between keyword-searchable ciphertexts and their corresponding encrypted files, and guarantee semantic security in both theory and practice. In addition, it allows the server to delete the intended ciphertexts according to the receiver's requirement. Then, we construct a DSPE instance with provable semantic security in the random oracle model. In terms of performance, the proposed instance also has the advantage that it only requires sublinear complexity to determine all matching ciphertexts or to delete the intended ciphertexts. Finally, we experimentally demonstrate the practicability of the instance.

Index Terms

Public-key Encryption with Keyword Search, Semantic Security, Dynamic Searchable Public-Key Encryption, Random Oracle

I. Introduction

CLOUD email system allows an enterprise to build an email system with much cheaper cost than the traditional on-premises solution. In 2017, the Radicati Group [1] showed the worldwide revenue forecast for Cloud Business Email, from 2017 to 2021. Figure 1 shows that the Cloud Business Email market is expected to generate nearly 43 billion by 2021. Hence, Cloud email system is a promising and important application due to its advantageous features. However, the honest-but-curious cloud platform makes users worry about their emails' privacy. To solve this problem, encrypting emails in public key setting has been recognized as a secure method by many famous companies, like Symantec, Voltage, Proofpoint and so on. The traditional public-key encryption (PKE) can keep the privacy of emails to the cloud platform. But it also makes the receivers of emails lose their capabilities to delegate keyword searches to the cloud platform. Boneh *et al.* first addressed this problem and proposed a new concept called public-key encryption with keyword search (PEKS) [2].

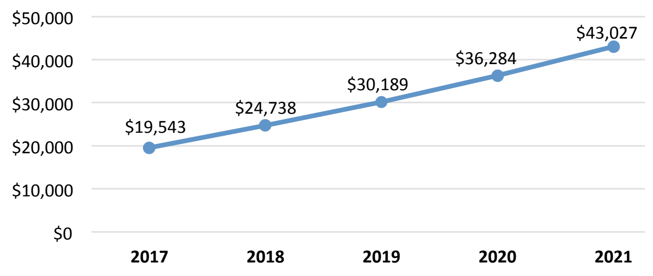


Figure 1: The worldwide revenue forecast for Cloud Business Email (unit: Million) [1].

PEKS has the advantage that all senders can generate keyword-searchable ciphertexts with a receiver's public key and upload these ciphertexts to a server. The receiver can delegate the keyword search to the server. Specifically, each

P. Xu, X. Gao and H. Jin are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. Emails: xupeng@mail.hust.edu.cn, 1040238358@qq.com, hjin@mail.hust.edu.cn.

W. Wang is with the Cyber-Physical-Social Systems Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. Email: viviawangww@gmail.com.

W. Susilo is with the Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Australia. Email: wsusilo@uow.edu.au.

Q. Wu is with the School of Electronic and Information Engineering, Beihang Univerisity, Beijing, China. Email: qianhong.wu@buaa.edu.cn.

sender separately encrypts his files and the extracted keywords using the receiver's public key, and sends the generated ciphertexts to the server; then, the receiver sends a keyword-search trapdoor to the server to retrieve the encrypted files with the expected keywords. Finally, the server finds all matching keyword-searchable ciphertexts, and sends the corresponding encrypted files to the receiver, and the receiver decrypts these files.

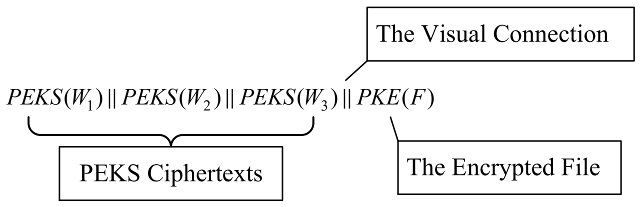


Figure 2: The Classic Appication of PEKS. Symbol $PEKS(W_i)$ denotes the keyword-searchable ciphertext generated by PEKS for keyword W_i. Symbol $PKE(F)$ denotes the encrypted file F using a traditional public-key encryption scheme. Symbol $||$ denotes the concatenation of strings.

In the previous works on PEKS, all keyword-searchable ciphertexts of a file must visually connect with the file's ciphertext, as shown in Figure 2. These connections make the function of deleting the searchable ciphertexts of an intended file easy. However, they imply that the server trivially knows that the keyword-searchable ciphertexts connected to the same file contain different keywords and the total number of keyword-searchable ciphertexts of a file. This leaked information breaks the semantic security of keywords [2] such that some keyword searchable ciphertexts are trivially distinguishable in the view of the server.

To address this problem, our basic idea is to hide the visible connections between keyword-searchable ciphertexts and the encrypted files. More specifically, suppose each file has a file identifier. In this scenario, a complete searchable ciphertext of a file consists of a keyword-searchable ciphertext, a file-identifier-searchable ciphertext and an encrypted file identifier of the file. Each searchable ciphertext and its encrypted file are separate, and all searchable ciphertexts are stored in a random order. Upon receiving a keyword-search trapdoor, the server identifies all matching keyword-searchable ciphertexts, decrypts the corresponding file identifiers and determines the corresponding encrypted files (the remaining operations are the same as those of the original PEKS). To delete all searchable ciphertexts of an intended file, the receiver first generates a file-delete trapdoor with the file's identifier and sends the trapdoor to the server; secondly, the server identifies all matching file-identifier-searchable ciphertexts and deletes the corresponding complete ciphertexts.

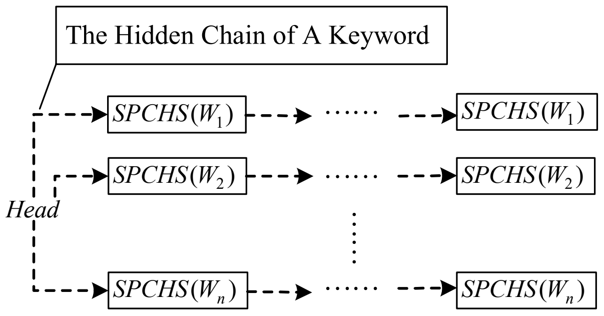


Figure 3: The Main Idea of SPCHS. Symbol $SPCHS(W_i)$ denotes the keyword-searchable ciphertext generated by SPCHS for keyword W_i. The dashed arrows denote the hidden relations.

The basic idea is effective in traditional PEKS schemes, but it is ineffective in PEKS schemes with both fast search performance and semantic security. This scheme is called SPCHS and was first proposed by Xu *et al.*[3]. In SPCHS, as shown in Figure 3, all searchable ciphertexts containing the same keyword, despite being in different files, are linked by a hidden chain, and a hidden relation links from a public *Head* to all the first searchable ciphertexts of these chains. When deleting a keyword-searchable ciphertext of a file, the corresponding chain of the keyword is broken, such that the keyword cannot be searched again. A similar problem also appeared in the early works of searchable symmetric-key encryption, such as in [4]. The problem was first solved by [5] in 2012, but our paper is the first to consider the problem in the field of PEKS.

A. Our Final Ideas

To solve the above problem, we first extend our basic idea to support fast keyword and file-identifier searches by constructing hidden relationships among searchable ciphertexts, as shown in Figure 4. All keyword-searchable ciphertexts

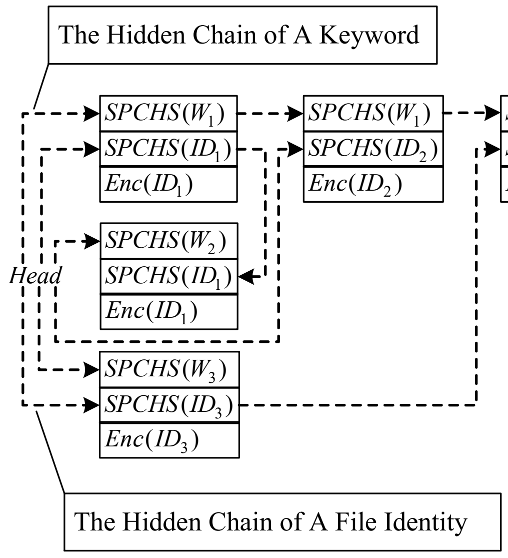


Figure 4: An Example of Our First Idea. Suppose there are three files ID_1, ID_2 and ID_3; file ID_1 has keywords W_1 and W_2, file ID_2 has keyword W_1, and file ID_3 has keywords W_3 and W_1. Each box denotes a complete searchable ciphertext, which consists of three parts. Symbol $SPCHS(W_i/ID_j)$ denotes the keyword/file-identifier-searchable ciphertext generated by SPCHS for keyword W_i or file identifier ID_j. Symbol $Enc(ID_i)$ denotes a ciphertext that can decrypt the file identifier ID_i by the corresponding keyword-search trapdoor.

of the same keyword are linked by a hidden chain. All file-identifier-searchable ciphertexts of the same file are also linked by a hidden chain, and a hidden relation links from a public *Head* to all the first searchable ciphertexts of these chains. With a keyword-search or file-delete trapdoor and the public *Head*, the server finds the first matching ciphertext by the corresponding relation from the Head at first. Secondly, another relation can be disclosed by the found ciphertext to guide the server to find the next matching ciphertext. By continuing in the same way, the server quickly finds all matching ciphertexts.

Second, to delete all searchable ciphertexts of a file, it is easy to delete all file-identifier-searchable ciphertexts of the file, but it is challenging to securely delete all keyword-searchable ciphertexts of the file and repair the broken chains. With a file-delete trapdoor, all keyword- and file-identifier-searchable ciphertexts of the file can be quickly found using the above mentioned idea. Since the found file-identifier-searchable ciphertexts have no relationship with other files' searchable ciphertexts, they can be directly deleted. However, deleting the found keyword searchable ciphertexts breaks the corresponding chains. To repair a broken chain, a straightforward method needs to know the former and latter keyword-searchable ciphertexts of the deleted ciphertext, and then link the former and latter ciphertexts to construct a complete chain. Clearly, this method is not compatible with semantic security since without the keyword-search trapdoor of a keyword, no one knows any information about the hidden chain corresponding to the keyword.

To overcome this challenge, we divide a complete deletion operation into two steps, logical and physical deletion. When deleting the keyword-searchable ciphertexts of a file, these ciphertexts are logically deleted by first tagging them with a special label. Upon receiving a keyword-search trapdoor, the server not only finds all matching keyword-searchable ciphertexts but also physically deletes their already tagged ciphertexts and repairs the broken chain. Clearly, this method avoids the above contradiction.

B. Our Work

We formally define the concept of dynamic searchable public-key encryption (DSPE) and its semantic security at first. DSPE is a general concept that includes not only the previous concepts of PEKS and SPCHS but also two new features: (1) each searchable ciphertext contains an encrypted file identifier and one can decrypt the file identifier with the corresponding keyword-search trapdoor; (2) with a file-delete trapdoor, the searchable ciphertexts of a file can be identified and deleted. It is worth noting that the first feature enables DSPE to construct a hidden connection between searchable ciphertexts and their corresponding encrypted files. Hence, DSPE can achieve semantic security not only in theory but also in practice.

The semantic security of DSPE is defined for all keywords, file identifiers and hidden relations. This means that (1) without any keyword-search or file-delete trapdoor, no one can distinguish any two searchable ciphertexts, and no information about the hidden relations is leaked, and (2) with a keyword-search or file-delete trapdoor, one can only disclose the corresponding relations, and the matching ciphertexts leak no information about the rest of the ciphertexts.

In contrast, the semantic security of PEKS is defined only for all keywords, and the semantic security of SPCHS is defined only for all keywords and hidden relations.

Following the formal definition of DSPE, this paper constructs a DSPE instance in which all searchable ciphertexts contain the hidden relationships shown in Figure 3. With the file-delete trapdoor of a file, the hidden chain of the file can be disclosed, and all ciphertexts of the file can be rapidly found and logically deleted. With a keyword-search trapdoor, the hidden chain of the coresponding keyword can be disclosed, and all ciphertexts of the keyword can be rapidly found. Simultaneously, identified ciphertexts that have been logically deleted by previous deletion operations are physically deleted, and the corresponding chain is repaired. Based on the computational bilinear Diffie-Hellman (CBDH) assumption [6], the instance is proven semantically secure in the random oracle (RO) model. In summary, our DSPE instance has the following advantages: fast search and delete performance and semantic security in both theory and practice.

C. Organization

Section II defines DSPE and its semantic security. Section III instantiates DPSE and proves the semantic security of the constructed instance. Section IV experimentally demonstrates the performance of our DSPE instance for searching keywords and deleting ciphertexts. Related work on PEKS is reviewed in Section V. This paper is concluded in Section VI.

II. Modeling DSPE

This section formalizes the model of DSPE and defines its security notion. In DSPE, suppose that each file has a unique file identifier ID and contains several keywords. Intuitively, DSPE extends SPCHS by defining two new algorithms $\mathbf{DeleteTrapdoor}$ and $\mathbf{Delete}$, which generate a file-delete trapdoor and delete the intended ciphertexts, respectively. In addition, DSPE extends the encryption algorithm of SPCHS by adding a file identifier as an input, such that the generated ciphertext can be searched by a file-delete trapdoor and decrypt the file identifier with the corresponding keyword-search trapdoor.

Definition 1 (DSPE). *Let $\mathcal{W}$ and $\mathcal{ID}$ be the keyword and file-identifier spaces, respectively. A DSPE scheme consists of seven algorithms:*

- $\mathbf{Setup}(1^k, \mathcal{W}, \mathcal{ID})$*: Take a security parameter 1^k, $\mathcal{W}$ and $\mathcal{ID}$ as inputs and probabilistically generate a pair of master public-and-secret keys $(\mathbf{PK}, \mathbf{SK})$, where $\mathbf{PK}$ includes $\mathcal{W}$, $\mathcal{ID}$ and the ciphertext space $\mathcal{C}$;*
- $\mathbf{StructureInitialization}(\mathbf{PK})$*: Take $\mathbf{PK}$ as input and probabilistically initialize a hidden structure by generating its private and public parts $(\mathbf{Pri}, \mathbf{Pub})$;*
- $\mathbf{Encryption}(\mathbf{PK}, w, id, \mathbf{Pri})$*: Take $\mathbf{PK}$, a keyword $w \in \mathcal{W}$, a file identifier $id \in \mathcal{ID}$ and a hidden structure's private part $\mathbf{Pri}$ as inputs, probabilistically generate a searchable ciphertext $C \in \mathcal{C}$ with the hidden structure of $\mathbf{Pri}$, and update $\mathbf{Pri}$;*
- $\mathbf{SearchTrapdoor}(\mathbf{SK}, w)$*: Take $\mathbf{SK}$ and a keyword $w \in \mathcal{W}$ as inputs and generate the corresponding keyword-search trapdoor T_w;*
- $\mathbf{DeleteTrapdoor}(\mathbf{SK}, id)$*: Take $\mathbf{SK}$ and a file identifier $id \in \mathcal{ID}$ as inputs and output a file-delete trapdoor T_{id} of id;*
- $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w'})$*: Take $\mathbf{PK}$, a hidden structure's public part $\mathbf{Pub}$, all searchable ciphertexts $\mathbb{C}$ and a keyword-search trapdoor $T_{w'}$ of keyword w' as inputs, disclose partial relations for guidance to identify the ciphertexts containing keyword w' with the hidden structure of $\mathbf{Pub}$, decrypt the identified ciphertexts and output the contained file identifiers;*
- $\mathbf{Delete}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{id'})$*: Take $\mathbf{PK}$, a hidden structure's public part $\mathbf{Pub}$, all searchable ciphertexts $\mathbb{C}$ and a file-delete trapdoor $T_{id'}$ of file id' as inputs, disclose partial relations for guidance to find out the ciphertexts containing file id' with the hidden structure of $\mathbf{Pub}$, and delete the found ciphertexts.*

Additionally, a DSPE scheme must be consistent in the following two senses: (1) given any keyword-search trapdoor $T_{w'}$ and any hidden structure's public part $\mathbf{Pub}$, all ciphertexts of keyword w' with the hidden structure $\mathbf{Pub}$ can be found by algorithm $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w'})$; (2) similarly, given any file-delete trapdoor $T_{id'}$ and any hidden structure's public part $\mathbf{Pub}$, algorithm $\mathbf{Delete}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{id'})$ finds all ciphertexts of file id' with the hidden structure $\mathbf{Pub}$.

Our DSPE definition also implies the traditional definition of PEKS if no hidden structure is initialized and the input parameters $\mathbf{Pri}$ and $\mathbf{Pub}$, respectively, of algorithms $\mathbf{Encryption}$ and $\mathbf{Delete}$ are null.

In the application of DSPE, a receiver sets up DSPE by running algorithm $\mathbf{Setup}$. Each sender initializes a hidden structure before the initial time to generate a searchable ciphertext using algorithm $\mathbf{StructureInitialization}$, generates searchable ciphertexts for his files using algorithm $\mathbf{Encryption}$, and uploads these ciphertexts to a server. Algorithm $\mathbf{SearchTrapdoor}$ enables the receiver to generate a keyword-search trapdoor. Upon receiving this trapdoor, the server runs algorithm $\mathbf{Search}$ for all senders' structures to obtain the file identifiers, which identify the files containing the queried keyword. Similarly, upon receiving a file-delete trapdoor generated by algorithm $\mathbf{DeleteTrapdoor}$ from the receiver, the server runs algorithm $\mathbf{Delete}$ for all senders' structures to delete the searchable ciphertexts of the intended file.

The goal of the semantic security of DSPE is to resist adaptively chosen keyword, file-identifier and structure attacks (SS-CKFSA). As in the semantic security of SPCHS, a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ is allowed to know the master public key and all structures' public parts, query the trapdoors for adaptively chosen keywords, and query the private parts for adaptively chosen structures. In addition, SS-CKFSA security allows the adversary $\mathcal{A}$ to query the trapdoors for adaptively chosen file identifiers and to query the searchable ciphertexts for adaptively chosen keywords, file identifiers and structures, including the targets that the adversary would like to be challenged. The adversary chooses two challenge triples, in which each triple consists of a keyword, a file identifier and a structure. SS-CKFSA security means that given a ciphertext of one of the two challenge triples, the adversary cannot determine which challenge keyword or which challenge file identifier or which challenge structure the challenge ciphertext corresponds to, if the adversary does not know the two challenge keywords search trapdoors, the two challenge file identifiers' delete trapdoors, and the two challenge structures private parts.

Definition 2 (SS-CKFSA Security)**.** *Suppose there are $N \in \mathbb{N}$ hidden structures at most . A DSPE scheme is SS-CKFSA secure, if any PPT adversary $\mathcal{A}$ has only a negligible advantage $Adv_{DSPE,\mathcal{A}}^{SS\text{-}CKFSA}$ to win in the following SS-CKFSA game:*

- ***Setup Phase**: A challenger sets up the DSPE scheme using algorithm $\mathbf{Setup}$ to generate a pair of master public-and-secret keys $(\mathbf{PK}, \mathbf{SK})$, initializes N hidden structures using algorithm $\mathbf{StructureInitialization}$ N times (let $\mathbf{PSet}$ be the set of all public parts of these N hidden structures), and sends $\mathbf{PK}$ and $\mathbf{PSet}$ to $\mathcal{A}$;*
- ***Query Phase 1**: Adversary $\mathcal{A}$ adaptively issues the following queries several times.*
  - ***Search-Trapdoor Query** $\mathcal{Q}_{STrap}(w)$: Taking a keyword $w \in \mathcal{W}$ as input, the challenger returns the keyword-search trapdoor of keyword w;*
  - ***Delete-Trapdoor** Query $\mathcal{Q}_{DTrap}(id)$: Taking a file identifier $id \in \mathcal{ID}$ as input, the challenger returns the file-delete trapdoor of file id;*
  - ***Privacy Query** $\mathcal{Q}_{Pri}(\mathbf{Pub})$: Taking a hidden structure's public part $\mathbf{Pub} \in \mathbf{PSet}$ as input, the challenger returns the corresponding private part;*
  - ***Encryption Query** $\mathcal{Q}_{Enc}(w, id, \mathbf{Pub})$: Taking a keyword $w \in \mathcal{W}$, a file identifier id and a hidden structure's public part $\mathbf{Pub}$ as inputs, the challenger generates and returns a searchable ciphertext of keyword w and file identifier id with the hidden structure $\mathbf{Pub}$.*
- ***Challenge Phase**: $\mathcal{A}$ sends two challenge triples $(w_0^*, id_0^*, \mathbf{Pub}_0^*) \in \mathcal{W} \times \mathcal{ID} \times \mathbf{PSet}$ and $(w_1^*, id_1^*, \mathbf{Pub}_1^*) \in \mathcal{W} \times \mathcal{ID} \times \mathbf{PSet}$ to the challenger. The challenger randomly chooses $d \in \{0,1\}$ and returns a challenge ciphertext C_d^* of $(w_d^*, id_d^*, \mathbf{Pub}_d^*)$ to $\mathcal{A}$.*
- ***Query Phase 2**: It is the same as **Query Phase 1**. Note that both in **Query Phase 1** and **Query Phase 2**, adversary $\mathcal{A}$ cannot query the corresponding private parts of $\mathbf{Pub}_0^*$ and $\mathbf{Pub}_1^*$, the keyword-search trapdoors of w_0^* and w_1^*, and the file-delete trapdoors of id_0^* and id_1^*.*
- ***Guess Phase**: $\mathcal{A}$ sends a guess d' to the challenger. We say that $\mathcal{A}$ wins if $d = d'$ and let $Adv_{DSPE,\mathcal{A}}^{SS\text{-}CKFSA} = Pr[d = d'] - \frac{1}{2}$ be the advantage of $\mathcal{A}$ to win in the above game.*

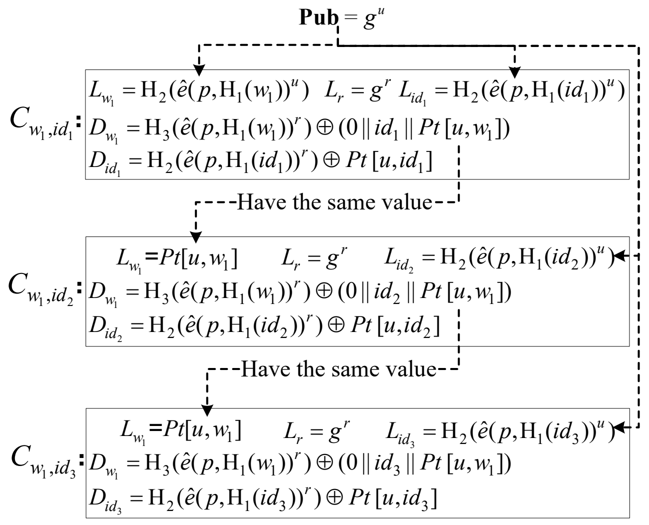
III. Instantiating DSPE

Let $x \xleftarrow{\$} \mathbf{X}$ denote an element x being randomly sampled from the set $\mathbf{X}$. Let $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map, where $\mathbb{G}$ and $\mathbb{G}_T$ denote two multiplicative groups of prime order q. Let g be a generator of $\mathbb{G}$. Let $\hat{e}$ be an efficiently computable and non-degenerate function with the bilinearity property $\hat{e}(g^a, g^b) = \hat{e}(g,g)^{ab}$, where $(a,b) \xleftarrow{\$} \mathbb{Z}_q^*$ and $\hat{e}(g,g)$ is a generator of $\mathbb{G}_T$. Let $\mathbf{BGen}(1^k)$ be an efficient bilinear map generator that takes a security parameter 1^k as input and probabilistically generates $(q, \mathbb{G}, \mathbb{G}_T, g, \hat{e})$. Let $\mathcal{W}$ and $\mathcal{ID}$ be the keyword and file identifier spaces, respectively. Suppose $\mathcal{W} \bigcap \mathcal{ID} = \emptyset$. Let $|\mathcal{ID}|$ denote the binary length of $\mathcal{ID}$. Suppose a deleted file identifier will never be re-used. A DSPE instance is then constructed as follows:

- $\mathbf{Setup}(1^k, \mathcal{W}, \mathcal{ID})$: Take a security parameter 1^k, $\mathcal{W}$ and $\mathcal{ID}$ as inputs, compute $(q, \mathbb{G}, \mathbb{G}_T, g, \hat{e}) \leftarrow \mathbf{BGen}(1^k)$, pick $s \xleftarrow{\$} \mathbb{Z}_q^*$, set $p = g^s$, set the ciphertext space $\mathcal{C} \subseteq \{0,1\}^{2k} \times \mathbb{G} \times \{0,1\}^{1+k+|\mathcal{ID}|} \times \{0,1\}^k$, choose three cryptographic hash functions $\mathbf{H}_1 : \{0,1\}^* \to \mathbb{G}$, $\mathbf{H}_2 : \mathbb{G}_T \to \{0,1\}^k$ and $\mathbf{H}_3 : \mathbb{G}_T \to \{0,1\}^{1+k+|\mathcal{ID}|}$, and finally generate the master public key $\mathbf{PK} = (q, \mathbb{G}, \mathbb{G}_T, g, \hat{e}, p, \mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3, \mathcal{W}, \mathcal{ID}, \mathcal{C})$ and the master secret key $\mathbf{SK} = s$.
- $\mathbf{StructureInitialization}(\mathbf{PK})$: Take $\mathbf{PK}$ as input, pick $u \xleftarrow{\$} \mathbb{Z}_q^*$, set $\mathbf{Pri} = (u)$ and $\mathbf{Pub} = g^u$, and initialize a hidden structure by generating its private and public parts $(\mathbf{Pri}, \mathbf{Pub})$. Note that $\mathbf{Pri}$ is a variable list formed as $\{u, (w, Pt[u,w]), (id, Pt[u,id]) | w \in \mathcal{W}, id \in \mathcal{ID}, Pt[u, w/id] \in \{0,1\}^k\}$, which is initialized as (u).
- $\mathbf{Encryption}(\mathbf{PK}, w, id, \mathbf{Pri})$: Take $\mathbf{PK}$, a keyword $w \in \mathcal{W}$, a file identifier $id \in \mathcal{ID}$ and a hidden structure's private part $\mathbf{Pri}$ as inputs, pick $r \xleftarrow{\$} \mathbb{Z}_q^*$, and perform the following steps to generate and output the searchable ciphertext $C = (L_w, L_{id}, L_r, D_w, D_{id})$:

1) Retrieve record $(w, Pt[u, w])$ by w in $\mathbf{Pri}$, set $L_r = g^r$;
2) If the record does not exist, add $(w, Pt[u, w] \xleftarrow{\$} \{0, 1\}^k)$ into $\mathbf{Pri}$, and set $L_w = \mathbf{H}_2(\hat{e}(p, \mathbf{H}_1(w))^u)$ and $D_w = \mathbf{H}_3(\hat{e}(p, \mathbf{H}_1(w))^r) \oplus (0||id||Pt[u, w])$;
3) Otherwise, pick $P_w \xleftarrow{\$} \{0, 1\}^k$, set $L_w = Pt[u, w]$ and $D_w = \mathbf{H}_3(\hat{e}(p, \mathbf{H}_1(w))^r) \oplus (0||id||P_w)$, update $Pt[u, w] = P_w$ in $\mathbf{Pri}$;
4) Retrieve record $(id, Pt[u, id])$ by id in $\mathbf{Pri}$;
5) If the record does not exist, add $(id, Pt[u, id] \xleftarrow{\$} \{0, 1\}^k)$ into $\mathbf{Pri}$, and set $L_{id} = \mathbf{H}_2(\hat{e}(p, \mathbf{H}_1(id))^u)$ and $D_{id} = \mathbf{H}_2(\hat{e}(p, \mathbf{H}_1(id))^r) \oplus Pt[u, id]$;
6) Otherwise, pick $P_{id} \xleftarrow{\$} \{0, 1\}^k$, set $L_{id} = Pt[u, id]$ and $D_{id} = \mathbf{H}_2(\hat{e}(p, \mathbf{H}_1(id))^r) \oplus P_{id}$, update $Pt[u, id] = P_{id}$ in $\mathbf{Pri}$.

- $\mathbf{SearchTrapdoor}(\mathbf{SK}, w)$: Take $\mathbf{SK}$ and a keyword $w \in \mathcal{W}$ as inputs and generate a keyword-search trapdoor $T_w = \mathbf{H}_1(w)^s$ of keyword w.
- $\mathbf{DeleteTrapdoor}(\mathbf{SK}, id)$: Take $\mathbf{SK}$ and a file identifier $id \in \mathcal{ID}$ as inputs and generate a file-delete trapdoor $T_{id} = \mathbf{H}_1(id)^s$ of file identifier id.
- $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w'})$: Take $\mathbf{PK}$, a hidden structure's public part $\mathbf{Pub}$, all searchable ciphertexts $\mathbb{C}$ and a keyword-search trapdoor $T_{w'}$ of keyword w' and perform the following steps:
  1) Initialize an empty set $\mathcal{I}$, two variables $i = 0$ and $j = 0$, and two temporary pointers $Pt = \mathbf{H}_2(\hat{e}(\mathbf{Pub}, T_{w'}))$ and $Pt' = NULL$;
  2) Set $i = i + 1$, seek a ciphertext $C^i = (L^i_w, L^i_{id}, L^i_r, D^i_w, D^i_{id})$ having $L^i_w = Pt$ in $\mathbb{C}$. If the ciphertext does not exist then return $\mathcal{I}$ and abort;
  3) Set $(Tag||id||Pt) = D^i_w \oplus \mathbf{H}_3(\hat{e}(L^i_r, T_{w'}))$;
  4) If $Tag = 0$, add id into $\mathcal{I}$, set $j = i$ and $Pt' = Pt$, and go to step 2);
  5) If $Tag = 1$ and $i = 1$, set $j = i$ and $Pt' = Pt$, and go to step 2);
  6) If $Tag = 1$ and $i > 1$, update the previously found ciphertext $C^j = (L^j_w, L^j_{id}, L^j_r, D^j_w, D^j_{id})$ by setting $D^j_w = D^j_w \oplus (0||0^{|\mathcal{ID}|}||Pt' \oplus Pt)$, physically delete the ciphertext C^i and go to step 2);
- $\mathbf{Delete}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{id'})$: Take $\mathbf{PK}$, a hidden structure's public part $\mathbf{Pub}$, all searchable ciphertexts $\mathbb{C}$ and a file-delete trapdoor $T_{id'}$ of file identifier id' as inputs and perform the following steps:
  1) Initialize a temporary pointer $Pt = \mathbf{H}_2(\hat{e}(\mathbf{Pub}, T_{id'}))$;
  2) Seek a ciphertext $C = (L_w, L_{id}, L_r, D_w, D_{id})$ having $L_{id} = Pt$ in $\mathbb{C}$;
  3) If the ciphertext does not exist, return $\perp$ and abort;
  4) Otherwise, logically delete the ciphertext by updating its $D_w = D_w \oplus (1||0^{k+|\mathcal{ID}|})$, set $Pt = D_{id} \oplus \mathbf{H}_2(\hat{e}(L_r, T_{id'}))$ and go to step 2).



Pub = g^u
C_{w1,id1}: L_{w1} = H2(ê(p,H1(w1))^u) L_r = g^r L_{id1} = H2(ê(p,H1(id1))^u)
D_{w1} = H3(ê(p,H1(w1))^r) ⊕ (0 || id1 || Pt[u,w1])
D_{id1} = H2(ê(p,H1(id1))^r) ⊕ Pt[u,id1]
Have the same value
C_{w1,id2}: L_{w1}=Pt[u,w1] L_r = g^r L_{id2} = H2(ê(p,H1(id2))^u)
D_{w1} = H3(ê(p,H1(w1))^r) ⊕ (0 || id2 || Pt[u,w1])
D_{id2} = H2(ê(p,H1(id2))^r) ⊕ Pt[u,id2]
Have the same value
C_{w1,id3}: L_{w1}=Pt[u,w1] L_r = g^r L_{id3} = H2(ê(p,H1(id3))^u)
D_{w1} = H3(ê(p,H1(w1))^r) ⊕ (0 || id3 || Pt[u,w1])
D_{id3} = H2(ê(p,H1(id3))^r) ⊕ Pt[u,id3]

Figure 5: An Example. Note that in each ciphertext, parameter $Pt[u, w_1]$ is randomly chosen before generating D_{w_1} according to step 2) and 3) of algorithm $\mathbf{Encryption}$, and parameter r will be randomly chosen.

An Example. Suppose a receiver has generated the master public and secret keys using algorithm $\mathbf{Setup}$. A sender would like to generate searchable ciphertexts for files id_1, id_2 and id_3, and these files have the same keyword w_1. The sender runs algorithm $\mathbf{StructureInitialization}$ to initialize a hidden structure with parameters $(\mathbf{Pri} = u, \mathbf{Pub} = g^u)$.

Then he takes (w_1, id_1), (w_1, id_2) and (w_1, id_3) as inputs to run algorithm **Encryption** and generates searchable ciphertexts C_{w_1,id_1}, C_{w_1,id_2} and C_{w_1,id_3}, which are uploaded to a server, as shown in Figure 5.

Since ciphertext C_{w_1,id_1} is the first of keyword w_1, the generated L_{w_1} of C_{w_1,id_1} by step 2) of algorithm **Encryption** implies a hidden relationship with $\mathbf{Pub}$. The hidden relationship can be disclosed by steps 1) and 2) of algorithm **Search** with keyword-search trapdoor T_{w_1}. In addition, ciphertexts C_{w_1,id_1}, C_{w_1,id_2} and C_{w_1,id_3} construct a hidden chain by step 3) of algorithm **Encryption**. In other words, the part D_{w_1} of the former ciphertext encrypts a point $Pt[u, w_1]$ having the same value as the part L_{w_1} of the latter ciphertext. Also, by using keyword-search trapdoor T_{w_1}, the hidden chain can be disclosed by steps 3) and 4) of algorithm **Search**. Similarly, there are hidden relationships existing between $\mathbf{Pub}$ and parts L_{id_1}, L_{id_2} and L_{id_3} of those ciphertexts, according to step 5) of algorithm **Encryption**. These hidden relationships can be disclosed by step 1) of algorithm **Delete** using the corresponding file-delete trapdoors.

Suppose the receiver would like to delete ciphertext C_{w_1,id_2}. He generates file-delete trapdoor T_{id_2} using algorithm **DeleteTrapdoor**. With trapdoor T_{id_2}, the server logically deletes ciphertext C_{w_1,id_2} using algorithm **Delete**. In other words, part D_{w_1} of ciphertext C_{w_1,id_2} is modified to $D_{w_1} = \mathbf{H}_3(\hat{e}(p, \mathbf{H}_1(w_1))^r) \oplus (1||id_2||Pt[u, w_1])$. Upon receiving keyword-search trapdoor T_{w_1}, the server physically deletes ciphertext C_{w_1,id_2} and repairs the broken chain by step 6) of algorithm **Search**.

Consistency. The essence of proving the consistency of our instance is to prove that the corresponding ciphertexts can be found with a keyword-search or file-delete trapdoor. In other words, with a keyword-search trapdoor T_w or a file-delete trapdoor T_{id}, the server can sequentially compute some pointers directed at the intended ciphertexts. The consistency of our instance is formally proved as follows.

Theorem 1. *Suppose that except with a negligible probability in parameter k, the hash functions $\mathbf{H}_1$, $\mathbf{H}_2$ and $\mathbf{H}_3$ are collision free. The above DSPE scheme is consistent, also except with a negligible probability in parameter k.*

Proof: Referring to Definition 1, a DSPE scheme must be consistent in the following two senses: (1) given any keyword-search trapdoor $T_{w'}$ and any hidden structure's public part $\mathbf{Pub}$, all the ciphertexts of keyword w' with the hidden structure $\mathbf{Pub}$ can be found using algorithm $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w'})$; (2) given any file-delete trapdoor $T_{id'}$ and any hidden structure's public part $\mathbf{Pub}$, algorithm $\mathbf{Delete}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{id'})$ finds all the ciphertexts of file id' with the hidden structure $\mathbf{Pub}$.

To prove the consistency in the first sense, without loss of generality, it can be proven that given the keyword-searchable trapdoor $T_{w_i} = H(w_i)^s$ of keyword $w_i \in \mathcal{W}$ and the hidden structure's public part $\mathbf{Pub} = g^u$, algorithm $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w_i})$ finds all ciphertexts of keyword w_i with the hidden structure $\mathbf{Pub}$. According to algorithm **Encryption**, all ciphertexts of keyword w_i in the hidden structure $\mathbf{Pub}$ construct a hidden chain, and the first ciphertext (let it be $C^1_{w_i} = (L^1_{w_i}, L^1_{id}, L^1_r, D^1_{w_i}, D^1_{id})$) of the chain has $L^1_{w_i} = \mathbf{H}_2(\hat{e}(p, \mathbf{H}_1(w_i))^u)$. According to the first step of algorithm **Search**, it is easy to find that $Pt = L^1_{w_i}$. Hence, the first ciphertext of keyword w_i can be found. To find the remaining ciphertexts of keyword w_i, the third step of algorithm **Search** decrypts a pointer that has the same value as the first part of the next ciphertext of keyword w_i. In other words, the pointer is directed toward the next ciphertext of keyword w_i. Hence, algorithm $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w_i})$ can find the second ciphertext of keyword w_i. By continuing in this manner, all ciphertexts of keyword w_i with the hidden structure $\mathbf{Pub}$ can be found. In addition, to complete the proof, we must consider the special case that some ciphertexts of keyword w_i will be deleted from the hidden chain when running algorithm $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w_i})$. Hence, we must prove that the hidden chain will be repaired well before the next running of algorithm $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w_i})$. According to the fifth step of algorithm **Search**, the first ciphertext of the hidden chain never be deleted. When deleting a middle ciphertext of the hidden chain, the sixth step of algorithm **Search** repairs the hidden chain. For example, suppose that there are three ciphertexts $C^1_{w_i} = (L^1_{w_i}, L^1_{id}, L^1_r, D^1_{w_i}, D^1_{id})$, $C^2_{w_i} = (L^2_{w_i}, L^2_{id}, L^2_r, D^2_{w_i}, D^2_{id})$ and $C^3_{w_i} = (L^3_{w_i}, L^3_{id}, L^3_r, D^3_{w_i}, D^3_{id})$ in the hidden chain, $D^1_{w_i}$ encrypts $L^2_{w_i}$, and $D^2_{w_i}$ encrypts $L^3_{w_i}$. Upon deleting ciphertext $C^2_{w_i}$, the sixth step updates ciphertext $C^1_{w_i}$, such that $D^1_{w_i}$ encrypts $L^3_{w_i}$ (not $L^2_{w_i}$). In other words, it results in ciphertext $C^1_{w_i}$ being relinked to ciphertext $C^3_{w_i}$, so the repaired hidden chain is also workable. In this way, we can prove that algorithm $\mathbf{Search}(\mathbf{PK}, \mathbf{Pub}, \mathbb{C}, T_{w_i})$ is correct.

Compared with the proof in the first sense, it is easy to prove consistency in the second sense. Hence, the details are omitted here. ■

Security Proof. The SS-CKFSA security of our DSPE instance relies on the CBDH assumption [6], which is defined as follows.

Definition 3 (The CBDH Assumption.)**.** *The CBDH problem in $\mathbf{BGen}(1^k) = (q, \mathbb{G}, \mathbb{G}_T, g, \hat{e})$ is defined as the probability of any PPT algorithm $\mathcal{B}$ to compute the value $\hat{e}(g, g)^{abc}$ when given (g^a, g^b, g^c), where $(a, b, c) \xleftarrow{\$} \mathbb{Z}_q^{*3}$. Let $Pr^{CBDH}_{\mathcal{B}}(1^k)$ denote that probability. We say that the CBDH assumption holds in $\mathbf{BGen}(1^k)$ if the probability $Pr^{CBDH}_{\mathcal{B}}(1^k)$ is negligible in parameter k.*

For the security proof, we prove that if an adversary can break the SS-CKFSA security of our DSPE instance in the RO model, then an algorithm can be constructed to solve the CBDH problem in $\mathbf{BGen}(1^k)$. Formally, we have the following Theorem 2.

Theorem 2. *Suppose that there are at most $N \in \mathbb{N}$ hidden structures and the sizes of the keyword space $\mathcal{W}$ and file-identifier space $\mathcal{ID}$ are polynomial. Let the hash functions $\mathbf{H}_1$, $\mathbf{H}_2$ and $\mathbf{H}_3$ be modeled as three random oracles $\mathcal{Q}_{H_1}(\cdot)$, $\mathcal{Q}_{H_2}(\cdot)$ and $\mathcal{Q}_{H_3}(\cdot)$, respectively. Suppose a PPT adversary $\mathcal{A}$ wins in the SS-CKFSA game of the above DSPE scheme with advantage $Adv^{SS\text{-}CKFSA}_{DSPE,\mathcal{A}}$, in which $\mathcal{A}$ makes at most q_1 queries to Oracle Query $\mathcal{Q}_{H_1}(\cdot)$, at most q_2 queries to Oracle Query $\mathcal{Q}_{H_2}(\cdot)$, at most q_3 queries to Oracle Query $\mathcal{Q}_{H_3}(\cdot)$, at most q_s queries to Search-Trapdoor Query$\mathcal{Q}_{STrap}(\cdot)$, at most q_d queries to Delete-Trapdoor Query $\mathcal{Q}_{DTrap}(\cdot)$, at most q_p queries to Privacy Query $\mathcal{Q}_{Pri}(\cdot)$ and at most q_e queries to Encryption Query $\mathcal{Q}_{Enc}(\cdot)$. Then, there is a PPT algorithm $\mathcal{B}$ that solves the CBDH problem in $\mathbf{BGen}(1^k)$ with probability greater than*

$$\frac{6^6 \cdot Adv^{SS\text{-}CKFSA}_{DSPE,\mathcal{A}}}{e^6 \cdot (q_s + q_d + q_p)^6 \cdot (q_3 + q_e) \cdot (|\mathcal{W}| + |\mathcal{ID}|) \cdot N \cdot \sigma^2}$$

, where e is the base of natural logarithms.

Proof: To prove this theorem, we construct a PPT algorithm $\mathcal{B}$ that plays the SS-CKFSA game with adversary $\mathcal{A}$ and utilizes the capability of $\mathcal{A}$ to solve the CBDH problem in $\mathbf{BGen}(1^k) = (q, \mathbb{G}, \mathbb{G}_T, g, \hat{e})$ with probability $Pr^{\text{CBDH}}_{\mathcal{B}}(1^k)$. Let $coin \overset{\sigma}{\leftarrow} \{0,1\}$ denote the operation that picks $coin \in \{0,1\}$ according to probability $Pr[coin = 1] = \sigma$ (the specified value of σ will be determined later). The constructed algorithm $\mathcal{B}$ in the SS-CKFSA game is as follows.

- **Setup Phase**: Algorithm $\mathcal{B}$ takes $(q, \mathbb{G}, \mathbb{G}_T, g, \hat{e}, g^a, g^b, g^c)$ and spaces $(\mathcal{W}, \mathcal{ID})$ as inputs and performs the following steps:
  1) Initialize the five lists $\mathbf{H_1List} = \emptyset \subseteq \{\mathcal{W}, \mathcal{ID}\} \times \mathbb{G} \times \mathbb{Z}_q^* \times \{0,1\}$, $\mathbf{H_2List} = \emptyset \subseteq \mathbb{G}_T \times \{0,1\}^k$, $\mathbf{H_3List} = \emptyset \subseteq \mathbb{G}_T \times \{0,1\}^{1+k+|\mathcal{ID}|}$, $\mathbf{Pt} = \emptyset \subseteq \{\mathcal{W}, \mathcal{ID}\} \times \mathbb{G} \times \{0,1\}^k$ and $\mathbf{SList} = \emptyset \subseteq \mathbb{G} \times \mathbb{Z}_q^* \times \{0,1\}$;
  2) Initialize N hidden structures by repeating the following steps for $i \in [1, N]$:
     a) Pick $u_i \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ and $coin_i \overset{\$}{\leftarrow} \{0,1\}$;
     b) If $coin_i = 1$, set $\mathbf{Pub}_i = g^{b \cdot u_i}$;
     c) Otherwise, set $\mathbf{Pub}_i = g^{u_i}$;
  3) Set $\mathbf{PSet} = \{\mathbf{Pub}_i | i \in [1, N]\}$ and $\mathbf{SList} = \{(\mathbf{Pub}_i, u_i, coin_i) | i \in [1, N]\}$;
  4) Set the master public key $\mathbf{PK} = (q, \mathbb{G}, \mathbb{G}_T, g, \hat{e}, p = g^a, \mathcal{W}, \mathcal{ID}, \mathcal{C})$, and send $\mathbf{PK}$ and $\mathbf{PSet}$ to adversary $\mathcal{A}$;
- **Query Phase 1**: Adversary $\mathcal{A}$ adaptively issues the following queries multiple times.
  – Oracle Query $\mathcal{Q}_{H_1}(w/id)$: Taking a keyword w or a file identifier id (which has never been queried before) as input, algorithm $\mathcal{B}$ picks $x \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ and $coin \overset{\sigma}{\leftarrow} \{0,1\}$, adds $(w/id, z = g^x, x, coin)$ into $\mathbf{H_1List}$ and outputs z if $coin = 0$, otherwise adds $(w/id, z = g^{c \cdot x}, x, coin)$ into $\mathbf{H_1List}$ and outputs z;
  – Oracle Query $\mathcal{Q}_{H_2}(v)$: Taking a value $v \in \mathbb{G}_T$ (which has never been queried before) as input, algorithm $\mathcal{B}$ picks $y \overset{\$}{\leftarrow} \{0,1\}^k$, adds (v, y) into $\mathbf{H_2List}$ and outputs y;
  – Oracle Query $\mathcal{Q}_{H_3}(v)$: Taking a value $v \in \mathbb{G}_T$ (which has never been queried before) as input, algorithm $\mathcal{B}$ picks $y \overset{\$}{\leftarrow} \{0,1\}^{1+k+|\mathcal{ID}|}$, adds (v, y) into $\mathbf{H_3List}$ and outputs y;
  – Search-Trapdoor Query $\mathcal{Q}_{STrap}(w)$: Taking a keyword $w \in \mathcal{W}$ as input, algorithm $\mathcal{B}$ queries oracle $\mathcal{Q}_{H_1}(w)$ if w has never been queried before, retrieves $(w, z, x, coin)$ according to w from $\mathbf{H_1List}$, outputs $g^{a \cdot x}$ if $coin = 0$, otherwise aborts and outputs $\perp$;
  – Delete-Trapdoor Query $\mathcal{Q}_{DTrap}(id)$: Taking a file identifier $id \in \mathcal{ID}$ as input, algorithm $\mathcal{B}$ queries oracle $\mathcal{Q}_{H_1}(id)$ if id has never been queried before, retrieves $(id, z, x, coin)$ according to id from $\mathbf{H_1List}$, outputs $g^{a \cdot x}$ if $coin = 0$, otherwise aborts and outputs $\perp$;
  – Privacy Query $\mathcal{Q}_{Pri}(\mathbf{Pub})$: Taking a hidden structure's public part $\mathbf{Pub} \in \mathbf{PSet}$ as input, algorithm $\mathcal{B}$ retrieves $(\mathbf{Pub}, u, coin)$ according to $\mathbf{Pub}$ from $\mathbf{SList}$, outputs u if $coin = 0$, otherwise aborts and outputs $\perp$;
  – Encryption Query $\mathcal{Q}_{Enc}(w, id, \mathbf{Pub}_i)$: Taking a keyword $w \in \mathcal{W}$, a file identifier $id \in \mathcal{ID}$ and a hidden structure's public part $\mathbf{Pub}_i$ as inputs, algorithm $\mathcal{B}$ queries oracle $\mathcal{Q}_{H_1}(w)$ or $\mathcal{Q}_{H_1}(id)$ if w or id have never been queried before, retrieves $(w, z_w, x_w, coin_w)$ and $(id, z_{id}, x_{id}, coin_{id})$ from $\mathbf{H_1List}$ according to w and id, respectively, retrieves $(\mathbf{Pub}_i, u_i, coin_i)$ from $\mathbf{SList}$ according to $\mathbf{Pub}_i$, picks $r \overset{\$}{\leftarrow} \mathbb{Z}_q^*$, computes $L_r = g^r$ and performs the following steps:
    1) Seek $(w, \mathbf{Pub}_i, Pt[u_i, w])$ by w and $\mathbf{Pub}_i$ in $\mathbf{Pt}$;
    2) If it does not exist, add $(w, \mathbf{Pub}_i, Pt[u_i, w] \overset{\$}{\leftarrow} \{0,1\}^k)$ to $\mathbf{Pt}$, compute $D_w = \mathcal{Q}_{H_3}(\hat{e}(g^a, z_w)^r) \oplus (0||id||Pt[u_i, w])$ and perform the following steps:
       a) If $coin_w = 1 \bigwedge coin_i = 1$, set $L_w \overset{\$}{\leftarrow} \{0,1\}^k$;
       b) If $coin_w = 0 \bigwedge coin_i = 1$, set $L_w = \mathcal{Q}_{H_2}(\hat{e}(g^a, g^b)^{x_w \cdot u_i})$;
       c) If $coin_i = 0$, set $L_w = \mathcal{Q}_{H_2}(\hat{e}(g^a, z_w)^{u_i})$;

3) Otherwise, pick $P_w \xleftarrow{\$} \{0,1\}^k$, set $L_w = Pt[u_i, w]$, compute $D_w = \mathcal{Q}_{H_3}(\hat{e}(g^a, z_w)^r) \oplus (0||id||P_w)$ and update $Pt[u_i, w] = P_w$;
4) Seek $(id, \mathbf{Pub}_i, Pt[u_i, id])$ by id and $\mathbf{Pub}_i$ in $\mathbf{Pt}$;
5) If it does not exist, add $(id, \mathbf{Pub}_i, Pt[u_i, id] \xleftarrow{\$} \{0,1\}^k)$ to $\mathbf{Pt}$, compute $D_{id} = \mathcal{Q}_{H_2}(\hat{e}(g^a, z_{id})^r) \oplus Pt[u_i, id]$ and perform the following steps:
   a) If $coin_{id} = 1 \bigwedge coin_i = 1$, set $L_{id} \xleftarrow{\$} \{0,1\}^k$;
   b) If $coin_{id} = 0 \bigwedge coin_i = 1$, set $L_{id} = \mathcal{Q}_{H_2}(\hat{e}(g^a, g^b)^{x_{id} \cdot u_i})$;
   c) If $coin_i = 0$, set $L_{id} = \mathcal{Q}_{H_2}(\hat{e}(g^a, z_{id})^{u_i})$;
6) Otherwise, pick $P_{id} \xleftarrow{\$} \{0,1\}^k$, set $L_{id} = Pt[u_i, id]$, compute $D_{id} = \mathcal{Q}_{H_2}(\hat{e}(g^a, z_{id})^r) \oplus P_{id}$ and update $Pt[u_i, id] = P_{id}$;
7) Output the searchable ciphertext $C = (L_w, L_{id}, L_r, D_w, D_{id})$;

- **Challenge Phase**: Adversary $\mathcal{A}$ sends two challenge tuples $(w_0^*, id_0^*, \mathbf{Pub}_0^*)$ and $(w_1^*, id_1^*, \mathbf{Pub}_1^*)$ to algorithm $\mathcal{B}$; $\mathcal{B}$ randomly chooses $d \in \{0,1\}$ and performs the following steps:
  1) Query $\mathcal{Q}_{H_1}$ for the elements in $(w_0^*, id_0^*, w_1^*, id_1^*)$ that have never been queried before;
  2) If there is any element in $(w_0^*, id_0^*, \mathbf{Pub}_0^*, w_1^*, id_1^*, \mathbf{Pub}_1^*)$ whose $coin = 0$ in $\mathbf{H_1List}$ and $\mathbf{SList}$, then abort and output $\perp$;
  3) Generate the challenge ciphertext $C_d^* = (L_{w_d^*}, L_{id_d^*}, L_{r_d^*}, D_{w_d^*}, D_{id_d^*})$ as follows:
     a) Set $L_{r_d^*} = g^b$, $D_{w_d^*} \xleftarrow{\$} \{0,1\}^{1+k+|\mathcal{ID}|}$ and $D_{id_d^*} \xleftarrow{\$} \{0,1\}^k$;
     b) Seek $(w_d^*, \mathbf{Pub}_d^*, Pt[u_d^*, w_d^*])$ by w_d^* and $\mathbf{Pub}_d^*$ in $\mathbf{Pt}$;
     c) If it does not exist, add $(w_d^*, \mathbf{Pub}_d^*, Pt[u_d^*, w_d^*] \xleftarrow{\$} \{0,1\}^k)$ to $\mathbf{Pt}$, set $L_{w_d^*} \xleftarrow{\$} \{0,1\}^k$;
     d) Otherwise, pick $P_{w_d^*} \xleftarrow{\$} \{0,1\}^k$, set $L_{w_d^*} = Pt[u_d^*, w_d^*]$ and update $Pt[u_d^*, w_d^*] = P_{w_d^*}$;
     e) Seek $(id_d^*, \mathbf{Pub}_d^*, Pt[u_d^*, id_d^*])$ by id_d^* and $\mathbf{Pub}_d^*$ in $\mathbf{Pt}$;
     f) If it does not exist, add $(id_d^*, \mathbf{Pub}_d^*, Pt[u_d^*, id_d^*] \xleftarrow{\$} \{0,1\}^k)$ to $\mathbf{Pt}$, set $L_{id_d^*} \xleftarrow{\$} \{0,1\}^k$;
     g) Otherwise, pick $P_{id_d^*} \xleftarrow{\$} \{0,1\}^k$, set $L_{id_d^*} = Pt[u_d^*, id_d^*]$ and update $Pt[u_d^*, id_d^*] = P_{id_d^*}$;
  4) Send the challenge ciphertext C_d^* to $\mathcal{A}$;
- **Query Phase 2**: This phase is the same as **Query Phase 1**. Note that in **Query Phase 1** and **Query Phase 2**, adversary $\mathcal{A}$ cannot query the corresponding private parts of $\mathbf{Pub}_0^*$ and $\mathbf{Pub}_1^*$, the corresponding search trapdoors of w_0^* and w_1^*, and the corresponding delete trapdoors of id_0^* and id_1^*.
- **Guess Phase**: Adversary $\mathcal{A}$ sends a guess d' to the challenger. Algorithm $\mathcal{B}$ randomly chooses a record (v, y) in $\mathbf{H_3List}$, retrieves the record $(w_d^*, g^{b \cdot x_{w_d^*}}, x_{w_d^*}, coin_{w_d^*})$ according to w_d^* from list $\mathbf{H_1List}$, and solves the CBDH problem by computing and returning $v^{1/x_{w_d^*}}$.

Suppose algorithm $\mathcal{B}$ does not abort in the above SS-CKFSA game. Under this assumption, we have the following facts:

- It is easy to find that all the above phases, excluding the encryption query $\mathcal{Q}_{Enc}(w, id, \mathbf{Pub}_i)$ in **Query Phase** and the generation of the challenge ciphertext in **Challenge Phase**, are indistinguishable from a real SS-CKFA game.
- In **Query Phase**, an encryption query $\mathcal{Q}_{Enc}(w, id, \mathbf{Pub}_i)$ will not generate a real ciphertext if the corresponding $coin$ values are $coin_w = 1$ or $coin_{id} = 1$ when $coin_i = 1$. However, this exception cannot be found by adversary $\mathcal{A}$ if he does not query $\mathcal{Q}_{H_2}(\hat{e}(g,g)^{abc \cdot x_w u_i})$ or $\mathcal{Q}_{H_2}(\hat{e}(g,g)^{abc \cdot x_{id} u_i})$ in **Query Phase 1** or **Query Phase 2**.
- In **Challenge Phase**, the challenge ciphertext is incorrectly generated, but adversary $\mathcal{A}$ cannot find this incorrectness (or exception) if he does not query $\mathcal{Q}_{H_2}(\hat{e}(g,g)^{abc \cdot x_{w_d^*} u_d^*})$, $\mathcal{Q}_{H_2}(\hat{e}(g,g)^{abc \cdot x_{id_d^*} u_d^*})$, $\mathcal{Q}_{H_2}(\hat{e}(g,g)^{abc \cdot x_{id_d^*}})$ or $\mathcal{Q}_{H_3}(\hat{e}(g,g)^{abc \cdot x_{w_d^*}})$ for $d \in \{0,1\}$ in **Query Phase 1** or **Query Phase 2**. Therefore, adveresary $\mathcal{A}$ has no advantage to win the above game since the challenge ciphertext in this case is independent of the challenge tuples $(w_0^*, id_0^*, \mathbf{Pub}_0^*)$ and $(w_1^*, id_1^*, \mathbf{Pub}_1^*)$.
- Recall that adversary $\mathcal{A}$ has advantage $Adv_{\text{DSPE},\mathcal{A}}^{\text{SS-CKFSA}}$ to win the SS-CKFSA game. Let $Query$ be the event that adversary $\mathcal{A}$ issues the above mentioned queries in **Query Phase** or **Challenge Phase**. Correspondingly, let $\overline{Query}$ be the opposite event of $Query$. We have

$$\begin{aligned}
& Adv_{\text{DSPE},\mathcal{A}}^{\text{SS-CKFSA}} \\
& = Pr[d = d'] - \frac{1}{2} \\
& = Pr[d = d' | Query] \cdot Pr[Query] + \\
& \qquad Pr[d = d' | \overline{Query}] \cdot Pr[\overline{Query}] - \frac{1}{2} \\
& = (Pr[d = d' | Query] - \frac{1}{2}) \cdot Pr[Query]
\end{aligned}$$

Furthermore, it implies that $Pr[Query] > Adv^{\text{SS-CKFSA}}_{\text{DSPE},\mathcal{A}}$.

- According to the *coin* values of all keywords, file identifiers and hidden structures, the maximum number of possible queries in the event $Query$ is $(|\mathcal{W}| + |\mathcal{ID}|) \cdot N \cdot \sigma^2$. Hence, the probability of the event that adversary $\mathcal{A}$ queries $\mathcal{Q}_{H_3}(\hat{e}(g,g)^{abc \cdot x_{w_d^*}})$ is greater than $\frac{1}{(|\mathcal{W}|+|\mathcal{ID}|)\cdot N \cdot \sigma^2} \cdot Adv^{\text{SS-CKFSA}}_{\text{DSPE},\mathcal{A}}$. This implies that algorithm $\mathcal{B}$ has probability greater than $\frac{1}{(q_3+q_e)\cdot(|\mathcal{W}|+|\mathcal{ID}|)\cdot N\cdot\sigma^2} \cdot Adv^{\text{SS-CKFSA}}_{\text{DSPE},\mathcal{A}}$to solve the CBDH problem.

Let $\overline{Abort}$ denote the event that algorithm $\mathcal{B}$ does not abort in the above game. According to the above game, the probability of the event $\overline{Abort}$ only relies on the following three parts: (1) the probability σ; (2) the number of times adversary $\mathcal{A}$ queries trapdoors $\mathcal{Q}_{STrap}$ and $\mathcal{Q}_{DTrap}$ and privacy $\mathcal{Q}_{Pri}$; (3) the *coin* values of all elements in the challenge tuples. We have that $Pr[\overline{Abort}] = (1-\sigma)^{q_s+q_d+q_p} \cdot \sigma^6$. Let $\sigma = \frac{6}{q_s+q_d+q_p+6}$ and $Pr[\overline{Abort}] \approx (\frac{6}{e\cdot(q_s+q_d+q_p)})^6$, where e is the base of natural logarithms.

Finally, we have that algorithm $\mathcal{B}$ approximately has probability greater than

$$\frac{6^6 \cdot Adv^{\text{SS-CKFSA}}_{\text{DSPE},\mathcal{A}}}{e^6 \cdot (q_s+q_d+q_p)^6 \cdot (q_3+q_e) \cdot (|\mathcal{W}|+|\mathcal{ID}|) \cdot N \cdot \sigma^2}$$

to solve the CBDH problem, where e is the base of natural logarithms. ■

IV. Performance

We coded our DSPE instance and tested the time costs of algorithm **Search** to find the matching ciphertexts and physically delete some ciphertexts. Table I shows the system parameters, including computer hardware, system software and the chosen elliptic curve. Assume that there are 10^4 searchable ciphertexts, each ciphertext contains one of the keywords shown in Table II, and the number of ciphertexts containing the same keyword is approximately equal to the real frequency of the corresponding keyword multiplied by 10^4.

Table I: System parameters.

Hardware	Intel Xeon CPU E5-2420 v2 @ 2.20GHz
OS	CentOS
Program Library	Pairing-Based Cryptography (PBC)
Mathematical Parameters	
Elliptic Curve	$y^2 = x^3 + x$
Base Field	8780710799663312522437781984754049815806883199414208211028653399266475630880222957078625179422662221423155858769582317459277713367317481324925129998224791
Order	730750818665451621361119245571504901405976559617
The default unit is decimal	

Table II: The frequencies of some keywords.

No.	Keyword	Freq.	No.	Keyword	Freq.
1	life	6.68%	2	child	6.8%
3	woman	6.91%	4	thing	8.12%
5	man	8.29%	6	day	8.73%
7	way	9.5%	8	people	13.98%
9	time	15.47%	10	year	15.52%

Figure 6 shows the time costs to search for 10 keywords. Since the search complexity of a keyword in our DSPE instance is linearly related to the number of ciphertexts containing the keyword, the time costs of the keywords are different. For example, the time cost is approximately 779 ms to search the keyword ”life”, which has the minimum frequency, and the time cost is approximately 1799 ms to search keyword ”year”, which has the maximum frequency.

Figure 7 shows the time costs to physically delete some ciphertexts. Specifically, we test the time costs to physically delete different numbers of ciphertexts. For example, the time cost is approximately 0.4 ms to physically delete 100 ciphertexts, and the time cost is approximately 4.3 ms to physically delete 1000 ciphertexts.

We do not test the time cost of algorithm **Delete** to logically delete some cipertexts since the related operations take much less time than bilinear-map operation when seaching a keyword or physically deleting a ciphertext.

In summary, the search performance of our DSPE instance is linearly related to the number of ciphertexts containing the queried keyword, and its physical delete performance is linearly related to the number of ciphertexts to be deleted. Moreover, without considering the time cost to retrieve ciphertexts from storage, the two performance indicators are independent of the total number of ciphertexts. Hence, it is clearer to demonstrate the performance of our DSPE scheme by listing the time cost to find one matching ciphertext or to physically delete one ciphertext. According to our above

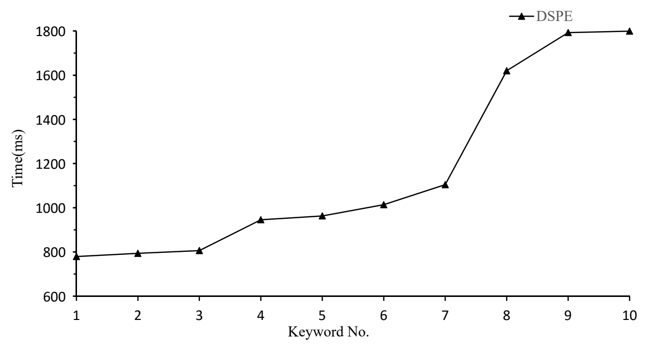


Figure 6: The time costs to search keywords.

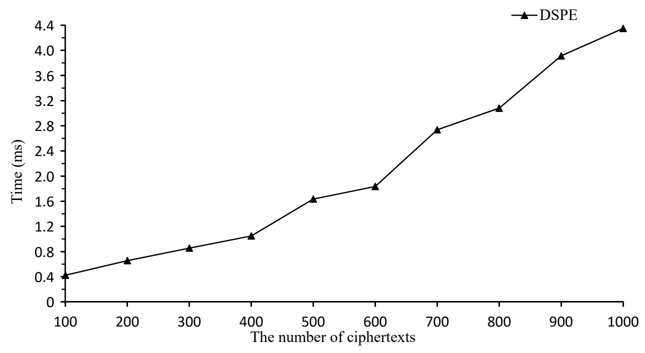


Figure 7: The time costs to physically delete some ciphertexts.

experiments, the time cost to find one matching ciphertext is approximately 1.16 ms, and the time cost to physically delete one ciphertext is approximately 0.004 ms.

V. Related Works

Most of the follow-up research to the seminal work on PEKS can be categorized as follows.

Traditional PEKS. Abdella *et al.* [7] revised the definition of PEKS's consistency and constructed some transformations among primitives related to PEKS. Crescenzo *et al.* [8] proposed a PEKS scheme using a variant of the quadratic residuosity problem instead of a bilinear map. Khader [9] proposed the first PEKS scheme in the standard model.

PEKS with Functional Search. The related works include conjunctive search [10], [11], [12], [13], [14], [15], range search [16], [17], [18], subset search [18], time-scope search [7], [19], similarity search [20], authorized search [21], [22], equality test for heterogeneous ciphertexts [23], and fuzzy keyword search [24].

Secure Channel Free PEKS. The traditional PEKS schemes require a secure channel to transmit keyword-search trapdoors. To remove this requirement, Baek *et al.* [15] proposed a secure channel-free PEKS (SCF-PEKS) scheme in which all keyword-search trapdoors are encrypted using the designated server's public key. Rhee *et al.* [25] proposed a SCF-PEKS scheme to achieve the enhanced security model of [15]. Fang *et al.* [26] proposed a SCF-PEKS scheme in the standard model. Emura *et al.* [27] proposed a SCF-PEKS scheme to resist adaptive attacks.

PEKS against Keyword Guessing Attack. Byun *et al.* first proposed the keyword guessing attack (KGA) [28] and launched an effective attack on some PEKS schemes [10]. Jeong *et al.* [29] proved that any PEKS scheme satisfying at least computationally indistinguishable consistency is subjected to KGA. Xu *et al.* [24] proposed a PEKS scheme with fuzzy keyword search to resist KGA in some scenarios. Furthermore, Chen *et al.* [30], [31] sequentially proposed two PEKS schemes to resist KGA and improve the performance. Sun *et al.* [32] proposed a new idea to resist KGA by limiting the capability to generate searchable ciphertexts.

Efficient PEKS. In the above PEKS schemes, the search complexity is linearly related to the total number of ciphertexts. To accelerate search performance, a chain-like structure is described in [33]. However, the proposed chain cannot guarantee the semantic security. Bellare *et al.* [34] proposed deterministic PKE with rapid keyword search as if the keywords were not encrypted. Subsequently, Bellare *et al.* [35] and Boldyreva *et al.* [36] independently proposed two deterministic PKE schemes, which are secure in the standard model. Deterministic PEKS schemes are applicable when the keyword space has a high min-entropy. Without sacrificing semantic security, Xu *et al.* [3] proposed the first PEKS scheme with sublinear search complexity.

VI. Conclusion

When applying the previous works on PEKS in practice, all keyword-searchable ciphertexts of a file must visually connect to the file's ciphertext. This feature is convenient for deleting some intended searchable ciphertexts; however, it is contradictory to the semantic security of keywords. In other words, one can trivially distinguish that the searchable ciphertexts of the same file's ciphertext contain different keywords. To solve this problem, we define a new concept, called DSPE, to extend the traditional concept of PEKS. By contrast, this new concept allows (1) the generated searchable ciphertexts to only have hidden connections with their files' ciphertexts in practice, (2) all searchable ciphertexts to construct some hidden structures to rapidly find the matching ciphertexts if one knows a keyword-search trapdoor, and (3) some intended ciphertexts to be rapidly deleted if one knows a file-delete trapdoor. Finally, we construct a DSPE instance to realize our aim, prove that the instance is semantically secure under the CBDH assumption in the RO model, and demonstrate the practicability of the instance through experiments.

References

[1] Radicati Group, Cloud Business Email Market, 2017-2021, http://www.radicati.com/wp/wp-content/uploads/2017/06/Cloud-Business-Email-Market-2017-2021-Brochure.pdf, (2017)

[2] Boneh Dan, Crescenzo Giovanni Di, Ostrovsky Rafail, Persiano Giuseppe: Public Key Encryption with Keyword Search. In: Cachin C. and Camenisch J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506-522. Springer, Heidelberg (2004)

[3] Xu Peng, Wu Qianhong, Wang Wei, Susilo Willy., Domingo-Ferrer Joseph, Jin Hai: Generating Searchable Public-Key Ciphertexts with Hidden Structures for Fast Keyword Search. IEEE Transactions on Information Forensics and Security, 10(9), pp. 1993-2006 (2015)

[4] Curtmola Reza, Garay Juan, Kamara Seny, Ostrovsky Rafail: Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In: ACM CCS 2006, pp. 79-88. ACM (2006)

[5] Kamara Seny, Papamanthou Charalampos, Roeder Tom: Dynamic searchable symmetric encryption. In ACM Conference on Computer and Communications Security, pp. 965976 (2012)

[6] Boneh Dan, Franklin Matt: Identity-Based Encryption from the Weil Pairing. In: Kilian J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213-239. Springer, Heidelberg (2001)

[7] Abdalla Michel, Bellare Mihir, Catalano Dario, Kiltz Eike, Kohno Tadayoshi, Lange Tanja, Malone-Lee John, Neven Gregory, Paillier Pascal, Shi Haixia: Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. In: Shoup V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 205-222. Springer, Heidelberg (2005)

[8] Crescenzo Giovanni Di, Saraswat Vishal: Public Key Encryption with Searchable Keywords Based on Jacobi Symbols. In: Srinathan K., Rangan C.P. and Yung M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 282-296. Springer, Heidelberg (2007)

[9] Khader Dalia: Public key encryption with keyword search based on K-resilient IBE. In: Gervasi O. and Gavrilova M. L. (eds.) ICCSA 2007. LNCS, vol. 4707, pp. 1086-1095. Springer, Heidelberg (2007)

[10] Park Dong Jin, Kim Kihyun, Lee Pil Joong: Public Key Encryption with Conjunctive Field Keyword Search. In: Lim C.H. and Yung M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 73-86. Springer, Heidelberg (2004)

[11] Golle Philippe, Staddon Jessica, Waters Brent: Secure Conjunctive Keyword Search over Encrypted Data. In: Jakobsson M., Yung M. and Zhou J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31-45. Springer, Heidelberg (2004)

[12] Ballard Lucas, Kamara Seny, Monrose Fabian: Achieving Efficient Conjunctive Keyword Searches over Encrypted Data. In: Qing S. et al. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 414-426. Springer, Heidelberg (2005)

[13] Hwang Yong Ho, Lee Pil Joong: Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-user System. In: Takagi T., Okamoto Tatsuaki, Okamoto E. and Okamoto Takeshi (eds.) Pairing 2007. LNCS, vol. 4575, pp. 2-22. Springer, Heidelberg (2007)

[14] Takagi Tsuyoshi, Ryu Eun-Kyung: Efficient Conjunctive Keyword-Searchable Encryption. In: 21st International Conference on Advanced Information Networking and Applications Workshops, pp. 409-414. IEEE (2007)

[15] Baek Joonsang, Safavi-Naini Reihaneh, Susilo Willy: Public Key Encryption with Keyword Search Revisited. In: Gervasi O. (ed.) ICCSA 2008. LNCS, vol. 5072, pp. 1249-1259. Springer, Heidelberg (2008)

[16] Bethencourt John, Chan T-H.Hubert, Perrig Adrian, Shi Elaine, Song Dawn: Anonymous Multi-Attribute Encryption with Range Query and Conditional Decryption. Technical Report CMU-CS-06-135 (2006)

[17] Shi Elaine, Bethencourt John, Chan T-H.Hubert, Song Dawn, Perrig Adrian: Multi-Dimensional Range Query over Encrypted Data. In: IEEE S&P 2007, pp. 350-364. IEEE (2007)

[18] Boneh Dan, Waters Brent: Conjunctive, Subset, and Range Queries on Encrypted Data. In: Vadhan S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535-554. Springer, Heidelberg (2007)

[19] Davis Darren, Monrose Fabian, Reiter Michael K.: Time-Scoped Searching of Encrypted Audit Logs. In: Lopez J., Qing S. and Okamoto E. (eds.) ICICS 2004. LNCS, vol. 3269, pp. 532-545. Springer, Heidelberg (2004)

[20] Cheung David W., Mamoulis Nikos, Wong W.K., Yiu S.M., Zhang Ye: Anonymous Fuzzy Identity-based Encryption for Similarity Search. In: Cheong O., Chwa K.-Y and Park K. (eds.) ISAAC 2010. LNCS, vol. 6505, pp. 61-72. Springer, Heidelberg (2010)

[21] Tang Qiang, Chen Xiaofeng: Towards asymmetric searchable encryption with message recovery and flexible search authorization. In: Li N. and Tzeng W.G. (eds.) ASIACCS 2013, pp. 253-264, ACM (2013)

[22] Ibraimi Luan, Nikova Svetla, Hartel Pieter, Jonker Willem: Public-Key Encryption with Delegated Search. In: Lopez J. and Tsudik G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 532-549. Springer, Heidelberg (2011)

[23] Yang Gguomin, Tan Chik How, Huang Qiong, Wong Duncan S.: Probabilistic Public Key Encryption with Equality Test. In: Pieprzyk J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 119-131. Springer, Heidelberg (2010)

[24] Xu Peng, Jin Hai, Wu Qianhong, Wang Wei.: Public-Key Encryption with Fuzzy Keyword Search: A Provably Secure Scheme under Keyword Guessing Attack. IEEE Transactions on Computers, 62(11), pp. 2266-2277 (2013)

[25] Rhee Hyun Sook, Park Jong Hwan, Susilo Willy, Lee Dong Hoon: Improved searchable public key encryption with designated tester. In Safavi-Naini R. and Varadharajan V. (eds.) ASIACCS 2009, pp. 376-379, ACM (2009)

[26] Fang Liming, Susilo Willy, Ge Chunpeng, Wang Jiandong: A secure channel free public key encryption with keyword search scheme without random oracles. In: Garay J.A., Miyaji A. and Otsuka A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 248258. Springer, Heidelberg (2009)

[27] Emura Keita, Miyaji Atsuko, Rahman Mohammad Shahriar, Omote Kazumasa: Generic constructions of secure-channel free searchable encryption with adaptive security. Security and Communication Networks, 8(8), pp. 15471560 (2015)

[28] Byun Jin Wook, Rhee Hyun Suk, Park Hyun-A, Lee Dong Hoon: Offline keyword guessing attacks on recent keyword search schemes over encrypted data. In: Jonker W. and Petkovi M. (eds) SDM 2006. LNCS, vol. 4165, pp. 75-83. Springer, Heidelberg (2006)

[29] Jeong Ik Rae, Kwon Jeong Ok, Hong Dowon, Lee Dong Hoon: Constructing PEKS schemes secure against keyword guessing attacks is possible?. Computer Communications, 32(2), pp. 394-396 (2009)

[30] Chen Rongmao, Mu Yi, Yang Guomin, Guo Fuchun, Wang Xiaofen: Dual-Server Public-Key Encryption With Keyword Search for Secure Cloud Storage. IEEE Transactions on Information Forensics and Security, 11(4): 789-798 (2016)
[31] Chen Rongmao, Mu Yi, Yang Guomin, Guo Fuchun, Huang Xinyi, Wang Xiaofen., Wang Yongjun: Server-Aided Public Key Encryption With Keyword Search. IEEE Transactions on Information Forensics and Security, 11(12): 2833-2842 (2016)
[32] Sun Lixue, Xu Chun Xiang, Zhang Mingwu, Chen Kefei, Li Hongwei: Secure Searchable Public Key Encryption against Insider Keyword Guessing Attacks from Indistinguishability Obfuscation. Science China Information Science, doi: 10.1007/s11432-017-9124-0 (2017)
[33] Camenisch Jan, Kohlweiss Markulf, Rial Alfredo, Sheedy Caroline: Blind and Anonymous Identity-Based Encryption and Authorised Private Searches on Public Key Encrypted Data. In: Jarecki S. and Tsudik G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 196-214. Springer, Heidelberg (2009)
[34] Bellare Mihir, Boldyreva Alexandra, O'Neill Adam: Deterministic and Efficiently Searchable Encryption. In: Menezes A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 535-552. Springer, Heidelberg (2007)
[35] Bellare Mihir, Fischlin Marc Fischlin, O'Neill Adam, Ristenpart Thomas: Deterministic Encryption: Definitional Equivalences and Constructions without Random Oracles. In: Wagner D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 360-378. Springer, Heidelberg (2008)
[36] Boldyreva Alexandra, Fehr Serge, O'Neill Adam : On Notions of Security for Deterministic Encryption, and Efficient Constructions without Random Oracles. In: Wagner D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 335-359. Springer, Heidelberg (2008)