2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications

*i*PEKS: Fast and Secure Cloud Data Retrieval from the Public-key Encryption with Keyword Search

Fu-Kuo Tseng, Rong-Jaye Chen, and Bao-Shuh Paul Lin

Department of Computer Science
National Chiao-Tung University
Hsinchu, Taiwan
{*fktseng, rjchen*}*@cs.nctu.edu.tw, bplin@mail.nctu.edu.tw*

***Abstract*—In recent years, considerable concern has arisen over the security of the data stored in the cloud. A number of studies have suggested the use of cryptographic primitives to protect the data. As these tools transform the data into an unintelligible form, secure and efficient retrieval of the encrypted data from the cloud becomes a major challenge. The public-key encryption with keyword search (PEKS) scheme and many of its variants have been proposed to respond to this challenge. However, given a large number of data (or searchable keywords) would be tested sequentially in these PEKS schemes, previous search results should be employed to improve the efficiency of future searches. In this paper, we present an interactive construction named *i*PEKS where the search time is linear to the total number of distinct searched keywords instead of the total number of the searchable keywords. The more the keywords have been searched previously, the better the efficiency can be improved. We provide theoretical analysis to show the security and privacy. In addition, implementation and performance experiments exhibit a great improvement in efficiency compared with the previous schemes.**

***Keywords*-cloud data, public-key encryption with keyword search, interactive search, search efficiency, implementation**

I. Introduction

As an increasing amount of data generated and processed over the years, moving the data to the cloud environment becomes a more cost-effective choice on a pay-per-use basis [1], [2]. However, because cloud users no longer have physical control of their data, the security and privacy of storing and retrieving becomes a major concern before adopting this paradigm shift [3], [4]. In addition, traditional encryption schemes do not support direct searches on the encrypted data [5], [6], [7]; therefore, cloud users have to downloading and decrypting all the cloud data first to search for their interested data locally. There is a high demand for a secure and more efficient scheme to store and retrieve the selected data with the help of the cloud [8], [9].

The public-key encryption with keyword search (PEKS) scheme and many of its variants have been proposed to respond to this demand [10], [11], [12], [13]. The system works as follows. Alice and any other users can generate data with encrypted keywords (aka. searchable keywords) and send to Bob's cloud storage. Bob can generate a token (aka. predicate trapdoor) to ask for the data matching the trapdoor. These schemes hide the content of both searchable keyword and predicate trapdoor from the cloud. Furthermore, the hidden-vector encryption (HVE) [11], [12] schemes enable equality, range and subset predicates, and conjunction of these predicates. Finally, the inner-product encryption (IPE) [13] scheme further realizes the combination of conjunctions/disjunctions of simple boolean predicates.

All of these schemes carry out searching by testing the specified predicate trapdoor with *ALL* the searchable keywords of the user files in the cloud. Given a large number of stored searchable keywords would be tested sequentially, previous search results could be employed to improve the efficiency of future searches. In addition, to support conjunction/disjunction of richer predicates, the test functions of these schemes involve a large number of time-consuming operations of elliptic curves and the corresponding bilinear pairings. Therefore, we would like to devise a more efficient test mechanism to enable fast retrieval of user data.

Contribution. In this paper, we present an interactive construction of PEKS termed *i*PEKS where the search time is proportional to the total number of distinct searched searchable keywords instead of ALL the stored searchable keywords. The more the keywords have been searched previously, the better the efficiency can be improved. In addition, we break down a complicated search predicate into several simple search predicates. The complex searches can be obtained more efficiently by combining several simple searches from the cloud. We provide a thorough theoretical analysis to show the security and privacy of *i*PEKS, and conduct performance experiments to show its efficiency and practicality for both the cloud and the users.

The rest of the paper is organized as follows. Related works are described in Section 2, while problem formulation is elaborated in Section 3. The design of *i*PEKS and concrete examples are detailed in Section 4. The security and performance analysis is presented in Section 5. Finally, the conclusion and future work are provided in Section 6.

978-0-7695-5022-0/13 $26.00 © 2013 IEEE
DOI 10.1109/TrustCom.2013.57

II. Related Works

In this section, we provide a classification of three models for the privacy-preserving search: *Searchable Private-key Encryption*, *Private Information Retrieval (PIR)* and *Searchable Public-key Encryption*. The common goal of these schemes is to enable users to store and retrieve the data in the server while revealing as little information about the access as possible to the server. The major differences among these three models are explained as follows.

▷ **Searchable Private-key Encryption.** In this setting, the user *possesses* the data and can organize the data in any convenient way, including suitable data structures before encryption. Later, the user encrypts the data (and corresponding data structures) using his/her private key and stores them to the server. Only someone with this private key can efficiently access the encrypted data in the server. In addition, the access pattern (i.e. which data is actually demanded) can be hidden from the server [14], [15].

▷ **Private Information Retrieval (PIR).** In this setting, the user retrieves data from a server storing *unencrypted* data (like stock quotes or patents) without revealing the access pattern which is the privacy of the user. Because the data is unencrypted, any scheme trying to hide the access pattern must touch all the data; otherwise, the server can learn the information about which data is interesting or not interesting to the user. Therefore, a single-database PIR scheme requires work at least linear in the size of the dataset [16], [17], [18].

▷ **Searchable Public-key Encryption.** In this setting, the user/sender, encrypting the keywords (using public-key of the receiver), can be different from the receiver (having the corresponding private key). The receiver can generate corresponding token (aka. predicate trapdoor) using his/her private key to retrieve the interested data by the stored encrypted keywords (aka. searchable keywords). Boneh *et al.* first proposed one sort of practical application called email routing system by devising the *public-key encryption with keyword search (PEKS)* scheme. The searchable keywords of a mail can be produced by the recipient's public key. The recipient can retrieve intended emails by delegating the predicate trapdoors to the routing server. The server gathers and returns the mails to the recipient [10].

To extend search predicates, Park *et al.* proposed *public-key encryption with conjunctive keyword search (PECK)* [19] to enable the conjunctions of equality, while Boneh *et al.* [11] further provided a *hidden-vector scheme (HVE)* supporting the conjunctions of equality, subset and range predicates [12]. Later, Katz *et al.* proposed *inner-product encryption (IPE)* supporting the combination of conjunctions/disjunctions of simple boolean predicates [13]. In all of these constructions, the searching by the server is performed sequentially to find the matched searchable keywords and corresponding files. Therefore, the processing time is proportional to the number of stored searchable keywords.

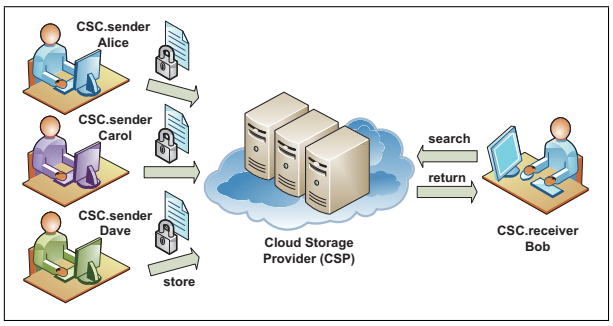


Figure 1. Cloud Storage Access Model

III. Problem Formulation

This section begins by defining the targeted system model including the system entities and related operations. Following that, the notations used throughout this paper are explained. Finally, we address the security/privacy threats to the search and define our design goals.

A. System Model

We consider a general cloud data storage architecture containing two system entities. (See Fig. 1)

1) *Cloud Storage Client (CSC)* stores a large number of data in the cloud. These data are either generated on his/her own or sent from other CSCs.
2) *Cloud Storage Provider (CSP)* provides search-based store/retrieval services for CSCs.

The CSC is further divided into two roles, the sender and receiver, based on their operations. The sender creates and sends encrypted data and searchable keywords (*aka*. PEKS ciphertexts). The CSP receives/stores the encrypted data together with PEKS ciphertexts. Later, the CSP performs search on receiving the predicate trapdoors from the receiver. The receiver generates trapdoors of intended predicate and sends it to the CSP to retrieve the matched data.

In particular, *i*PEKS have two novel approaches to accelerate the retrieval time of the CSCs (or the search time of the CSPs). First, the CSP caches previously-searched results to avoid the search on all the stored PEKS ciphertexts. Secondly, the CSC in *i*PEKS further utilizes the search history by breaking down one complex predicate into several simple predicates. The CSC can collect/combine the qualified file sets of these simple predicates from the CSP. Note that the retrieval of these sets can be fast given that most of these sets have been cached and maintained accordingly. The CSC combines these qualified file sets of the simple predicates to form the qualified file set of the original (complex) predicate. The CSC sends this qualified file set to the CSP and retrieves the intended files. By introducing caching and interaction approaches, *i*PEKS can greatly reduce the search complexity with reasonable storage/computatation overheads on CSPs when the CSC stores the file. The detailed construction is elaborated and discussed in Section IV.

Table I
NOTATIONS

Notation	Descriptions
$\mathcal{F}$	The file collection
$w_{i,j}$	The keyword of category i, with value j
f_{id}	The file with the file identifier id
$C_{f_{id}}$	The encrypted data with the file identifier id
$\mathcal{W}$	The universal keyword set of size n
$\mathcal{P}(\mathcal{W})$	The power set of the universal keyword set $\mathcal{W}$
$\mathcal{W}_{f_{id}}$	The keyword set specified for f_{id}
$\mathcal{F}_{w_{i,j}}$	The set of the file identifier containing $w_{i,j}$
$\mathcal{Q}$	The universal set for search predicates over $\mathcal{W}$
q	The demanded search predicate $q \in \mathcal{Q}$
$\mathcal{W}_q$	The keyword set of q converted to simple predicates
T_q	The set of trapdoors of simple predicates for q
$S_{f_{id}}$	The set of searchable keywords for file f_i
$\mathcal{QF}_q$	The qualified file sets for simple predicates in q
$\mathcal{FS}_q$	The qualified file sets for complex predicate q

For basic operations and keys, we follow the notations from the PEKS scheme by Boneh et al. [10] to describe our proposed *i*PEKS. Their scheme is described below:
(1) $\texttt{KeyGen}(s)$: On input a security parameter s and output a public/private key pair (A_{pub}, A_{priv}). (2) $\texttt{PEKS}(A_{pub}, w)$: On input A_{pub} and a specified keyword w, it outputs the searchable keywords s_w. (3) $\texttt{Trapdoor}(A_{priv}, w')$: On input A_{priv} and intended keyword w', it outputs the trapdoor $t_{w'}$. (4) $\texttt{Test}(A_{pub}, t_{w'}, s_w)$: On input $t_{w'}$ and s_w, it returns 'yes' if the keyword w' in $t_{w'}$ is the same as the w in s_w.

The receiver Bob runs $\texttt{KeyGen}(s)$ to obtain his public/private key pair (A_{pub}, A_{priv}). The sender Alice uses Bob's public key A_{pub} to generate one searchable keyword of keyword w for a file. Later, Bob uses his private key A_{priv} to generate the predicate trapdoor $t_{w'}$ for the intended keyword w'. The CSP tests each stored searchable keywords s_w with the received predicate trapdoor $t_{w'}$. If the test returns 'yes', send the corresponding file to Bob.

The file keywords can be extracted directly from the file content or specified by the file owner. On the one hand, we assume each receiver maintains its own keyword set $\mathcal{W}$ and keeps the file collection $\mathcal{F}$ in the cloud (See Table I). On the other, the keyword set $\mathcal{W}$ is composed of m keyword categories, each of which has n_i possible values. Thus, we have $n{=}\sum_{i=1}^{m} n_i$ possible keyword values to choose for a file. We will use the keyword and keyword values interchangeably throughout the paper. In Table II, $\mathcal{W}$ contains three keyword categories: w_1:(article) type, w_2:author and w_3:(published) year, thus m is 3. For the 'type' category, there are three possible values: $w_{1,1}$:[type:conference], $w_{1,2}$:[type:journal] and $w_{1,3}$:[type:tech. report], thus n_1 is 3. Similarly, n_2 is 3, while n_3 is 4. Therefore, the size of $\mathcal{W}$ is $n{=}\sum_{i=1}^{3} n_i{=}10$.

Table II
COMPLEX PREDICATES TO SETS OF SIMPLE PREDICATES

	Complex Predicates	Set of Simple Predicates
AND-1 (at least)	$q_1 = w_{2,3}w_{3,3}$	$\mathcal{W}_{q_1}{=}\{w_{2,3}, w_{3,3}\}$
AND-2 (exactly)	$q_2 = w_{2,3}w_{3,3}$	$\mathcal{W}_{q_2}{=}\{w_{2,3}, w_{3,3}, w_{2,1}, w_{2,2}\}$
OR	$q_3 = w_{1,1}w_{1,2}$	$\mathcal{W}_{q_3}{=}\{w_{1,1}, w_{1,2}\}$
RANGE	$q_4 = w_{3,1} \sim w_{3,3}$	$\mathcal{W}_{q_4}{=}\{w_{3,1}, w_{3,2}, w_{3,3}\}$

$\mathcal{W} = \{w_{1,1}, w_{1,2}, w_{1,3}, w_{2,1}, w_{2,2}, w_{2,3}, w_{3,1}, w_{3,2}, w_{3,3}, w_{3,4}\}$
· $w_{1,1}$:[type:conference]; $w_{1,2}$:[type:journal]; $w_{1,3}$:[type:tech. report]
· $w_{2,1}$:[athr:bplin]; $w_{2,2}$:[athr:rjchen]; $w_{2,3}$:[athr:fktseng]
· $w_{3,1}$:[yr:2010], $w_{3,2}$:[yr:2011], $w_{3,3}$:[yr:2012], $w_{3,4}$:[yr:2013]

Each file f in $\mathcal{F}$, denoted as f_{id}, is given a unique file identifier *id* to support file management through these identifiers. Similarly, the encrypted version of f_{id} is denoted as $C_{f_{id}}$. The encryption algorithms can be symmetric or asymmetric. For the file collection, $\mathcal{W}_{f_{id}}$ represents the keywords specified for f_{id}, while $\mathcal{F}_{w_{i,j}}$ denotes the set of the file identifiers where the keyword $w_{i,j}$ is specified for the corresponding files. In addition, a file can be specified one single value or multiple values from the keyword category w_i. In Table II, the keywords in w_1:type is mutually exclusive, only an article type can be specified for a file. Multiple keyword values in w_2:author is selected to express that multiple authors own this file. To better describe *i*PEKS, $\mathcal{F}_{w_i}{+}\mathcal{F}_{w_j}$ denotes the union of these two sets, while $\mathcal{F}_{w_i}{\cdot}\mathcal{F}_{w_j}$ (or shorthanded as $\mathcal{F}_{w_i}\mathcal{F}_{w_j}$), the intersection of the two. Finally, $\mathcal{F}_{w_i}{-}\mathcal{F}_{w_j}$ denotes the set difference.

For the search predicate, the universal set of all possible predicate is denoted as $\mathcal{Q}$. *i*PEKS supports the conjunction and disjunction of all possible keyword values $w_{i,j} \in \mathcal{W}$. Therefore, the size of $\mathcal{Q}$ is $2^{\mathcal{P}(\mathcal{W})}$, where $\mathcal{P}(\mathcal{W})$ is the power set of $\mathcal{W}$ and $|\mathcal{P}(\mathcal{W})| = 2^n$. In Table II, q_1 is to retrieve the files with keyword values $w_{2,3}$:[athr:fktseng] and $w_{3,3}$:[yr:2012], the $\mathcal{W}_{q_1}$ to the CSP is $\{w_{2,3}, w_{3,3}\}$ and the $\mathcal{QF}_{q_1}$ from the CSP is $\{\mathcal{F}_{w_{2,3}}, \mathcal{F}_{w_{3,3}}\}$. The predicate q_1 is to retrieve the files owned by *at least* the author 'fktseng' in '2012'. To have equivalent predicates, the CSC intersects the sets in $\mathcal{W}_{q_1}$ to form the qualified file set $\mathcal{FS}_{q_1}{=}\mathcal{F}_{w_{2,3}}{\cdot}\mathcal{F}_{w_{3,3}}$ and sends it to CSP to retrieve the intended files. Secondly, the predicate q_2 is to retrieve the files owned *solely* by 'fktseng' in '2012'. The CSC has the $\mathcal{W}_{q_2}{=}\{w_{2,3}, w_{3,3}, w_{2,1}, w_{2,2}\}$ to the CSP and obtains the $\mathcal{QF}_{q_2}{=}\{\mathcal{F}_{w_{i,j}} | w_{i,j}{\in}\mathcal{W}_{q_2}\}$ from it. The qualified file set $\mathcal{FS}_{q_2} = \mathcal{F}_{w_{2,3}}\mathcal{F}_{w_{3,3}}{-}\mathcal{F}_{w_{2,3}}\mathcal{F}_{w_{3,3}}\mathcal{F}_{w_{2,1}}{-}\mathcal{F}_{w_{2,3}}\mathcal{F}_{w_{3,3}}\mathcal{F}_{w_{2,2}}$. Next, the predicate q_3 is divided into two simple predicates and obtains $\mathcal{W}_{q_3}$ and $\mathcal{QF}_{q_3}$. The CSC unites all the sets in $\mathcal{QF}_{q_3}$ to have all the files either of conference or journal papers. Finally, the predicate q_4 contains three simple predicates to retrieve the files whose year is from 2010

to 2012. The qualified file set is obtained by uniting all the sets in $\mathcal{QF}_{q_4}$. The transformation between complex predicates and simples predicates are summarized in Table II. Finally, $S_{f_{id}}=\{s_{w_{i,j}}\}_{w_{i,j}\in\mathcal{W}_{id}}$ is the set of searchable keywords of file f_{id}, while $T_q=\{t_{w_{i,j}}\}_{w_{i,j}\in\mathcal{W}_q}$ denotes the set of trapdoors for simple predicates in q (See Table I).

B. Thread Model and Design Goal

The CSP is assumed to be honest-but-curious in *i*PEKS. The CSP follows the specified protocols, but may attempt to learn extra information from transactions. In face of malicious CSPs, the authentication mechanisms like those in [20], [21] can be employed to avoid unexpected tempering of the stored files and search results. The design goals are as follows: (1) *Search Privacy:* The searchable keywords reveal nothing about the content of the underlying keywords. Similarly, the search trapdoors leak no information about the corresponding predicates. The CSP can only use the search trapdoors to tell if the searchable keyword of a file satisfies the intended predicate in the predicate trapdoor. (2) *Search Efficiency*: The search should be carried out efficiently for CSPs to help users retrieve their cloud data in time.

IV. Proposed Scheme

Overview. We propose *i*PEKS, an interactive searchable public-key encryption consisting of two phases: the sending phase and the retrieval phase. The CSP maintains two lists of tuples: the cache list C-List and the unclassified list N-List. C-List comprises the tuple $\langle t_{w_{i,j}}, s_{w_{i,j}}, \mathcal{F}_{w_{i,j}}\rangle$, while N-List includes the tuple expressed as $\langle s_{w_{i,j}}, f\rangle$. Two lists are initially empty. In the sending phase, the sender prepares ciphertext C_f and searchable keywords S_f to the CSP. The CSP includes the file identifier f to $\mathcal{F}_{w_{i,j}}$ in the C-List if the $s_{w_{i,j}}$ has been searched; otherwise, the CSP includes the remaining $\{s_{w_{i,j}}\}$ to N-List. In the retrieval phase, when a simple predicate for $w_{i,j}$ is issued, the CSP searches the cache C-List first to obtain $\mathcal{F}_{w_{i,j}}$; otherwise, the CSP searches the unclassified list N-List to obtain $\mathcal{F}_{w_{i,j}}$ and includes this information to C-List. Note that the receiver breaks down the predicate q into simple predicates stored in $\mathcal{W}_q$. The CSP returns the qualified file sets $\mathcal{QF}_q=\{\mathcal{F}_{w_{i,j}}\}_{w_{i,j}\in W_q}$. The receiver builds $\mathcal{FS}_q$ according to q and sends $\mathcal{FS}_q$ to the CSP. The CSP returns the ciphertexts specified in $\mathcal{FS}_q$. The receiver decrypts $\{C_{f_{id}}\}_{id\in\mathcal{FS}_q}$ to obtain the plaintexts $\{f_{id}\}_{id\in\mathcal{FS}_q}$. The detailed description of these two phases is as follows:

▷ **Sending Phase:**

Step 1. The sender sets $C=(C_f, S_k)$, where C_{f_k} is the encrypted form of f and $S_f=\{s_{w_{i',j'}}=\texttt{PEKS}(A_{pub}, w_{i',j'})|\ w_{i',j'}\in\mathcal{W}_{f_k}\}$. The sender sends C to the CSP.

Step 2. The CSP sets as k the file identifier of the stored file C. For each $s_{w_{i',j'}}\in S_{f_k}$, the CSP executes $\texttt{Test}(A_{pub}, s_{w_{i',j'}}, t_{w_{i,j}})$ for each tuple $\langle t_{w_{i,j}}, s_{w_{i,j}}, \mathcal{F}_{w_{i,j}}\rangle$ in the C-List. If Test returns 'yes' for the tuple $\langle t_{w_{i,j}}, s_{w_{i,j}}, \mathcal{F}_{w_{i,j}}\rangle$, add the file identifier k into $\mathcal{F}_{w_{i,j}}$; otherwise, test the next tuple in the C-List. If Test fails for all tuples in the C-List, add the tuple $\langle f_k, s_{w_{i',j'}}\rangle$ to the N-List.

▷ **Retrieval Phase:**

Step 1. The receiver sets one search predicate $q\in\mathcal{Q}$ and W_q. The receiver sets $T_q=\{t_{w_{i',j'}}|t_{w_{i',j'}}=\texttt{Trapdoor}(A_{priv}, w_{i',j'})$ and $w_{i',j'}\in W_q\}$. The receiver sends T_q to the CSP.

Step 2. For each $t_{w_{i',j'}}\in T_q$, the CSP executes $\texttt{Test}(A_{pub}, s_{w_{i,j}}, t_{w_{i',j'}})$ for each tuple $\langle t_{w_{i,j}}, s_{w_{i,j}}, \mathcal{F}_{w_{i,j}}\rangle$ in the C-List. If Test returns 'yes' for the tuple $\langle t_{w_{i,j}}, s_{w_{i,j}}, \mathcal{F}_{w_{i,j}}\rangle$, add $\mathcal{F}_{w_{i',j'}}$ to the qualified file set $\mathcal{QF}_q$; otherwise, test the next tuple in the C-List. If Test fails for all tuples in the C-List, execute $\texttt{Test}(A_{pub}, s_{w_{i,j}}, t_{w_{i',j'}})$ for each tuple $\langle f, s_{w_{i,j}}\rangle$ in the N-List. $\mathcal{F}_{w_{i',j'}}$ collects the file identifier f with $\texttt{Test}(A_{pub}, s_{w_{i,j}}, t_{w_{i',j'}})$ returning 'yes' and adds $\langle t_{w_{i',j'}}, s_{w_{i,j}}, \mathcal{F}_{w_{i',j'}}\rangle$ in the C-List. The CSP returns the qualified file sets $\mathcal{QF}_q=\{\mathcal{F}_{w_{i',j'}}\}_{w_{i',j'}\in\mathcal{W}_q}$.

Step 3. The receiver builds one set of file identifiers for each of the terms in q. Then the receiver unites all these sets to obtain the targeted set of file identifiers $\mathcal{FS}_q$ and sends $\mathcal{FS}_q$ to the CSP. Note that $q=b_1\mathcal{F}_{w_{1,1}}+\cdots+b_n\mathcal{F}_{w_{m,n_m}}+b_{n+1}\cdot\mathcal{F}_{w_{1,1}}\mathcal{F}_{w_{1,2}}+\cdots+b_{2^{2^n}}\mathcal{F}_{w_{1,1}}\mathcal{F}_{w_{1,2}}\cdots\mathcal{F}_{w_{m,n_m}}$, $b_i\in\{\emptyset, U\}$. If $b_i=\emptyset$, the corresponding term is not included in q.

Step 4. The CSP returns the ciphertexts $\{C_{f_{id}}\}_{id\in\mathcal{FS}_q}$.

Step 5. The receiver decrypts the ciphertexts $\{C_{f_{id}}\}_{id\in\mathcal{FS}_q}$ to obtain the plaintexts $\{f_{id}\}_{id\in\mathcal{FS}_q}$. □

Example. Set $S_{f_1}=\{s_{w_{1,1}}, s_{w_{2,2}}, s_{w_{3,4}}\}$ and $S_{f_2}=\{s_{w_{1,3}}, s_{w_{2,4}}, s_{w_{3,4}}\}$. The C-List and N-list are initially empty. When a sender Alice stores S_{f_1} and S_{f_2} to the CSP for the receiver Bob, the CSP adds the tuple $\langle f_1, s_{w_{1,1}}\rangle$, $\langle f_1, s_{w_{2,2}}\rangle$ and $\langle f_1, s_{w_{3,4}}\rangle$ to the N-List because the C-List is empty. Also, the CSP stores $\langle f_2, s_{w_{1,3}}\rangle$, $\langle f_2, s_{w_{2,4}}\rangle$ and $\langle f_2, s_{w_{3,4}}\rangle$ to the N-List. Refer to Table II for the keyword set $\mathcal{W}$.

The receiver Bob would like to search with a predicate $q=(w_{2,1}+w_{2,2})\cdot w_{3,4}$ which means retrieving the files in 2013 containing at least 'bplin' and 'rjchen' as authors. Bob breaks down q into three simple predicate (stored in $\mathcal{W}_q$) and generates $T_q=\{t_{w_{2,1}}, t_{w_{2,2}}, t_{w_{3,4}}\}$ to the CSP. Because the C-List is empty, the CSP executes $\texttt{Test}(A_{pub}, s_{w_{i,j}}, t_{w_{i',j'}})$ for all $s_{w_{i,j}}\in N$-list and $t_{w_{i',j'}}\in T_q$. After finishing all the tests for q, the C-List have entries with $\mathcal{F}_{w_{i,j}}$, where $\mathcal{F}_{w_{1,1}}=\mathcal{F}_{w_{2,2}}=\{1\}$, $\mathcal{F}_{w_{1,3}}=\mathcal{F}_{w_{2,4}}=\{2\}$ and $\mathcal{F}_{w_{3,4}}=\{1,2\}$. The CSP inserts $\langle t_{w_{1,1}}, s_{w_{1,1}}, \mathcal{F}_{w_{1,1}}\rangle$, $\langle t_{w_{2,2}}, s_{w_{2,2}}, \mathcal{F}_{w_{2,2}}\rangle$, $\langle t_{w_{1,3}}, s_{w_{1,3}}, \mathcal{F}_{w_{1,3}}\rangle$, $\langle t_{w_{2,4}}, s_{w_{2,4}}, \mathcal{F}_{w_{2,4}}\rangle$ and $\langle t_{w_{3,4}}, s_{w_{3,4}}, \mathcal{F}_{w_{3,4}}\rangle$ to the C-List. The CSP returns $\mathcal{QF}_q=\{\mathcal{F}_{w_{2,1}}, \mathcal{F}_{w_{2,2}}, \mathcal{F}_{w_{3,4}}\}$ to Alice, where $\mathcal{F}_{w_{2,1}}=\emptyset$. Alice first builds $\mathcal{F}_{w_{3,4}}=\{1,2\}$ and $\mathcal{F}_{w_{2,1}}+\mathcal{F}_{w_{2,2}}=\{1\}$, and intersects these sets to obtain the targeted set of file identifiers $\mathcal{FS}_q=\{f_1\}=(\mathcal{F}_{w_{2,1}}+\mathcal{F}_{w_{2,2}})\mathcal{F}_{w_{3,4}}$. The CSP prepares and returns C_{f_1} to Alice. Alice decrypts $\{C_{f_1}\}$ to obtain $\{f_1\}$.

The sender Carol sends $S_{f_3}=\{s_{w_{1,1}}, s_{w_{2,3}}, s_{w_{3,4}}\}$ to the CSP intended to the receiver Alice. The CSP executes

$\texttt{Test}(A_{pub}, s_{w_{i',j'}}, t_{w_{i,j}})$ for all $s_{w_{i',j'}} \in S_{f_3}$ and $t_{w_{i,j}}$ in the tuples in the C-List. The $\texttt{Test}$ returns 'yes' for the tuple $\langle t_{w_{3,4}}, s_{w_{3,4}}, \mathcal{F}_{w_{3,4}} \rangle$, add the file identifier 3 into $\mathcal{F}_{w_{3,4}}$. The CSP stores $\langle f_3, s_{w_{1,1}} \rangle$ and $\langle f_3, s_{w_{2,3}} \rangle$ into the N-List.

The receiver Bob would like to search the CSP with predicate $q'=(w_{1,3}+w_{3,4})$. Bob breaks down q' into simple predicates and generates $T_{q'}=\{t_{w_{1,3}}, t_{w_{3,4}}\}$ to the CSP. The CSP executes $\texttt{Test}(A_{pub}, s_{w_{i,j}}, t_{w_{i',j'}})$ for all $s_{w_{i,j}} \in N$-list and $t_{w_{i',j'}} \in T_{q'}$. After finishing all the tests, $\mathcal{F}_{w_{1,3}}=\{2\}$ and $\mathcal{F}_{w_{3,4}}=\{1,3\}$. The CSP returns $\mathcal{QF}_{q'}=\{\mathcal{F}_{w_{1,3}}, \mathcal{F}_{w_{3,4}}\}$ to Alice. Alice unites these two sets to obtain the targeted set of file identifiers $\mathcal{FS}_{q'}=\{f_1, f_2, f_3\}=\mathcal{F}_{w_{1,3}}+\mathcal{F}_{w_{3,4}}$. The CSP prepares and returns C_{f_1}, C_{f_2} and C_{f_3} to Alice. Alice decrypts $\{C_{f_1}, C_{f_2}, C_{f_3}\}$ to obtain $\{f_1, f_2, f_3\}$.

V. Security Analysis and Performance Evaluation

In this section, we present the security and privacy of *i*PEKS by giving formal arguments. Next, we provide detailed description of a systematic attack possibly carried out by the CSP and provide one corresponding countermeasure to *i*PEKS. Moreover, we evaluate the storage, computation and communication costs for the receiver and CSP in *i*PEKS.

A. Security Analysis

On the one hand, the security of *i*PEKS is based on the security of the underlying PEKS [10]. The security of the PEKS relies on the *bilinear Diffie-Hellman problem* (BDHP). The BDHP is widely believed to be hard that no algorithm is now available to efficiently find the solution. Thus we assume BDHP is computationally intractable, that is, the malicious CSC or CSP could not tell the content of the searchable keyword and predicate trapdoor because these actions are involved in solving the hard problem. Formally, *i*PEKS is semantically secure against an adaptive chosen keyword attack assuming BDHP is hard [10].

On the other hand, the interactive retrieval of data in *i*PEKS may reveal some useful information for the CSP to find the intended search predicate. The CSP can use $\mathcal{FS}_q$ from the CSC to find the targeted q. Given $\mathcal{FS}_{q'} = b_1\mathcal{F}_{w_{1,1}}+\cdots+b_n\mathcal{F}_{w_{m,n_m}}+b_{n+1}\mathcal{F}_{w_{1,1}}\mathcal{F}_{w_{1,2}}+\cdots+b_{2^{2^n}}\mathcal{F}_{w_{1,1}}\mathcal{F}_{w_{1,2}}\cdots\mathcal{F}_{w_{m,n_m}}$, $b_k \in \{\emptyset, U\}$. $\mathcal{FS}_{q'}$ is the union of at most 2^m terms. Each term is one member of the power set of m set of file identifiers, $\mathcal{F}$. The number of possible search predicates $|\mathcal{Q}|=2^{\mathcal{P}(\mathcal{W})}$ and the CSP can try each of the possible search predicates. If the result for one q' is the same as $\mathcal{FS}_{q'}$, q' is one candidate of the targeted predicate q. The CSP has to test all possible q to obtain all the possible search predicates q'. This systematic attack on *i*PEKS by the CSP is described in Algorithm 1.

Once the candidate predicates, off-line keyword guessing can be used to find the underlying simple predicate $w_{i,j}$ from the searchable keyword $s_{w_{i,j}}$. We can specify more simple predicate trapdoors than what is contained in $\mathcal{W}_q$ and some of them are of no interest to the receiver. To make it infeasible to the CSP to do exhaust search on all the possible search predicates, if $|\mathcal{W}_q|$ is fewer than 7, randomly generate simple predicate trapdoors to make $|\mathcal{W}_q|$ at least 7. The number of possible predicates is $|\mathcal{Q}|=2^{2^7} = 2^{128}$, which is computation-infeasible for the CSP to find the candidate predicates set $\{q'\}$ for the targeted search predicate q.

Algorithm 1. systematic attack on *i*PEKS by the CSP

▷ Input: $\mathcal{FS}_q$, the set of matched file identifiers given q

▷ Output: Q_q, the set of search predicates q' where $\mathcal{FS}_{q'}=\mathcal{FS}_q$

for $j = 1$ to $2^{\mathcal{P}(\mathcal{W})}(= 2^{2^m})$ **do**

  Write j as binary of length 2^m and treat the bit k as b_k in q'_j.

  If $k = 0$, set b_k as $\emptyset$; otherwise, set b_k as U in q'_j.

  Construct $\mathcal{FS}_{q'_j}$ using $\{\langle t_{w_{i,j}}, s_{w_{i,j}}, \mathcal{F}_{w_{i,j}} \rangle\}$ in the C-List.

  If $\mathcal{FS}_q = \mathcal{FS}_{q'_j}$ **then**

    $Q_q = Q_q \cup \{q'_j\}$

  end of if

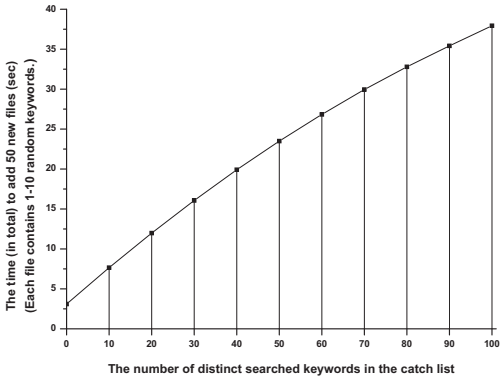
end of for

return Q_q

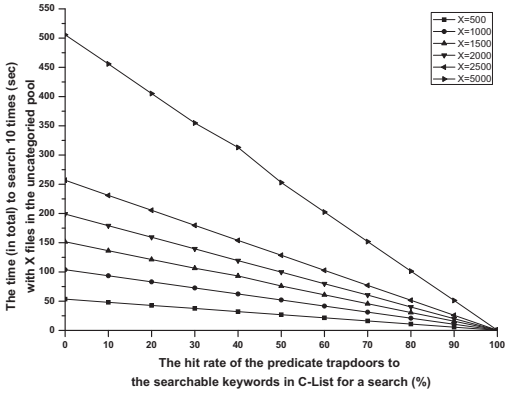
B. Computation and Communication Evaluation

We use the server with Intel Xeon processor E5620 at 2.40 GHz running Ubuntu 11.10. Elliptic curves are using GMP [22] and PBC [23] libraries. We use a supersingular curve over one base field of size 1024 bits and the embedding degree is 2. The size of one group element in G_1 is 2048 bits. The cost of scalar multiplication in G_1 is 2.24 ms, while that of the bilinear pairing is 1.8 ms. Hashing one string to a G_1 point takes 26.5 ms, while multiplication in G_2 needs 7.3 ms. The rest operations cost less than 1 ms.

When the sender stores files to the CSP, CSP has to update the C-List and N-List accordingly. Figure $2(a)$ presents the time in total to send 50 new files to the CSP when there are 0 to 100 distinct searched keywords in the C-List. These files contain 1 to 10 randomly-selected keywords. As the figure shows, when these are only small number of distinct searched keywords in the C-List, most of the searchable keywords are added to N-List and the time for testing through the C-List can be omitted. On the other hand, the more the distinct searched keywords in the C-List, the more time can be saved to maintain the C-List and N-List. The searchable keyword has higher probability to match a tuple early in the C-List, which offsets the length increase of the C-List. Figure $2(b)$ demonstrates the time in average to perform 10 searches with 500 to 5000 files in the N-List. Each file has unclassified searchable keywords ranging from 1 to 10. With the hit rate 0%, all the searchable keywords are in the N-List, which is equivalent to the original PEKS. With the increase of the hit rate, the search time is reduced

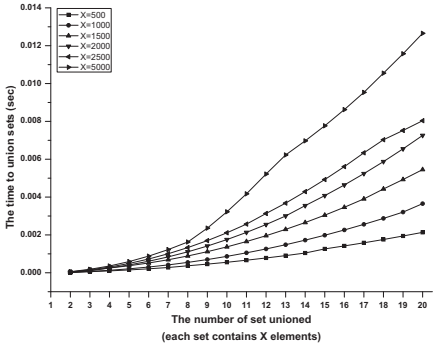


(a) Store files into cache line/ uncategoried pool by the CSP

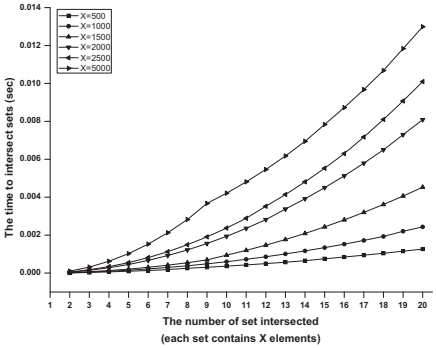


(b) Retrieve files from the cache line/ uncategoried pool by the CSP

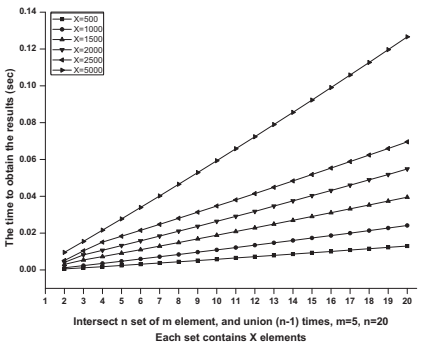
Figure 2. Store/Retrieve operation performed by the CSP



(a) Union of sets



(b) Intersection of sets



(c) Combination of Union/Intersection of sets

Figure 3. Set operation performed by the CSC.receiver

accordingly and affected slightly by the number of tuples in the N-List. If all the searchable keywords have been searched previously, the search time is 0.28 *s* in average because there is no need to search the N-List.

On the other hand, the receiver has to build one set of file identifiers for each of the terms in q. Then the receiver unites all these sets to obtain the targeted set of file identifiers $\mathcal{FS}_q$ and sends $\mathcal{FS}_q$ to the CSP. Figure 3(a) demonstrates the time to unite 2 to 20 sets whose size (that is, the number of file identifiers) is from 500 to 5000. The union of 20 sets (with 5000 file identifier each) is still efficient and can be carried out within 14 *ms*. The interaction of sets is also conducted in the same way as shown in Figure 3(b). The result is also as efficient. Finally, we assume the receiver issues predicate trapdoors requiring n set intersection of m element, and union $(n-1)$ times. The experiment sets m=5 and n=20, which means the predicate q involves 20 terms and each terms has 5 randomly-selected keyword sets. The construction of the set of the qualified file identifiers is very efficient (less than 1.4 *ms*) for the receiver (see Figure 3(c)).

As for the comparative work HVE [12], the computation time is related to all possible keyword values in $\mathcal{W}$. Assume the number of stored PEKS ciphertexts is X=1500, the number of keyword category is m=3, each of which has 5 values; therefore, the equivalent number of HVE ciphertexts is 100. When storing a file, the CSP in *i*PEKS spends extra 0.76 to 0.06 *s* to maintain the C-List and N-List given that the C-List is from empty to half-full (the corresponding hit rate is from $0\%\sim50\%$. In addition, the size of C-List and N-List is proportional to the number of contained entries. Each entry head costs about 512 B, the following file *id* is 20 bit each if there is one million files. The overall storage for C-List/N-List is fewer than 1 GB. The work for the CSC in *i*PEKS is corresponding to $\mathcal{W}_{f_{id}}$, while that in HVE is proportional to the number of HVE ciphertexts. If the file to be stored has 10 keywords, the CSC in *i*PEKS works around 2 times faster than that in HVE scheme to generate predicate trapdoors. As for testing, the work for the CSP in *i*PEKS is proportional to $\mathcal{W}_q$, while that in HVE is corresponding to the number of HVE ciphertexts. If the predicate contains 5 simple predicates and the hit rate is 80%-100%, the time for the CSP in *i*PEKS to obtain the $\mathcal{QF}_q$ is from 15.28 *s* down to 0.14 *s* and that in HVE is 71.72 *s* in average, which is 4-500 times slower. Therefore, *i*PEKS boost retrieval efficiency by introducing mild store/computation overheads in the CSP without increasing the work for the CSC.

VI. Conclusion

In this paper, we present an interactive construction of PEKS termed *i*PEKS where the search time is proportional to the size of C-List instead of *ALL* the stored searchable keywords. The more the keywords have been searched previously, the better the efficiency can be improved. In addition, the receiver can break down a complex search predicate into several simple predicates to speed up the retrieval. We provide the theoretical analysis to show its security and privacy, and conduct performance experiments to show its efficiency and practicality. The comparison between *i*PEKS and the HVE is also provided. For future work, we would like to exploit *i*PEKS to support richer predicates and evaluate its efficiency with the counterparts.

Acknowledgment

This research is supported by National Science Council, Taiwan under contract No. 101-2221-E-009-138-, and Delta Electronics, Inc. under contract No. 102C003.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing (draft)," *NIST special publication*, vol. 800, p. 145, 2011.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] N. Virvilis, S. Dritsas, and D. Gritzalis, "Secure cloud storage: Available infrastructures and architectures review and evaluation," in *Trust, Privacy and Security in Digital Business*, ser. LNCS. Springer, 2011, vol. 6863, pp. 74–85.

[4] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 1–11, 2011.

[5] Nist, "Fips pub 197: Announcing the advanced encryption standard (aes)," *NIST*, 2001.

[6] J. Jonsson and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," no. 3, February.

[7] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," *SIAM J. of Computing*, vol. 32, no. 3, pp. 586–615, 2003, extended abstract in Crypto'01.

[8] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1467 –1479, aug. 2012.

[9] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 451–459.

[10] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," vol. 3027, pp. 506–522, 2004.

[11] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, S. Vadhan, Ed. Springer Berlin / Heidelberg, 2007, vol. 4392, pp. 535–554.

[12] V. Iovino and G. Persiano, "Hidden-vector encryption with groups of prime order," in *Proceedings of the 2nd international conference on Pairing-Based Cryptography*, ser. Pairing'08. Berlin,Heidelberg: Springer-Verlag, 2008, pp. 75–88.

[13] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," *Advances in Cryptology–EUROCRYPT 2008*, pp. 146–162, 2008.

[14] R. Ostrovsky, "Efficient computation on oblivious rams," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, ser. STOC '90. New York, NY, USA: ACM, 1990, pp. 514–523.

[15] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," 1993.

[16] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: single database, computationally-private information retrieval," in *Foundations of Computer Science, 1997. Proceedings, 38th Annual Symposium on*, Oct, pp. 364–373.

[17] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, ser. STOC'04. New York, NY, USA: ACM, 2004, pp. 262–271.

[18] ——, "Cryptography from anonymity," in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 239–248.

[19] D. Park, K. Kim, and P. Lee, "Public key encryption with conjunctive field keyword search," in *Information Security Applications*, ser. Lecture Notes in Computer Science, C. Lim and M. Yung, Eds. Springer Berlin Heidelberg, 2005, vol. 3325, pp. 73–86.

[20] F.-K. Tseng, Y.-H. Liu, and R.-J. Chen, "Toward authenticated and complete query results from cloud storages," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, june 2012, pp. 1204 –1209.

[21] ——, "Ensuring correctness of range searches on encrypted cloud data," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, dec. 2012, pp. 570 –573.

[22] *GMP: The GNU Multiple Precision Arithmetic Library*, Free Software Foundation, Inc, 2006, available at http://gmplib.org/.

[23] B. Lynn, *PBC: Pairing-Based Cryptography Library*, 2008, available at http://crypto.stanford.edu/pbc/.