JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 40, 1115-1138 (2024)
DOI: 10.6688/JISE.202409_40(5).0012

A Partial-Filename Search Mechanism for Encrypted Filenames in a P2P Network*

SHIN-YAN CHIOU[1,2,3]
[1]*Department of Electrical Engineering, College of Engineering*
Chang Gung University
Taoyuan, 333 Taiwan
[2]*Department of Nuclear Medicine*
Linkou Chang Gung Memorial Hospital, Taoyuan, Taiwan
[3]*Department of Neurosurgery*
Keelung Chang Gung Memorial Hospital, Keelung, Taiwan
E-mail: ansel@mail.cgu.edu.tw

The efficiency and accuracy of data search, storage, confidentiality and security of P2P systems is a serious concern. Previous studies have proposed using a partial filename search function which allows the user to input a partial filename to search for associated filenames on the remote system. However, if the file name is encrypted for enhanced security, this search function is ineffective. This paper proposes a search mechanism for encrypted filenames. In the case of filename encryption, the proposed method can also achieve a partial filename search function while maintaining filename confidentiality. The proposed system was implemented on an Android smartphone to simulate encrypted filename search. To the best of our knowledge, this is the first work done on partial-filename search for encrypted filenames.

Keywords: encrypted filename search, partial filename search, P2P, security, confidentiality

1. INTRODUCTION

In recent years, P2P service applications have become more widespread. Users can search for P2P-based files from any location, at any time, and on any device. This raises significant user and file security issues, and maintaining file and filename confidentiality while offering convenient access to authorized users is an important problem.

P2P Systems: P2P systems can be divided into two types: structured and unstructured. Structured systems adopt a specific rule topology which provides better search efficiency at the cost of reduced fault tolerance. Unstructured systems use a random mesh topology, which is more fault tolerant, providing a higher search rate but with reduced efficiency. In his study of unstructured P2P systems, Doulkeridis *et al.*, [5] proposed a self-organizing P2P method to transform unstructured P2P networks into a super-peer architecture to improve search efficiency. In the study of structured P2P systems, Genesan *et al.* [11] and Zhao *et al.* [12] used a distributed hash table (DHT) to convert files into values and publish them to the responsible node, and to guide them to specific nodes during the search process to improve efficiency. Liu *et al.*, [13] combined a trust mechanism and Q-learning method (*i.e.*, SMITQ), which can not only improve P2P search efficiency, but can also be applied to Internet-based routing aware designs.

Received February 7, 2023; revised May 27, 2023; accepted July 27, 2023.
Communicated by Raylin Tso.
* This work is partially supported by the National Science and Technology Council under Grant NSTC 112-2221-E-182-007-MY2 and by the CGMH project under Grant BMRPB46.

Verifiable File Search on the P2P system: Verification of file search results is an important issue for P2P systems. These research efforts can be divided into two categories [2]: P2P storage auditing and encrypted keyword search. Of these categories: P2P storage auditing mechanisms [7, 9, 10] can ensure the integrity of outsourced data, and encrypted keyword search mechanisms [8, 21-31] provide protection for user privacy by encrypting data outsourced to the P2P. Chin *et al.* proposed a basic protocol to implement a verifiable P2P file search function [4] and further proposed a fully-fledged protocol [2]. Section 2 provides a detailed introduction to the complete protocol.

Query Authentication: Large-scale storage services on storage services can be unreliable and vulnerable to various internal and external threats. Chandrasekhar and Singhal [18] proposed a query authentication for P2P-based storage systems with multiple data sources based on multi-trapdoor hash functions, allowing clients to efficiently verify the authenticity and integrity of the retrieved data. In advance, Xu *et al.* [19] proposed an access-policy-preserving (APP) signature to provide both query authentication and access control. The APP signature is used to derive customized signatures for unauthorized users to achieve the zero-knowledge confidentiality.

Secure Deletion on P2P Storage: In addition to search efficiency [18, 20], secure processing of P2P files (including secure file encryption, search and deletion) is a critical consideration. Yu *et al.*, [6] proposed a valid protocol for secure deletion on P2P storage, using a key established on the user device and a P2P-based hash table to allow for the safe and efficient deletion of P2P-based files.

Partial Filename Queries on P2P system: Lee *et al.*, [1] proposed a P2P-based partial filename search method. In the context of structured P2P systems and DHT architectures, the partition of filename and the calculation of the filename key are used to be capable of search for P2P-based partial filenames (the complete method [1] is introduced in Section 2.1).

Multi-Keyword Search over Encrypted Data on P2P system: However, in the real world, user-submitted keywords may be akin to a synonym rather than an exact match. Existing search methods for encrypted filenames only support precise matches, and this lack of tolerance for synonym substitutes reduces search efficiency and accuracy. Therefore, Krishna *et al*., [3] proposed a synonym-based fuzzy multi-keyword ranked search method that sorts the keyword search results according to degree of relevance and is automatically corrected. This approach can be used to achieve privacy of P2P-based encrypted files.

Other methods related to file search: Zhao *et al.* [32] present a verifiable and privacy-preserving ranked multi-keyword search (VPS) scheme based on the difficulty of factorization on large integers. Ge *et al.* [33] designed an accumulative authentication tag (AAT) based on the symmetric-key cryptography, and proposed a new secure index composed by a search table ST based on the orthogonal list and a verification list VL containing AATs. Xu *et al.* [34] designed a multi-keyword verifiable searchable symmetric encryption scheme based on blockchain, which provides an efficient multi-keyword search and fair verification of search results. Chenam and Ali [35] proposed a concept called dmCLPAECKS (a designated cloud server-based multi-user certificateless authenticated encryption with conjunctive keyword search scheme), supporting conjunctive keyword search. The same document (or e-mail) is only encrypted once and can be retrieved by different recipients. Liu *et al.* [36] proposed a privacy preserving keyword search scheme with full verifiability and forward security. Their scheme provides Forward Secure Accumulative Authentica-

tion Tag (FSAAT) with incremental property. Zhao *et al.* [37] proposed a forward privacy multi-keyword search scheme based on the classic MRSE scheme. Their scheme makes the cloud cannot obtain the actual match results of the past query with the newly updated files by adding the well-chosen dummy elements to the original index and query vectors. Gan *et al.* [38] presented an efficient VSSE (verifiable Searchable symmetric encryption) scheme, building on OXT protocol (Cash *et al.*, CRYPTO 2013), for conjunctive keyword queries with sublinear search overhead. Their VSSE scheme is based on a privacy-preserving hash-based accumulator, by leveraging a well-established cryptographic primitive, symmetric hidden vector encryption (SHVE).

Proposed Partial-filename Search Mechanism for Encrypted Filenames on P2P System: Although the method proposed in [1] can perform partial filename searches for P2P data, and method [3] can provide synonym search of encrypted data, they are unable to perform partial filename search and search verification for P2P-encrypted filenames. This paper proposes a partial filename search mechanism for P2P-encrypted filenames which can perform partial filename search for encrypted filenames and files, and while still protecting user privacy and data authentication.

Overview of Results: This article proposes a method for searching for partial filenames of encrypted file and filenames. In the data upload phase, the file and associated filename are encrypted, ensuring privacy. In the data search phase, the file and filename are kept encrypted to ensure privacy during search execution. Finally, during the filename receiving phase, after the data owner receives the P2P server result, the data owner can calculate the correct filename from the received files using his/her private key, thus ensuring filename privacy, data confidentiality, and filename validation. It can be used in any centralized or decentralized P2P-based file storage, which is public, insecure, and need to be protected.

Paper Contribution: This paper proposes a novel, efficient and secure mechanism for searching remote encrypted filenames via partial-filename input while maintaining security properties: anonymity, filename privacy and confidentiality, resistance to asynchronous attacks and tracking attack, and filename privacy-preserved storage and search. Security analyses, formal proofs, and feature comparisons are conducted, and the results showed that our scheme is secure and has better features and performance. Finally, the proposed system was implemented on a personal computer and an android smartphone to simulate encrypted filename search in a P2P-based environment, which show that our scheme can be easily applied to a P2P-base storage system. To the best of our knowledge, this is the first work done on partial-filename search for encrypted filenames.

Paper structure: This paper is divided into six sections. Section two reviews the relevant literature to explain and analyze the referenced schemes. Section three describes the proposed system's contents and structures. Section four analyze the proposed of seven system requirements and seven security requirements. Section five describes an actual implementation using Android phone and section six draws a conclusion.

2. RELATED WORKS

This section provides an in-depth introduction and discussion on the partial filename search method proposed by Lee *et al.*, [1], and the verification method for remote filename search proposed by Chen *et al.* [2, 4].

Table 1. Notations.

Notation	Description
f	a filename
s	a query phrase
p	the length of filename
d	the dimension in the mapping function
n_r	the number of characters on the right of "*"
n_l	the number of characters on the left of "*"
$+^{(n)}$	n continuous '+' (*e.g.* $+^{(5)}$ means "+++++")
t_X	the extracted current time of X
T_{th_i}	the ith time threshold
r	the range in each dimension
a_i	the ith character
l	the max. limitation of filename length
a	the total amount of filename set
n_+	the total amount of '+' in query
IS	index sequence
H_s	set of index values
F	partial filename sequence
S	query phrase
Z	the total amount of filename and partial filename
ID_X	the ID of X
SID_X	the pseudo ID of X
$BSID_X$	the backup pseudo ID of X
$h()$	hash function
$HMAC()$	HMAC function
$E_K(\cdot)/D_K(\cdot)$	symmetric encryption/decryption using key K

2.1 Partial Filename Query

In 2012, Lee *et al.*, [1] proposed a method for partial filename search in P2P systems, including the File Publishing and Query Processing phases. The steps are described in detail below.

(1) File Publishing

When the data owner is in the file publishing stage, the filename is first divided into several fragments. When then calculate the Keys for these individual fragments, and finally upload the filename and these index values (*i.e.* the Keys) to the P2P system for storage. Table 1 defines the symbols and parameters used, and the steps are described in detail below:

Step 1 (Filename fragmentation): The data owner selects the length d, and divides the file name $f = (a_0, a_1, a_2, \ldots, a_{p-1})$ with length p into $p - d + 1$ segments $(a_i, a_{i+1}, \ldots, a_{i+d-1})$ with length d, $0 \le i \le p - d + 1$, where a_i represents the $i + 1$ character of the filename f.
Step 2 (Index values computation): Data owner computes index sequence $IS = \{IS_j\}$ using Eqs. (1) and (2),

$$f(a_i) = \begin{cases} h(a_i)r, \text{ if } a_i \neq '+', \ 0 \le i \le p-1 \\ \text{random value from 0 } to\ r\text{-1, if } a_i = '+' \end{cases}, \quad (1)$$

$$IS_j = (f(a_j), f(a_{j+1}), \ldots, f(a_{j+d-1})), 0 \le j \le p-d \quad (2)$$

and calculate the index value $Key(IS_j)$ via Eq. (3),

$$Key(IS_j) = \sum_{i=0}^{d-1} (f(a_{j+i}) \cdot r^i), 0 \le j \le p-d. \quad (3)$$

Step 3 (Index value upload): The data owner then uploads the filename and the $(p-d+1)$ index values $Key(IS_j)$ to the P2P server.
Step 4 (Table construction): The P2P server then creates a comparison table from the received filename and index value $Key(IS_j)$.

(2) Query Processing

In the search processing stage, the user first enters the query term, which is then divided into several segments using the sliding window partition method. He/she then analyzes and selects the most suitable segment, and calculates its individual index values, which are then uploaded to the P2P server. The P2P server then compares them using the database's comparison table before returning the result to the user. The steps are explained in detail as follows:

Step 1 (Filename segmentation): The user enters the query term with a length p, which is then divided using the sliding window partition method into $p-d+1$ segments with a length d, and the most suitable segment is used to execute Step 2.
Step 2 (Index value calculation): The user applies Eqs. (4) and (5) to calculate the query phrase QP,

$$m(s_i) = \begin{cases} h(s_i) \bmod r, & \text{if } s_i \ne '+' \\ -1, & \text{if } s_i = '+' \end{cases} \quad (4)$$

$$QP = m(s_0), m(s_1), \ldots, m(s_{d-1}) \quad (5)$$

and the index value $Key(QP)$ of the partial filename is calculated using Eq. (6).

$$Key(QP) = m(s_i) * r^0 + m(s_{i+1}) * r^1 + \ldots m(s_{d-1}) * r^{d-1} \quad (6)$$

(The symbol "+" in the query denotes the search for a single unknown word.)

Step 3 (Index value upload): The user uploads the partial filename's index value $Key(QP)$ to the P2P server.
Step 4 (Filename search): The P2P server compares the received index value $Key(QP)$ to the database comparison table to obtain the search result (one filename, multiple filenames, or no result).
Step 5 (Return results): The P2P server then returns the search results to the user.

2.2 Verifiable File Search (Basic Protocol)

Chen *et al.*, [4] proposed a basic protocol for implementing a verifiable P2P filename search function using a key to achieve filename verification.

This protocol assumes that the data owner, user and P2P server all know all the filenames. The protocol is executed in the following order: (1) Key generation; (2) Outsource; (3) Query; (4) Search; and (5) Verify. However, this protocol does not protect filename privacy. Table 1 defines the symbols and parameters used. The steps are described in detail as follows,

Step 1 (Key Generation): Data owner generates key K.
Step 2 (Outsource): The data owner then uses all possible filename f from a predetermined alphabet, applying Eq. (7) with a specified filename length limit l, to calculate the "existing filename set F_1" and the "non-existing filename set F_2",

$$\begin{cases} F_1 = (f, HMAC(f, K), \text{file content}) \\ F_2 = (f, HMAC(f, K), \text{null}) \end{cases}, \tag{7}$$

and then F_1 and F_2 are uploaded to the P2P server for storage in the comparison table, where $HMAC()$ is the message authentication code function.
Step 3 (Query): The user then transfers the filename f to be queried to the P2P server.
Step 4 (Search): The P2P server then compares the received filename f to the comparison table and returns a query result (f, $HMAC(f, K)$), file content) or (f, $HMAC(f, K)$, null) to the user.
Step 5 (Verify): The user obtains the search result (f, $HMAC(f, K)$, file content) or (f, $HMAC(f, K)$, null) from the P2P server, and uses the key K and the filename f to calculate $HMAC'(f, K)$, and then compares the received data $HMAC(f, K)$ to verify its integrity.

2.3 Verifiable File Search (Fully-Fledged Protocol)

Chen *et al.*, [2] further proposed a fully-fledged protocol for implementing verified P2P file search using two keys to achieve filename verifiability, filename privacy and user differentiation. In this protocol, the data owner has two keys K_1 and K_2, where K_1 is generated by the data owner, and K_2 is a key shared by the P2P server and the Data owner. The protocol flow proceeds through five steps: (1) Key Generation; (2) Outsource; (3) Query; (4) Search; and (5) Verify, and uses K_1 and K_2 to achieve filename verification and filename privacy. The steps are described in detail as follows:

Step 1 (Key Generation): The Data owner generates the keys K_1 and K_2.
Step 2 (Outsource): Then, the Data owner uses all possible filename f from a preset alphabe, based on Eq. (8) the specified filename limit l, to calculate "existing filename set F_1" and "non-existent filename set F_2". F_1 and F_2 are then uploaded to the P2P server for storage in the comparison table.

$$\begin{cases} F_1 = (HMAC(f, K_2), HMAC(HMAC(f, K_2), K_1), \text{file content}) \\ F_2 = (HMAC(f, K_2), HMAC(HMAC(f, K_2), K_1), \text{null}) \end{cases} \tag{8}$$

Step 3 (Query): The user then splits the filename f to be queried into all possible filename beginning with f_i, and calculates $HMAC(f_i, K_2)$ which is then sent to the P2P server for searching, where $i = 1, 2, \ldots, p + 2$.
Step 4 (Search): The P2P server then receives the $HMAC(f_i, K_2)$ from the database com-

parison table and returns the search result ($HMAC(f, K_2)$, $HMAC(HMAC(f, K_2), K_1)$, file content) or ($HMAC(f, K_2)$, $HMAC(HMAC(f, K_2), K_1)$, null) to the user.
Step 5 (Verify): The user then obtains the search result from the P2P server, and uses K_1 and the $HMAC(f_i, K_2)$ returned from the P2P server to calculate $HMAC'(HMAC(f, K_2), K_1)$, which is then compared with $HMAC(HMAC(f, K_2), K_1)$ from the P2P server to verify the integrity of the data.

3. PROPOSED SCHEME

This section introduces a mechanism of partial-filename search for encrypted filenames on P2P storage (PSEF), which can perform a partial filename search function for encrypted files, with all user data executed in the encrypted state, and achieve user privacy, data authentication, and filename privacy. This section first lists the requirements (including system requirements, attacker model, and security requirements), and then details the proposed scheme (including registration phase, initial phase, outsourcing phase and query phase).

3.1 Scheme Requirements

This section explains the system requirements and the detailed scheme of the proposed system.

3.1.1 System requirement

System requirements of the proposed system are described as Definition 1.

Definition 1 (System requirements): The proposed scheme should meet the following conditions.
(1) Only data owners are allowed to search their files.
(2) Only data owners can obtain complete filenames.
(3) Encrypted filenames are searchable.
(4) Partial filenames can be used for filename search.
(5) The symbol "+" can be used to search for single unknown words.
(6) The symbol "#" can be used to search for zero or one unknown words.
(7) The symbol "*" can be used to search for zero, one, or more than one unknown words.

3.1.2 Attacker model

In our scheme, any identity communicates with each other via an insecure public channel, offering adversaries opportunities to intercept. In the following, we present the assumptions of the attacker model.
(1) An adversary may eavesdrop on all communications between protocol actors over the public channel.
(2) An attacker can modify, delete, resend and reroute the eavesdropped message.
(3) An attacker can be a legitimate user.
(4) An attacker cannot be a legitimate Server.
(5) The attacker knows the protocol description, which means the protocol is public.

3.1.3 Security requirement

The security requirements of the proposed system are described as Definition 2.

Definition 2 (Security requirements): The proposed scheme should meet the following conditions.

(1) **Anonymity:** Aside from the server, the user's and identity should not be disclosed to anyone from eavesdropped information.
(2) **Filename privacy:** Attackers cannot disclose any filename or partial-filename information from eavesdropped information.
(3) **Filename confidentiality:** The Server (or an attacker) cannot disclose any filename or partial-filename information (from stolen database information).
(4) **Resistance to asynchronous attacks:** Attackers cannot block data transmissions, causing the server or users to be unable to synchronously update, and thus undermining the following authentication iteration.
(5) **Resistance to tracking attack:** Attackers cannot access information from the messages transmitted through the protocol to determine which users are involved a given communication session.
(6) **Filename privacy-preserved storage:** Filenames cannot be disclosed in the procedure that P2P server stores filenames.
(7) **Filename privacy-preserved search:** Filenames cannot be disclosed in the procedure that P2P server compares matching tables in database.

3.2 Proposed Scheme

The proposed scheme entails four phases: registration phase, initial phase, file-publishing phase, and query phase.

3.2.1 Registration phase

User U uses ID_U to register on the Server. From the first key field Key, the Server then selects a shared key K_{SU}, which is then transmitted through a secret channel to U. Finally, both sides calculate $SID_U \leftarrow h(ID_U, K_{SA}, 0)$ (Fig. 1). The secret channel can be a hypertext transfer protocol secure (https) protocol, a Short Message Service (SMS), a concealed Pin-Letters, or other methods.

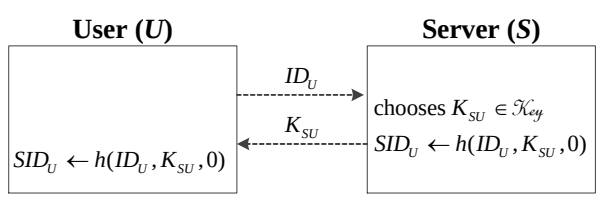

Fig. 1. Registration phase.

3.2.2 Initial phase

From the second key field Key 2, the user U then selects a symmetrically encrypted

private key K_U, using symmetric encryption methods such as AES [15] (or DES [16]).

3.2.3 Outsourcing phase

This phase entails four steps: filename segmentation, upload, storage and return, and identifier update (Fig. 2). Notations are shown in Table 1 and the algorithm is shown in Table 2. The steps are described in detail as follows.

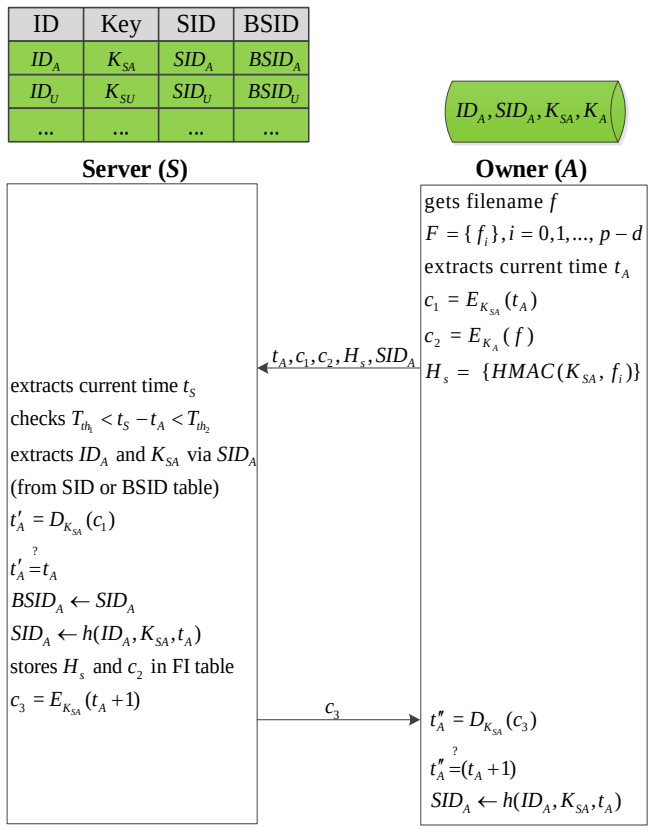

Fig. 2. Outsourcing phase.

Step 1 (filename segmentation): Data owner A enters a filename $f = (a_0, a_1, a_2, \ldots, a_{p-1})$ with a filename length p, selects a segmentation length d, and produces segmented filename sequence $F = \{f_i\}$ with a sequence length of $\max(1, p - d + 1)$ where $a \in \{a, b, \ldots, z\}$ indicating that the $(i + 1)$ component of f, the sequence length indicates the number of elements in the sequence, and f_i is calculated using Eq. (9) as follows,

$$f_i = \begin{cases} a_i a_{i+1} \ldots a_{i+d-1}, & i = 0,1,2,\ldots, p-d, \text{ if } p > d \\ f, & i = 0, \text{ if } p \le d \end{cases}. \tag{9}$$

Examples are shown in Table 3.

Step 2 (filename upload): A obtains the current t_A, and then calculates $c_1 = E_{K_{SA}}(t_A)$, $c_2 = E_{K_A}(f)$, and $H_s = \{HMAC(K_{SA}, f_i)\}$, and then uploads t_A, c_1, c_2, H_s and SID_A to P2P server S (Fig. 2).

Step 3 (storage and return): S obtains the current time t_S and verifies the establishment of $T_{th_1} < t_S - t_A < T_{th_2}$, and then uses SID_A to search for ID_A and K_{SA}. If SID_A cannot be found in the *SID* field, then it searches *BSID*. Next, calculate $t'_A = D_{K_{SA}}(c_1)$ and determine whether $t'_A \stackrel{?}{=} t_A$ is established. Then update $BSID_A \leftarrow SID_A$ and $SID_A \leftarrow h(ID_A, K_{SA}, t_A)$. Finally, H_s and c_2 are stored in the File Index table (Table 4), and $c_3 = E_{K_{SA}}(t_A + 1)$ is returned to the Data owner.

Table 2. Outsourcing algorithm.

Algorithm Outsourcing
Input:
  f: filename
  d: dimension
Procedure Publish(f, d)
1: Compute $E_{K_{SA}}(f)$
2: **if** (f.length >= d)
3: for (int i = 0; i <= f.length – d; i++)
4: f_i = substring (f, i, i+d–1)
5: Compute $HMAC(K_{SA}, f_i)$
6: Upload $E_{K_{SA}}(f)$, $\{HMAC(K_{SA}, f_i)\}$ and SID_A to P2P Server
7: **else** // f.length < d
8: Compute $HMAC(K_{SA}, f_0)$
9: Upload $E_{K_{SA}}(f)$, $HMAC(K_{SA}, f_i)$ and SID_A to P2P Server
10: end

Table 3. Example of filename partition in outsourcing phase.

Filename	Partition result (d = 4)
dog	dog
book	book
apple	appl, pple
message	mess, essa, ssag, sage

Table 4. Filename index table.

SID	Encrypted filename (c_2)	index (H_s)
SID_A	$E_{K_A}(f^1)$	$\{HMAC(K_{SA}, f_i^1)\}$
SID_A	$E_{K_A}(f^2)$	$\{HMAC(K_{SA}, f_i^2)\}$
⋮	⋮	⋮
SID_A	$E_{K_A}(f^n)$	$\{HMAC(K_{SA}, f_i^n)\}$

Step 4 (ID update): A calculates $t''_A = D_{K_{SA}}(c_2)$ and determines whether $t''_A \stackrel{?}{=} (t_A + 1)$ is established before updating $SID_A \leftarrow h(ID_A, K_{SA}, t_A)$.

3.2.4 Query phase

This phase includes two parts: **symbol transformation** and **detail steps**.

Part 1. Symbol transformation

Our scheme can use symbols ("*", "#" or "+") in unknown-word search to make search results more precise, where "+" means exactly one unknown word, "#" means zero or one un-known words, and "*" means zero, one, or more than one unknown words. When the user inputs query $S = (s_0, s_1, \ldots, s_{b-1})$ including any symbol, it triggers a symbol transformation.

A. Symbol ∗ transformation T_*

$S = (s_0, s_1, \ldots, s_{b-1})$ is transformed to $S^{\#}$ via "Symbol ∗ transformation" T_*:

$$\tau(s_i = '*' \mid d, n_r, n_l, S) = \begin{cases} "\#^{(d-n_r)}", \text{ if } n_l = 0 \\ "\#^{(d-n_l)}", \text{ if } n_r = 0 \\ "\#^{(d-1)}", \text{ if } n_l \neq 0 \,\&\, n_r \neq 0 \end{cases} \tag{10}$$

T_* uses d, n_r, n_l and $S = (s_0, s_1, \ldots, s_{b-1})$ to transform "∗" to one or more "#", where "$\#^{(a)}$" indicates continuous a #(*e.g.*, "$\#^{(3)}$" indicates "###"), n_r indicates the number of letters between c_{this} and c_{right}, and n_l indicates the number of letters between c_{this} and c_{left}, and c_{this} means the current character, c_{right} indicates the first symbol (*i.e.* '*', '#', or '+') on the right of c_{this} (if a symbol exists) or the final letter s_{b-1} (if it does not exist), and c_{left} indicates the first symbol (*i.e.* '*', '#', or '+') on the left of c_{this} (if a symbol exists) or the first letter s_0 (if it does not exist). The calculation of n_r and n_l is based on three scenarios depending on where "∗" appears in the term:

(1) *Prefix*: when '∗' appears in the first character of the search term (*i.e.*, s_0 = '∗'), calculate

$$n_r = \begin{cases} \min\{j\} \mid (s_j = '*', j > 0), \text{if } '*' \in (s_1, s_2, \ldots, s_{b-1}) \\ b-1, \text{ if } '*' \notin (s_1, s_2, \ldots, s_{b-1}) \end{cases},$$

and then '∗' is transformed to $d - n_r$ '#'. (Note: $n_l = 0$ in this scenario.)

(2) *Mid-term*: when '*' is in the middle of the search term (*i.e.*, s_i = '∗', $i \in [2, b-2]$, directly transform '*' as $d - 1$ '#'. (Note:

$$n_r = \begin{cases} \min\{j\} - i \mid (s_j = '*', j > i), \text{if } '*' \in (s_{i+1}, s_{i+2}, \ldots, s_{b-1}) \\ b - i - 1, \text{ if } '*' \notin (s_{i+1}, s_{i+2}, \ldots, s_{b-1}) \end{cases},$$

$$n_l = \begin{cases} i - \max\{j\} - 1 \mid (s_j = '*', j < i), \text{if } '*' \in (s_0, s_1, \ldots, s_{i-1}) \\ i, \text{if } '*' \notin (s_0, s_1, \ldots, s_{i-1}) \end{cases}.$$

(3) *Suffix*: when '∗' is at the end of the search term (*i.e.*, s_{b-1} = '∗'), calculate

$$n_l = \begin{cases} b - \max\{j\} - 2 \mid (s_j = '*', j < i), \text{if } '*' \in (s_0, s_1, \ldots, s_{b-2}) \\ b - 1, \text{if } '*' \notin (s_0, s_1, \ldots, s_{b-2}) \end{cases},$$

and then '*' is transformed to $d - n_l$ '#'. (Note: $n_r = 0$.)

B. Symbol # transformation $T_{\#}$

Then QC is transformed to QP_i^+ via symbol # transformation $T_{\#}$:

$$\tau(q_i = \text{'\#'}^{(n)} \mid QC) = \{\phi, +, ++, ..., +^{(n)}\}. \quad (11)$$

$T_{\#}$ uses $QC = (qc_0, qc_1, \ldots, qc_{d-1})$ to transform '#' to all possible '+'s.

C. Symbol + transformation T_+

Then $QP = (q_0, q_1, \ldots, q_{d-1})$ is transformed to *IND* via symbol + transformation T_+:

$$\tau(q_i = \text{'+'} \mid QP) = \{\phi, a, b, ..., y, z\}. \quad (12)$$

T_+ uses $QP = (q_0, q_1, \ldots, q_{d-1})$ to transform '+' to all possible letters.

Part 2. Detailed Steps

The process includes five steps: (1) Symbol processing and filename segmentation; (2) filename upload; (3) ID update; (4) message return; and (5) filename encryption (Fig. 3). The examples shown in Table 5 and the algorithm is shown in Table 6.

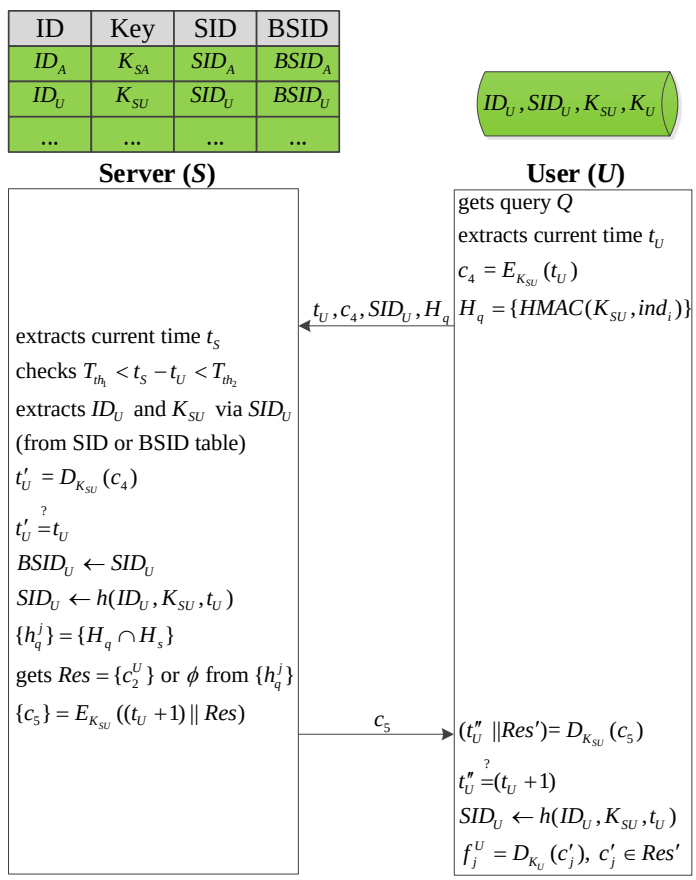

Fig. 3. Query phase.

Table 5. Example of file partition in Query phase ($d = 4$).

Query phrase (S)	Position of "*"	n_l	n_r	T_*	T_* result ($S^{\#}$)	$PS^{\#}$	QC	QP_i^+	T_+	T_+ result (IND)
mess+	N/A	–	–	N/A	mess+	mess, ess+	mess	mess	N/A	mess
+ook	N/A	–	–	N/A	+ook	+ook	+ook	+ook	$\tau(+, QP_1)$	aook, book, …, zook
*ge	s_0	0	2	$\tau(s_0\|4,2,0,S)$	##ge	##ge	##ge	ge ‖ +ge ‖ ++ge	$\tau(+, QP_1)$ $\tau(+, QP_2)$	ge ‖ age, bge, …, zge ‖ aage, abge, …, zzge
d*	s_1	1	0	$\tau(s_1\|4,0,1,S)$	d###	d###	d###	d ‖ d+ ‖ d++ ‖ d+++	$\tau(+, QP_1)$ $\tau(+, QP_2)$ $\tau(+, QP_3)$	d ‖ da, db, …, dz ‖ daa, dab, …, dzz ‖, daaa, daab, …, dzzz
a*s+	s_1	1	1	$\tau(s_1\|4,1,1,S)$	a###s+	a###, ###s ##s+	##s+	s+ ‖ +s+ ‖ ++s+	$\tau(+, QP_1)$ $\tau(+, QP_2)$ $\tau(+, QP_3)$	sa, sb, …, sz ‖ asa, asb, …, zsa ‖ aasa, aasb, zzsz
m*s+ge	s_1	1	1	$\tau(s_1\|4,1,1,S)$	m###s+ge	m###, ###s, ##s+, #s+g, s+ge	s+ge	s+ge	$\tau(+, QP_1)$	sage, sbge, …, szge
*s*ge	s_0	0	1	$\tau(s_0\|4,1,0,S)$	###s*ge,	###s, ##s#, #s##, s###, ###g, ##ge	##ge	ge ‖ +ge ‖ ++ge	$\tau(+, QP_1)$ $\tau(+, QP_2)$	ge ‖ age, bge, …, zge ‖ aage, abge, …, zzge
	s_2	1	2	$\tau(s_2\|4,2,1,S)$	*s###ge					
	s_0 & s_2	–	–	–	###s###ge					

Table 6. Search algorithm.

Algorithm Search
Input:
S: query
d: dimension
K_{SU}: key
Procedure Search(*S*, *d*)
1: for (int *i* = 0; *i* < *S.length*; *i*++)
2: if(char(*i*) = *)
3: *S* = '*'transfer
4: for (int *i* = 0 ; *i* < *S.length* ; *i*++)
5: if(char(*i*) = #)
6: *S* = '#' transfer
7: for (int *i* = 0; *i* < *S.length* ; *i*++)
8: if (char(*i*) = #)
9: *S* = '+' transfer
10: Compute {$HMAC(K_{SU}, S)$}
11: Send {$HMAC(K_{SU}, S)$} to P2P server
12: end

Step 1 (Symbol transformation and filename segmentation): user *U* inputs a search term $S = (s_0, s_1, \ldots, s_{b-1})$ with a length *b*. If *S* includes a symbol "*", "#" or "+", then first perform Eqs. (10)-(12) to execute symbol processing (examples shown in Table 5). The process includes the following steps:
(1) *Symbol* "*" *transformation* T_*: If *S* includes the symbol "*", then use Eq. (10) to transform *S* to $S^{\#}$. If *S* does not contain "*" or "#", then $S^{\#} = S$. (See Table 5 for example.)
(2) *Filename segmentation*: If the length of $S^{\#}$ exceeds *d*, use the sliding window partition method to segment *s* into $p - d + 1$ partial filename sequence $PS^{\#}$ with a length *d*. If the length of $S^{\#}$ is less than or equal to *d*, then $PS^{\#} = S^{\#}$.
(3) *Select search term*: If the number of the elements in $PS^{\#}$ is 1, then select the search term $QC = PS^{\#}$. If the element number in $PS^{\#}$ is greater than 1, then select the smallest number of components "#" as *QC*. If the number of "#" is the same, then select the component "#" on the left side or in the middle as *QC*.
(4) *Symbol* "#" *transformation* $T_{\#}$: If *QC* includes the symbol "#", then use Eq. (11) to transform *QC* to QP_i. If *QC* does not include "#", then $QP_i = QC$.
(5) *Symbol* "+" *transformation* T_+: If QP_i^+ includes the symbol "+", then use Eq. (12) to transform QP_i^+ to $IND = \{ind_i\}$. If QP_i^+ does not include "+", then $IND = QP_i^+$.
Step 2 (Filename upload): *U* obtains current time t_U, calculates $c_4 = E_{K_{SU}}(t_U)$, $H_q = \{HMAC(K_{SU}, ind_i)\}$, and uploads t_U, c_4, H_q and SID_U to P2P server *S*.
Step 3 (ID update): *S* obtains the current time t_S to verify the establishment of $T_{th_1} < t_S - t_U < T_{th_2}$, and then uses SID_U to search for ID_U and K_{SU}. If it can't be found in the *SID* field, then search *BSID*. Next, calculate $t'_U = D_{K_{SU}}(c_4)$ and determine $t'_U = t_U$. Once t'_U and t_U are confirmed to be identical, update $BSID_U \leftarrow SID_U$ and $SID_U \leftarrow h(ID_U, K_{SU}, t_U)$.
Step 4 (Result return): *S* then compares the received H_q to the comparison table H_s. If a suitable filename $\{c_2^U\}$ is found, then $Res = \{c_2^U\}$. Otherwise, *Res* = null. Next, transmit $c_5 = E_{K_{SU}}((t_U + 1) \| Res)$ to the user.
Step 5 (Filename encryption): *U* calculates $D_{K_{SU}}(c_5)$ to obtain t''_U and *Res*′. Determine $t''_U \stackrel{?}{=} (t_U + 1)$. Confirm t''_U and $(t_U + 1)$ are identical and then calculate $f^U = D_{K_U}(Res')$ and update $SID_U \leftarrow h(ID_U, K_{SU}, t_U)$.

4. COMPARISON AND SECURITY ANALYSIS

This section analyses and compares the properties and securities including the seven system requirements in Definition 1 and the seven security requirements in Definition 2. Table 7 summarizes the comparison of the properties and securities for the proposed method and those schemes proposed by Lee *et al.* [1], Chen *et al.* [2], Chen *et al.* [4] and Yu and Choi [6].

Table 7. Comparison of features and security properties.

	Lee *et al.* [1]	Chen *et al.* [2]	Chen *et al.* [4]	Choi [6]	Ours
(1-1)		√	√	√	√
(1-2)				√	√
(1-3)				√	√
(1-4)	√				√
(1-5)	√				√
(1-6)					√
(1-7)	√				√
(2-1)	√	√	√	√	√
(2-2)		√		√	√
(2-3)					√
(2-4)					√
(2-5)		√		√	√
(2-6)		√		√	√
(2-7)	√	√	√	√	√

4.1 Property Analysis

Our proposed system provides seven major properties.

(1) (1-1) Only data owners are allowed to search their files: In addition to the segment length d, each Data owner U can possess a different key K_U, thus identical filenames for files belonging to different users will calculate different index H_S, and only the Data owner using key K_U can search successfully.
(2) (1-2) Only data owners can obtain complete filenames: Since the complete filename is encrypted using key K_U, only the Data owner can use K_U to decrypt and obtain the complete filename.
(3) (1-3) Encrypted filenames are searchable: On the P2P server, the file index table can be used to search for encrypted filenames.
(4) (1-4) Partial filenames can be used for filename search: The user can search for a filename from a partial filename query using the outsourcing algorithm, symbol transformation, and search algorithm.
(5) (1-5) The symbol "+" can be used to search for single unknown words: The user can enter the character "+" which can be transformed using T_+ in Eq. (12) to search for exactly one unknown word.
(6) (1-6) The symbol "#" can be used to search for zero or one unknown words: The uses can enter the character "#", which can be transformed to "+" using $T_\#$ in Eq. (11) to search for zero or one unknown words.

(7) (1-7) The symbol "*" can be used to search for zero, one, or more than one unknown words: The user can input the character "*", which can be transformed to "#" using T_* in Eq. (10) to search for zero, one, or more than one unknown words.

4.2 Security Analysis

By using the proof concept [39, 40], we analyze the security of our protocols according to the requirements defined in Definition 2 as follows.

4.2.1 Anonymity

Because $SID_U^{(i)} = h(ID_U, K_{SA}, t_U^{(i-1)})$, an attacker may eavesdrop $SID_U^{(i)}$ and $t_U^{(i-1)}$ to try to evaluate ID_U. However, ID_U is not able to be evaluated because the hash function is irreversible, thus the proposed method achieves anonymity. Theorem 1 proves the property of anonymity from Definition 3.

Definition 3 (Partial hash problem): Let $a, b, c \in Z$ and $h_1 = h(a, b, c)$. If a can be evaluated from given c and h_1, then we say the *partial hash problem* is solved. (The probability of solving this problem is denoted as $\Pr(a \mid h_1, c) = \varepsilon_1$.)

Theorem 1 (Anonymity): In our scheme, if an attacker can evaluate ID_U from SID_U, then the *partial hash problem* can be solved.

Proof. In our scheme, assume an adversary tries to compute ID_U from two-round eavesdropped $SID_U^{(i)}$ and $t_U^{(i-1)}$, where $SID_U^{(i)}$ stands for the current-round SID_U, $t_U^{(i-1)}$ means the previous-round t_U, and $SID_U^{(i)} = h(ID_U, K_{SA}, t_U^{(i-1)})$. Let RO_1 be a random oracle: input $SID_U^{(i)}$ and $t_U^{(i-1)}$ to output ID_U (*i.e.* $RO_1(t_U^{(i-1)}, SID_U^{(i)}) \rightarrow ID_U$). In Definition 3, let $t_U^{(i-1)} \leftarrow c$ and $SID_U^{(i)} \leftarrow h_1$ be input parameters of RO_1 and obtain output ID_U. Let $a \leftarrow ID_U$ then a is evaluated. Therefore, $\Pr(ID_U \mid SID_U^{(i)}, t_U^{(i-1)}) \le \Pr(a \mid h_1, c) = \varepsilon_1$, which means the *partial hash problem* can be solved if RO_1 exists.

4.2.2 Filename privacy

Only $c_2 = E_{K_A}(f)$ can be eavesdropped in the outsourcing phase. Therefore, an attacker (or the P2P server) is not able to disclose the filename f without key K_A, thus the proposed method achieves filename privacy. Theorem 2 proves the property of filename privacy from Definition 4.

Definition 4 (Symmetric decryption problem): Let $x \in Z$ and $c = E_K(x)$ stands for a symmetric encryption (*e.g.* AES) of x using key K. If x can be evaluated from given c, then we say the *Symmetric decryption problem* is solved. (The probability of solving this problem is denoted as $\Pr(x \mid c) = \varepsilon_2$.)

Theorem 2 (Filename privacy): In our scheme, if an attacker can evaluate f from eavesdropped $c_2 = E_{K_A}(f)$, then the *Symmetric decryption problem* can be solved.

***Proof*:** In our scheme, assume an adversary tries to evaluate f from eavesdropped $c_2 = E_{K_A}(f)$. Let RO_2 be a random oracle: input c_2 to output f (*i.e.* $RO_2(c_2) \rightarrow f$.) In Definition 4, Let $c_2 \leftarrow c$ be input parameters of RO_2 and obtain output f. Let $x \leftarrow f$ then x is evaluated. Therefore, $\Pr(f \mid c_2) \leq \Pr(x \mid c) = \varepsilon_2$, which means the *Symmetric decryption problem* can be solved if RO_2 exists.

4.2.3 Resistance to synchronous attacks

In our scheme, both user and server have *SID* Table and *BSID* Table. If an attacker tries to block SID_A and makes the updates of SID_A out of sync., server can still use the *BSID* Table to identify user identities. Therefore, our scheme can achieve synchronized attack resistance.

4.2.4 Resistance to tracking attack

Only $SID_U^{(n)}$, $SID_V^{(m)}$, $t_U^{(n-1)}$ and $t_V^{(m-1)}$ can be eavesdropped from continuous three- or four-round query-phase procedure. Because $SID_A \leftarrow h(ID_A, t_A)$ is altered with t_A in each communication, Therefore, it is unable to identify whether $SID_U^{(n)}$ and $SID_V^{(m)}$ are the same user, thus the proposed method achieves resistance to tracking attack. Theorem 3 proves the property of resistance to tracking attack from Definition 4.

Definition 5 (Partial pre-hashed-message tracking problem): Let $a_1, a_2, b_1, b_2, c_1, c_2 \in Z$, $h_1 = h(a_1, b_1, c_1)$ and $h_2 = h(a_2, b_2, c_2)$. If $isEqual(a_1, a_2)$ can be evaluated from given h_1, h_2, c_1, and c_2, then we say the *partial pre-hashed-message tracking problem* is solved, where $c_1 \neq c_2$ and $isEqual(a_1, a_2)$ is 0 (if $a_1 \neq a_2$) or 1 (if $a_1 = a_2$). (The probability of solving this problem is denoted as $\Pr(isEqual(a_1, a_2) \mid h_1, h_2, c_1, c_2) = \varepsilon_3$).

Theorem 3 (Resistance to tracking attacks): In our scheme, if an attacker can evaluate the value of $isEqual(ID_U^{(n)}, ID_V^{(m)})$ from eavesdropped $SID_U^{(n)}$, $SID_V^{(m)}$, $t_U^{(n-1)}$ and $t_V^{(m-1)}$, then the *partial pre-hashed-message tracking problem* can be solved, where $SID_U^{(n)} / SID_V^{(m)}$ stands for the n/mth-round SID_U / SID_V, $t_U^{(n-1)} / t_V^{(m-1)}$ means the $(n-1)/(m-1)$th-round t_U/t_V, $SID_U^{(n)} = h(ID_U, K_{SU}, t_U^{(n-1)})$, $SID_V^{(m)} = h(ID_V, K_{SU}, t_V^{(n-1)})$, $isEqual(x, y)$ is 0 (if $x \neq y$) or 1 (if $x = y$), and $t_1 \neq t_2$.

***Proof*:** In our scheme, assume an adversary tries to track a user A from eavesdropped $SID_U^{(n)}$, $SID_V^{(m)}$, $t_U^{(n-1)}$ and $t_V^{(m-1)}$. Let RO_3 be a random oracle: Input $SID_U^{(n)}$, $SID_V^{(m)}$, $t_U^{(n-1)}$ and $t_V^{(m-1)}$ to output $isEqual(ID_U^{(n)}, ID_V^{(m)})$. (*i.e.* $RO_3(SID_U^{(n)}, SID_V^{(m)}, t_U^{(n-1)}$ and $t_V^{(m-1)}) \rightarrow isEqual(ID_U^{(n)}, ID_V^{(m)})$.) In Definition 5, let $SID_U^{(n)} \leftarrow h_1$, $SID_V^{(m)} \leftarrow h_2$, $t_U^{(n-1)} \leftarrow c_1$ and $t_V^{(m-1)} \leftarrow c_2$ be input parameters of RO_3 and obtain output $isEqual(ID_U^{(n)}, ID_V^{(m)})$. Let $isEqual(a_1, a_2) \leftarrow isEqual(ID_U^{(n)}, ID_V^{(m)})$, then $isEqual(a_1, a_2)$ is evaluated. Therefore, $\Pr(isEqual(ID_U^{(n)}, ID_V^{(m)}) \mid SID_U^{(n)}, SID_V^{(m)}, t_U^{(n-1)}, t_V^{(m-1)}) \leq \Pr(isEqual(a_1, a_2) \mid h_1, h_2, c_1, c_2) = \varepsilon_3$, which means the *partial pre-hashed-message tracking problem* can be solved if RO_3 exists.

4.2.5 Filename privacy-preserved search

Only $c_5 = E_{K_{SU}}((t_U + 1) \| Rec)$ and $H_q = \{HMAC(K_{SU}, ind_i)\}$ can be eavesdropped in the query phase. Therefore, an attacker (or the P2P server) is not able to disclose the query

result *Rec* or query information ind_i, thus the proposed method achieves filename privacy-preserved search. Theorem 4 proves the property of filename privacy-preserved search from Definition 6.

Definition 6 (Partial symmetric decryption problem): Let a, $x \in Z$ and $c = E_K(a \| x)$ stands for a symmetric encryption (*e.g.* AES) of $a \| x$ using key K. If x can be evaluated from given c, then we say the *Partial symmetric decryption problem* is solved. (The probability of solving this problem is denoted as $\Pr(x|c) = \varepsilon_4$.)

Theorem 4 (Filename privacy-preserved search): In our scheme, if attacker can evaluate *Rec* when server search filename $c_5 = E_{K_{SU}}((t_U + 1)\|Rec)$, then the *Partial symmetric decryption problem* can be solved.

***Proof*:** In our scheme, assume an adversary tries to evaluate *Rec* from eavesdropped $c_5 = E_{K_{SU}}((t_U + 1)\|Rec)$. Let RO_4 be a random oracle: input c_5 to output *Rec* (*i.e.* $RO_4(c_5) \rightarrow Rec$.) In Definition 6, Let $c_5 \leftarrow c$ be an input parameter of RO_4 and obtain output *Rec*. Let $x \leftarrow Rec$ then x is evaluated. Therefore, $\Pr(Rec \,|\, c_5) \le \Pr(x \,|\, c) = \varepsilon_4$, which means the *Partial symmetric decryption problem* can be solved if RO_4 exists.

4.2.6 Filename privacy-preserved storage

Only $c_2 = E_{K_A}(f)$ and $H_s = \{HMAC(K_{SA}, f_i)\}$ can be eavesdropped in the outsourcing phase. Therefore, an attacker is not able to disclose the information of filename f or partial filenames f_i, thus the proposed method achieves filename privacy-preserved storage. Theorem 5 proves the property of filename privacy-preserved storage from Definition 7.

Definition 7 (Joint HMAC and symmetric decryption problem): Let $x \in Z$, $c = E_{K_1}(x)$ stands for a symmetric encryption (*e.g.* AES) of q-length x using key K_1, and $H = \{HMAC(K_2, y_i)\}$ means the set of *HMAC* value of K_2 and y_i, where $x = (b_0, b_1, \ldots, b_{q-1})$ and $y_i = (b_i, b_{i+1}, \ldots, b_{i+d-1})$, $i = 0, 1, 2, \ldots, q - d'$. If y_j can be evaluated from given d', c and H, then we say the *Joint HMAC and symmetric decryption problem* is solved, where $j = 0, 1, 2, \ldots, q - d'$. (The probability of solving this problem is denoted as $\Pr(y_j \,|\, d', c, H) = \varepsilon_5$).

Theorem 5 (Filename privacy-preserved storage): In our scheme, if attacker can evaluate f_i from eavesdropped $c_2 = E_{K_A}(f)$, and $H_s = \{HMAC(K_{SA}, f_i)\}$, then the *Joint HMAC and symmetric decryption problem* can be solved.

***Proof*:** In our scheme, assume an adversary tries to evaluate f_i from $c_2 = E_{K_A}(f)$, d and $H_s = \{HMAC(K_{SA}, f_i)\}$. Let RO_5 be a random oracle: input d, c_2 and H_s to output f_i (*i.e.* $RO_5(d, c_2, H_s) \rightarrow f_i$.) In Definition 7, Let $d \leftarrow d'$, $c_2 \leftarrow c$ and $H_s \leftarrow H$ be input parameters of RO_5 and obtain output f_i. Let $y_j \leftarrow f_i$ then y_j is evaluated. Therefore, $\Pr(f_i \,|\, d, c_2, H_s) \le \Pr(y_i \,|\, d', c, H) = \varepsilon_5$, which means the **Joint HMAC and symmetric decryption problem** can be solved if RO_5 exists.

4.2.7 Filename confidentiality

Only $c_2 = E_{K_A}(f)$, and $H_s = \{HMAC(K_{SA}, f_i)\}$can be obtained from database. Therefore,

the Server (or an attacker) is not able to disclose the information of filename f or partial filenames f_i from database, thus the proposed method achieves filename confidentiality. Theorem 6 proves the property of filename confidentiality from Definition 7.

Theorem 6 (Filename confidentiality): In our scheme, if attacker can evaluate f_i from database-stolen $c_2 = E_{K_A}(f)$, and $H_s = \{HMAC(K_{SA}, f_i)\}$, then the **Joint HMAC and decryption problem** can be solved.

***Proof*:** In our scheme, assume an adversary tries to evaluate f_i from $c_2 = E_{K_A}(f)$, d and $H_s = \{HMAC(K_{SA}, f_i)\}$. Let RO_5 be a random oracle: input d, c_2 and H_s to output f_i (*i.e.* $RO_5(d, c_2, H_s) \rightarrow f_i$). In Definition 7, Let $d \leftarrow d'$, $c_2 \leftarrow c$ and $H_s \leftarrow H$ be input parameters of RO_5 and obtain output f_i. Let $y_j \leftarrow f_i$ then y_j is evaluated. Therefore, $\Pr(f_i \mid d, c_2, H_s) \le \Pr(y_j \mid d, c_2, H) = \varepsilon_5$, which means the **Joint HMAC and symmetric decryption problem** can be solved if RO_5 exists.

Table 8. Comparison of computation cost in outsourcing phase.

	[1]	[2]	[4]	[6]	Ours
No. of Key	$p-d+1$	2	1	2	2
No. of filename partition	$p-d+1$	0	0	0	
Hash	$d(p-d+1)$	0	0	3	0
HMAC	0	$2*\sum_{i=1}^{l} C_i^a * i!$	$\sum_{i=1}^{l} C_i^a * i!$	1	$p-d+1$
Multiplication	$d(p-d+1)+d-1$	0	0	0	0
Addition	$(d-1)(p-d+1)$	0	0	0	0
Encryption	0	0	0	1	1

Table 9. Comparison of computation cost in query phase.

	[1]	[2]	[4]	[6]	Ours
No. of Key	$p-d+1$	2	1	1	1
No. of filename partition	$p-d+1$	0	0	0	$p-d+1$
hash	$d(p-d+1)$	0	0	1	0
HMAC	0	$2(p+2)$	$p+2$	1	$p-d+1$
Multiplication	$d(p-d+1)+d-1$	0	0	0	0
Addition	$(d-1)(p-d+1)$	0	0	0	0

Table 10. Comparison of server pairing cost in outsourcing phase.

Conditions	[1]	[2]	[4]	[6]	Ours
Without Symbol	$Z*(p-d+1)$	$Z*(n+2)$	$Z*(n+2)$	Z	$Z*(p-d+1)$
With Symbol	27^{n_+}	N/A	N/A	N/A	27^{n_+}

Table 11. Average search time for 500-record database (d = 4).

	n(+)	n(#)	n(*)	Position	Condition	Example	Time
1	0	0	0	N/A	N/A	computer	485
2	1	0	0	back	$n_l = 3$	com+	568
3	1	0	0	back	$n_l = 2$	co+	572
4	0	1	0	back	$n_l = 3$	com#	581
5	0	1	0	back	$n_l = 2$	co#	586
6	0	0	1	front	$n_r = 3$	*ter	578
7	0	0	1	front	$n_r = 2$	*er	2699
8	0	0	1	back	$n_l = 3$	com*	594
9	0	0	1	back	$n_l = 2$	co*	2702
10	0	0	1	middle	$n_r = n_l = 3$	com*ter	597
11	0	0	1	middle	$n_r = n_l = 2$	co*er	2717
12	1	0	1	front & back	$n_r^* = 3, n_l^+ = 3$	*ute+	601

$n(x)$: number of symbol x; Time unit: millisecond (ms)

4.3 Computation Cost Analysis

In this section, we analyze the computation and communication performance of our proposed method from four aspects: computation cost in outsourcing phase (Table 8), computation cost in query phase (Table 9), Server pairing cost in filename-search phase (Table 10), and communication times (Table 11), where l is the maximum limitation of filename length and a stand for the total amount of filename sets F_1 and F_2 in Eq. (7).

5. IMPLEMENTATION

This section presents the implementation of the proposed scheme. We use one personal computer and one android phone to implement a P2P server and a personal user respectively. The personal computer implementation used Windows 10 with an Intel (R) core (TM) i7-7700HQ CPU @ 2.80GHz and 16G RAM. Android phone implementation used HTC U11 based on Android 7.11 and Qualcomm S835 2.45GHz. They communicate to each other through wireless networks such as 3G, 4G or WiFi. Moreover, the hash function used is SHA-512 [14], the symmetric encryption algorithm is DES [16]. Table 11 shows the average search time of ten queries for 500-record database in each condition, with the network connection upload speed 15.62Mbps, and the download speed 92.7Mbps.

In our implementation, we assume that the system time of the server and the Android smartphone are synchronized. However, the system times on the server and the Android smartphone are difficult to be synchronized. Fortunately, the experience in implementation shows that the system time difference between the server and the phones is within milliseconds. By assuming the maximum system time difference between the server and phones is 1000 milliseconds and the value $t_S - t_M$ is between 10 ms and 30 ms, we suggest to set T_{th_1} and T_{th_2} to –990 ms and 1030 ms, respectively. (In real situation, the value $t_S - t_M$ is suggested to be measured again for much accuracy.) The application flowchart is shown in Fig. 4.

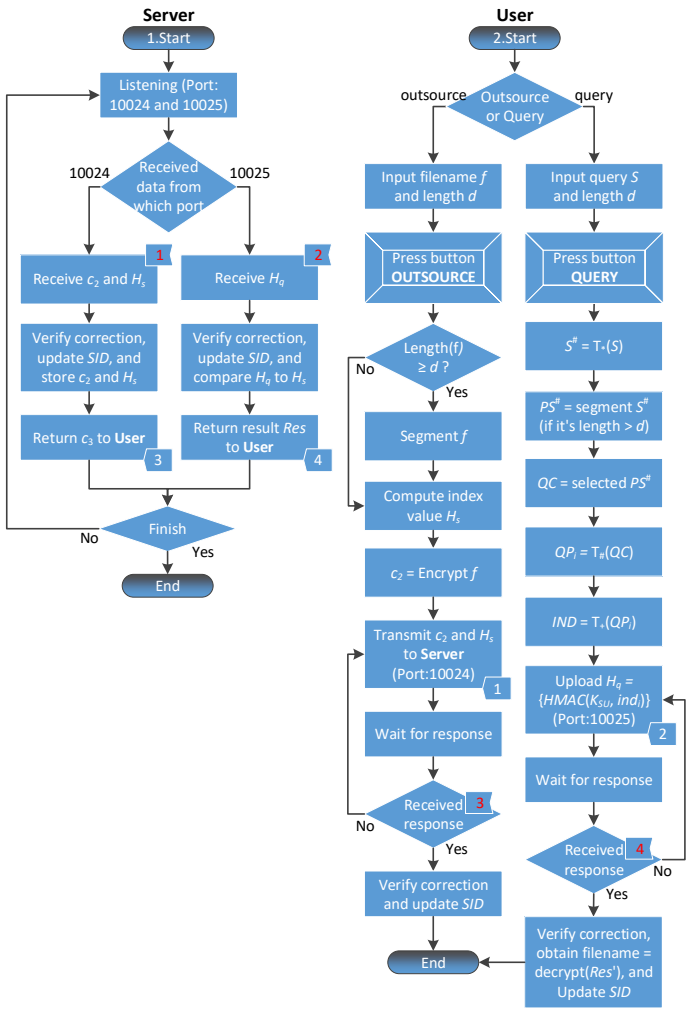

Fig. 4. Application flowchart.

6. CONCLUSION

P2P-based computing is becoming more popular, with users increasingly storing and backing up files on P2P networks. In addition to security consideration for filename encryption, users are also concerned with searching for files using partial filenames. The present study proposes a P2P-based search function in which an encrypted filename can be searched for successfully using a partial filename while simultaneously providing convenience and security. The proposed protocol provides seven types of functionality and seven types of security. The proposed system was implemented in a mobile device to demonstrate feasibility. Future work will seek to provide private partial filenames search in multi languages, and improve search performance to further facilitate P2P-based applications.

REFERENCES

1. G. Lee, S. L. Peng, Y. C. Chen, and J. S. Huang, "An efficient search mechanism for supporting partial filename queries in structured peer-to-peer overlay," *Peer-to-Peer Networking and Applications*, Vol. 5, 2012, pp. 340-349.
2. F. Chen, T. Xiang, X. Fu, and W. Yu, "User differentiated verifiable file search on the cloud," *IEEE Transactions on Services Computing*, Vol. 11, 2018, pp. 948-961.
3. C. R. Krishna and S. A. Mittal, "Privacy preserving synonym based fuzzy multi-keyword ranked search over encrypted cloud data," in *Proceedings of International Conference on Computing, Communication and Automation*, 2016, pp. 1187-1194.
4. F. Chen, T. Xiang, X. Fu, and W. Yu, "Towards verifiable file search on the cloud," in *Proceedings of IEEE Conference on Communications and Network Security*, 2014, pp. 346-354.
5. C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis,"Peer-to-peer similarity search over widely distributed document collections," in *Proceedings of ACM Workshop on Large-Scale Distributed Systems for Information Retrieval*, 2008, pp. 35-42.
6. J. W. Yu and H. K. Choi, "Efficient protocol for searchable encryption and secure deletion on cloud storages," in *Proceedings of IEEE International Conference on Consumer Electronics*, 2017, pp. 444-447.
7. A. Juels and B. S. Kaliski Jr, "PORs: Proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 584-597.
8. N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, 2014, pp. 222-233.
9. B. Wang, B. Li, and H. Li, "Panda: Public auditing for shared data with efficient user revocation in the cloud," *IEEE Transactions on Services Computing*, Vol. 8, 2015, pp. 92-106.
10. H. Wang, "Identity-based distributed provable data possession in multicloud storage," *IEEE Transactions on Services Computing*, Vol. 8, 2015, pp. 328-340.
11. P. Ganesan, Q. Sun, and H. Garcia-Molina, "Yappers: A peer-to-peer lookup service over arbitrary topology," in *Proceedings of the 22nd Annual Joint Conference of IEEE Computer and Communications*, 2002, pp. 1250-1260.
12. B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, Vol. 22, 2004, pp. 41-53.
13. H. L. Liu, G. X. Chen, Y. Chen, and Q. B. Chen, "A trust-based P2P resource search method integrating with Q-learning for future Internet," *Peer-to-Peer Networking and Applications*, Vol. 8, 2015, pp. 532-542.
14. P. Gallagher and A. Director, "Secure Hash Standard (SHS)," *FIPS PUB*, Vol. 180, 1995, p. 183.
15. J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*, ISC, Vol. 196, Springer Berlin, Heidelberg, 2002.
16. W. Diffie and M. Hellman, "Exhaustive cryptanalysis of the NBS data encryption standard," *IEEE Computer Society Technical Committee on Security and Privacy*, Vol. 76, 1997, pp. 72-84.

17. N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, Vol. 48, 1987, pp. 203-209.
18. S. Chandrasekhar and M. Singhal, "Efficient and scalable query authentication for cloud-based storage systems with multiple data sources," *IEEE Transactions on Services Computing*, Vol. 10, 2017, pp. 520-533.
19. C. Xu, J. Xu, H. Hu, and M. H. Au, "When query authentication meets fine-grained access control: A zero-knowledge approach," in *Proceedings of International Conference on Management of Data*, 2018, pp. 147-162.
20. Y. Zhang, G. Cui, S. Deng, F. Chen, Y. Wang, and Q. He, "Efficient query of quality correlation for service composition," *IEEE Transactions on Services Computing*, Vol. 14, 2018, pp. 695-709.
21. X. Ge, J. Yu, C. Hu, H. Zhang, and R. Hao, "Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *IEEE Access*, Vol. 6, 2018, pp. 45725-45739.
22. T. Peng, Y. Lin, X. Yao, and W. Zhang, "An efficient ranked multi-keyword search for multiple data owners over encrypted cloud data," *IEEE Access*, Vol. 6, 2018, pp. 21924-21933.
23. D. Wu, Q. Gan, and X. Wang, "Verifiable public key encryption with keyword search based on homomorphic encryption in multi-user setting," *IEEE Access*, Vol. 6, 2018, pp. 42445-42453.
24. L. Zhang, Y. Zhang, and H. Ma, "Privacy-preserving and dynamic multi-attribute conjunctive keyword search over encrypted cloud data," *IEEE Access*, Vol. 6, 2018, pp. 34214-34225.
25. Q. Zhang, Q. Liu, and G. Wang, "PRMS: A personalized mobile search over encrypted outsourced data," *IEEE Access*, Vol. 6, 2018, pp. 31541-31552.
26. Y. Zhang, R. H. Deng, J. Shu, K. Yang, and D. Zheng, "TKSE: Trustworthy keyword search over encrypted data with two-side verifiability via blockchain," *IEEE Access*, Vol. 6, 2018, pp. 31077-31087.
27. C. Guo, R. Zhuang, C.-C. Chang, and Q. Yuan, "Dynamic multi-keyword ranked search based on bloom filter over encrypted cloud data," *IEEE Access*, Vol. 7, 2019, pp. 35826-35837.
28. C. Hu, P. Liu, R. Yang, and Y. Xu, "Public-key encryption with keyword search via obfuscation," *IEEE Access*, Vol. 7, 2019, pp. 37394-37405.
29. C. Hu, X. Song, P. Liu, Y. Xin, Y. Xu, Y. Duan, and R. Hao, "Forward secure conjunctive-keyword searchable encryption," *IEEE Access*, Vol. 7, 2019, pp. 35035-35048.
30. J. Sun, L. Ren, S. Wang, and X. Yao, "Multi-keyword searchable and data verifiable attribute-based encryption scheme for cloud storage," *IEEE Access*, Vol. 7, 2019, pp. 66655-66667.
31. Y. Wu, J. Hou, J. Liu, W. Zhou, and S. Yao, "Novel multi-keyword search on encrypted data in the cloud," *IEEE Access*, 2019, Vol. 7, pp. 31984-31996.
32. M. Zhao, L. G. Liu, Y. Ding, Y. Wang, H. Liang, S. Tang, B. Wen, and W. Liang, "Verifiable and privacy-preserving ranked multi-keyword search over outsourced data in clouds," in *Proceedings of IEEE 15th International Conference on Big Data Science and Engineering*, 2021, pp. 95-102.

33. X. Ge, J. Yu, Jia H. Zhang, C. Hu, Z. Li, Z. Qin, and R. Hao, "Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification," *IEEE Transactions on Dependable and Secure Computing*, Vol. 18, 2019, pp. 490-504.
34. W. Xu, J. Zhang, Y. Yuan, X. Wang, Y. Liu, and M. I. Khalid, "Towards efficient verifiable multi-keyword search over encrypted data based on blockchain," *PeerJ Computer Science*, Vol. 8, 2022, p. e930.
35. V. B. Chenam and S. T. Ali, "A designated cloud server-based multi-user certificateless public key authenticated encryption with conjunctive keyword search against IKGA," *Computer Standards & Interfaces*, Vol. 81, 2022, p. 103603.
36. Y. Liu, J. Yu, M. Yang, W. Hou, and H. Wang, "Towards fully verifiable forward secure privacy preserving keyword search for IoT outsourced data," *Future Generation Computer Systems*, Vol. 128, 2022, pp. 178-191.
37. S. Zhao, H. Zhang, X. Zhang, W. Li, F. Gao, and Q. Wen, "Forward privacy multi-keyword ranked search over encrypted database," *International Journal of Intelligent Systems*, Vol. 37, 2022, pp. 7356-7378.
38. Q. Gan, J. K. Liu, X. Wang, X. Yuan, S. F. Sun, D. Huang, C. Zuo, and J. Wang, "Verifiable searchable symmetric encryption for conjunctive keyword queries in cloud storage," *Frontiers of Computer Science*, Vol. 16, 2022, pp. 1-19.
39. M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM Conference on Computer and Communications security*, 1993, pp. 62-73.
40. S.-Y. Chiou and Z.-Y. Liao, "A real-time, automated and privacy-preserving mobile emergency-medical-service network for informing the closest rescuer to rapidly support mobile-emergency-call victims," *IEEE Access*, Vol. 6, 2018, pp. 35787-35800.

Shin-Yan Chiou (邱錫彥) received the Ph.D. degree in Electrical Engineering from National Cheng Kung University, Taiwan, in 2004. From 2004 to 2009, he worked at Industrial Technology Research Institute as an RD Engineer. Since 2009, he joined the faculty of the Department of Electrical Engineering, Chang Gung University, Taoyuan, Taiwan, where he is currently a Professor. He has published a number of journals and conference papers in the areas of information security, social network security and mobile security. His research interests include information security, cryptography, social network security, and secure applications between mobile devices.