applied sciences

MDPI

Article

Research on Dynamic Searchable Encryption Method Based on Bloom Filter

Ziqi Jin [1], Dongmei Li [1,*], Xiaomei Zhang [1] and Zhi Cai [2]

[1] School of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai 201620, China; jzq552desh@163.com (Z.J.); xmzhang@sues.edu.cn (X.Z.)
[2] Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China; caiz@bjut.edu.cn
* Correspondence: dongmeilee@126.com

Abstract: Data outsourcing has become more and more popular due to its low cost and flexibility. However, there is a problem that the cloud server used to store data is partially trusted. Searchable encryption is an efficient technology that is devoted to helping people conduct accurate searches without leaking information. Nonetheless, most existing schemes cannot support dynamic updates or meet the privacy requirements of all users. There have been some experiments to solve these issues by implementing a dynamically searchable asymmetric encryption scheme. This paper proposes an efficient searchable encryption scheme based on the Authenticator Bloom Filter (ABF). The solution can support dynamic updates and multiple users and meet forward and backward security. This paper uses an ABF to improve the efficiency of searches and updates while playing a significant role in dynamic updates. This paper designs a new token encryption scheme and file set encryption scheme, which not only helps users reduce time in searches and updates but also supports multi-user modes. Experiments show that the proposed scheme takes less time in searching and updating algorithms, especially when the keyword does not exist. The solution also takes into account the problem of history storage when updating, which reduces the unnecessary consumption of memory and avoids multiple storage states for the same file.

Keywords: forwardprivacy; backward privacy; asymmetric searchable encryption; multi-user; Bloom filter

Citation: Jin, Z.; Li, D.; Zhang, X.; Cai, Z. Research on Dynamic Searchable Encryption Method Based on Bloom Filter. *Appl. Sci.* **2024**, *14*, 3379. https://doi.org/10.3390/app14083379

Academic Editor: Luis Javier Garcia Villalba

Received: 29 February 2024
Revised: 11 April 2024
Accepted: 12 April 2024
Published: 17 April 2024

1. Introduction

In the age of big data, both individuals and businesses need to store large amounts of data. The identification information, preferences, and habits generated by users when using various applications are stored and analyzed. To protect the privacy of users, cloud servers fade in people's sight. As cloud servers are semi-trusted, unencrypted information being stored in a server can be insecure in two ways: In the first case, some malicious users will access the server. These malicious users will copy the information from the server, which will cause the user's information to be compromised. In the second scenario, the cloud server is honest but curious. In Chai and Gong [1], the definition of an honest but curious server in the paper is: (1) storing outsourced data without modifying it; (2) honestly performing all operations such as searching and returning text data separately; and (3) attempting to learn the users' initial data.

An honest but curious adversary is also defined as a legitimate server that will try to find out all the useful information from the obtained content but will not deviate from the set protocol in the communication channel mentioned by Paverd et al. [2]. As a result, users need to encrypt their important information and store it on cloud servers; otherwise, their information security will be at risk. For example, threat actors broke into Amazon's web servers and caused a breach of the sensitive information of 3.7 million users. The stolen data were then posted on various hacking forums for sale. In the same time frame, the FlexBooker cloud server was also compromised and the personal data of up to 19 million

users were leaked. The investigation found that the company was using AWS S3 storage buckets to store data but had not implemented any security measures. It is therefore essential that data in cloud servers are kept encrypted.

While encrypting the data keeps them from being compromised, it also prevents the cloud server from being able to manipulate the data. The reality is that users do not want to download data and process them again; they want to be able to add, delete, change, and check their encrypted data directly on the cloud server. As a result, the concept of symmetric searchable encryption (SSE) has been introduced and investigated. SSE enables the execution of keyword searches in ciphertext, one of the most basic data operations [3].

Users encrypt their private data and outsource them to a semi-trusted server, after which they send a search token to the server to perform a keyword search without revealing sensitive information [4–8]. Searchable encryption is divided into symmetric searchable encryption and asymmetric searchable encryption. In earlier research, symmetric searchable encryption was mainly studied in the static case. However, the static case is not applicable to practical work. Dynamic searchable encryption implements dynamic updates on the basis of the former. However, in order to improve the efficiency of the search, each of these options allows some information to be divulged within certain limits. The file injection attack was confirmed by Zhang et al. [9]. This attack is performed by injecting a relatively small number of files to learn a large portion of the keywords searched by the client. To resist this attack, forward security has received attention.

Bost et al. [10] proposed a definition of forward security for searchable encryption and proposed a scheme for a type of DSSE based on forward security. Later, Bost et al. [11] proposed a definition of backward security and gave several schemes. He et al. [12] propose a searchable solution that satisfies backward and forward security. However, this scheme is only applicable to individual users for searching their own data stored in cloud servers and is not applicable to practical applications.

As most practical application environments are not closed, the implementation of symmetric searchable encryption always falls short of the requirements. We therefore introduce asymmetric searchable encryption.

Our main contributions are summarized as follows.

(1) In this paper, a new multi-user dynamic searchable scheme is proposed on the basis of the predecessors. A new validation Bloom filter structure (ABF) based on the existing Bloom filter is proposed. The new ABF not only includes the original features of the Bloom filter but also adds a counter module, which makes the solution easy to implement in dynamic updates and greatly reduces the error rate.
(2) In this paper, a new file set encryption scheme is designed, which uses a lightweight algorithm to reduce the overhead of initialization and update. At the same time, the ABF and state op of encrypted files are used to realize dynamic update of data.
(3) The scheme satisfies forward and backward safety. Forward security is satisfied by updating the search token, and backward security is realized by a new file set encryption scheme. Compared with other schemes, the scheme in this paper not only has a great advantage in time cost but also fully considers the problem of historical storage, avoiding multiple different storage states for the same file.

2. Related Work

Asymmetric encryption utilizes a pair of keys, known as the public key and private key. The public key is made publicly available and is used for encrypting data, while the private key is kept secret and is used for decrypting data. For asymmetric searchable encryption, it was first proposed by again Boneh et al. [13] in their article. But this scheme requires a secure communication channel to pass the trapdoor. However, establishing a secure communication channel is very difficult and expensive. Therefore, Beak et al. devised a scheme that does not require a secure communication channel [14]. Tang and Chen et al. [15] designed a PKI-based asymmetric searchable encryption scheme. It improves on the flaw that an attacker can obtain the relationship between the trapdoor and the ciphertext, as

proposed by Baek et al. Park et al. [16] propose two structures for link keyword search with public key encryption. However, this solution involves many connections between the user and the server and has a large storage overhead.

Guo et al. [17] analyze the scheme of Li et al. [18] and demonstrate that its trapdoor indistinguishability is not satisfied. A security scheme that satisfies the requirements of the test-specified server and provides stronger security guarantees for the confidentiality of keywords is also proposed. However, these two schemes do not address the encryption algorithm for files and only enhance the security of keywords, without much advantage in terms of practical application.

A spatial keyword query satisfying forward–backward security was proposed by Wang et al. [19]. The article uses Hilbert curves to simplify geometric range queries to range queries and uses prefix encoding to cover range queries. This solution allows other users to search for data, but this solution is not designed to update steps regarding data that already exist on the cloud server. Although this solution proposes a change to the bitmap when updating, the part of the update algorithm and storage file involved is not further described. Chen et al. [20] proposed a blockchain-based public-key searchable encryption scheme in the paper. The scheme, BSPEFB, not only makes use of smart contracts for searching, which can ensure the correctness and immutability of the returned results, but also satisfies backward and forward security. The solution reduces the number of computationally intensive operations and has a high search efficiency. However, each trapdoor in the scheme corresponds to a separate keyword, which causes a huge inconvenience to the data owner each time the data user requests a search token while giving the data owner an idea of the range of keywords the data user is interested in. If a malicious data owner uses this for analysis, it could easily compromise the data user's privacy.

3. Preliminaries

3.1. Forward and Backward Privacy

The definition of forward–backward security was first proposed by Stefanov et al. [21] with the first scheme to support dynamic keywords. Then, forward–backward security was first formalized by Bost et al. [10,11]. As for the definition of forward security, Bost et al. argue that an update does not reveal information about the updated keywords. Importantly, the server does not know if the updated file matches the keyword that the user searched for. The definition is as follows:

Definition 1. *The update leak function L^{Updt} can be written as:*

$$L^{Updt}(op, in) = L^{'}(op, \{ind_i, \mu_i\}) \tag{1}$$

For (op, in) pairs, op is the update query and in is the input. $\{ind_i, \mu_i\}$ is a collection of update files, in which μ_i is the key of the update and ind is the updated document. If the update function can be written as the above expression, then the L adaptive secure SSE scheme is forward private.

Bost et al.'s [10] definition here extends forward privacy as proposed by Stefanov et al. [21]. Also, this definition focuses only on adding documents, not updating them.

In backward security, the server cannot associate the currently searched keyword with the results of a previous search. That is, every time a user adds a document ind corresponding to a keyword to the database, then it is removed later [21]. After this series of operations, when searching this keyword, the search result will not appear in the document ind, so the SSE scheme is backward safe. Bost et al. defined three types of backward security in their paper: backward privacy with insertion pattern, backward privacy with update pattern, and weak backward privacy. In this paper, backward security is judged according to the second one: backward privacy with update mode. It leaks the documents currently matching w, when they were inserted, and when all the updates on w happened (but not their content).

3.2. Bilinear Pairing

Let q be a prime, $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cycle groups of prime order q, on which the operations are addition and multiplication, respectively. The bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ from $\mathbb{G}_1$ to $\mathbb{G}_2$ satisfies the following properties:

1. Bilinearity: For all $P, Q, R \in \mathbb{G}_1$ and $a, b \in Z$, there are $e(aP, bQ) = e(P, Q)^{ab}$, or $e(P + Q, R) = e(P, R) \cdot e(Q, R)$ and $e(P, Q + R) = e(P, Q) \cdot e(P, R)$ then the mapping is said to be bilinear.
2. Non-degeneracy: If P is the generator of $\mathbb{G}_1$, then $e(P, P)$ is the generator of $\mathbb{G}_2$.
3. Computability: For all $P, Q \in \mathbb{G}_1$, there is an efficient algorithm for calculating $e(P, Q)$.

3.3. Bloom Filter

The Bloom filter was proposed by Bloom in 1970 [22]. It is actually a very long binary vector and a series of random mapping functions. The Bloom filter can be used to search whether an element is in a set; its advantage is that the space efficiency and query time are much better than the general algorithm, and the disadvantage is that there is a certain misrecognition rate and deletion difficulty.

For each datum, the data owner hashes it into the Bloom filter through k unbiased hash functions. For the number of unbiased hash functions k, this paper uses the following formula to calculate:

$$k = (\frac{m}{n}) * ln(2), \tag{2}$$

where m is the size of the filter's bit array, and n is the number of elements expected to be inserted.

For the false positive rate of the Bloom filter, the following definition is given:

$$BFR = (1 - e^{-\frac{kn}{m}})^k, \tag{3}$$

where k is the number of the hush function, n is the number of elements to be stored, and m is the size of the bit array. In this paper the false positive rate of 10^{-6} is specified based on the size of the experimental data.

4. Proposed Construction

We propose a new chain structure which includes two parts: keyword-security encryption and file-security encryption. Forward–backward security can be satisfied by performing these two parts.

4.1. System Model

In our design, there are three parts: data owner (DO), data user (DU) and cloud server (CS). The cloud server stores and manages the data owner's ciphertext set and helps legitimate data users search for the corresponding data. The system model is shown in Figure 1. First, the data owner collects the public keys of all legitimate users to be used to compute the relevant data pp for the search token. In the second step, the data owner sends the encrypted EDB, ABF, and B to the cloud server for storage. The above is the initialization preparation. Next, if there is a user (legitimate or not), he/she can request the data about the search token from the data owner (Step 3). After that, the data owner sends pp to the data user (Step 4). In step 5, the data user uses the data pp to calculate the corresponding keyword search token. Here, only legitimate users can calculate the correct search token using their private key; otherwise, they will only obtain the wrong data. In the next step, the data user sends the search token to the cloud server to apply for the search. In the seventh step, the cloud server sends a collection of encrypted files from the search to the data user, and finally, in the eighth step, the data user decrypts the data in the res to obtain the plaintext.

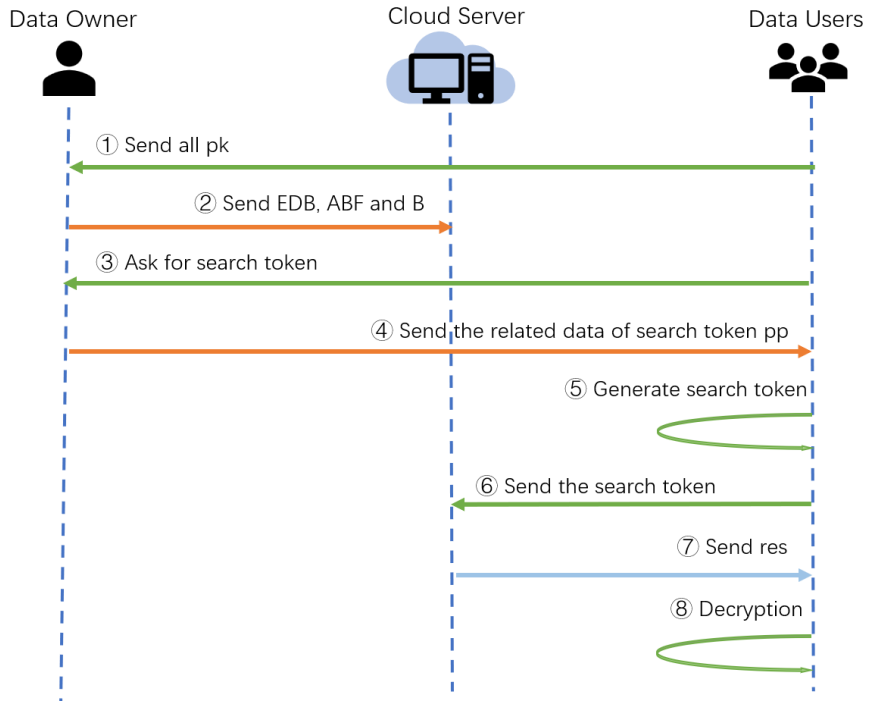

Figure 1. System model.

The scheme proposed in this paper consists of eight algorithms:

$Setup(\lambda) \rightarrow para$: Input the security parameter λ and generate the hash function, pseudo-random function, bilinear mapping, and other data required in the next step.

$KeyGen(H_0, id, g) \rightarrow sk_{du_i}, pk_{du_i}$: Each DU generates her/his own public/private keys using its own id.

$Initial(W_{ind}, para) \rightarrow (ABF, EDB, B, NE)$: data owner initializes the related data. Encrypts (keyword, file set) pairs and sends the encrypted data to cloud server.

$Trapdoor(w, para, h, s) \rightarrow st_w$: DU runs this algorithm, enters its private key (the data are sent from the DO into the algorithm), calculates the search token, and sends it to the cloud server.

$Search(st_w, para, EDB, ABF) \rightarrow res$: The data user sends the keyword search token to the cloud server, and the server runs the algorithm to send the corresponding encrypted file set to the data user .

$Update(Doc, para, EDB, ABF)$: The algorithm is used to update the data. This algorithm is run by the DO to encrypt the newly stored keywords or files and put them in the corresponding location.

$UpdateST(W, para, EDB, ABF) \rightarrow EDB, ABF$: This algorithm is run by the DO, which updates all keyword search tokens at the end of each update, and then sends all the updated data to the server.

$Dec(res, w, H_2) \rightarrow tal$: The data user decrypts the collection from the server to obtain the required file.

4.2. Keyword-Security Encryption

For the encryption of the search token of the keyword, this paper sets the following definition in order to meet the search conditions of multiple users.

Let $U = (id_1, id_2, id_3, id_4, \ldots, id_n)$ be the id set of legitimate search users and the number of users be n. Then, set $\vec{X} = (x_1, x_2, x_3, \ldots, x_l) = \left((H_0(id)\ )^0, (H_0(id)\ )^1, (H_0(id)\ )^2, \ldots, (H_0(id)\ )^l,\right)$ to the hashed set of a user's id, where $x_0 = 1$. Set $\vec{Z} = (z_1, z_2, z_3, \ldots, z_l)$, where z_j in $\vec{Z}$ is the coefficient of z^j of the expansion of $\prod_{j=1}^{n}\left(z - H_0\left(id_j\right)\ \right)$, and id_j is the id of the j-th user. In this article, the data owner sends the following data to the data user:

$$pp = (p_0, p_1, p_2, para)\ , \tag{4}$$

where $p_0 = g^r$, andr is the random number; $p_1 = g^{r\prod_{i=1}^{l}(z_i)}$, and z_i is the data of $\vec{Z}$; $p_2 = g^{z_0}$.

Assuming that a data user with id wants to search for the keyword w, all the data need to be organized into the following form:

$$\begin{aligned} t_w &= e\left(g^{\prod_{i=1}^{l}(x_i)}, p_1\right) e(g^w p_2, p_0) \\ &= e\left(g^{\prod_{i=1}^{l}(x_i)}, g^{r\prod_{i=1}^{l}(z_i)}\right) e(g^w g^{z_0}, g^r) \\ &= e(g,g)^{-x_0 z_0 r} e(g,g)^{wr+z_0 r}. \end{aligned} \tag{5}$$

As $H_0(id)$ is the root of $\prod_{j=1}^{n}(z - H_0(id_j))$, and $\left\langle \vec{X}, \vec{Z} \right\rangle = \prod_{i=0}^{l} x_i z_i = 0$, so that $\prod_{i=1}^{l} x_i z_i = -x_0 z_0$. However, $x_0 = 1$, we can obtain $x_0 z_0 r = z_0 r$.

4.3. Keyword Storage Scheme

For keyword storage, this paper designs an Authenticator Bloom Filter (ABF). As shown in Figure 2, the Bloom filter has been modified to add a counting module, and the authenticator is designed to support dynamic updates.

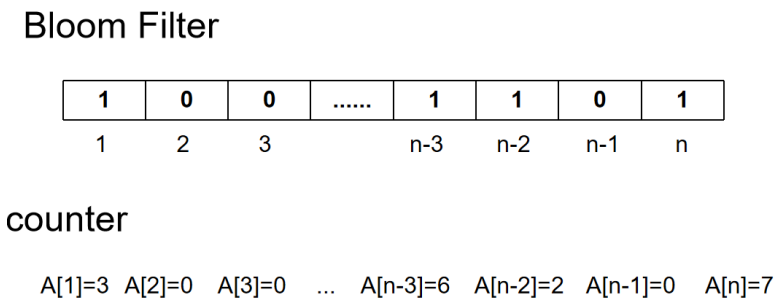

Figure 2. Authenticator Bloom Filter.

In the ABF structure, for each keyword, the Data Wwner hashes it into the Bloom filter through k unbiased hash functions. For the problem that there may be multiple keywords corresponding to one location, this article adds the counter A[]. Each time A keyword is computed and mapped to the Bloom filter, the count is increased by one for each position A[i]. This means that there is a keyword mapped in the i-th position.

Figures 3 and 4 show the process of adding and deleting data for the ABF. Add data as shown in Figure 3. Hash keyword A and map it to bits 1, 3, 5, and 8 in the Bloom filter. Since bits 1, 5, and 8 are already mapped with keywords, only one is added to the counter. On the third bit, not only a one is added to the counter, but also a one is placed on the corresponding bit of the filter. The deletion process is shown in Figure 4. After keyword B is hashed, it is mapped to the first, fifth, sixth, and eighth bits. First, the corresponding counters are reduced by one, and it is found that the eighth counter is reduced to 0. This means that there are no more keywords mapped to this location, so place 0 in the Bloom filter.

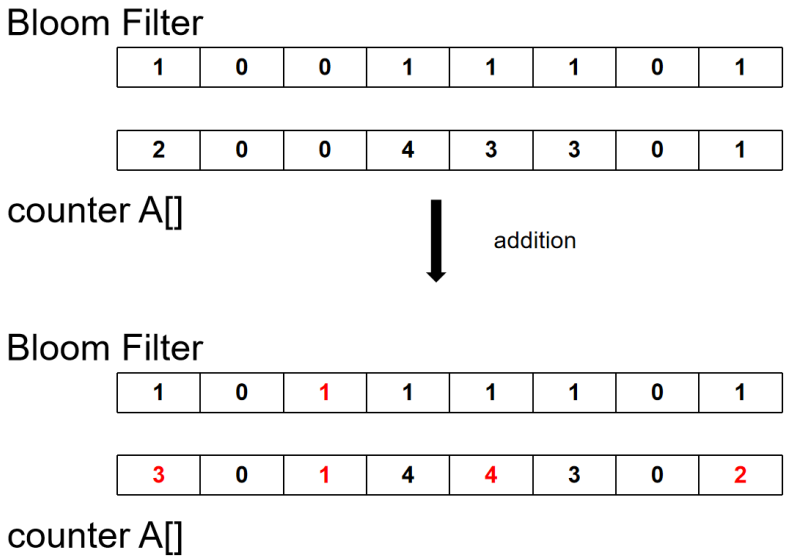

Figure 3. Authenticator Bloom Filter (addition). (The red font is the changed data).

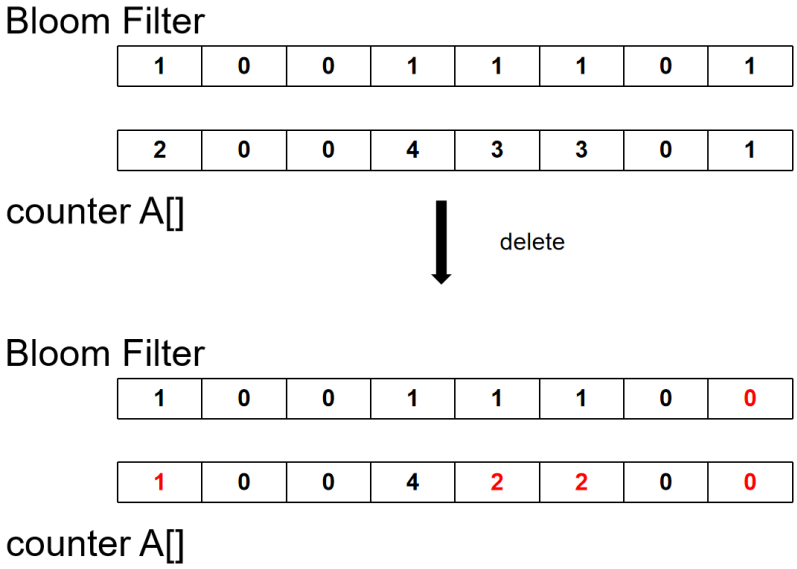

Figure 4. Authenticator Bloom Filter (deletion). (The red font is the changed data).

4.4. File-Security Encryption

The ind in this paper's scheme refers to the address of the file, and the user can find the encrypted file by decrypting to obtain the ind plaintext. This paper uses a symmetric encryption scheme to encrypt the contents of the file, which is not specifically described because it is not very relevant to the scheme of this paper.

The encryption for the document set is as follows:

$$IE^{j}_{st_w} = H_2(w||j) \oplus (ind[j]||op), \tag{6}$$

where op is the state of the file (add/del).

The form of the encrypted file collection is put into the server, but the form of the first key-value pair of each keyword is different from the other; the first set of key-value pairs is as follows:

$$add^{1}_{st_w} = H_3(st_w), \tag{7}$$

$$val^1_{st_w} = \left(IE^1_{st_w} || rn_1\right) \oplus H_3(st_w). \tag{8}$$

Here the first set of key-value pairs requires a search token and a randomly generated number that is used to search for the next key-value pair, and the rest of the key-value pairs are as follows:

$$add^i_{st_w} = H_3(rn_{i-1}), \tag{9}$$

$$val^i_{st_w} = \left(IE^1_{st_w} || rn_i\right) \oplus H_3(rn_{i-1}). \tag{10}$$

Each key-value pair here is calculated from the previous set of key-value pairs, as shown in Figure 5.

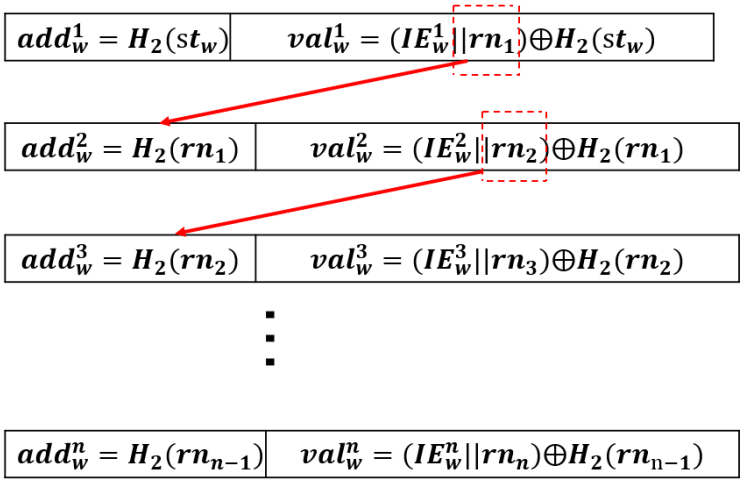

Figure 5. Encrypted file storage structure.

For document deletion operations, this article does not physically delete an existing document but sets the op state corresponding to the document to delete. When the server runs the search algorithm, it obtains an encrypted file, thus supporting backward security.

5. Construction

In this section, we introduce our method. This method can be used in many different situations. We will describe and analyze the following Algorithms 1–8.

Algorithm 1 Setup

Input: λ
Output: *para*
1: `Generates the paramenters about the pairing operation` $(\mathbb{G}_1, \mathbb{G}_2, e, g, q)$
2: **Generate the sets** $\vec{Z}$
3: `Select the Hash functions` $\left(H_{i\in\{0,2\}}, h_{i\in\{1,4\}}\right)$
4: $para = \left(\mathbb{G}_1, \mathbb{G}_2, e, g, q, H_{i\in\{0,2\}}, h_{i\in\{1,4\}}\right)$

Algorithm 2 KeyGen

Input: H_0, id, g
Output: sk_{du_i}, pk_{du_i}
1: **Generate the sets** $\vec{X}$ **of user i**
2: $sk_{du} = g^{\prod_{i=1}^{l} x_i}$
3: $pk_{du} = H_0(id)$

Algorithm 3 Initial

Input: $W_{ind}, para$
Output: ABF, EDB, B, NE

while $W_{ind} \neq null$ **do**
  $w_{ind} \xleftarrow{R} W_{ind}$
  $W_{ind} \leftarrow W_{ind} \setminus \{w_{ind}\}$
  $Parse \quad w_{ind} \quad as \quad (w, st_w, ind[j])$
  $ABF \leftarrow H' = (h_1(st_w), h_2(st_w), h_3(st_w), h_4(st_w))$
  $c_{st_w} = 1, B[st_w] = c_{st_w}$
  $IE^1_{st_w} = H_2(w||1) \oplus (ind[1]||op)$
  $add^1_{st_w} = H_3(st_w)$
  $rn_1 \leftarrow \{0,1\}^\lambda$
  $val^1_{st_w} = \left(IE^1_{st_w}||rn_1\right) \oplus H_3(st_w)$
  $EDB[add^1_{st_w}] = val^1_{st_w}, NE[st_w] = rn_1$
  for $i = 2$ **to** j **do**
    $IE^i_{st_w} = H_2(w||i) \oplus (ind[i]||op)$
    $rn_i \leftarrow \{0,1\}^\lambda, rn \leftarrow NE[st_w]$
    $add^i_{st_w} = H_3(rn)$
    $val^i_{st_w} = \left(IE^i_{st_w}||rn_i\right) \oplus H_3(st_w)$
    $EDB[add^i_{st_w}] = val^i_{st_w}$
    $NE[st_w] = rn_i, c_{st_w} \leftarrow B[st_w]$
    $B[st_w] = c_{st_w} + 1$
  end for
end while
send ABF,EDB,B and NE to the cloud server

Algorithm 4 Trapdoor

Input: w, pp
Output: st_w

$st_w = H_0\left(e\left(g^{\prod_{i=1}^{l}(x_i)}, p_1\right)e(g^w p_2, p_0)\right)$
$Send \quad st_w \quad to \quad Cloud \quad Server$

Algorithm 5 Search

Input: $st_w, para, EDB, B$
Output: res

$res \leftarrow \emptyset$
$H' = (h_1(st_w), h_2(st_w), h_3(st_w), h_4(st_w))$
if $H' \quad can't \quad be \quad mapped \quad to \quad ABF, \quad break;$
else
  $c_{st_w} \leftarrow B[st_w]$
  $val^1_{st_w} \leftarrow EDB[H_3(st_w)]$
  $\left(IE^1_{st_w}||rn_1\right) = val^1_{st_w} \oplus H_3(st_w)$
  $res = res \cup IE^1_{st_w}, temp = rn_1$
  for $y = 2$ **to** c_{st_w} **do**
    $val^y_{st_w} \leftarrow EDB[H_3(temp)]$
    $\left(IE^y_{st_w}||rn_y\right) = val^y_{st_w} \oplus H_3(temp)$
    $res = res \cup \left\{IE^y_{st_w}\right\}, temp = rn_y$
  end for
send res to DataUser

Algorithm 6 Update

Input: $Doc, para, EDB, B$
Output:

while $Doc \neq null$ **do**
  $doc \xleftarrow{R} Doc, Doc \leftarrow Doc \setminus \{doc\}, flag = 0$
  $Prase \quad doc \quad as \quad (w, ind, st_w, op')$
  While $flag == 0$ **do**
    if st_w not exit
      update ABF
      $c_{st_w} = 1, B[st_w] = c_{st_w}$
      $add^1_{st_w} = H_3(st_w), rn_1 \xleftarrow{R} \{0,1\}^\lambda$
      $IE^1_{st_w} = H_2(w||1) \oplus (ind[1]||op')$
      $val^1_{st_w} = \left(IE^1_{st_w}||rn_1\right) \oplus H_3(st_w)$
      $flag = 1, update \quad EDB, NE$
      $Put \quad search \quad token \quad st_w \quad in \quad the \quad ABF$
    else
      $c \leftarrow B[st_w], rn \leftarrow NE[st_w]$
      $val^1_{st_w} \leftarrow EDB[H_3(st_w)]$
      $\left(IE^1_{st_w}||rn_1\right) = val^1_{st_w} \oplus H_3(st_w)$
      $(ind[1]||op) = IE^1_{st_w} \oplus H_2(w||1)$
      $temp = rn_1$
      if $ind == ind[1]$
        $op = op', flag = 1$
      else
        for $y = 2$ **to** c **do**
          $val^y_{st_w} \leftarrow EDB[H_3(temp)]$
          $\left(IE^y_{st_w}||rn_y\right) = val^y_{st_w} \oplus H_3(temp)$
          $(ind[y]||op) = IE^y_{st_w} \oplus H_2(w||y)$
          if $ind == ind[y]$
            $op = op', flag = 1$
        end for
      $rn_{c+1} \xleftarrow{R} \{0,1\}^\lambda, NE[st_w] = rn_{c+1}$
      $add^{c+1}_{st_w} = H_3(rn)$
      $IE^{c+1}_{st_w} = (ind[c+1]||op) \oplus H_2(w||c+1)$
      $val^{c+1}_{st_w} = \left(IE^{c+1}_{st_w}||rn_{c+1}\right) \oplus H_3(rn), flag = 1$
  end while
end while

Algorithm 7 UpdateST

Input: $W, para$
Output: EDB, ABF

$r \xleftarrow{R} \{0,1\}^\lambda$
for each keyword $w_i \in W$ **do**
  $t_w = e(g,g)^{w_i r}$
  $st_w = H_0(t_w)$
  $add^1_{st_w} = H_3(st_w)$
  $val^1_{st_w} = \left(IE^1_{st_w}||rn_1\right) \oplus H_3(st_w)$
  $Update \quad EDB \quad ABF$
end for
$Send \quad EDB \quad ABF \quad to \quad Cloud \quad Server$

Algorithm 8 Dec

Input: res, w, H_2
Output: tal
1: $tal \leftarrow \phi$
2: **for** $j = 1$ **to** $|res|$ **do**
3: $IE^{j}_{st_w} \leftarrow res$
4: $(ind[j]||op) = IE^{j}_{st_w} \oplus H_2(w||j)$
5: **if** $op == add$
6: $tal = tal \cup \{ind[j]\}$
7: **end for**
8: **return tal**

$Setup(\lambda) \rightarrow para$: The algorithm is run by the DO and the initialization parameters are defined. First, the data owner inputs the security parameter λ to the algorithm, then generatea the addition group $\mathbb{G}_1$, whose order is a prime q. Let the multiplicative group $\mathbb{G}_2$ have the same order. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a map. So, we have g as the generator of group $\mathbb{G}_1$. Then, the Hash function is selected and we also need to generate vector $\vec{Z}$ based on the set of public keys of all users.

$KeyGen(H_0, id, g) \rightarrow sk_{du_i}, pk_{du_i}$: Each DU generates her/his own public/private keys using its own id.

$Initial(W_{ind}, para) \rightarrow (ABF, EDB, B, NE)$: The data owner runs Algorithm 3 to initialize all the data. The data owner encrypts all the data before sending it to the cloud server. The initialization data for each keyword are placed in $w_{ind} = (w, st_w, ind[j])$, where $ind[j]$ is a document collection of keywords (line 4–5). The search token corresponding to this keyword is evaluated by four hashes and mapped to the Bloom filter, while the counter A[i] at each corresponding position of the filter is increased by one.

The next step is to encrypt the file set. The first document of each keyword is encrypted differently from the other documents, so it needs to be calculated separately. This scheme requires key-value pairs to store encrypted file sets. Key-value pairs are represented in this paper by (add/val), and the corresponding val value is represented in this paper by $EDB[add]$. This scheme needs to create an NE to store the random number generated by the latest keyword file for future updates. Finally, ABF, EDB, B, NE are sent to the cloud server.

$Trapdoor(w, pp) \rightarrow st_w$: The DU runs this algorithm, enters its private key and the data sent from the DO into the algorithm, and calculates the search token and sends it to the cloud server.

$Search(st_w, para, EDB, ABF) \rightarrow res$: The CS runs this algorithm, uses the keyword search token to put its corresponding set of encrypted files into the set res, and sends the res to the DU. First, the cloud server needs to map the search token to the vABF to determine whether the token exists (Line 2–3). If the search token exists, the key-value pair of the encrypted file is found through the token. First, the first value $val^1_{st_w}$ is found by searching the token, and then the encrypted file and rn value are calculated by $val^1_{st_w}$. Then, the key pair of the next encrypted file is found by the rn value found in turn, and all the encrypted files IE are put into the set res through calculation. Finally, the cloud server sends the set res to the DU.

$Update(Doc, para, EDB, ABF)$:This algorithm is run by the DO to encrypt the newly stored keywords or files and put them in the corresponding location.

Updates in this scheme are batch updates (including additions and deletions), and the data owner packages the files that need to be added along with other keywords, update status, and search tokens into a quadruple doc, and puts all the docs into a collection, Doc.

There are four situations that need to be determined during the update:

1. When the keyword corresponding to the updated document does not exist (line 8–18). At this point, you need to initialize the keyword and its files and update the ABF;

2. When the keyword exists, and the corresponding first document is the target document (line 20–28). When determining that the first document is the target document, change the status op directly to the target op′;
3. When the keyword exists, and the target file corresponds to a subsequent known file set (line 30–36). Check whether all the files correspond to the target file at one time, and change the corresponding state of the file op to the target state op′ (add or del state) if found;
4. If the target file has not been stored (line 38–43). Add the target file to the end of the file set while updating EDB and file counters B and NE.

$UpdateST(W, para, EDB, ABF) \rightarrow EDB, ABF$: This algorithm is run by the DO, which updates all keyword search tokens at the end of each update and then sends all the updated data to the server. This algorithm is run when the data owner is sure that all the data that need to be updated have been updated. When updating the search token, the data owner needs to randomly select a random number r to replace the original r to achieve the purpose of data update. Since the generation of a key-value pair for the first document of the encrypted document set corresponding to each keyword involves a search token, the EDB needs to be updated after each search token is updated. Finally, the new EDB and ABF are sent to the cloud server.

$Dec(res, w, H_2) \rightarrow tal$: The DU runs this algorithm to take the encrypted data from the cloud server and decrypt it one by one. After obtaining the file state op, determine whether it is the state add, and if it is, put the file into the collection tal. Finally, send tal to the DU.

6. Security Analysis

6.1. Forward–Backward Privacy

First, forward security means that an update does not reveal any information about the updated keywords. Since the hash function is one-way, the server cannot decrypt the stored identifier unless the client can generate a previous search token. At the same time, every time the data owner updates, the updated keyword search token is updated, so even if the previous search token is leaked, it will not affect future security. Therefore, the scheme in this paper realizes forward privacy.

Backward security ensures that search queries do not show indexes that were previously added but later removed. In this scenario, the file and its file state op are encrypted. Because the search results are still in ciphertext, even if it is stored in a curious server, an attacker cannot learn useful information about the index without knowing exactly what the keyword is. Thus, we support backward privacy.

6.2. Adaptive Security

In order to improve the efficiency of the solution, most existing solutions will leak some information to the cloud server. Therefore, the confidentiality of searchable encryption schemes means that no more information is leaked than is allowed. To demonstrate confidentiality, we follow a true-ideal simulation paradigm similar to the work [23].

Let $\Pi = (Setup, KeyGen, Initial, Trapdoor, Search, Update, UpdateST, Dec)$ be this article's scheme, S be the simulator, and $\mathcal{A}$ be the adversary. We defined the following two games:

$Real_{\mathcal{A}}^{\Pi}(\lambda)$: Run the algorithm $Setup(\lambda)$ and the algorithm $KeyGen(para)$. Then, the game is published $(para, pk_{du})$ and sk_{du} is saved. After that, The attacker then selects a database DB, executes various queries against it, including update queries, search queries, and decryption queries, and returns the answers to these queries by executing the corresponding algorithms or protocols update, search, and dec, respectively. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

$Ideal_{\mathcal{A}}^{\Pi}(\lambda)$: In an ideal world, the opponent selects A safety parameter, and the simulator selects the leak functions $\mathcal{L}^{Setup}$ and $\mathcal{L}^{KeyGen}$ to generate system parameters and return them to the $\mathcal{A}$. The adversary then selects a database, DB, and executes

various queries against it, including update queries, search queries, and decryption queries. The experiment returns the answers to these queries by calling the leak function $\mathcal{L} = \left(\mathcal{L}_{Setup}, \mathcal{L}_{KeyGen}, \mathcal{L}_{Initial}, \mathcal{L}_{Trapdoor}, \mathcal{L}_{Search}, \mathcal{L}_{Update}, \mathcal{L}_{Dec}\right)$. Finally, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$.

Theorem 1. *Let H be the password hash function. The scheme is $\mathcal{L}$ adaptively safe in the stochastic prediction model, where the set of leakage functions $\mathcal{L}$ is defined as follows:*

$$\mathcal{L} = \left(\mathcal{L}_{Setup}, \mathcal{L}_{KeyGen}, \mathcal{L}_{Initial}, \mathcal{L}_{Trapdoor}, \mathcal{L}_{Search}, \mathcal{L}_{Update}, \mathcal{L}_{Dec}\right), \tag{11}$$

where $\mathcal{L}_{Setup} = \perp, \mathcal{L}_{KeyGen} = \perp, \mathcal{L}_{Initial} = \perp, \mathcal{L}_{Trapdoor} = \perp, \mathcal{L}_{Dec} = \perp$.

Proof. Our proof uses a hybrid argument consisting of a series of games. The first game is exactly the same as the game in the real world, while the last game is exactly the same as the game in the ideal world.

G0: This game is the real world SSE security game Real. So, we can obtain :

$$Pr\left[Real_{A}^{EDAEFB}(\lambda) = 1\right] = Pr[Game_0 = 1]. \tag{12}$$

G1: In this game, we need to randomly select the user's public key ID_i to replace the original public key $pk_{du_i} = H_0(id_i)$. It is easy to see here that G1 and G0 are indistinguishable.

$$Pr[Game_0 = 1] = Pr[Game_1 = 1]. \tag{13}$$

G2: In this game, we create a table $TOKEN$ to store search tokens. Each search token is replaced by a random number. Whenever a keyword search token is called, we call the number in the table $TOKEN$ instead of the number in the text. In the case of updates, we will randomly select a string in $\{0,1\}^{\lambda}$ to act. Here we have:

$$|Pr[G_2 = 1] - Pr[G_1 = 1]| \leq Adv_{\mathcal{A}}^{hash}(\lambda) \tag{14}$$

G3: In this game, we need to create four tables Ha_1, Ha_2, Ha_3, Ha_4 to answer the random oracle query, which are used to record the $h_{i\in\{1,4\}}$ that needs to be mapped to the ABF. In the game, whenever these four values need to be calculated, they are directly taken at random from $\{0,1\}^{\lambda}$ and put into the four Ha_1, Ha_2, Ha_3, Ha_4 tables. If the opponent can distinguish between game 2 and game 3, then the hash function can be distinguished from the real random function, which is obviously impossible. Thus, we have:

$$|Pr[G_3 = 1] - Pr[G_2 = 1]| \leq Adv_{\mathcal{A}}^{hash}(\lambda). \tag{15}$$

G4: In this game, two tables, H1 and H2, need to be created to answer $\mathcal{A}$'s query. $H1$ is to record the response to $H_2(w||j)$ and H2 is to record the response to $H_3()$. In our game, we only consider the leak function in the algorithms update, so we can define $\mathfrak{L}_{Update}(DOC) = \Sigma_{w\in W}|EDB(w)|$, which only leaks the number of keyword/document pairs. In game 2, we generate the search token st_w in the update algorithm as a random string instead of the search token generated in the algorithm. In addition, the $H_1(st_w, w_i)$ and $H_2^{c_{stw}}(st_w)$ during token generation is also replaced by the random strings. If the adversary can distinguish between games 2 and 3, we can distinguish between hashed and truly random functions. Then, we have:

$$Pr[Game_4 = 1] - Pr[Game_3 = 1] \leq Adv_{\mathcal{A}}^{hash}(\lambda)\ . \tag{16}$$

G5: In this game, we maintain a table $UPDATE$ to generate the encrypted document. In the update protocol, game 5 uses random numbers instead of encrypted document IE. It can be seen that games 4 and 5 are the same.

$$Pr[Game_4 = 1] = Pr[Game_5 = 1]. \quad (17)$$

G6: Simulator S simulates the adversary's point of view with a leak function L that includes search patterns and add history. From the opponent's point of view, G4 and G5 are exactly the same. Thus, they are indistinguishable:

$$Pr[Game_6 = 1] = Pr[Game_5 = 1] = Pr\left[Ideal_A^{\Pi}(\lambda) = 1\right]. \quad (18)$$

Conclusion: To sum up the contributions of G0, G1, G2, G3, G4, G5, and G6 we have:

$$Pr\left[Real_{\mathcal{A}}^{\Pi}(\lambda) = 1\right] = Pr\left[Ideal_{\mathcal{A}}^{\Pi}(\lambda) = 1\right] \leq Adv_{\mathcal{A}}^{hash}(\lambda) \quad (19)$$

□

Since the hash function is a one-way function, this scheme is an $\mathcal{L}$-adaptively-secure searchable encryption scheme.

7. Performance Analysis

This chapter analyzes our scheme through performance and experiments. Comparing multiple thesis schemes with the scheme of this paper, we draw a conclusion.

We use python cryptographic libraries on a machine with 16 GB of RAM, Intel CORE i7-9700(8-core, 3.6 GHz), running Windows 10 to implement our algorithm. The experiment took Enron email as the data set, mainly tested the algorithm of updating and searching, and compared the time spent in processing all the keywords and file pairs. In the experiment, the security parameter λ = 128 was set, and MD5 was used to implement the hash function. The scheme in this paper will also be compared with the schemes in papers of Chen1 [24], Liu [25], and Chen2 [20].

7.1. Functional Comparison

The functional comparison is shown in Table 1. The scheme of Chen1 et al. [24] can satisfy the anterograde safety. However, this scheme uses symmetric encryption and does not satisfy multiple users. Liu et al.'s [25] scheme satisfies forward security but not backward security or multiple users and uses symmetric key encryption. The scheme proposed by Chen2 et al. [20] satisfies both forward and backward security, supports multiple users, and uses asymmetric key encryption. However, the multiple users of this scheme will consume more time but not in the main scheme, which the article author only mentioned in the article. As can be seen from the table, the scheme in this paper is one of the best.

Table 1. Functional comparison.

Scheme	FP	BP	Multi-User	Cryptosystem
Chen1 [24]	✓	✓	×	symmetric
Liu [25]	✓	×	×	symmetric
Chen2 [20]	✓	✓	✓	asymmetric
our	✓	✓	✓	asymmetric

7.2. Time Consuming for Different ABF Hash Function Numbers

In order to improve the search efficiency, this paper uses the ABF to search keywords. In the ABF, the main factor affecting its efficiency is the number of hashes. Since the use of Bloom filters saves on the error rate, there are generally two solutions: increase the number of hashes and increase the storage array. Since the ABF is stored in the cloud, the error rate can be reduced by increasing the array size so only a few hash functions need to be considered for the most efficient update time. As shown in Figures 6 and 7, the experiment compares the time spent adding and deleting the hash numbers of 4, 6, 8, and 10. It can be

seen that when the number of hashing times is four, the time required is the least, and the time required increases slowly as the number of keywords increases.

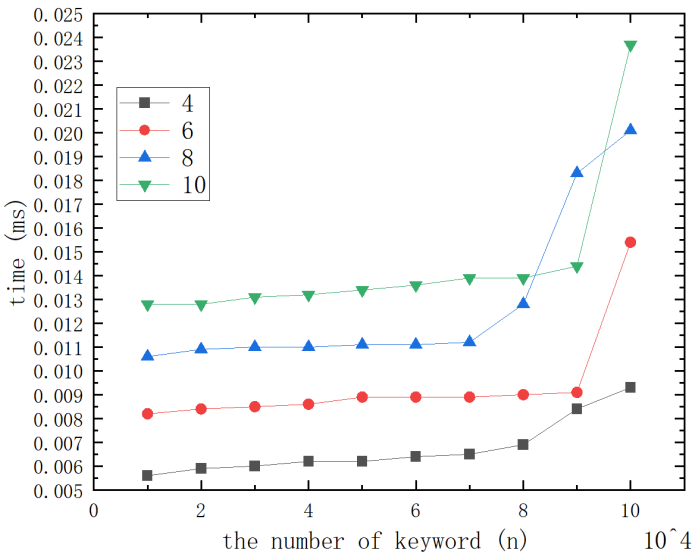

Figure 6. Impact of different hashing times on search time (add).

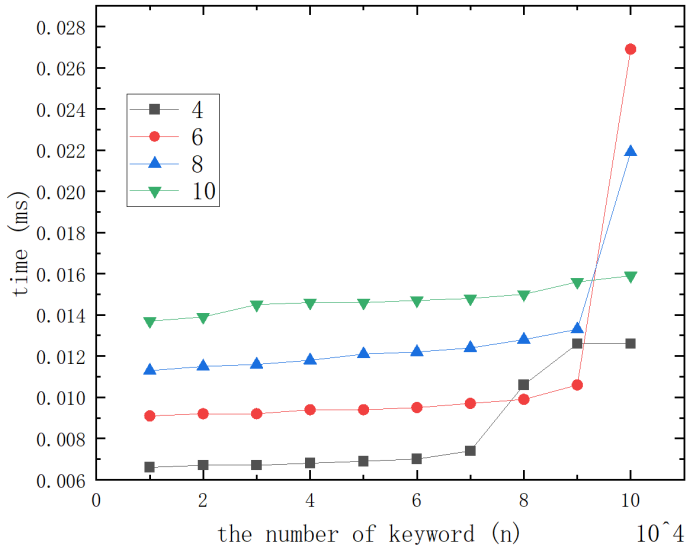

Figure 7. Impact of different hashing times on search time (del).

Based on the conclusion drawn above, it can be determined that the number of hash functions used by this paper's scheme in the ABF is four. Therefore, Formula (3) can be utilized to calculate the size of the bit set in this paper. It can be obtained as m = 47,925,315 bits so that the false positive rate of this paper scheme can be 10^{-6}.

7.3. Time Cost of Search Algorithm

When the user initiates a query operation, a token for the corresponding keyword is generated, and the token is then sent to the server. Figure 8 shows the relationship between the number of keyword document pairs and the search time when the server performs a search. In the experiment, the three schemes Chen1 [24], Liu [25], and Chen2 [20] were compared. In order to obtain a more fair result, the experiment added up the search token generation time of each scheme for comparison, equivalent to calculating the total process of the data user to obtain the encrypted file of the file.

As we can see from Figure 8, the search time in this article is less than in other scenarios, whether the keyword is present or not. If it does not exist, simply return and prompt. As can be seen from the Figure 8, the search time without keywords is between 0.03 ms and 0.05 ms, which can greatly improve the search rate and reduce the waiting time for users to receive feedback. Compared with other experiments, it is necessary to conduct a chain search for all keywords and then give feedback.

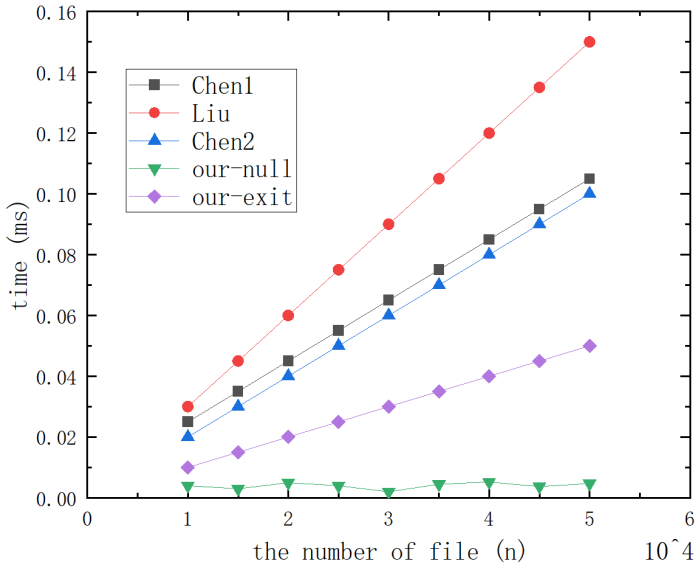

Figure 8. Time cost of search algorithm [20,24,25].

For the search of non-existent keywords, since the other three experiments are all using the same chain storage mode, they are combined into one for comparison (shown in Figure 9). It can be seen that once the keywords exceed 100,000, the chain search method will gradually increase the time. However, the time consumed in the search method in this paper is basically stable, and the time consumed is not increased due to the growth of the total number of searches. The data will only fluctuate in a small range, and the feedback time of users will be greatly shortened.

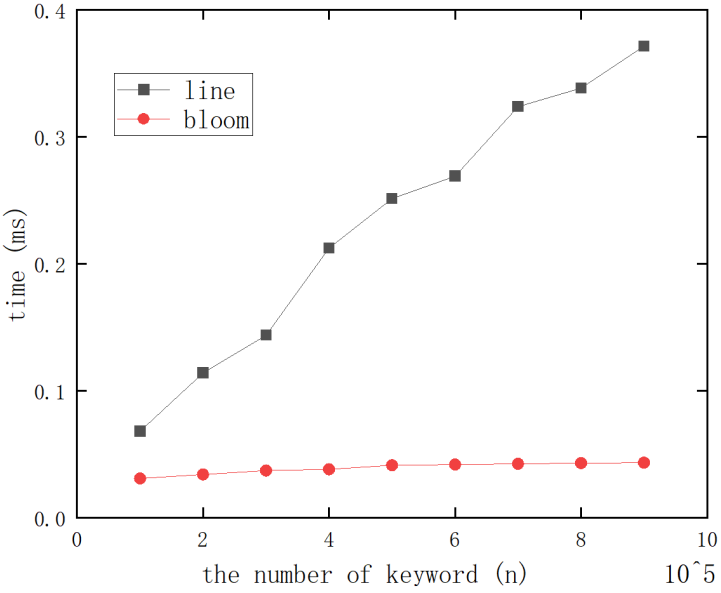

Figure 9. Time spent searching for a keyword when the keyword does not exist.

7.4. Time Cost of Update Algorithm

In the update algorithm, this paper is compared with the schemes of Chen1 [24], Liu [25], and Chen2 [20]. Since these schemes are added and deleted with this algorithm, they are all shown in the following figure.

As can be seen from Figure 10, the solution update of Chen2 et al. [20] took more time. In this scheme, for each keyword that needs to be updated, the file corresponding to it needs to be re-encrypted once. That is, the data owner does not care about the previous encrypted file set; she/he re-encrypts the new file set, and then sends it directly into the database. The reason for the time consumed is that some calculations used in the encryption process, such as pseudo-random function F, bilinear pair e, etc., will be more time-consuming than hashing operations. In addition, the update scheme of Chen2 et al.'s [20] scheme does not take into account whether the newly added file has been added before, which will lead to repeated searches, or whether the new file is deleted and the old version is added, resulting in the deleted file being obtained by the data user.

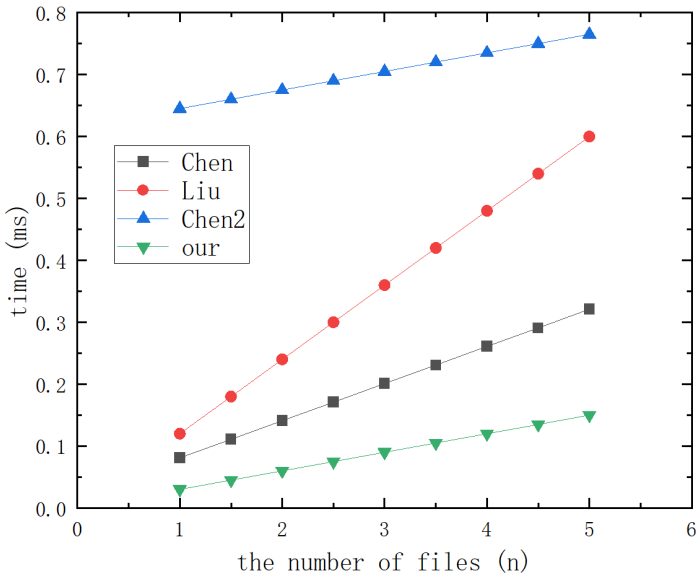

Figure 10. Time cost of update algorithm [20,24,25].

In Liu et al.'s [25] scheme, the previous addition of files was also not taken into account in the update. Although the time is shorter, with the increase in the number of files, the time is the fastest growing, even exceeding the original time-consuming Chen2 scheme.

Among the schemes proposed by Chen1 et al. [24], the time is second only to that proposed in this paper. However, the problem is the same as that in the previous two schemes; the existing files and their status are not considered. This not only adds unnecessary storage space but also affects subsequent search results. Finally, the scheme in this paper is superior to the other three schemes both in terms of rationality and time.

8. Conclusions

In this paper, we design a new dynamic searchable scheme which satisfies forward and backward security. A new ABF based on the original Bloom filter is proposed to reduce the misjudgment rate. At the same time, new key encryption and file encryption schemes are designed. The solution supports forward and backward security, multiple users, and dynamic updates. Compared with other existing schemes on the premise of forward and backward security, especially when the keyword does not exist, the scheme in this paper greatly reduces the time and improves the efficiency. And the keyword search time in this paper has been maintained between 0.03 ms and 0.05 ms. The scheme of this paper takes into account the history of file storage, avoids the situation of multiple storage states of a file when searching, and greatly meets the needs of users.

Author Contributions: Supervision, D.L.; Writing—original draft, Z.J.; Review, X.Z. and Z.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Open Foundation of Shanghai Key Laboratory of Integrated Administration Technologies for Information Security (Grant No. AKG2019005), and Scientific and Technological Innovation 2030—"New Generation Artificial Intelligence" Major Project (Grant No. 2020AAA0109300), and Shanghai Local Colleges and Universities Science and Technology Innovation Capacity-Building Project (Grant No. 23010501800).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in the article.

Acknowledgments: Throughout the writing of this dissertation I have received a great deal of support and assistance. I would also like to thank my tutor, Dongmei Li, for her valuable guidance throughout my studies. You provided me with the tools that I needed to choose the right direction and successfully complete my dissertation. I am also extremely grateful to all my friends and classmates who have kindly provided me assistance and companionship in the course of preparing this paper. In addition, I would like to thank Shanghai University of Engineering Science for providing me with a good learning environment.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chai, Q.; Gong, G. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In Proceedings of the 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, Canada, 10–15 June 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 917–922.
2. Paverd, A.; Martin, A.; Brown, I. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep.* **2014**, 1–14.
3. Song, D.X.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000, Berkeley, CA, USA, 14–17 May 2000; IEEE: Piscataway, NJ, USA, 2000; pp. 44–55.
4. Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. Searchable symmetric encryption: Improved definitions and efficient constructions. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 3 November 2006; pp. 79–88.
5. Chase, M.; Kamara, S. Structured encryption and controlled disclosure. In *Advances in Cryptology-ASIACRYPT 2010: Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, 5–9 December 2010*; Proceedings 16; Springer: Berlin/Heidelberg, Germany, 2010; pp. 577–594.
6. Cash, D.; Jarecki, S.; Jutla, C.; Krawczyk, H.; Roşu, M.C.; Steiner, M. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013: Proceedings of the 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2013*; Proceedings, Part I; Springer: Berlin/Heidelberg, Germany, 2013; pp. 353–373.
7. Cash, D.; Tessaro, S. The locality of searchable symmetric encryption. In *Advances in Cryptology–EUROCRYPT 2014: Proceedings of the 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, 11–15 May 2014*; Proceedings 33; Springer: Berlin/Heidelberg, Germany, 2014; pp. 351–368.
8. Asharov, G.; Naor, M.; Segev, G.; Shahaf, I. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing, Cambridge, MA, USA, 19–21 June 2016; pp. 1101–1114.
9. Zhang, B.; Zhang, F. An efficient public key encryption with conjunctive-subset keywords search. *J. Netw. Comput. Appl.* **2011**, *34*, 262–267. [CrossRef]
10. Bost, R. Σοφος: Forward secure searchable encryption. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 1143–1154.
11. Bost, R.; Minaud, B.; Ohrimenko, O. Forward and backward private searchable encryption from constrained cryptographic primitives. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1465–1482.
12. He, K.; Chen, J.; Zhou, Q.; Du, R.; Xiang, Y. Secure dynamic searchable symmetric encryption with constant client storage cost. *IEEE Trans. Inf. Forensics Secur.* **2020**, *16*, 1538–1549. [CrossRef]
13. Boneh, D.; Di Crescenzo, G.; Ostrovsky, R.; Persiano, G. Public key encryption with keyword search. In *Advances in Cryptology-EUROCRYPT 2004: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, 2–6 May 2004*; Proceedings 23; Springer: Berlin/Heidelberg, Germany, 2004; pp. 506–522.

14. Baek, J.; Safavi-Naini, R.; Susilo, W. Public key encryption with keyword search revisited. In *Computational Science and Its Applications–ICCSA 2008: Proceedings of the International Conference, Perugia, Italy, 30 June–3 July 2008*; Proceedings, Part I 8; Springer: Berlin/Heidelberg, Germany, 2008; pp. 1249–1259.
15. Tang, Q.; Chen, L. Public-key encryption with registered keyword search. In *Public Key Infrastructures, Services and Applications*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 163–178.
16. Park, D.J.; Kim, K.; Lee, P.J. Public key encryption with conjunctive field keyword search. In *Information Security Applications*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 73–86.
17. Guo, J.; Han, L.; Yang, G.; Liu, X.; Tian, C. An improved secure designated server public key searchable encryption scheme with multi-ciphertext indistinguishability. *J. Cloud Comput.* **2022**, *11*, 14. [CrossRef]
18. Li, H.; Huang, Q.; Shen, J.; Yang, G.; Susilo, W. Designated-server identity-based authenticated encryption with keyword search for encrypted emails. *Inf. Sci.* **2019**, *481*, 330–343. [CrossRef]
19. Wang, X.; Ma, J.; Liu, X.; Miao, Y.; Liu, Y.; Deng, R.H. Forward/backward and content private dsse for spatial keyword queries. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 3358–3370. [CrossRef]
20. Chen, B.; Wu, L.; Wang, H.; Zhou, L.; He, D. A blockchain-based searchable public-key encryption with forward and backward privacy for cloud-assisted vehicular social networks. *IEEE Trans. Veh. Technol.* **2019**, *69*, 5813–5825. [CrossRef]
21. Stefanov, E.; Papamanthou, C.; Shi, E. Practical dynamic searchable encryption with small leakage. *Cryptol. ePrint Arch.* **2013**. Available online: https://eprint.iacr.org/2013/832 (accessed on 28 February 2024).
22. Bloom, B.H. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **1970**, *13*, 422–426. [CrossRef]
23. Song, X.; Dong, C.; Yuan, D.; Xu, Q.; Zhao, M. Forward private searchable symmetric encryption with optimized I/O efficiency. *IEEE Trans. Dependable Secur. Comput.* **2018**, *17*, 912–927. [CrossRef]
24. Chen, B.; Xiang, T.; He, D.; Li, H.; Choo, K.K.R. BPVSE: Publicly Verifiable Searchable Encryption for Cloud-Assisted Electronic Health Records. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 3171–3184. [CrossRef]
25. Liu, Y.; Yu, J.; Yang, M.; Hou, W.; Wang, H. Towards fully verifiable forward secure privacy preserving keyword search for IoT outsourced data. *Future Gener. Comput. Syst.* **2022**, *128*, 178–191. [CrossRef]