applied sciences

MDPI

Article

A Searchable Encryption with Forward/Backward Security and Constant Storage

Suzhen Cao [1], Junjian Yan [1,*], Zixuan Fang [1] and Caifen Wang [2]

[1] College of Computer Science and Engineering, Northwest Normal University, Lanzhou 730070, China
[2] College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China
* Correspondence: 2020222072@nwnu.edu.cn; Tel.: +86-183-0341-2303

Abstract: Dynamic searchable encryption satisfies users' needs for ciphertext retrieval on semi-trusted servers, while allowing users to update server-side data. However, cloud servers with dynamically updatable data are vulnerable to information abuse and file injection attacks, and current public key-based dynamic searchable encryption algorithms are often complicated in construction and high in computational overhead, which is not efficient for practical applications. In addition, the client's storage costs grow linearly with the number of keywords in the database, creating a new bottleneck when the size of the keyword set is large. To solve the above problems, a dynamic searchable encryption scheme that uses a double-layer structure, while satisfying forward and backward security, is proposed. The double-layer structure maintains a constant client-side storage cost while guaranteeing forward and backward security and further reduces the algorithm overhead by avoiding bilinear pairings in the encryption and decryption operations. The analysis results show that the scheme is more advantageous in terms of security and computational efficiency than the existing dynamic searchable encryption scheme under the public key cryptosystem. It is also suitable for the big data communication environment.

Keywords: searchable encryption; public key cryptosystem; forward security; backward security

check for updates

Citation: Cao, S.; Yan, J.; Fang, Z.; Wang, C. A Searchable Encryption with Forward/Backward Security and Constant Storage. *Appl. Sci.* **2023**, *13*, 2181. https://doi.org/10.3390/app13042181

Academic Editor: Christos Bouras

Received: 16 December 2022
Revised: 16 January 2023
Accepted: 3 February 2023
Published: 8 February 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the development of network communication technologies such as 5G [1], Internet users choose to store local data in the cloud for use anywhere and anytime, while it is often believed that cloud servers are semi-trusted and sensitive data uploaded by users may create new security risks. Therefore, data users choose to encrypt their data locally before handing it over to cloud servers for hosting, and the ability to manipulate encrypted data stored on remote and untrustworthy servers is gradually becoming a basic need for users [2,3].

Song et al. [4] first proposed the concept of searchable encryption (SE) by using a symmetric cryptosystem, which was used to solve the problem of ciphertext retrieval of data stored on the cloud server [5]. In a general searchable encryption algorithm, the data owner is responsible for encrypting and uploading the data, and the cloud server stores and retrieves the data ciphertext. The data user can perform an exact search by generating and submitting a search trapdoor. Symmetric searchable encryption is widely used due to its low computational overhead, among other reasons, but its key distribution is complicated, and its use is limited in some areas. To expand the application scope of searchable encryption techniques, Boneh et al. [6] proposed a searchable encryption scheme based on the public key cryptosystem. Early research on SE focused on static storage systems, in which users transfer encrypted documents to a cloud server and then generate trapdoors to send search requests to the server. Based on this model, numerous static searchable encryption schemes have been proposed [7,8]. Due to the high overhead of public key searchable encryption in the algorithm construction process, many scholars have

focused on improving the efficiency of the algorithm [9–12]. The computational overhead can be significantly reduced by reducing the operations of bilinear pairs in the trapdoor matching process, and the optimization in the storage structure can likewise lessen the system operation load. At present, the static SE scheme has been increasingly improved, but has also gradually revealed its shortcomings: the data in the cloud cannot be updated in a timely manner, and the system can only be reinitialized when there is an updated demand, which is problematic in practical use.

To achieve real-time database updates in the cloud, more scholars have focused on dynamic searchable encryption (DSE), which is used to achieve real-time data updates in the cloud without reinitializing the algorithm. For better performance, usually, some data information is allowed to be leaked to the cloud server in DSE schemes [13,14], and the leakage may occur during the algorithm initialization phase, as well as during the query algorithm, etc. Since updating the database may leak some information to the cloud server, DSE will bring a whole new challenge regarding data leakage. The possible leakage for dynamic, searchable encryption was first proposed by Stefanov et al. [15] at the 2014 NDSS conference and formally defined by Bost et al. [16]. Forward security [17,18] means that no information about keywords will be leaked when performing update operations or that old search trapdoors cannot be used to search updated files; backward security [19] means that the search algorithm should not leak file identifiers that have been deleted, i.e., subsequent searches will not leak the index information corresponding to deleted files. Song et al. [20] propose a symmetric cryptography-based and efficiently searchable encryption scheme based on symmetric cryptography, with high I/O efficiency, which only satisfies the forward security, and still exhibits the risk of leakage for the keyword search algorithm that has been deleted. To solve the above problem, based on the work of Bost et al. [16], Ghareh et al. [17] and Sun et al. [19] proposed schemes that also provide forward and backward security, with improvements in both operational efficiency and algorithm construction. Chen et al. [9,21] successively proposed a searchable encryption scheme based on a public key cryptosystem, which also satisfies forward and backward security. The pseudo-random permutation function is used as the core of the algorithm. The computational cost of the algorithm increases rapidly with the increase in keywords, and the existence of bilinear pairs in the algorithm also affects the computational efficiency, to some extent. However, these algorithms use a pseudo-random substitution function as the core of the algorithm, and the computational overhead of the algorithm grows faster with the growth in the number of keywords. In most of the current backward and forward secure DSE schemes, the user needs to maintain a local inverted index that stores the current state of each keyword to generate a search trapdoor. The state needs to be updated synchronously when keyword/document pair operations are performed to add or remove keywords. In the existing scheme, with the increase in keywords, the local storage increases sharply. This shows that it is necessary to design lightweight searchable encryption algorithms with low storage overhead. In summary, most of the current SEs possess the following problems: First, some proposed schemes achieve forward and backward security through pseudorandom functions [22], an inverted index, and other technologies, but the above schemes will create a serious server load under a large number of encryption and update operations. This is limited in practical application. Second, the client's state information may generate leakage. That is, DSE is still needed to improve the efficiency of the algorithm based on further reducing the possibility of leakage.

Our main contributions are summarized as follows.

We proposed a lightweight public-key searchable encryption scheme, which eliminates the bilinear pairing operation in the algorithm and improves the efficiency.

Our scheme adopts a double-layer structure. The upper layer is responsible for the trapdoor of search keywords, which generates fields by an iterative hash function, and can effectively reduce the storage cost. The lower layer is responsible for storing the document index to optimize the computational cost and reduce the reconstruction overhead. Users can

generate search trapdoors that satisfy forward and backward security, avoiding maintaining a state list for each keyword.

We compare our scheme with the existing partially searchable encryption schemes in many aspects. The experimental results show that our scheme offers considerable advantages.

2. Preliminaries

2.1. Hard Problem Assumption

Based on the $\widetilde{n} - RSA$ Trapdoor Decisional Diffie–Hellman (n-TDDH) assumption [12]: let $N = (zp+1)(2q+1)$ be the modulus of the RSA algorithm, and group G be the group of order pzq. The Jacobi symbol of all of elements of G is one, so the group G is cyclic. Given the parameters $(N, g^{ps_1}, g^{ps_2}, g_3 = g^p, (h_{i_1} = y_{i_1}p + zqk_{i_1}, h_{i_2} = y_{i_2}p + k_{i_2}zq, d_i = (s_1y_{i_1} + s_2y_{i_2})^{-1} \mathrm{mod} zq)$ with $h_{i_1} = y_{i_1}p + zqk_{i_1}, h_{i_2} = y_{i_2}p + zqk_{i_2}$ as input. It is difficult for the user to distinguish the tuple $(g_3, {g_3}^r, {g_3}^{(y_{i_1}s_1+y_{i_2}s_2)}, {g_3}^{r(y_{i_1}s_1+y_{i_2}s_2)})$ from $(g_3, {g_3}^r, {g_3}^{(y_{i_1}s_1+y_{i_2}s_2)}, T)$ by a non-negligible probability, where T is a random integer. The advantage of adversary A is defined as follows.

$$|\Pr[A(g_3, {g_3}^r, {g_3}^{x_i}, {g_3}^{rx_i}) = 0] - \Pr[A(g_3, {g_3}^r, {g_3}^{x_i}, T)]| \geq \varepsilon$$

which $x_i = s_1y_{i_1} + s_2y_{i_2}$.

2.2. Identifiers

The document doc consists of the identifier $ind \in \{0,1\}^l$ and the set of keywords $W_{ind} \subseteq \{0,1\}^*$. The document database DB consists of n documents doc and can be represented as $DB = (ind_i, W_{ind_i})_{i=1}^n$; the set of documents containing the keywords ω is represented by $DB(\omega) = \{ind_i | \omega \in W_{ind_i}\}$; N is a keyword/document pair and can be represented as $N = \sum_{i=1}^n |W_{ind_i}|$. The function $Match(A, a)$ represents a lookup of the corresponding value of the element a in the dictionary A. The notations in our DSE schemes are shown in Table 1.

Table 1. Parameter identifiers.

Notations	Meanings
$x \xleftarrow{\$} X$	randomly selected x from X
λ	security parameters
doc	the document
ind	identifier of document
W_{ind}	a collection of keywords containing the document identifier ind
DB	the document database
$CLen$	chain length
EDB	encrypted document database
n	number of documents in the database
kt_ω	the keyword identifier of ω
T_ω	trapdoors of keyword ω
m	total number of keywords
op	data operators: add or delete
$\|\|$	data concatenation operator
$\oplus$	bitwise exclusive OR(XOR) operator

2.3. Double-Layer Structure

To optimize the storage structure, our scheme uses a double-layer structure, as shown in Figure 1. This structure consists of two parts: a key index chain (KIC) at the upper level and a document index chain (DIC) at the lower level. In this structure, the space occupied by the state information is constant.

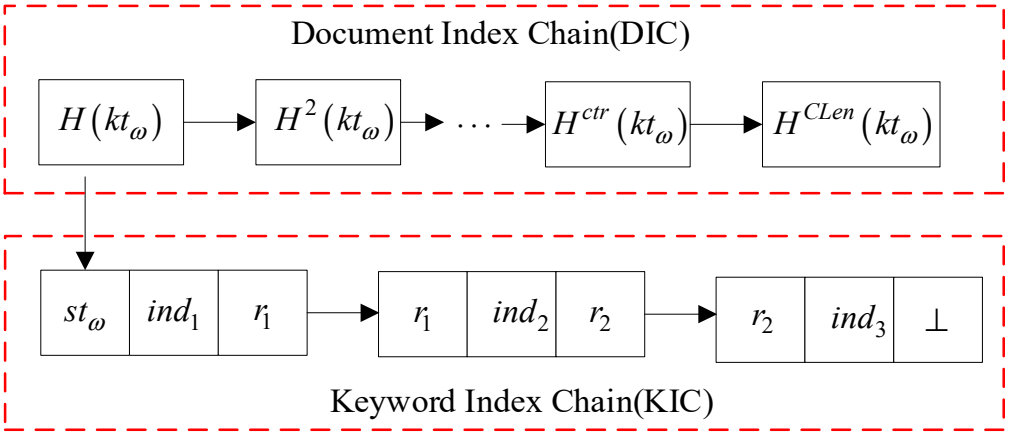


Figure 1. Double-layer structure.

Keyword index chain: the client is responsible for storing a global counter and a key. With this structure, users can generate search trapdoors for a keyword without knowing the keyword's current state. The scheme uses a keyword index chain design to achieve a constant client storage cost.

Let $H_0(*)$ be a cryptographic hash function, and the keyword index chain of the keyword ω consists of $CLen$ search trapdoors, $H_0(T_\omega), {H_0}^2(T_\omega), \ldots, {H_0}^{CLen-1}(T_\omega), {H_0}^{CLen}(T_\omega)$, where kt_ω is the keyword identifier of the keyword ω and ${H_0}^j(T_\omega)\left\{\begin{array}{ll} H_0(kt_\omega) & j=1 \\ H_0(H_0{}^{j-1}(kt_\omega)) & j\geq 2 \end{array}\right\}$. With the search trapdoors of the keyword ω and the global counter $ctr(1 \leq ctr \leq CLen)$, the server can calculate all the search trapdoors of the keyword index chain before the position of ${H_0}^{ctr}(kt_\omega)$ by iteratively calling the hash function $H_0(*)$, i.e., the server can obtain the search trapdoors of $H_0(kt_\omega)$ to ${H_0}^{ctr}(kt_\omega)$. The basic structure of the operation is shown in Figure 2. The server cannot reverse the keyword index chain. Additional overhead is incurred for the $CLen - ctr$ hash operations performed on the server side, but the operational overhead of this stage is acceptable in practice because the hash function operation overhead is very low.

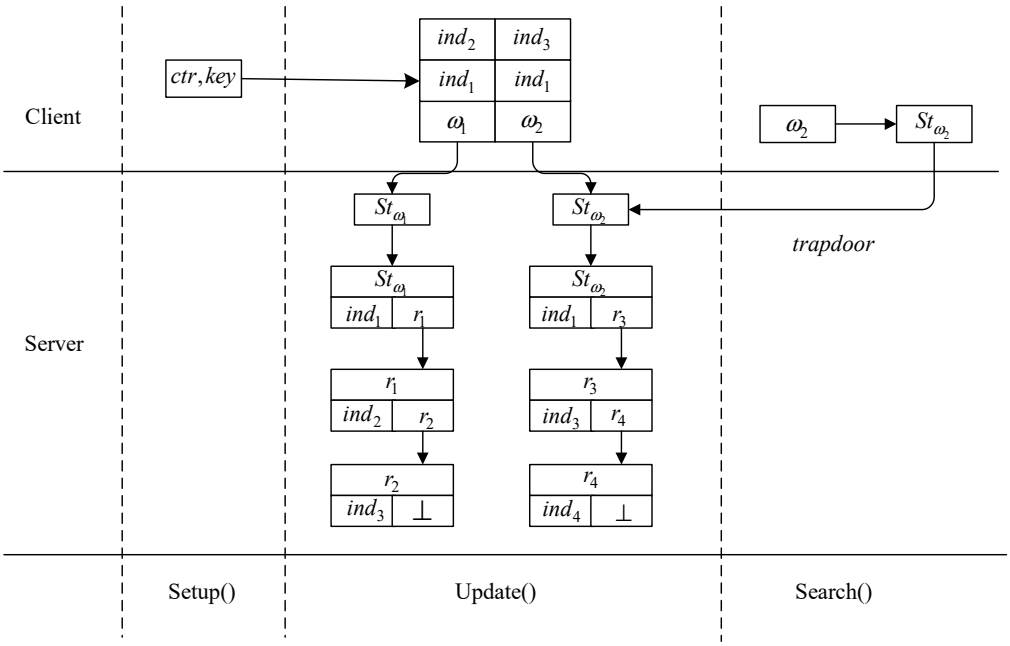


Figure 2. Server operation process.

Document index chain: If a trapdoor corresponds to a set of keyword/document pairs, the global counter *ctr* is quickly consumed when the number of keywords is large. To solve this problem, we designed a lower-level structure. Each search trapdoor node in the keyword index chain corresponds to a document index chain node, and a set of keyword/document pairs can be encoded for a specific keyword in the update.

2.4. Security Model

We adopt the indistinguishable identity chosen ciphertext adversary (IND-ID-CCA) security model proposed by Boneh and Franklin [23]. In this model, a challenger interacts with the adversary as follows:

Setup: In this phase, the challenger will be given a hard problem as an example, and it computes the public parameters.

Phase 1: The adversary adaptively requires the challenger to answer a maximum of n queries, with no more than the following questions:

Key generation query $(ID_i, params)$: the adversary asks the challenger to generate the private key d_i of any arbitrary ID_i.

Decrypt the query $(CT, ID_i, params)$: the adversary asks the challenger to decrypt the ciphertext CT_j of any arbitrary ID_i.

Challenge: The adversary $\mathcal{A}$ chooses two equal-length messages $m_0, m_1 \in Z_N^*$, and an arbitrary ID^* that it aims to attack. The challenger picks a random bit $\gamma \in \{0, 1\}$ and computes the ciphertext CT^*, then returns it to the adversary as a challenge.

Phase 2: This phase is basically similar to Phase 1, but does not allow the adversary $\mathcal{A}$ to generate the private key for ID^* or to ask the challenger $\mathcal{C}$ to decrypt CT^*.

Guess: The adversary outputs a challenge value $\gamma^{'} \in \{0, 1\}$; if $\gamma^{'} = \gamma$, the challenger wins the game.

The adversary in the above game is called the IND-ID-CCA adversary. The advantage of the adversary to win the game can be defined as:

$$Adv_A^{\varepsilon} = \left| \Pr[\gamma = \gamma^{'}] - \frac{1}{2} \right|$$

3. System Model

The system model adopted by our proposed scheme is shown in Figure 3 and has three entities: the client, the PKG, and the server.

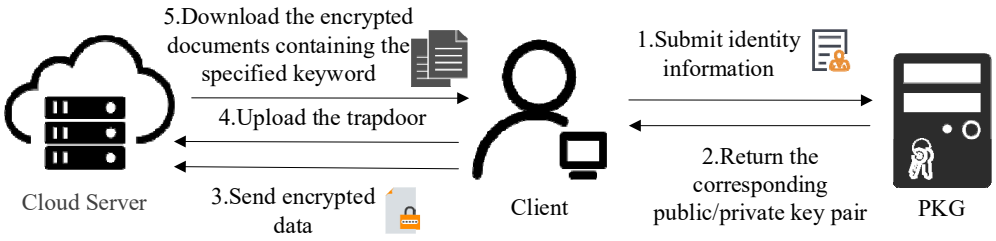


Figure 3. Model of entities and data interaction.

Client: Data encryption and decryption are performed on the client. The client user uploads the ciphertext to the server after encrypting the plaintext with the public key. When searching, the client user can use the keywords to generate trapdoors for a precise search. After receiving the ciphertext result returned by the server, the private key is used to decrypt it.

Server: The server has powerful computing and storage capabilities and can store encrypted data. It is an honest but curious entity. It executes the preset program honestly, but it may try to learn the information from the stored messages and infer what the client user is searching for. In addition, the server processes the search request through the check trapdoor. After the verification, the server sends the encrypted data to the corresponding user.

Private Key Generator (PKG): The PKG is the core of the IBE system. Its functions are as follows: when the IBE system is initialized, it generates the master key and public parameters, and publishes the public parameters. The PKG verifies the user's identity and sends it to the corresponding user through a secret channel as a private key.

The client is responsible for encrypting/updating the documents and uploading them to the storage server (double-layer structure), as well as generating keyword-specific search tokens for keyword searches. The server is responsible for constructing the double-layer system for storage and retrieval and uploading the encrypted data to the DIC part of the double-layer structure.

4. Our Proposed Algorithm

4.1. Setup Phase

Input the security parameter λ, and the KGC generates two $(\lambda/2)$ bit prime numbers $\dot{p}$ and $\dot{q}$ where $\dot{p} = zp + 1$ and $\dot{q} = 2q + 1$. The KGC generates the modulus $N = \dot{p}\dot{q}$ of the RSA algorithm and selects $s_1, s_2 \in Z_N^*$ and g as the generating elements of the group G of order pzq. Compute, $g_1 = g^{ps_1} \mathrm{mod} N$ $g_2 = g^{ps_2} \mathrm{mod} N$, and $g_3 = g^p \mathrm{mod} N$. Define the hash functions $H_0 : \{0,1\}^* \to \{0,1\}^\lambda$, $H_1 : \{0,1\}^* \to \{0,1\}^\lambda$, $H_2 : \{0,1\}^* \to \{0,1\}^\lambda$, $H_3 : \{0,1\}^\lambda \to \{0,1\}^\eta$ where $\eta = \log_2 M$. $F()$ is a pseudo-random function that outputs the system public parameters P_{pub}: g_1, g_2, g_3, H_0, H_1, H_2, H_3, N, g, $F()$.

4.2. Key Generation (KeyGen)

PKG calculates the public/private key for the user with the identity ID_i.

The system calculates $h_{i_1} = H_1(ID_i)$, $h_{i_2} = H_2(ID_i)$, since any integer greater than $pzq - p - zq$ (Frobenius number) can be expressed as a linear combination of p, zq. Therefore, it can also be expressed as: $h_{i_1} = y_{i_1}p + k_{i_1}zq$; $h_{i_2} = y_{i_2}p + k_{i_2}zq$. The inverse solution yields the public key $pk_i = (y_{i_1}, y_{i_2})$:

$$y_{i_1} = p^{-1}h_{i_1} \mathrm{mod} zq = p^{-1}(y_{i_1}p + k_{i_1}zq) \mathrm{mod} zq; = y_{i_1}pp^{-1} \mathrm{mod} zq$$

$$y_{i_2} = p^{-1}h_{i_2} \mathrm{mod} zq = p^{-1}(y_{i_2}p + k_{i_2}zq) \mathrm{mod} zq = y_{i_2}pp^{-1} \mathrm{mod} zq$$

PKG generates the private key sk_i of user ID_i:

$$sk_i = (y_{i_1}s_1 + y_{i_2}s_2)^{-1} \mathrm{mod} zq$$

4.3. Trapdoor Generation (Trapdoor)

The user calculates the keyword token $kt_\omega \leftarrow F(k_1, \omega)$; in turn, the user generates the search trapdoor $T_\omega \leftarrow H^{ctr}(kt_\omega)$.

4.4. Encryption (Enc)

The user calculates $F = g_3^r \mathrm{mod} N$; $e_i = g_1^{h_{i_1}} \times g_2^{h_{i_2}}$, where e_i is constant if the public key does not change and needs to be calculated only once. Calculate the ciphertext $C = (C_1, C_2)$:$C_2 = e_i^r \mathrm{mod} N = (g_1^{y_{i_1}} \times g_2^{y_{i_2}})^r = g_3^{(y_{i_1}s_1 + y_{i_2}s_2)r}$, $F = C_2^{sk_i} = g_3^r \mathrm{mod} N$, $C_1 = (ind||op) \oplus H_3(F)$.

4.5. Update

The update function algorithm is expressed by Algorithm 1.

Algorithm 1 Update function

Parse doc **as** (ind, W_{ind});
while $W_{ind} \neq 0$ **do**
  $\omega \leftarrow W_{ind} \backslash \{\omega\}$;
  $T_\omega \leftarrow Match(KW, \omega)$;
  if $T_\omega = \perp$, **then**
    $T_\omega \leftarrow Trapdoor(PK_i, \omega)$;
    $KW \leftarrow KW \cup (\omega, T_\omega)$;
    $dic \leftarrow \varnothing$;
  $C \leftarrow Enc(PK_i, ind||op)$;
  $k \leftarrow H_0(T_\omega||0)$;
  $v \leftarrow match(dic, k)$;
if $v = null$, **then**
    $dic \leftarrow dic \backslash \{(k, v)\}$;
    $rt \xleftarrow{\$} \{0,1\}^\lambda$;
    $dic \leftarrow dic \cup \{(k, H_0(st||1) \oplus (ind||op||rt))\}$;
    $dic \leftarrow dic \cup \{(H_0(rt||0), H_0(rt||1) \oplus H_0(st||1) \oplus v)\}$;
else
    $v \leftarrow H_0(st||1) \oplus (ind||op||\perp)$;
    $dic \leftarrow dic \cup \{(k, v)\}$;
  $dic' \leftarrow dic$;
Send dic'_ω **to Server**.

4.6. Search

The search function algorithm is expressed by Algorithm 2.

Algorithm 2 Search function

Def $j \leftarrow ctr, RES \leftarrow \varnothing$;
while $j \leq CLen$ **do**
 $k \leftarrow H_0(T_\omega)$;
  $v \leftarrow March(EDB, k)$;
 if $v \neq \perp$ **then**
   $(C||rt) \leftarrow v \oplus H(T_\omega||1)$;
   while $rt \neq \perp$ **do**
     $res \leftarrow res \cup (C)$;
     $v \leftarrow Match(EDB, H_0(rt||0))$;
     $(C||rt) \leftarrow v \oplus H_0(rt||1)$;
   $res \leftarrow res \cup (C)$;
 $j += 1$;
$T_\omega \leftarrow H_0(T)$;
return T_ω.

4.7. Decryption

The decryption function algorithm is expressed by Algorithm 3.

Algorithm 3 Decryption function

Parse as (C_1,C_2)
$F = C_2^{sk_i} = g_3^r \bmod N$;
$ind||op = C_1 \oplus H_3(F)$;
if $op \neq add$, **then**
  $DB(\omega) \leftarrow DB(\omega) \backslash \{ind\}$;
else
  $DB(\omega) \leftarrow DB(\omega) \cup \{ind\}$.

5. Security Analysis

5.1. Indistinguishable Identity Chosen Ciphertext Adversary (IND-ID-CCA)

In this section, we prove the security of the proposed scheme based on the IND-ID-CCA model using the following two theorems. In Theorem 1, we will show that an adversary with access to *n* private keys cannot obtain any information about the keys of p, z, q, and PKG. The security of this scheme relies on the difficulty of decomposing the RSA grand mode n. In other words, if an attacker manages to obtain p and q, then it can compute the private keys of all users. In Theorem 2, the security IND-ID-CCA model of the proposed scheme is proved.

Theorem 1. *Let* $N = \widetilde{pq} = (pz+1)(2q+1)$ *be an RSA module with unknown factorization, and G be a cyclic group of order zq, and* $d_i = {x_i}^{-1} mod zq$*. Given the number of private keys* (d_i) *in the proposed scheme, it is infeasible to compute the decomposition of the mod N if the value of* x_i *is unknown, and for any two instances* x_i *and* x_j*, the values of* $x_i - x_j$ *and* $x_i \cdot x_{\mathrm{j}}^{-1}$ *are unknown to the adversary* $\mathcal{A}$*.*

Proof. Suppose there is an adversary $\mathcal{A}$ that can obtain a number of instances of d_i satisfying the conditions of Theorem 1, and adversary $\mathcal{A}$ can compute a private key d_k that does not satisfy the conditions of Theorem 1. Then prove that there exists an algorithm A that decomposes the computed modulus in approximately the same running time. The algorithm consists of the following three parts. □

Initialization: The algorithm A and N are given as the challenger $\mathcal{C}$ generates N a random integer $d_i \in_R Z_{zq}^*$ and sends it to adversary $\mathcal{A}$. Note that the challenger $\mathcal{C}$ does not know the factorization of N, and there exists a unique element $x_i = {d_i}^{-1} \mathrm{mod} zq$ that makes $x_i = d_i^{-1} \mathrm{mod} zq$.

Challenge: The algorithm A requires the adversary $\mathcal{A}$ to select one element in d_i and compute the private key d_k that does not satisfy the conditions of Theorem 1 associated with the challenge private key.

Response: If adversary $\mathcal{A}$ computes d_k and $x_i - x_k$, then send them to algorithm A, and algorithm A can decompose N as follows:

$$d_k d_i(x_i) = d_k \mathrm{mod} zq$$

$$d_k d_i(x_k) = d_i \mathrm{mod} zq$$

$$d_k d_i(x_i - x_k) = d_k - d_i \mathrm{mod} zq$$

$$d_k d_i(x_i - x_k) - d_k + d_i = 0 \mathrm{mod} zq$$

which is a multiple of zq.

If the adversary $\mathcal{A}$ computes d_k and $v = x_k \cdot {x_i}^{-1}$, then give them to algorithm A, and N can be factored by algorithm A as follows:

$$d_i(x_i) = 1 \mathrm{mod} zq$$

$$d_k(x_i v) = 1 \mathrm{mod} zq$$

$$d_i x_i - d_k x_i v = 0 \mathrm{mod} zq$$

$$x_i = d_i - d_k v \mathrm{mod} zq$$

$$d_i x_i = 0 \mathrm{mod} zq$$

Although the rule set of private keys is finite, it is difficult to compute another valid private key without knowing the factorization of the computational modes.

Theorem 2. *Let* $N = \widetilde{p}\widetilde{q} = (pz+1)(2q+1)$ *be a module of the RSA algorithm and G be the group of* Z_N^*. *The proposed scheme is secure for the adversary of* $IND-ID-CCA$ *if the* $\widetilde{n}-TDDH$ *problem holds in G.*

Proof. Assuming that the adversary $\mathcal{A}$ solves the proposed solution with the advantage ε, there exists an algorithm B that solves the problem $\widetilde{n}-TDDH$ with the advantage $\frac{\varepsilon}{e(n+1)}$. The goal of algorithm B is to distinguish two tuples $(g, g^x modN, g^r modN, g^{rx} modN)$ and $(g, g^x modN, g^r modN, T)$, where T is a random integer. In the next game, algorithm B interacts with adversary $\mathcal{A}$.□

Initialization: In this phase, adversary $\mathcal{A}$ is given an instance of $\widetilde{n}-TDDH$ as input, where $\left\{h_{i_1} = y_{i_1} + k_{i_1} zq, h_{i_2} = y_{i_2} + k_{i_2} zq, d_i = (y_{i_1} s_1 + y_{i_2} s_2)^{-1} \mathrm{mod} zq\right\}_{i=1}^{n}$ and $h_{c_1} = y_{c_1} p + zqk_{c_2}$.

The hash functions $H_1(\cdot)$ and $H_2(\cdot)$ are used to simulate the random oracle $O_1(\cdot)$ and $O_2(\cdot)$; hash function $H_3(\cdot)$ is used to simulate the random oracle $O_3(\cdot)$, which is controlled by the challenger $\mathcal{C}$ and $\mathcal{C}$ can get the hash value by requesting it from random oracle.

The random oracle function $O_1(ID_i)$: Challenger $\mathcal{C}$ maintains the list $List_1$, assigns the user's identity ID_i to $O_1(\cdot)$ as input, and returns the hash value. When the adversary $\mathcal{A}$ requests ID_i, the challenger $\mathcal{C}$ queries $List_1$ and if it finds a tuple corresponding to the user's ID_i identity, it returns that tuple to adversary $\mathcal{A}$. Otherwise, challenger $\mathcal{C}$ chooses a random bit $b_i \in \{0,1\}$, and sends h_{c_1} and h_{c_2} to the adversary if $b_i = 0$; otherwise, it chooses a random integer $j \in \{1, \ldots, n\}$, and returns h_{j_1} and h_{j_2} to adversary $\mathcal{A}$. Finally, it saves h_{j_1}, h_{j_2}, b_i, as well as the identity ID_i, to the list $List_1$. $O_2(\cdot)$ and $O_3(\cdot)$ are similar to $O_1(\cdot)$.

Phase 1: The adversary $\mathcal{A}$ is allowed to make up to n queries, each of which is a key generation query (ID_i) or a decryption query (CT, ID_i), where the ID_i and CT are specified by $\mathcal{A}$. The algorithm B sends a private key d_i corresponding to plaintext m or the specified ID_i and returns it to $\mathcal{A}$.

Key generation query $(ID_i, params)$: The challenger $\mathcal{C}$ searches its database to find the corresponding tuple; if $b_i = 1$, it returns the private key d_i, which is equal to $(y_{i_1} s_1 + y_{i_2} s_2)^{-1} \mathrm{mod} zq$, to adversary $\mathcal{A}$. Otherwise, it rejects the query and ends the game.

Decrypt the query $(CT, ID_i, params)$: The challenger $\mathcal{C}$ will search the database $O_1(\cdot)$ to find the tuple containing ID_i. If $b_i = 1$, it extracts the d_i value from the tuple, computes $O_2({C_2}^{d_i}) \oplus C_1$, and sends it to adversary $\mathcal{A}$. Otherwise, challenger $\mathcal{C}$ will reject the query.

Challenge: First, adversary $\mathcal{A}$ chooses two equal-length messages $m_0, m_1 \in Z_N^*$, and an arbitrarily ID^*, and sends them to algorithm B. Second, algorithm B searches the database $O_1(\cdot)$ for the corresponding tuple. If $b^* = 0$, it generates a random bit $\gamma \in \{0,1\}$ and calculates $CT^* = (O_2({g_3}^r) \oplus M_\gamma, T)$, then sends it to the adversary. Otherwise, it rejects the query and ends the game.

Assuming that adversary $\mathcal{A}$ has the ability to decrypt CT^*, then γ is available by $T = {g_3}^{r(y_{c_1} s_1 + y_{c_2} s_2)} \mathrm{mod} N$, but if T is a random element in G, then adversary $\mathcal{A}$ cannot obtain γ.

Phase 2: This phase is basically similar to Phase 1, but does not allow adversary $\mathcal{A}$ to generate the private key for ID^* or ask challenger $\mathcal{C}$ to decrypt CT^*.

Guess: Output the result $\widetilde{\gamma} \in \{0,1\}$, if $\gamma = \widetilde{\gamma}$, then $T = {g_3}^{r(y_{c_1} s_1 + y_{c_2} s_2)} \mathrm{mod} N$; if $\gamma \neq \widetilde{\gamma}$, then T is a random integer.

Analysis: If the game completes the challenge phase and the Other query of adversary $\mathcal{A}$, which includes key generation queries and decryption queries, is not rejected, the game is successfully completed. Let $b = 1$ have the probability δ, the probability of not rejecting any adversary's n queries is $(\delta)^n$, and the probability of not terminating the game during the challenge phase is $1 - \delta$. Therefore, the game ends successfully with the probability

of $(\delta)^n(1-\delta)$, and the probability is maximum at $\delta_{opt} = \frac{n}{n+1}$. If adversary $\mathcal{A}$, through the above steps, can obtain $Adv^{\varepsilon}_{\mathcal{A}} = \varepsilon$ in time t and at most $Adv^{\varepsilon}_{\mathcal{A}} = \varepsilon$ queries and is able to solve the solution, algorithm B is able to solve the $\widetilde{n} - TDDH$ problem with the advantage of $\varepsilon\delta^n(1-\delta) = \varepsilon\left(\frac{n}{(n+1)}\right)^n\left(1-\frac{n}{(n+1)}\right) \approx \frac{\varepsilon}{e(n+1)}$. In fact, the hard problem is unsolvable, so this scheme satisfies IND-ID-CCA security.

5.2. Forward and Backward SECURITY

In this section, the security of our scheme is demonstrated by the following steps. The cloud server in the system model only has access to the encrypted form of *ind*, and when EDB is updated, the system needs to generate a new search trapdoor for the new entry. The entries observed by the server at the time of the update are invalid, and again the operation type is not available. The search algorithm does not leak any information to the cloud server other than the number of keyword-related entries and the time of update. Specifically, it cannot determine the operations related to deletion and addition. Forward security means that updates can be performed without revealing any information about the keywords, or that previous search trapdoors cannot be used to search the updated files. Backward security means that the search algorithm should not leak the identifier of the deleted file; that is, the subsequent search will not leak the index information corresponding to the deleted file. The scheme satisfies the forward security, since the hash function is a one-way function, and the server derives the keyword index chain backward, so it cannot access the identifiers stored in DIC. Each time we update the EDB, we need to generate new search tokens for new entries. Therefore, the server cannot distinguish the target entry from the random entry during the update. Besides, the server cannot obtain the type of operation. The above properties can guarantee that the scheme satisfies backward security. We will use the following games to prove the forward and backward security of our scheme.

There is a leakage in our scheme, so the leak function $\mathcal{L} = (\mathcal{L}_{setup}, \mathcal{L}_{search}, \mathcal{L}_{update})$ is defined as follows.

$$\mathcal{L}_{setup}(CLen, \lambda) = \perp;$$

$$\mathcal{L}_{search(\omega)} = (sp(\omega), HistDB(\omega));$$

$$\mathcal{L}_{update}(DOC, op) = (\sum_{\omega \in W} |DB(\omega)|, op).$$

The following proof makes an ideal reality argument by discussing the comparison of true random functions with pseudo-random functions. The Game0 is identical to the searchable encryption (SE) game, with pseudo-random functions, while the last game Game3 is identical to the SE, with true random functions. The argument is carried out by comparing the performance of the true and pseudo-random functions in the SE scheme.

Game0: This section is the definition of a realistic SE program. $\Pr[Real_A(\lambda) = 1] = \Pr[Game_0 = 1]$.

Game1: In this section, the system maintains a table of pseudo-random functions indexed by keywords KEY, and for each keyword ω, the system records the KEY bound to it. When a keyword is queried for the first time, a pseudo-random sequence is generated for it and written to the table KEY, which is returned when the keyword is queried again. We map the algorithm program to obtain $\Pr[Real_A(\lambda) = 1$, and if the adversary $\mathcal{B}_1$ can distinguish Game 1 from this algorithm, it can distinguish the pseudo-random function F from the true random function, represented as follows: $\Pr[Real_A(\lambda) = 1] - \Pr[Game_1 = 1] \leq Adv^{prf}_{\mathcal{B}_1}(\lambda)$.

Game2: In this section, the system maintains three tables: H_0, H_1 and H_2, which correspond to ${H_0}^{ctr}(kt_\omega)$, $H_0(T_\omega||0)$, and $H_0(T_\omega||1)$, respectively. When a keyword is queried, the system traverses the hash table and returns the hash value directly if the hash value corresponding to the keyword exists. Otherwise, a hash value is generated for it by the hash function, and it is stored in the table to be answered in the next query. Since the add operation is currently considered, the leak function of the update is defined as follows: $\mathcal{L}_{ADD}(DOC) = \sum_{\omega \in W}|DB(\omega)|$, which will leak only the number of keyword/document

pairs. The search trapdoor T_ω is generated as a random string in the *Update* algorithm. The hash functions h_1, h_2 are also replaced by random strings. Each time the hash function is called, we return the corresponding hash values through the tables H_0, H_1 and H_2, respectively. If the adversary can distinguish between Game2 and Game1, then it will be able to distinguish between the hash function H and the true random function. To summarize, we define the adversary $\mathcal{B}_2$ as follows: $\Pr[Game_1] - \Pr[Game_2 = 1] \leq Adv^{hash}_{H,\mathcal{B}_2}(\lambda)$.

Game3: In this game, the system maintains the table *Update* for generating search trapdoors, which is indexed by keywords. In the search algorithm, Game3 generates the search trapdoor T_ω by running the *ctr* hash function several times. Instead of mapping the keywords to the table $Update$ of values in DIC, the algorithm uses the table $Update$ that maps to the update count. This leads to $\Pr[Game_2 = 1] = \Pr[Game_3 = 1]$.

Except in $Ideal_{\mathcal{L}}$, Game3 is the same as the ideal environment, which leads to $\Pr[Game_3 = 1] = \Pr[Ideal_A(\lambda) = 1]$.

Conclusion: Through the above game interaction, in which the attacker queries and the system feeds back, by stating that $F()$ is a pseudo-random function and the hash function $H()$ is a one-way function, there exist two adversaries, $\mathcal{B}_1$, $\mathcal{B}_2$, expressed through the following equations: $\Pr[Real_A = 1] - \Pr[Ideal_A(\lambda) = 1] \leq Adv_{B_1}(\lambda) + Adv_{B_2}(\lambda)$.

Then, the theorem is proved.

6. Performance Evaluation

In this section, the scheme is compared with similar recent schemes in terms of the computational and storage overhead required for each phase of the algorithmic scheme operation. We tested our scheme on a Windows 10 operating system (64-bit), Intel® Core(TM) i5-6200 CPU @ 3.10 GHz, and a local virtual machine, VMware (4 GB RAM), running the open source project OpenStack for performance testing, which can provide sufficient computing resources for computation. Basic cryptographic operations were implemented using the Python cryptographic package pycrypto. Table 2 shows the time required for different operations.

Table 2. Computation time.

Symbols	Meaning
T_{sm}	scalar multiplication
T_p	bilinear pair operations
T_h	hash operations
T_{match}	$Match()$ function operation
T_F	pseudo-random function operations
T_{Div_G}	division operation in group G
T_{map}	mapping calculation
T/T^{-1}	T/T^{-1} operation
T_{mul_G}	multiplication in group G
T_e	exponential arithmetic

6.1. Data Sets

The test uses a dataset from SNAP [24]: Amazon reviews, which consists of reviews from Amazon. Reviews include product and user information, ratings, and plain text reviews. In the above review dataset, 2,441,053 pieces of product information were extracted and used as search keywords.

6.2. Evaluation of Computation Cost

This section focuses on the comparison from the encryption and update, as well as the search phase, and compares our scheme with others regarding the computational overhead generated for the processing of the same amount of data, so as to analyze the differences between the algorithms of different schemes. Table 3 shows the comparison of different DSE schemes.

Table 3. Comparison of different DSE schemes.

Scheme	Encryption and Update	Search	Security	Cryptosystem
Chen et al. [21]	$n*(T_F+T_p+2T_{sm}+2T_h)$	$n*T_F+T_p+T_h$	F.	Public Key
Bost et al. [16]	$n(3T_h+T_p)+T_h$	$n(2T_h+T^{-1})$	F. and B.	Symmetries
Ding et al. [18]	$T_{map}+T_F+4T_h+T_e+n(2T_h+2T_e)+3T_{sm}$	$n*2T_h+4T_h$	F. and B.	Public Key
Ours	$3T_h+2T_F+n(2T_h+T_e)$	$n(T_{match}+T_h)$	F. and B.	Public Key

F.: forward security; B.: backward security.

Since the main computational overhead of the algorithm comes from the encryption and search phase, we test the time required by different schemes when the number of keywords increases from 100 to 2000. Figures 4 and 5 show the performance of different schemes in the encryption and search phase.

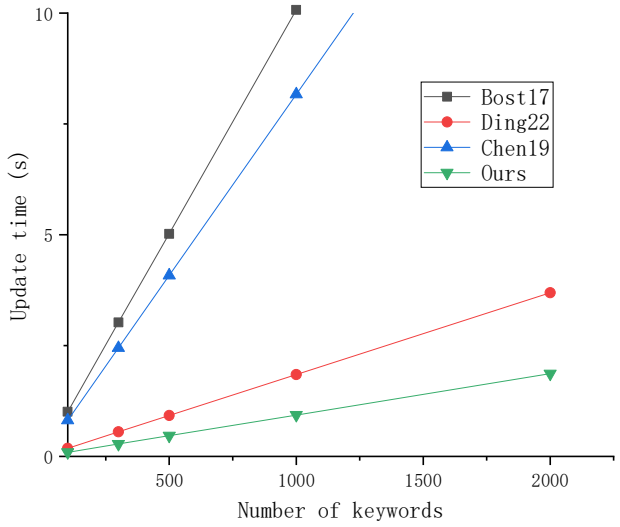


Figure 4. Performance comparison with Bost17 [16], Ding22 [18], and Chen19 [9] in the encryption and update phase.

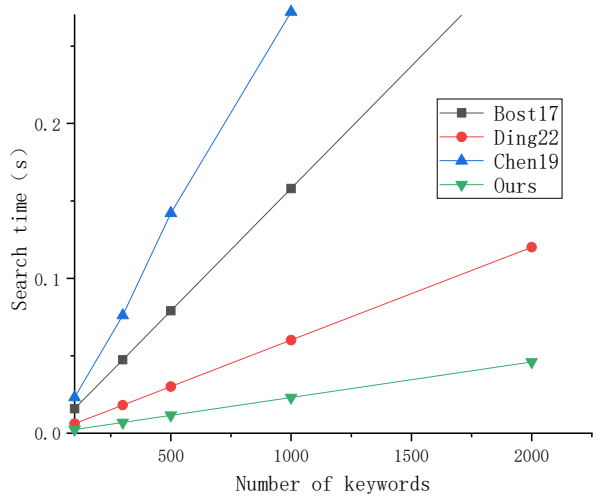


Figure 5. Performance comparison with Bost17 [16], Ding22 [18], and Chen19 [9] in the search phase.

Figure 4 describes the performance of different schemes in the encryption and update. Figure 4 shows that among the above four comparison schemes, the running time of Bost et al. [16] and Chen et al. [21] exceeds 5 s when the number of keywords is 1000, and the increase is large. It can be expected that in the environment of more keywords, the scheme will bring great pressure to the server. The scheme of Ding et al. [18] shows a certain advantage in running time when compared to our scheme. Furthermore, when our scheme processes up to 2000 keywords, the time required by our scheme is only 55% of that of Ding et al. [18]. This will bring great advantages to our scheme in the overall operation phase.

From the analysis in Figure 5, it is clear that the overhead of Chen et al. [21] is larger, which is due to the fact that a large number of exponential operations are used in the search process, and each cryptographic operation needs to be re-invoked, resulting in a negative impact, requiring more power operations, on its overall scheme. The scheme of Bost et al. [16] requires long-term maintenance of the inverted index, which puts great computational pressure on the server, and it also uses more bilinear pairings operations, which further reduces the performance of the scheme algorithm. Ding et al. [18] use bilinear pairings operations, which have some impact on the algorithm efficiency. These only need to be calculated once when the system is established, guaranteeing the efficient operation of the subsequent algorithms, to some extent. Our scheme boasts a lightweight construction and adopts a bilinear pair-free design in the search phase. This is a huge improvement to the scheme, and the advantage becomes more obvious as the number of keywords increases.

It can be observed from Figures 4 and 5 that the overhead of the above scheme is much higher in the encryption and update phase than in the search phase. Therefore, the impact of the encryption phase performance on the overall performance of the scheme is decisive. Therefore, the advantages in the encryption and update phase determine that the overall performance of our scheme is excellent. Figure 6 shows a comparison of the performance analysis of the overall scheme in the whole process of encryption, updates, trapdoor generation, and searching. The overall performance test in Figure 6 demonstrates that our solution outperforms the others.

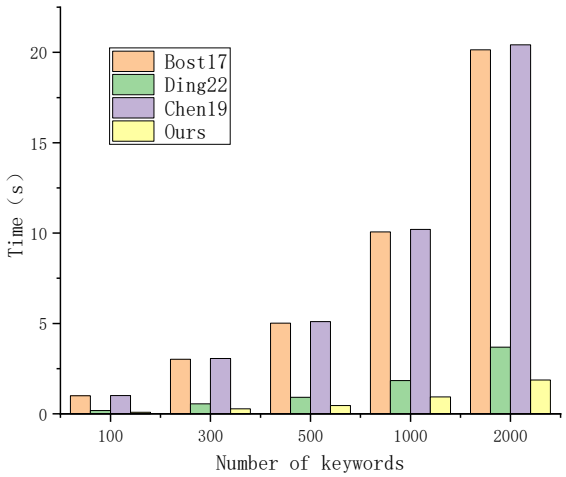


Figure 6. Performance comparison with Bost17 [16], Ding22 [18], and Chen19 [9] in the search phase.

6.3. Evaluation of Storage Cost

In order to compare the storage pressure generated by different keyword databases using various similar solutions, we used three different sizes of keyword databases and analyzes their advantages and disadvantages by comparing the storage overheads generated by different solutions under the same circumstances. The Amazon review data source was used at this phase of the test, which was divided into three datasets based on the number of keywords (Dataset 1–3). Table 4 shows that the number of files, the number of keywords,

and the number of keyword/document pairs in Dataset 1, Dataset 2, and Dataset 3 increase in descending order.

Table 4. Test data table.

Dataset	Number of Files	Number of Keywords	Keyword/Document Pairs
Dataset 1	2000	7563	55,323
Dataset 2	5000	12,315	263,597
Dataset 3	10,000	29,681	438,762

In Table 5, the storage space of Bost et al. [16], Ding et al. [18] and Chen et al. [21], without the optimization of the storage structure, increases significantly with the increase in keywords. Some of the above schemes use inverted indexing to maintain the keyword state, while our scheme uses a global counter generated by a key and a hash function to maintain the keyword state. Although the iterative use of hashing incurs additional overhead, the feasibility of the scheme is demonstrated in the next phase of performance evaluation.

Table 5. Performance of each scheme under the effect of different databases.

Scheme	Dataset 1	Dataset 2	Dataset 3
[21]	3842 kb	7613 kb	8922 kb
[16]	1974 kb	4012 kb	6474 kb
[18]	3963 kb	5563 kb	10,072 kb
Ours	843 kb	843 kb	843 kb

6.4. Discussion of Comparison

The above two parts of the performance evaluation are evaluated in terms of computational efficiency and storage overhead, respectively. In terms of storage overhead, in order to reduce the occupation of local storage resources, the design of iteratively nested hash functions is used to ensure that the size of each node is consistent, and that constant storage overhead is maintained without increasing the length of the keyword index chain. The above structure creates a large number of hash operations, and the experimental analysis shows that the hash function generates negligible efficiency to the operation of the algorithm. In comparison with similar schemes, our scheme still has some advantages in terms of computational overhead. As a result, we achieve constant storage overhead and high encryption and decryption efficiency. Future research work should explore improvements in function, time cost, and safety on the basis of this scheme. These results prove that our scheme can be applied to a wider range of applications, such as low-performance computers.

7. Conclusions

In this article, we analyze the problems of the related dynamic searchable encryption schemes and the solutions proposed by different scholars according to various aspects. We proposed a dynamic searchable encryption scheme with constant storage, which adopts a double-layer structure and iteratively uses hash functions. The scheme satisfies forward and backward security, while also achieving low computational overhead. The experimental evaluation shows the superiority of our scheme relative to the current mainstream algorithms. Our proposed scheme can be applied to a small encryption environment to provide privacy protection, i.e., the internal network of a company or scientific research institution, where the ciphertext is stored centrally to ensure that unauthorized users cannot have access. At present, our research has not implemented a multi-keyword search. In future work, we will consider including a multi-keyword search to improve accuracy and protect user search patterns vs. access patterns.

Author Contributions: Conceptualization, S.C. and J.Y.; methodology, J.Y.; software, J.Y.; validation, J.Y. and Z.F.; formal analysis, J.Y.; investigation, S.C.; resources, J.Y.; data curation, J.Y.; writing—original draft preparation, J.Y.; writing—review and editing, S.C.; visualization, Z.F.; supervision, C.W.; project administration, S.C. and J.Y.; funding acquisition, J.Y. and S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (No. 62262060) and the Industrial Support Plan Project of the Gansu Provincial Department of Education (2022CYZC-17).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Namasudra, S. Data access control in the cloud computing environment for bioinformatics. *Int. J. Appl. Res. Bioinform. (IJARB)* **2021**, *11*, 40–50. [CrossRef]
2. Ocansey, S.K.; Wang, C. Searchable Encryption for Integrating Cloud and Sensor Networks with Secure Updates. *Ad hoc Sens. Wirel. Netw.* **2021**, *50*, 1–14.
3. Sadeeq, M.M.; Abdulkareem, N.M.; Zeebaree, S.R.; Ahmed, D.M.; Sami, A.S.; Zebari, R.R. IoT and Cloud computing issues, challenges and opportunities: A review. *Qubahan Acad. J.* **2021**, *1*, 1–7. [CrossRef]
4. Song, D.X.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, S&P 2000, Berkeley, CA, USA, 14 May 2000.
5. Li, J.; Gan, Q.; Wang, X.; Wang, F. Efficient forward secure searchable encryption supporting multi-keyword query. In Proceedings of the 2nd International Conference on Computing and Data Science, online, 28 January 2021.
6. Boneh, D.; Crescenzo, G.D.; Ostrovsky, R.; Persiano, G. Public key encryption with keyword search. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Perugia, Italy, 30 June 2004.
7. Yang, N.; Xu, S.; Quan, Z. An efficient public key searchable encryption scheme for mobile smart terminal. *IEEE Access* **2020**, *8*, 77940–77950. [CrossRef]
8. Zhang, Y.; Liu, X.; Lang, X.; Zhang, Y.; Wang, C. VCLPKES: Verifiable certificateless public key searchable encryption scheme for industrial Internet of Things. *IEEE Access* **2020**, *8*, 20849–20861. [CrossRef]
9. 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]
10. Wei, Y.; Lv, S.; Guo, X.; Liu, Z.; Huang, Y.; Li, B. FSSE: Forward secure searchable encryption with keyed-block chains. *Inf. Sci.* **2019**, *500*, 113–126. [CrossRef]
11. Salmani, K. An Efficient, Verifiable, and Dynamic Searchable Symmetric Encryption with Forward Privacy. In Proceedings of the 2022 19th Annual International Conference on Privacy, Security & Trust (PST), Fredericton, NB, Canada, 22 August 2022.
12. Salimi, M. A New Efficient Identity-Based Encryption Without Pairing. *Cryptol. Eprint Arch.* **2021**, *2021*, 1–11.
13. He, K.; Chen, J.; Zhou, Q.; Du, R.; Xiang, Y. Secure dynamic searchable symmetric encryption with constant client storage cost. *IEEE Trans. Inf. Secur.* **2020**, *16*, 1538–1549. [CrossRef]
14. Cui, J.; Sun, Y.; Xu, Y.; Tian, M.; Zhong, H. Forward and backward secure searchable encryption with multi-keyword search and result verification. *Sci. China Inf. Sci.* **2022**, *65*, 1–3. [CrossRef]
15. Stefanov, E.; Papamanthou, C.; Shi, E. Practical dynamic searchable encryption with small leakage. *Cryptol. Eprint Arch.* **2013**, *2013*, 832–845.
16. 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, New York, NY, USA, 30 October 2017.
17. Ghareh Chamani, J.; Papadopoulos, D.; Papamanthou, C.; Jalili, R. New constructions for forward and backward private symmetric searchable encryption. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 15 October 2018.
18. Ding, X.; Cao, S.; Wang, C. Smart contract-assisted dynamically searchable encryption scheme with forward and backward security. *Comput. Eng.* **2022**, *48*, 141–150.
19. Sun, S.-F.; Yuan, X.; Liu, J.K.; Steinfeld, R.; Sakzad, A.; Vo, V.; Nepal, S. Practical backward-secure searchable encryption from symmetric puncturable encryption. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15 October 2018.
20. 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]

21. Chen, B.; Wu, L.; Kumar, N.; Choo, K.-K.R.; He, D. Lightweight searchable public-key encryption with forward privacy over IIoT outsourced data. *IEEE Trans. Emerg. Top. Comput.* **2019**, *9*, 1753–1764. [CrossRef]
22. Boneh, D.; Waters, B. Constrained pseudorandom functions and their applications. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, 1–5 December 2013.
23. Boneh, D.; Boyen, X. Efficient selective-ID secure identity-based encryption without random oracles. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, 2–6 May 2004.
24. Web Data: Amazon Reviews. Available online: https://snap.stanford.edu/data/web-Amazon.html (accessed on 12 December 2022).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.