

Article

Fast Multi-User Searchable Encryption with Forward and Backward Private Access Control

Salim Sabah Bulbul ¹, Zaid Ameen Abduljabbar ^{2,*}, Duaa Fadhel Najem ³, Vincent Omollo Nyangaresi ⁴, Junchao Ma ^{5,*} and Abdulla J. Y. Aldarwisch ²

- ¹ Directorate General of Education Basra, Ministry of Education, Basra 61004, Iraq; pgs2185@uobasrah.edu.iq
² Department of Computer Science, College of Education for Pure Sciences, University of Basrah, Basrah 61004, Iraq
³ Department of Cyber Security, College of Computer Science and Information Technology, University of Basrah, Basrah 61004, Iraq
⁴ Department of Computer Science and Software Engineering, Jaramogi Oginga Odinga University of Science & Technology, Bondo 40601, Kenya
⁵ College of Big Data and Internet, Shenzhen Technology University, Shenzhen 518118, China
* Correspondence: zaid.ameen@uobasrah.edu.iq (Z.A.A.); majunchao@sztu.edu.cn (J.M.)

Abstract: Untrusted servers are servers or storage entities lacking complete trust from the data owner or users. This characterization implies that the server hosting encrypted data may not enjoy full trust from data owners or users, stemming from apprehensions related to potential security breaches, unauthorized access, or other security risks. The security of searchable encryption has been put into question by several recent attacks. Currently, users can search for encrypted documents on untrusted cloud servers using searchable symmetric encryption (SSE). This study delves deeply into two pivotal concepts of privacy within dynamic searchable symmetric encryption (DSSE) schemes: forward privacy and backward privacy. The former serves as a safeguard against the linkage of recently added documents to previously conducted search queries, whereas the latter guarantees the irretrievability of deleted documents in subsequent search inquiries. However, the provision of fine-grained access control is complex in existing multi-user SSE schemes. SSE schemes may also incur high computation costs due to the need for fine-grained access control, and it is essential to support document updates and forward privacy. In response to these issues, this paper suggests a searchable encryption scheme that uses simple primitive tools. We present a multi-user SSE scheme that efficiently controls access to dynamically encrypted documents to resolve these issues, using an innovative approach that readily enhances previous findings. Rather than employing asymmetric encryption as in comparable systems, we harness low-complexity primitive encryption tools and inverted index-based DSSE to handle retrieving encrypted files, resulting in a notably faster system. Furthermore, we ensure heightened security by refreshing the encryption key after each search, meaning that users are unable to conduct subsequent searches with the same key and must obtain a fresh key from the data owner. An experimental evaluation shows that our scheme achieves forward and Type II backward privacy and has much faster search performance than other schemes. Our scheme can be considered secure, as proven in a random oracle model.



Citation: Bulbul, S.S.; Abduljabbar, Z.A.; Najem, D.F.; Nyangaresi, V.O.; Ma, J.; Aldarwisch, A.J.Y. Fast Multi-User Searchable Encryption with Forward and Backward Private Access Control. *J. Sens. Actuator Netw.* **2024**, *13*, 12. <https://doi.org/10.3390/jsan13010012>

Academic Editors: Donald Elmazi, Elis Kulla and Hakima Chaouchi

Received: 7 December 2023

Revised: 10 January 2024

Accepted: 18 January 2024

Published: 2 February 2024

Keywords: symmetric encryption; cloud computing; access control; multiple user; backward privacy



Copyright: © 2024 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

Following the rapid emergence of cloud computing, individuals and enterprises can now outsource storage and computation to the cloud [1–3]. The encryption of outsourced data before uploading to the cloud can prevent privacy leaks; however, data availability will be reduced if traditional encryption is used, as querying outsourced data is impossible in this case [4].

Searchable symmetric encryption (SSE) allows encrypted documents to be searched securely, thereby solving the problem described above [5–7]. Two methods can be used to design searchable encryption schemes: the first considers a single user who uploads encrypted data to the server and retrieves them when needed by requesting a query [8,9], and the second involves the data owner, who sends encrypted data to the server and allows many users to access them [10–13]. In addition to searching for documents, users sometimes need to upload their documents in realistic scenarios: for instance, collaborative e-health applications and projects are examples of situations in which a crowd of members need to upload relevant information, and then to search for and access each other's contributions depending on their roles. SSE was originally introduced by Song et al. [5] with the aim of facilitating efficient access to encrypted data while minimizing privacy constraints. During the execution of a user's query, SSE reveals some information to the server, which is known as leakage. An access pattern and a search pattern are usually included in the leak. Search patterns reveal which searches have the same keyword, whereas access patterns reveal the files returned by searches.

Dynamic searchable encryption with leakage mitigation SSE was applied to static datasets until 2009, when the principle of updating encrypted databases (described as dynamic SSE schemes) was first presented [14,15]. Existing dynamic symmetric searchable encryption (DSSE) schemes encounter substantial challenges, particularly in terms of storage complexity and overhead costs. While traditional searchable symmetric encryption (SSE) schemas designed for static databases excel in efficiently searching encrypted data, they fall short in accommodating dynamic updates, such as deletions and additions. The demand for adaptability in dynamic environments, characterized by constant changes in data, poses a significant hurdle for current schemes. Storage complexity has emerged as a primary concern within DSSE schemes, with many struggling to manage escalating storage requirements as datasets expand or undergo evolution. Furthermore, the security guarantees of certain schemes may be compromised when adapted for dynamic operations, creating a critical gap in ensuring the confidentiality and integrity of sensitive information. However, despite the added functionality, dynamic SSE raises privacy concerns. Dynamic, searchable cryptographic schemes face many challenges, including forward and backward specificity. The notion of forward privacy is straightforward: within a database, a file may encompass keywords that have been previously queried, even after their inclusion in the database. A brief discussion of this concept was presented in [16], and a more detailed discussion in [17–19]. Recently, file injection attacks were introduced in [20]. Dynamic SSE schemes now require forward privacy as an essential attribute. The concept of backward privacy is the second concept related to dynamic SSE: when a file is searched for in the present, the system does not reveal whether it was previously deleted from the encrypted database. This concept was first mentioned in [18], but without a formal definition, and has been expanded on by Bost et al. [21], who provided a formal definition of leakage with three levels and proposed a system that eliminated it.

- **Type I backward privacy leakage:** When a search for a keyword w is carried out, Type I schemes reveal how many updates were made to w , what type of updates were made, and when each update was made.
- **Type II backward privacy leakage:** As well as the leakage at the first level, the second level may also reveal the times at which all updates associated with the keyword were made.
- **Type III backward privacy leakage:** The last level is considered the weakest, as it reveals the deletion of a previously added keyword, in addition to the leakage at the above levels.

Multi-user searchable encryption (MUSE) is a cryptographic technique designed to facilitate encrypted data querying by multiple users, ensuring rigorous standards of privacy and security. It serves as a robust solution for integrating search capabilities with encrypted information, particularly in applications like cloud storage, where shared access to data is often necessary. MUSE leverages encryption to enable data exploration without requiring decryption, employing the searchable symmetric encryption (SSE) method. SSE empowers users to conduct searches within encrypted data while preventing unauthorized access. In the MUSE framework, each user possesses an individualized secret key critical for the encryption and decryption of data, enhancing the overall security and privacy measures. The encrypted data are subsequently stored on a server tasked with enabling search operations on the data. Whenever a user initiates a search for specific information, a search query is transmitted to the server, which employs the SSE scheme to conduct a search of the encrypted data and delivers the results back to the user. Throughout this process, data remain in an encrypted state, thereby preserving the paramount principles of privacy and security.

Two pressing issues remain unresolved, although researchers have provided solutions for certain scenarios such as that in [22]. The first problem is to find a lightweight approach to multi-user access control. Due to their intricate nature and the significant computational demands involved, existing multi-user SSE schemes lack a suitable access control method for all cases. The second requirement is for dynamic multi-user scenarios. In order to ensure secure uploading, forward privacy must also be provided [7,18] during the upload process. We developed an efficient multi-user SSE scheme that preserves forward privacy and efficiently controls access. The levels of access control privilege in our construction are symmetrical, and it is a privilege-based multi-user SSE scheme that is simpler and faster than attribute-based schemes with complex access structures.

In this paper, we present a novel searchable symmetric encryption (SSE) scheme designed to achieve robust access control in dynamic multi-user environments. Our approach introduces a unique combination of pseudorandom functions (PRFs) and forward-backward privacy preservation mechanisms, ensuring that updates are indistinguishable from random numbers, thereby guaranteeing both Type II backward and forward privacy. The novelty of our scheme lies in its efficient yet secure design, demonstrated through a comprehensive security analysis and experimental evaluations, where it consistently outperforms existing schemes, including *SSMDO* [23] and *Najafi* [24], in terms of index generation, search process efficiency, and overall system performance. In our approach, the server maintains an encrypted index that correlates individual keywords with specific sets of file identifiers, which denote the files containing those keywords. Each keyword entry is encrypted with a unique, fresh key, which is determined based on the keyword itself, the number of times that keyword has been updated, and a random value. A fresh random value is generated after each search query to ensure forward privacy. This fresh key is derived using a pseudorandom function from a master key held by the owner, and the owner is therefore required to keep records of the number of updates for each keyword and to present such a random value when needed. To conduct a keyword search, the owner must generate and disclose a ‘once’ key, which enables the user to interact with the relevant entries in the encrypted index, retrieve file identifiers associated with the keyword, and subsequently retrieve the corresponding encrypted files. Upon retrieval, the user decrypts the encrypted files, ensuring that results stored on the server are promptly deleted to achieve backward privacy. The key is then automatically updated to prevent its reuse by the same user or any other users for subsequent searches. This comprehensive system establishes a secure environment suitable for multiple users. We can summarize the main contributions of our model as follows:

1. We propose a multi-user searchable symmetric encryption scheme that uses symmetric keys, thus significantly reducing the computation cost of the scheme.
2. Our scheme is shown to be secure in multi-user scenarios via theoretical analyses of forward privacy and user security. Several experiments are also carried out to demonstrate that our design is efficient from the perspectives of computation and storage.
3. Our scheme achieves reverse privacy: search queries should not reveal deleted files, because the updating key is changed when each search query is finished. This type of privacy is lacking in most similar constructions.
4. We employ an optimized indexing system that marks and subsequently removes accessed entries, as the access pattern inadvertently reveals these results, and their encryption similarly exposes them.

The remainder of the paper is structured as follows. Related work is discussed in Section 2, and we present the problem statement in Section 3. The primitive tools and secure definitions used in our scheme are introduced in Section 4. The proposed scheme is described in detail in Section 5. Section 6 presents an analysis of the system and its performance. Section 7 concludes the paper.

2. Previous Work

Kamara et al. [15] presented the first scheme to explicitly support efficient database updates. In their later work, Kamara and Papamanthou [14] were able to reduce the amount of leakage in the first method. In a seminal paper [16], the concept of forward privacy was introduced as a novel idea for encryption schemes; however, it is important to highlight that the studies in [14,15] did not consider this concept in the design of their systems, meaning that there is potential susceptibility to privacy breaches of the data. Since this concept was introduced, it has become one of the basic pillars of preserving privacy for any searchable encryption scheme and has been extensively studied and applied, as described in references [17–19,25–27]. The concept of backward privacy was introduced for the first time by Stefanov et al. [18], although they provided no definition or construction of this concept. To the best of our knowledge, Bost et al. [21] were the first and only researchers to focus on the notion of backward privacy and to offer schemes for implementing it, as well as to present a formal definition of backward privacy. Specific constructions have been proposed to bolster privacy in various adversarial scenarios, as detailed in [28]. Nevertheless, it is imperative to note that these constructions do not accommodate multiple users and are tailored exclusively to an individual user of the system.

A multi-user scenario has not been seriously considered in SSE schemes for a long time. In one research study [22], an attempt was made to develop a multi-client SSE scheme based on an attribute-based encryption ABE scheme and OXT scheme, as described in [6]. In this proposed scheme, clients had the capability to establish access policies for their documents using their individual attributes, and subsequently to upload these documents to a server. Other clients could access these documents concurrently only if their own attributes were aligned with the access policy for the document. However, two key limitations were identified in [22]. Firstly, the integration of ABE introduces a considerable and potentially unnecessary overhead, given that more straightforward access control methods, such as privilege levels, would suffice for most use cases. Secondly, the system permits any client to upload documents without imposing constraints to address the corresponding security concerns, specifically the issue of forward or backward privacy. Stefanov et al. [18] claimed forward privacy by stating that the same search tokens should not match a newly inserted document. File injection attacks [20] can destroy encrypted systems when an SSE scheme lacks forward privacy. Owing to the need for and importance of accommodating multiple users in some systems, several schemes have been developed with the aim of addressing this requirement, as documented in [23,24,29]. The constructions outlined in these articles employed asymmetric encryption techniques, resulting in elevated computational costs during search and decryption processes. In addition to the substantial computational expenses associated with these systems [23,29],

they also failed to ensure backward privacy. We compare our proposed scheme with other schemes involving multiple users. To the best of our knowledge, there have been no previous works that have supported several users while offering backward privacy, as noted in Table 1. In addition, the approach of Zhang et al. [22] is the only one to support multiple users. In the scenarios considered here, fine-grained access control is not needed and imposes excessive computational burdens.

Table 1. Comparison of the proposed scheme with alternatives in the literature.

Scheme	Computation		Communication		BP	Multi-User
	Search	Update	Search	Update		
Moneta [17]	$\hat{O}(a_w \log N + \log^3 N)$	$\hat{O}(\log^2 N)$	$\hat{O}(a_w \log N + \log^3 N)$	$\hat{O}(\log^3 N)$	I	✗
Diana _{del} [17]	$O(a_w)$	$O(\log a_w)$	$O(n_w + n_w \log a_w)$	$O(1)$	III	✗
SD _a [30]	$O(a_w + \log N)$	$O(\log N)$	$O(a_w + \log N)$	$O(\log N)$	II	✗
π_{WBP} [28]	$O(o'_w)$	$O(1)$	$O(n_w)$	$O(1)$	III	✗
Najafi [24]	$O(n_w)$	$O(m)$	$O(SR)$	$O(1)$	III	✓
SSMDO [23]	$O(1)$	$O(m)$	$O((o_w + 1) * Q)$	$O(m)$	✗	✓
Ours	$O(o'_w)$	$O(1)$	$O(n_w)$	$O(1)$	II	✓

BP represents backward privacy, N indicates how many (keyword, identifier) mappings there are. m is the number of distinctive keywords. An addition operation on w is represented by a_w , a delete operation on w is denoted as d_w , and an update operation on w is represented by o_w (i.e., $o_w = a_w + d_w$). According to our last search, there have been o'_w updates, while n_w refers to the number of documents we currently have shared with w . \hat{O} masks the $\log N$ contents. ✓ indicates satisfied, while ✗ indicates not satisfied. SR is the size of the result set for w . A conjunctive query $Q = \{w_1, \dots, w_{|Q|}\}$, $|Q| > 1$.

3. Problem Statement

3.1. System Model

Figure 1 shows the SSE system, which consists of three distinct entities: data owners, cloud servers, and users, as follows:

- a. Data owners bear responsibility for the maintenance of the system and resolving issues. They play a pivotal role in the allocation and distribution of access tokens to users. Data owners are tasked with generating encryption keys, modifying databases, and creating keys for users upon request, particularly when users initiate search operations.
- b. Users have the authority to upload their own data to the cloud and to perform searches across documents uploaded by other users. Access tokens, which are furnished by data owners, enable users to formulate access policies when uploading documents. Users can only access documents for which their access tokens align with the access policies, thus ensuring secure access during query execution.
- c. Cloud servers within the SSE system are equipped with advanced computational and storage capabilities, thereby facilitating robust data processing and storage. When a user uploads ciphertext accompanied by its associated index, the cloud server assumes responsibility for data processing. In cases where a user dispatches a search token to the cloud server, it initiates the search operation and subsequently returns the corresponding ciphertext results. The functionality of this server is crucial in terms of ensuring the security and efficiency of the system's search operations.

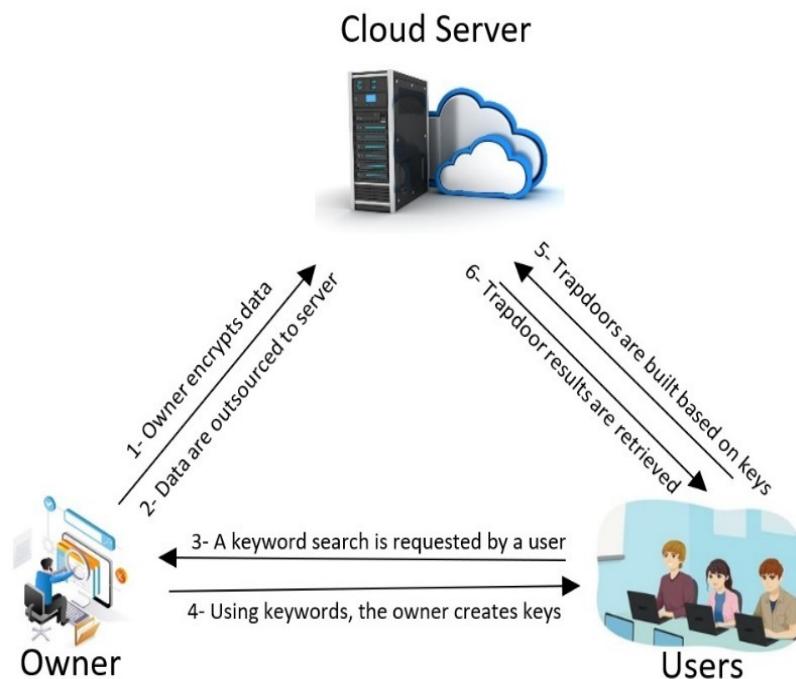


Figure 1. Model of the proposed system.

3.2. Security Assumptions

In the proposed scheme, the cloud server is presumed to adhere to the principles of integrity and some inquisitiveness. It is assumed that the cloud server will faithfully execute the prescribed protocols and procedures within this framework. Simultaneously, the cloud server undertakes the task of aggregating and scrutinizing the documents uploaded by users to extract additional privacy-related information. The owner generates a temporary key when a user requests a search on the server, which expires upon completion of the search operation. The key is essential for accessing encrypted data stored on the server, and users are unable to access the data without it.

3.3. Design Objectives

We present an efficient scheme for dynamic multi-user environments with the aim of facilitating straightforward queries while ensuring a high level of data user privacy. Our design therefore considered several security and performance objectives, as follows:

1. The owner has the exclusive capability to generate tokens from a master primitive key, which can subsequently be entrusted to authorized users as necessary. Subsequently, these users can perform queries on an encrypted database by utilizing the securely stored keywords at their disposal.
2. A multi-user environment is provided. Using the access tokens assigned to each user, the cloud server allows them to outsource their documents on behalf of all users and to search for outsourced documents that other users have contributed.
3. The computational cost is low. In our scheme, access control is implemented using only symmetric encryption. We contend that our proposed scheme will be less computationally expensive than previous schemes.
4. Forward and backward privacy are preserved. To achieve privacy, we use a counter based on keywords and a random number that is modified after each search to generate fresh keys. For these reasons, the cloud server cannot establish a connection between new documents that have been uploaded and previous search tokens.

4. Preliminaries

4.1. Notation

We use $r \xleftarrow{\$} \mathbf{X}$ to refer to an element r that has been randomly selected from a set of elements \mathbf{X} . Strings of length l are denoted as $\{0, 1\}^l$, while strings of arbitrary length are denoted as $\{0, 1\}^*$. Two strings a and b are concatenated using $a||b$. λ is the security parameter. We use $|S|$ to denote the cardinality of a set S and the symbol \mathcal{A} to represent the adversary server. \mathcal{L} denotes information that has been leaked during the operation of the system. $P(x; y)$ refers to a protocol executed in the context of a client–server setting where the protocol P is executed using the client’s input, denoted as x , and the server utilizes the input y , in accordance with the provided parameters. To implement the search protocol, we typically need $\mathcal{L}^{srch} = (ap, qp)$, where ap represents the *access pattern* and qp is the *query pattern* in the protocol.

4.2. Forward and Backward Privacy and Leakage Functions

In dynamic searchable encryption (DSE) systems, leakage functions play a crucial role in maintaining the integrity of the encrypted index as new documents are either added to or removed from the collection.

An effective searchable encryption scheme should disclose the least possible amount of information. Leakage can be captured using leakage functions. The setup protocol exclusively reveals \mathcal{L}^{stup} , encompassing solely the count of documents and keywords within the database DB , as well as the size of the database. In more formal terms, \mathcal{L}^{srch} maintains a history $Hist$ containing the history of all queries qi .

Forward privacy. Informally, forward private schemes are those where it is impossible to relate an operation to one that occurred previously. There is a strong property within DSSE that prevents the leakage of update operations during an update. In principle, updating a keyword requires that no information about it is leaked in the update query. Formally, it is defined as follows:

Definition 1. (Forward Privacy). SSE schemes with \mathcal{L} -adaptive security are forward private if used for an update query.

$$\mathcal{L}^{upd}(op, w, ind) = \mathcal{L}'^{upd}(op, ind).$$

A stateless function \mathcal{L}' is used here. This definition is the same as that in [17], where multiple keywords appear in the update query.

Backward privacy. Backward privacy is a critical property of searchable encryption schemes that ensures the confidentiality of previously submitted queries even in the event of server compromise or collusion. By deleting some entries from prior searches for a keyword w , the server does not learn any new information. When a search for a keyword w found in a previously deleted entry is performed, backward privacy guarantees that the server remains unaware of any additional information beyond the initial search. In an ideal scenario, the SSE scheme should not conceal previously removed entries, or at least their file identifiers, from an adversary [18]. The author of [21] provides a formal definition of three types of backward privacy, where Type I reveals the least information and Type III the most. In accordance with the notation used in [21], there are several functions that need to be added before we give our final definition.

Each query executed is represented by one entry in a list Q . In the case of a search, the entry will take the form (t, w) , where t denotes the timestamp for the query and w denotes the keyword searched. An example of an update would be $(t, op, (w, ind))$, where $= add$ or del , and ind is the file that has been modified. Let $\text{TimeDB}(w)$ be a function that, for a keyword w , returns a list of all the timestamps/file identifier pairs associated with keyword w that have

been added to DB during the term of this keyword and that have not been deleted during this time.

$$\text{TimeDB}(\mathbf{w}) = \{(t, \text{ind}) | (t, \text{add}, (w, \text{ind})) \in Q \text{ and } \forall t', (t', \text{del}, (w, \text{ind})) \notin Q\}$$

A function called **Updates(w)** records updates, timestamps t , deletions, and additions to a specific keyword w . Formally, we have the following:

$$\text{Updates}(\mathbf{w}) = \{t | (t, \text{add}, (w, \text{ind})) \in Q \text{ or } (t, \text{del}, (w, \text{ind})) \in Q\}.$$

A function called **DelHist(w)** returns to the adversary all pairs of insertion timestamps and deletion timestamps that are associated with deleted entries. The deletions corresponding to additions are explicitly revealed.

$$\text{DelHist}(\mathbf{w}) = \left\{ \left(t^{\text{add}}, t^{\text{del}} \right) \mid \exists \text{ind} : \left(t^{\text{add}}, \text{add}, (w, \text{ind}) \right) \in Q \text{ and } \left(t^{\text{del}}, \text{del}, (w, \text{id}) \right) \in Q \right\}$$

Clearly, leakage from these functions is increasing. As a result, backward privacy is now formally defined for different types of leaks.

Definition 2. ([21]). An \mathcal{L} -adaptively-secure DSSE scheme has backward privacy where $\mathcal{L} = (\mathcal{L}^{\text{stup}}, \mathcal{L}^{\text{srch}}, \mathcal{L}^{\text{upd}})$:

TypeI (BP level - I) : iff $\mathcal{L}^{\text{upd}}(op, w, \text{ind}) = \mathcal{L}'^{\text{upd}}(op)$ and $\mathcal{L}^{\text{srch}}(w) = \mathcal{L}''^{\text{srch}}\{\text{TimeDB}(w), a_w\}$

TypeII (BP level - II) : iff $\mathcal{L}^{\text{upd}}(op, w, \text{ind}) = \mathcal{L}'^{\text{upd}}(op, w)$ and $\mathcal{L}^{\text{srch}}(w) = \mathcal{L}''^{\text{srch}}\{\text{TimeDB}(w), \text{Updates}(w)\}$

TypeIII (BP level - III) : iff $\mathcal{L}^{\text{upd}}(op, w, \text{ind}) = \mathcal{L}'^{\text{upd}}(op, w)$ and $\mathcal{L}^{\text{srch}}(w) = \mathcal{L}''^{\text{srch}}\{\text{TimeDB}(w), \text{DelHist}(w)\}$

In this case, \mathcal{L}' and \mathcal{L}'' represent a stateless function.

A leak happens when a user actually retrieves the files. The definition given above presupposes that a scheme results in the inadvertent exposure of files currently holding the data w . Specifically, **TimeDB(w)** has the capability to reveal the indexes of the documents returned.

Pseudorandom functions. Let $\text{GenPRF}(1^\lambda) \in \{0,1\}^\lambda$ for key generation, and let the function $G: \{0,1\}^\lambda \times \{0,1\}^l \rightarrow \{0,1\}^{l'}$ be a family of pseudorandom functions (PRFs). For all PPT adversaries Adv , G is a secure PRF family, $|Pr[K \leftarrow \text{GenPRF}(1^\lambda); Adv_{G_K}(\cdot)(1^\lambda) = 1] - Pr[Adv_R()(1^\lambda) = 1]| \leq v(\lambda)$, where $R: \{0,1\}^l \rightarrow \{0,1\}^{l'}$ means the function is truly random.

5. Proposed Forward and Backward Multi-User Scheme

5.1. Overview

This section presents the proposed scheme, which to the best of our knowledge is the first to support multiple users with backward and forward privacy. We employ a straightforward method to store encrypted records in such a manner that no information is divulged to the server during updates, which include file insertions and deletions. The only information revealed to the server is the timing of each update that occurred while the records were stored in an encrypted state. An inverted index (address value) is used in this construction to store encrypted data of the form (ind, op) , where op refers to operation insertions or deletions and ind is the name of a specific file associated with these operations. Using a pseudorandom function, the owner can generate the set of locations that are associated with a keyword w for a given search operation using the address at which the values are stored in the hash table. This ensures that the user can efficiently produce a set of locations associated with a particular keyword. The author of [12] introduced a similar syntax based on asymmetric encoders; however, our scheme is based

on symmetric encoders, which makes searching and decoding much faster. Our scheme is also more secure, as privacy is ensured in both the forward and backward directions, which is not possible in [12]. This issue was effectively resolved in our scheme, which permits the creation of locations directly on the user's device. The decryption of entries also occurs locally on the user's device, thereby obviating the need to transmit the decryption key to the server for the purpose of creating location addresses and decrypting results. In the context of dynamic symmetric searchable encryption (DSSE), the assurance of dissociating previous search queries from subsequent ones necessitates forward privacy. Updates process, while backward privacy ensures the non-association of subsequent search requests for the retrieval of deleted documents from the past. This study introduces a comprehensive method for preserving both forward and backward privacy in DSSE. Notably, this strategy marks the development of the first operational and non-interactive Type II backward-private DSSE framework without reliance on secure execution environments. The proposed DSSE scheme achieves Type II backward and forward privacy by generating a unique one-time (fresh) key, denoted as K_g , for each search query, resulting in distinct encryption outcomes for each iteration. Additionally, it safeguards against the server by discerning the underlying operation (delete or add) embedded in the update query.

Setup. Managing the setup is the responsibility of the owner of the data. A secret key K_{SKE} is generated by the setup algorithm (Algorithm 1) upon entering the security parameters λ . The owner starts with four blank maps ($E_{DB}, P_{DB}, key_{Cnt}, Srch_{rn}$). In the first map E_{DB} , entries are encrypted and a cashed plaintext identifier is stored in P_{DB} ; these are then transmitted to the server, where they are stored for later modification through the update and search protocols. The other two maps are stored locally with the owner. A counter is stored on one map key_{Cnt} for each keyword, and a random value is stored on another map $Srch_{rn}$ for each keyword search.

Update. A specific file is updated (added or deleted) during the update procedure shown in Algorithm 2. In addition to receiving the file ind that contains a list of keywords, the owner will also receive the operation op (add or remove). The owner has access to the K_{SKE} key, the values of the local state search random $Srch_{rn}$, and keyword counter key_{Cnt} . $Srch_{rn}$ will be modified after each distinct keyword is searched for, and key_{Cnt} is a counter that indicates how many updates have been made for each distinct keyword. The generation of unique fresh keys for each update operation, resulting in distinct values for the same keyword on each update, introduces additional computational complexity cost to the system. However, the significant benefits derived from this practice include heightened data privacy protection and the alleviation of the need for clients to manage an extensive array of keys. The system incorporates one private key (K_{SKE}) exclusively stored with the data owner, reinforcing the security infrastructure. A fresh key K_g is generated by using K_{SKE} with a random value $Srch_{rn}$. Only one search is performed with this key K_g , and the search engine considers this to have expired once it has been used, so that users cannot use it again. Through the generation of a fresh key, denoted as K_g , for each update query, distinct encryption outcomes are achieved during each update operation (deletion or addition) for the same keyword. This approach aims to prevent information leakage and uphold the security properties of Type II backward privacy and forward privacy. Furthermore, it provides protection against the server by elucidating the underlying operation (delete or add) inherent in the update query.

Let us examine an illustrative example to elucidate the concept embodied by the entry (add, ind), signifying a request for the inclusion of a file in the encrypted database. Subsequently, we shall elucidate the operation of the update algorithm in the following manner. To add a keyword, it must be associated with a file. $FileCnt[w]$ is first checked for configuration by the owner. When this happens, the counter value key_{Cnt} for w is set to zero, and $Srch_{rn}$ to a random value. The counter is incremented in both cases (lines 2–4). To ensure backward privacy and one-time use when searching, a renewable key K_g is generated for each keyword. Next, the owner runs the PRF G with key K_g twice and calculates $G(K_g, w || key_{Cnt}[w] || 0)$ and $G(K_g, w || key_{Cnt}[w] || 1)$. $(ind || op)$ will be encrypted and

stored on the server side at the location $Addr_w$ extracted based on the first PRF output. In contrast, the second PRF output is XORed with the entry $(ind||op)$, resulting in an encrypted value Val_w , which is stored by the server (lines 6–7). A pair of addresses and values $(Addr_w, Val_w)$ is sent to the server, which stores them as $E_{DB}[Addr_w] = Val_w$ (line 9).

Algorithm 1 Setup $(\lambda; \perp)$

Owner:

- 1: $K_{SKE} \xleftarrow{\$} GenPRF(1^\lambda)$
 - 2: $E_{DB}, P_{DB}, key_{Cnt}, Srch_m \leftarrow$ empty map
 - 3: $\partial \leftarrow \{key_{Cnt}, Srch_m\}$ store in **Owner**
 - 4: Send E_{DB} To the **Server side**.
-

Algorithm 2 Update $(\partial, f, op, K_{SKE}; E_{DB})$

Owner:

- 1: For all $w \in f$
- 2: If $\partial[w] \leftarrow \{\}$
- 3: $key_{Cnt} = 0, Srch_m \xleftarrow{\$} \{0, 1\}^\lambda$
- 4: $key_{Cnt} ++$
- 5: $K_g \leftarrow G(K_{SKE}, w||Srch_m)$
- 6: $Addr_w \leftarrow G(K_g, w||key_{Cnt}[w]||0)$
- 7: $Val_w \leftarrow (ind||op) \oplus G(K_g, w||key_{Cnt}[w]||1)$
- 8: Send $\{Addr_w, Val_w\}$ to server

Server:

- 9: $E_{DB}[Addr_w] \leftarrow Val_w$
-

Search. Finally, we describe the algorithm that is used to search for keywords (Algorithm 3). A request containing a keyword is sent to the owner when the user U_k requests a search of the encrypted database stored on the cloud server. The owner searches for the keyword in the local state ∂ maps. A one-time key K_g is generated for the search and decryption if it exists, and the user U_k can use this. As well as generating a tag value K_t for w , the server also returns a list of document identifiers (if any exist) that are a match for w in the last search query, allowing the server to return the set of documents that match w (lines 2–3). The key K_g , K_t and the keyword counters $key_{Cnt}[w]$ are sent over a secure channel by the owner to U_k . $key_{Cnt}[w]$ represents a cumulative count of the updates made to w , which is used by U_k when searching for files containing this keyword. By employing key K_g and counter, a user generates a list L_{addr} that shows where corresponding entries in E_{DB} are located. This is accomplished by evaluating the PRF G on input $(K_g, w||i||0)$ for $i = 1, \dots, key_{Cnt}[w]$. Since G is a deterministic function, the locations that are calculated are the same as those obtained during the previous updates to w . The server then receives the list L_{addr} of locations (lines 6–9). E_{DB} contains the values of all locations in L_{addr} , which the server retrieves from E_{DB} as a result E_{Res} of the search query and the results of previous searches from $P_{DB}[K_t]$ (lines 10–15). When the encrypted values E_{Res} are received by the user, they are decrypted using the PRF outputs $G(K_g, w||i||1)$ for $i = 1, \dots, key_{Cnt}[w]$ and then XORed with the i-th encrypted value and the i-th PRF output. A key K_g that corresponds to a particular keyword will expire after a search is completed and the value $Srch_m$ will update. As a result, after the user filters the results Res , the server stores the plaintext version of the results in the map P_{DB} since is deemed an *access pattern* leakage that would be useless if it were encrypted (lines 10–15).

Algorithm 3 Search($\partial, w, K_{SKE}; E_{DB}$)

```

1: User  $U_k$  asks the owner to search for keyword  $w$ 
Owner:
2:  $K_g \leftarrow G(K_{SKE}, w||Srch_m)$ 
3:  $K_t \leftarrow G(K_{SKE}, w)$ 
4:  $Srch_m \leftarrow \{0,1\}^\lambda$ 
5: Send  $K_g, K_t, key_{Cnt}[w]$  to user  $U_k$ 
User  $U_k$ :
6:  $L_{addr} = \{\}$ 
7: for  $i = 1$  to  $key_{Cnt}[w]$ 
8:    $L_{addr} \leftarrow L_{addr} \cup G(K_g, w||i||0)$ 
9: Send  $\{L_{addr}, K_t\}$  to Server
Server:
10:  $E_{Res} = \{\}, Res = \{\}$ 
11:  $Res \leftarrow P_{DB}[K_t]$ 
12: for  $i = 1$  to  $|L_{addr}|$ 
13:    $E_{Res} \leftarrow E_{Res} \cup E_{DB}[L_{addr}[i]]$ 
14:   Delete  $E_{DB}[L_{addr}[i]]$ 
15: Send  $\{E_{Res}, Res\}$  to User  $U_k$ 
User  $U_k$ :
16: for  $i = 1$  to  $|E_{Res}|$ 
17:    $(ind || op) \leftarrow E_{Res}[i] \oplus G(K_g, w||i||1)$ 
18:   If  $op = add$ 
19:      $Res \leftarrow Res \cup ind$ 
20:   Else
21:      $Res \leftarrow Res / ind$ 
22: Send  $\{Res\}$  to Server
Server:
23:  $P_{DB}[K_t] \leftarrow Res$ 

```

5.2. Security Analysis

Type II backward and forward privacy is achieved by our scheme. As a result of the pseudorandom nature of G and the fact that the PRF consumes a different input during each update, forward privacy is assured when the two values ($Addr_w, Val_w$) observed by the server are indistinguishable from random numbers. Servers are not even notified of which operation has been executed (additions/deletions), so leakage cannot occur. In the process of querying for ‘w’, the server encounters numerous pseudorandom function (PRF) evaluations that it has previously encountered during updates, thereby risking potential breaches of backward privacy. This situation prompts the server to promptly disclose the timing of each update operation pertaining to ‘w.’ Moreover, the server’s perspective remains restricted to revealing only the information supplied by the owner, meaning it will not gain insight into operations such as deletions negating additions or vice versa.

Now that we have declared our scheme secure as the full proof follows later in the next section, we can state the following theorem:

Theorem 1. Assuming G is a secure PRF, our scheme is an adaptively secure DSSE scheme with

$$\mathcal{L}^{Stup} = \perp, \mathcal{L}^{Updt}(op, w, ind) = \perp \text{ and } \mathcal{L}^{Srch}(w) = (TimeDB(w), Updates(w))$$

6. Experimental Evaluation

6.1. Comparison of the Performance of Our Scheme

Proof of Theorem 1. In our scheme, the transcript is performed as follows: At each update request, data are stored as ($Addr, Val$) pairs in the encrypted database EDB on the server. EDB is initially empty. Also included are lists L_{addr} of locations of encrypted values retrieved from the server when a keyword is searched for. We assume that q is the total

length of the transcript. The simulator Sim operates as follows. On starting, Sim creates an empty EDB and an empty I list. By randomly sampling from G , Sim calculates $(\text{Addr}, \text{Val})$ during an update query. Let the timestamp of the update be i , and let t be the time it occurred. An entry in Sim is stored as $I(m) = (\text{Addr}, \text{Val})$. Null entries $I(m)$ are returned for timestamps m that do not correspond to updates. Leakage functions such as TimeDB (w) and Updates (w) are used in the simulator Sim during a keyword search. Using Updates (w), the timestamps of previously updated keywords are derived based on the timestamps of updates from Updates (w), represented as $M = (m_1, \dots, m_{a_w})$, and the addresses are sent to the server as they are stored in $I(m_i)$ for $m = 1, \dots, a_w$. The inference of document identifiers containing the search keyword from TimeDB(w) is carried out by the Sim . This is carried out when a keyword search is completed. Sim is used here to prove the security of our scheme. \square

Game₀. In this hybrid, it deals with the real system $DSSE_{\text{Real}}$, as shown in the following figure.

```

 $b \leftarrow DSSE_{\text{Real}}^{\Pi} q(\lambda)$ 
1:    $N \leftarrow \mathcal{A}(1^\lambda); (K_{SKE}, \partial_0, E_{DB0}) \leftarrow \text{Initialise}(1^\lambda, N);$ 
2:   for  $n = 1$  to  $q$  do
3:      $(type_n, ind_n, w_n) \leftarrow \mathcal{A}(1^\lambda, E_{DB0}, t_1, \dots, t_{n-1});$ 
4:     if  $type_n = \text{search}$  then
5:        $(\partial_n, DB(w_n); E_{DBn}) \leftarrow \text{Search}(K_{SKE}, w_n, \partial_{n-1}; E_{DBn-1})$ 
6:     else if  $type_n = \text{update}$  then
7:        $(\partial_n; E_{DBn}) \leftarrow \text{Update}(K_{SKE}, op, (ind_n, w_n), \partial_{n-1}; E_{DBn-1})$ 
11:     $b \leftarrow \mathcal{A}(1^\lambda, E_{DB0}, t_1, \dots, t_q);$ 
12:    return  $b;$ 

```

$$\mathbb{P}[DSSE_{\text{Real}}^{\Pi} q(\lambda) = 1] - \mathbb{P}[Game_0 = 1] = 0$$

Game₁. This hybrid is the same as for the previous game, except for one difference. A keyword's value $G(K_g, w || \text{key}_{C_{nt}}[w] || m)$ is computed during an update for m and is either zero or one, and the values are chosen randomly from the range of the function G . q entries are maintained in a list I . In the case where the i -th operation is an update, the entry $I(j) = (\text{Addr}, \text{Val}, w, ind)$ will be used to save the sampled random values along with the operation input (w, ind) for $j = 1, \dots, q$. If it is not an update operation, the entry will contain a null value. When searching for a particular keyword w , the game initiates a scan of I to find entries that are a match for w . Subsequently, the game sends the associated Addr values to the server and awaits a response. Afterward, the game conducts another scan of I to infer Res , which denotes the collection of documents that currently hold w . This set is then transmitted to the server.

The security of the PRF ensures that $Game_1$ cannot be differentiated from $Game_0$, since the PRF is never evaluated with the same input during updates in $Game_0$.

$$\mathbb{P}[Game_0 = 1] - \mathbb{P}[Game_1 = 1] \leq \mathcal{A}_G^{prf}(\lambda) \leq negl(\lambda)$$

Game₂. The game referred to as $DSSE_{\text{Ideal}}$ is formally defined in Figure 3a, and utilizes the Sim simulator as described above. It is evident that the generated transcript adheres to the same probability distribution as the one created during $Game_1$.

This is because the leakage functions correspond to identical values computed in that game, and when a uniformly random value rv is selected, the resulting $ind || op \oplus rv$ also follows a uniform random distribution.

```

 $b \leftarrow DSSE_{Ideal}^{\Pi}_{\mathcal{A}, Sim} q(\lambda)$ 
1:    $N \leftarrow \mathcal{A}(1^\lambda); (\partial_{Sim}, E_{DB0}) \leftarrow SimInitialise(1^\lambda, N);$ 
2:   for  $n = 1$  to  $q$  do
3:      $(type_n, ind_n, w_n) \leftarrow \mathcal{A}(1^\lambda, E_{DB0}, t_1, \dots, t_{n-1});$ 
4:     if  $type_n = \text{search}$  then
5:        $(\partial_{Sim}; t_n, E_{DBn}) \leftarrow SimSearch(\partial_{Sim}, \mathcal{L}^{srch}(w_n); E_{DBn-1})$ 
6:     else if  $type_n = \text{update}$  then
7:        $(\partial_{Sim}; E_{DBn}) \leftarrow SimUpdate(\partial_{Sim}, \mathcal{L}^{upd}(w_n); E_{DBn-1})$ 
11:     $b \leftarrow \mathcal{A}(1^\lambda, E_{DB0}, t_1, \dots, t_q);$ 
12:    return  $b;$ 

```

$$P\left[DSSE_{Real}^{\Pi}_{\mathcal{A}} q(\lambda) = 1\right] - P\left[DSSE_{Ideal}^{\Pi}_{\mathcal{A}, Sim} q(\lambda) = 1\right] \leq \mathcal{A}_G^{prf}(\lambda) \leq negl(\lambda)$$

In regard to the accuracy of our approach, it should be mentioned that if G is not a pseudorandom function (as it is in our implementation), there is a possibility of collisions arising when calculating $Addr$ and Val for various w and ind pairs. This probability can be significantly reduced by expanding the range of G .

6.2. Performance Evaluation

The SSMDO [23] and Najafi [24] share similarities with our proposed approach in terms of achieving privacy for multiple users, and we, therefore, compared the execution times of these algorithms. This comparison included evaluating the time taken for index generation, client storage, and the search process. We carried out these experiments on a single computer, and the encrypted database was kept in the memory of the same computer, without the need for a WAN network. Two computers in different countries (Iraq and the UAE) were used to calculate the round-trip time for the data, and the result was approximately 2 ms. Our main focus was to calculate the time spent on searching and updating operations, in addition to the communication size and the amount of storage required for each plan on the client side.

To implement the proposed scheme and assess its efficiency, we used the Java language on the Windows 11 operating system, with an x64-based processor. The comparison schemes were applied to a real dataset called Enron [31]. The selected ranges in the experimental evaluation, including dataset size, dictionary size, and other parameters, are pivotal for comprehending the performance and generalizability of the proposed (DSSE) scheme in real-world scenarios. The dataset comprises an extensive collection of textual email exchanges among Enron employees. Notably, the dataset is characterized by its considerable size, offering a diverse and realistic sample that proves valuable for testing information retrieval systems. The test results were obtained on a computer equipped with an Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz, 8GB of RAM, and a 512 GB SSD hard disk. To enable a comparative analysis of our proposed scheme with existing ones, the time cost and running time results for SSMDO and Al-Najafy, as reported in [24], have been referenced.

Metrics include index generation time, measuring the efficiency of the setup process; search process efficiency, evaluating the time complexity of the search algorithm; communication overhead, quantifying data transmission during operations; storage requirements, analyzing client-side storage; and round-trip time for data transfer, simulating real-world scenarios. The Enron [31] dataset served as the foundation for evaluations, conducting experiments on computers across various countries to authentically assess communication expenditures. Recognized as an authentic database and a live illustration of the system's application in a real-world environment, this approach allowed for a comprehensive and realistic evaluation of the system's performance. The results, analyzed in the experimental evaluation, reveal the DSSE scheme's remarkable efficiency in index generation, outper-

forming existing methodologies. The search process efficiency, communication overhead, and storage requirements demonstrate the lightweight nature of the design. The round-trip time for data transfer showcases the scheme's practicality in a distributed environment. The DSSE scheme ensures privacy preservation and exhibits superior efficiency. The comprehensive experimental evaluation validates the approach, with future work focusing on enhancing system security through a verification mechanism for the authenticity and integrity of results.

Index generation. In the proposed scheme, the encrypted index is created by performing a continuous update process for each keyword. The index is represented as a set of ordered pairs consisting of a value and a key. We note that our scheme utilizes symmetric encryption. The updating procedure exhibits a time complexity of $O(1)$, indicating that as the size of the encrypted index increases, the required time for the process does not escalate linearly. This aspect is straightforward in our scheme. In the *SSMDO* and *Najafi* methodologies for constructing encrypted indexes, the index data structure comprises two numerical components along with a vector component. It also keeps track of the size of the dictionary used in the system.

The *Najafi* approach has two additional numerical components in the index structure, meaning that more exponentiation operations need to be performed. The time it takes to generate the index is influenced by two factors: the number of documents in the dataset and the total number of keywords defined in the system.

From Figure 2, we can see the variations in the time it takes to generate an index due to changes in these two factors across the three different schemes (*SSMDO*, *Najafi*, and our proposed scheme). By comparing the diagrams, we can see that our scheme performs the index generation process in a shorter amount of time than the other schemes, indicating that it is the most efficient in terms of index generation.

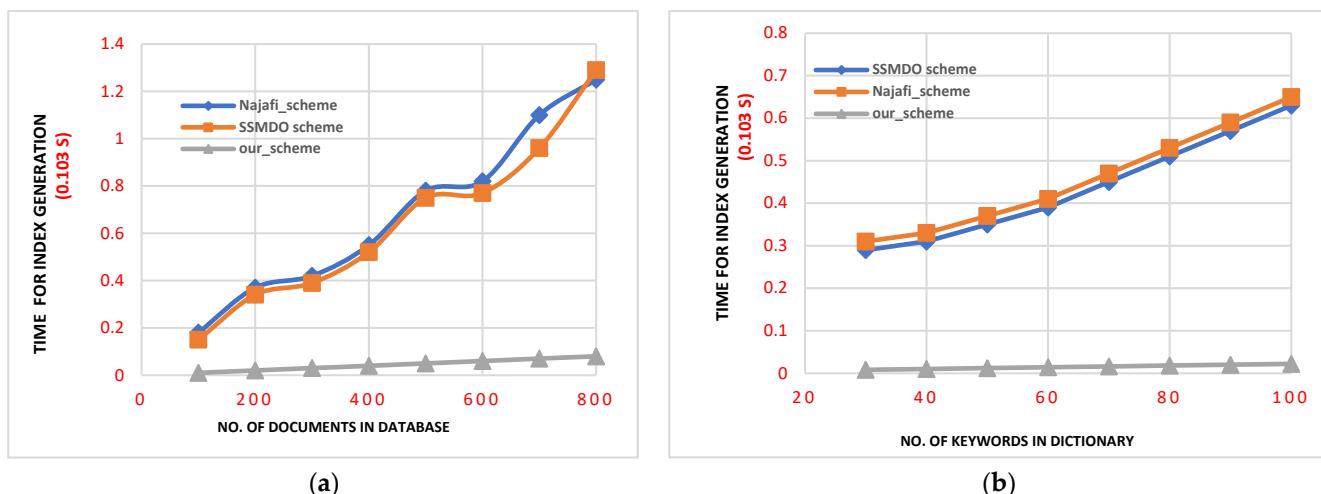


Figure 2. (a) The impact of dataset size variations with a constant dictionary size of $n = 50$. (b) The effects of dictionary size variations while maintaining a constant dataset size of $m = 262$.

Figure 2 shows the time taken to generate an encrypted index in two different scenarios. Figure 2a shows the impact of varying the dataset size while keeping the dictionary size constant at $n = 50$, while Figure 2b shows the effect of varying the dictionary size while maintaining a constant dataset size of $m = 262$. The figures provide insights into how the index generation time changes in relation to these factors.

Trapdoor generation. It should be noted that the schemes under comparison outperform our proposed scheme for trapdoor generation. This distinction arises from the fact that our scheme calculates the locations of values in the encrypted database for a specific keyword at the client before initiating the search query for that keyword. In contrast, in our scheme, the time expended by the server in retrieving results is notably quicker than in comparable schemes. This discrepancy underscores the efficiency of our scheme in the subsequent comparison.

Search process. The selection of dataset size and dictionary size for experimental variations is a nuanced process influenced by practical considerations and relevance to real-world scenarios. The chosen ranges (100 to 800 for dataset size and 30 to 100 for the number of keywords) are motivated by several factors. They may reflect realistic expectations in the application domain, mirroring document or record quantities in a typical database, and the diversity of searchable terms. Additionally, the ranges enable scalability assessments, testing the system's performance with varying dataset and dictionary sizes. Considerations for resource constraints, trade-off analysis, benchmarking against other schemes, and statistical significance underscore the deliberate and comprehensive nature of the experimental design.

The figures presented above illustrate a direct correlation between the size of the result set and the corresponding search time across all schemes, indicating a linear increase in search time as result size grows. Bilinear pairing plays an important role in the search algorithm for both the *SSMDO* and *Najafi* schemes; however, this imposes a significant computational overhead, and the frequency of its application directly impacts the time complexity of the search algorithm. From Figure 3, we can observe how the time complexity changes for each scheme as we increase two key factors: the dataset size and the dictionary size. For all of the executions, our scheme consistently outperformed the *Najafi* scheme, which is the only other scheme with the same type of information leakage. The predominant portion of the time is allocated to client-side operations within our proposed scheme, with the server primarily tasked with retrieving precomputed location results, thus minimizing its computational workload. Figure 3a demonstrates that as the dataset size increases, the time cost of the search process also increases almost linearly. Interestingly, despite using the same number of pairings, the *SSMDO* and *Najafi* schemes show a steeper increase in the search time cost compared to our scheme. This suggests that the search algorithm used in our approach is more efficient. Furthermore, Figure 3b shows that the time cost of the search algorithm remains relatively independent of the size of the dictionary; in other words, the size of the dictionary does not significantly impact the time required for the search process. Overall, these findings highlight the efficiency and effectiveness of the proposed scheme's search algorithm in comparison to the *SSMDO* and *Najafi* schemes, considering the computational overhead of the bilinear pairing map. This result is not surprising, since the *SSMDO* and *Najafi* schemes rely on bilinear pairings to achieve privacy in both directions, whereas our scheme uses symmetric encryption. Figure 3 shows the time taken for a search operation, measured in terms of the time cost. Figure 3 is divided into two parts: Figure 3a represents the variations in search time with different dataset sizes while maintaining a constant dictionary size of $n = 50$, and Figure 3b showcases the changes in search time with different dictionary sizes while keeping the dataset size constant at $m = 262$. These figures help us understand how the search time is influenced by these factors.

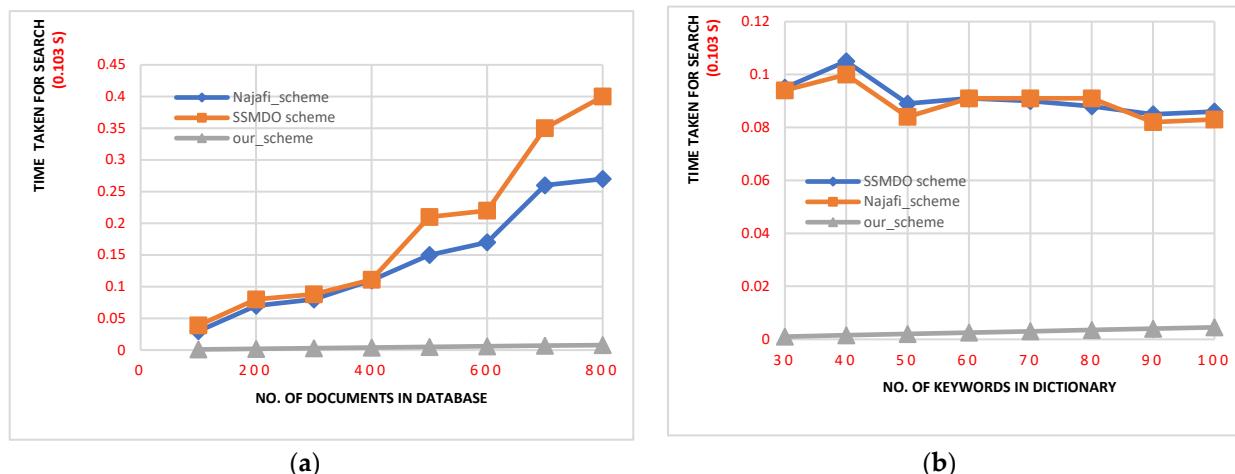


Figure 3. (a) The relationship between search time and different dataset sizes, holding a constant dictionary size of $n = 50$. (b) The impact of various dictionary sizes on search time, with a fixed dataset size of $m = 262$.

7. Conclusions

This paper has presented a searchable symmetric encryption scheme that can be used to achieve effective access control in a dynamic multi-user environment. We must contend with two essential properties when designing an SSE scheme: (forward and backward) privacy and efficiency. Since granular access control faces problems when used in similar scenarios, an SSE based on simple tools was introduced to achieve high security and retrieval speed. We also noted the importance of advanced privacy in multi-user scenarios and showed that our scheme achieved forward and backward privacy preservation through a security analysis. Extensive experiments also indicated that our proposed scheme was feasible for real-world scenarios involving multiple users due to the accounts, storage, and connections used, which meant that our build was lightweight.

The paper identifies potential limitations and outlines areas for future research in the proposed scheme. It acknowledges the risk of collisions in the pseudorandom function (G) when calculating $Addr$ and Val , suggesting future research to mitigate these risks through strategies like expanding the function range. Another avenue for improvement involves implementing a verification mechanism to ensure the authenticity and integrity of results, warranting the exploration of robust mechanisms. This additional layer of security will provide a robust defence against any potential attacks, making the system completely secure and reliable. Scalability, resource utilization, real-world deployment challenges, and extension to different use cases are highlighted as further research areas. The proposed scheme can also be integrated with fine-grained access control technology. Additionally, a call for a more quantitative analysis of security guarantees in various threat models is emphasized for future investigations, aiming to enhance the proposed scheme's contributions and practical applicability.

Author Contributions: All authors have contributed equally to this article. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the Natural Science Foundation of Top Talent of SXTU (grant No. 20211061010016).

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

References

1. Al Sibahee, M.A.; Lu, S.; Abduljabbar, Z.A.; Liu, X.; Abdalla, H.B.; Hussain, M.A.; Hussien, Z.A.; Ghrabat, M.J.J. Lightweight secure message delivery for E2E S2S communication in the IoT-cloud system. *IEEE Access* **2020**, *8*, 218331–218347. [[CrossRef](#)]
2. Abduljabbar, Z.A.; Jin, H.; Ibrahim, A.; Hussien, Z.A.; Hussain, M.A.; Abbdal, S.H.; Zou, D. Secure Biometric Image Retrieval in IoT-Cloud. In Proceedings of the 2016 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), Hong Kong, China, 5–8 August 2016; pp. 1–6.
3. Hussain, M.A.; Hussien, Z.A.; Abduljabbar, Z.A.; Ma, J.; Al Sibahee, M.A.; Hussain, S.A.; Nyangaresi, V.O.; Jiao, X. Provably throttling SQLI using an enciphering query and secure matching. *Egypt. Inform. J.* **2022**, *23*, 145–162. [[CrossRef](#)]
4. Abduljabbar, Z.A.; Jin, H.; Ibrahim, A.; Hussien, Z.A.; Hussain, M.A.; Abbdal, S.H.; Zou, D. Sepim: Secure and efficient private image matching. *Appl. Sci.* **2016**, *6*, 213. [[CrossRef](#)]
5. 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, 14–17 May 2000; pp. 44–55.
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 Proceedings of the Advances in Cryptology—CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2013; Proceedings, Part I. Springer: Berlin/Heidelberg, Germany, 2013; pp. 353–373.
7. Lai, S.; Patranabis, S.; Sakzad, A.; Liu, J.K.; Mukhopadhyay, D.; Steinfeld, R.; Sun, S.-F.; Liu, D.; Zuo, C. Result Pattern Hiding Searchable Encryption for conjunctive Queries. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 745–762.
8. Abduljabbar, Z.A.; Ibrahim, A.; Hussain, M.A.; Hussien, Z.A.; Al Sibahee, M.A.; Lu, S. EEIRI: Efficient encrypted image retrieval in IoT-cloud. *KSII Trans. Internet Inf. Syst. (TIIS)* **2019**, *13*, 5692–5716.
9. Sun, S.-F.; Zuo, C.; Liu, J.K.; Sakzad, A.; Steinfeld, R.; Yuen, T.H.; Yuan, X.; Gu, D. Non-interactive multi-client searchable encryption: Realization and implementation. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 452–467. [[CrossRef](#)]
10. Huaze, L.; Kaiping, X.; David, S.L.W.; Ruidong, L. An efficient multi-user multi-keyword fuzzy search scheme over encrypted cloud storage. *J. Univ. Sci. Technol. China* **2021**, *51*, 1361–1382.
11. Sun, S.-F.; Liu, J.K.; Sakzad, A.; Steinfeld, R.; Yuen, T.H. An Efficient Non-Interactive Multi-Client Searchable Encryption with Support for Boolean Queries. In *European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 154–172.
12. Du, L.; Li, K.; Liu, Q.; Wu, Z.; Zhang, S. Dynamic multi-client searchable symmetric encryption with support for boolean queries. *Inf. Sci.* **2020**, *506*, 234–257. [[CrossRef](#)]
13. Al Sibahee, M.A.; Lu, S.; Abduljabbar, Z.A.; Ibrahim, A.; Hussien, Z.A.; Mutlaq KA, A.; Hussain, M.A. Efficient encrypted image retrieval in IoT-cloud with multi-user authentication. *Int. J. Distrib. Sens. Netw.* **2018**, *14*, 1550147718761814. [[CrossRef](#)]
14. Kamara, S.; Papamanthou, C. Parallel and Dynamic Searchable Symmetric Encryption. In *International Conference on Financial Cryptography and Data Security*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 258–274.
15. Kamara, S.; Papamanthou, C.; Roeder, T. Dynamic Searchable Symmetric Encryption. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 965–976.
16. Chang, Y.-C.; Mitzenmacher, M. Privacy Preserving Keyword Searches on Remote Encrypted Data. In *International Conference on Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 442–455.
17. Bost, R. Σ οφοց: 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.
18. Stefanov, E.; Papamanthou, C.; Shi, E. Practical Dynamic Searchable Encryption with Small Leakage. *Cryptology ePrint Archive* **2013**.
19. Etemad, M.; Küpcü, A.; Papamanthou, C.; Evans, D. Efficient dynamic searchable encryption with forward privacy. *Proc. Priv. Enhancing Technol.* **2018**, *2018*, 5–20. [[CrossRef](#)]
20. Huang, Y.; Lv, S.; Liu, Z.; Song, X.; Li, J.; Yuan, Y.; Dong, C. Cetus: An efficient symmetric searchable encryption against file-injection attack with SGX. *Sci. China Inf. Sci.* **2021**, *64*, 182314. [[CrossRef](#)]
21. 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.
22. Zhang, K.; Wen, M.; Lu, R.; Chen, K. Multi-client sub-linear boolean keyword searching for encrypted cloud storage with owner-enforced authorization. *IEEE Trans. Dependable Secur. Comput.* **2020**, *18*, 2875–2887. [[CrossRef](#)]
23. Guo, Z.; Zhang, H.; Sun, C.; Wen, Q.; Li, W. Secure multi-keyword ranked search over encrypted cloud data for multiple data owners. *J. Syst. Softw.* **2018**, *137*, 380–395. [[CrossRef](#)]
24. Najafi, A.; Bayat, M.; Javadi, H.H.S. Fair multi-owner search over encrypted data with forward and backward privacy in cloud-assisted Internet of Things. *Future Gener. Comput. Syst.* **2021**, *124*, 285–294. [[CrossRef](#)]
25. Kim, K.S.; Kim, M.; Lee, D.; Park, J.H.; Kim, W.-H. Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1449–1463.
26. Cash, D.; Tessaro, S. The Locality of Searchable Symmetric Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 351–368.

27. Naveed, M.; Prabhakaran, M.; Gunter, C.A. Dynamic Searchable Encryption via Blind Storage. In *2014 IEEE Symposium on Security and Privacy*; IEEE: Berlin/Heidelberg, Germany, 2014; pp. 639–654.
28. Chatterjee, S.; Puria, S.K.P.; Shah, A. Efficient backward private searchable encryption. *J. Comput. Secur.* **2020**, *28*, 229–267. [[CrossRef](#)]
29. Yang, J.; Liu, F.; Luo, X.; Hong, J.; Li, J.; Xue, K. Forward Private Multi-Client Searchable Encryption with Efficient Access Control in Cloud Storage. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*; IEEE: Berlin/Heidelberg, Germany, 2022; pp. 3791–3796.
30. Alyousif, A.; Yassin, A.; Abduljabbar, Z.; Xu, K. Improving the performance of searchable symmetric encryption by optimizing locality. *J. Basrah Res.* **2023**, *49*, 102–113. [[CrossRef](#)]
31. Enron Email Dataset. Available online: <https://www.cs.cmu.edu/~enron/> (accessed on 10 November 2023).

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.