

Efficient Strong Privacy-Preserving Conjunctive Keyword Search Over Encrypted Cloud Data

Chang Xu, *Member, IEEE*, Ruijuan Wang, Liehuang Zhu, *Member, IEEE*, Chuan Zhang, *Member, IEEE*, Rongxing Lu, *Fellow, IEEE*, and Kashif Sharif, *Senior Member, IEEE*

Abstract—Searchable symmetric encryption (SSE) supports keyword search over outsourced symmetrically encrypted data. Dynamic searchable symmetric encryption (DSSE), a variant of SSE, further enables data updating. Most DSSE works with conjunctive keyword search primarily consider forward and backward privacy. Ideally, the server should only learn the result sets involving all keywords in the conjunction. However, existing schemes suffer from keyword pair result pattern (KPRP) leakage, revealing the partial result sets containing two of query keywords. We propose the first DSSE scheme to address aforementioned concerns that achieves strong privacy-preserving conjunctive keyword search. Specifically, our scheme can maintain forward and backward privacy and eliminate KPRP leakage, offering a higher level of security. The search complexity scales with the number of documents stored in the database in several existing schemes. However, the complexity of our scheme scales with the update frequency of the least frequent keyword in the conjunction, which is much smaller than the size of the entire database. Besides, we devise a least frequent keyword acquisition protocol to reduce frequent interactions between clients. Finally, we analyze the security of our scheme and evaluate its performance theoretically and experimentally. The results show that our scheme has strong privacy preservation and efficiency.

Index Terms—strong privacy-preserving, DSSE, conjunctive keyword search.

1 INTRODUCTION

WITH the advent of cloud computing, outsourcing the storage and processing of large data collection to third-party servers has gained significant popularity. However, data privacy is a primary concern as the third-party cloud server cannot be fully trusted. These privacy issues may cause reputational damage and/or location leakage [1], [2]. Generally, users do not desire that their sensitive data be disclosed to an untrusted server. A straightforward solution is to encrypt the data before uploading it to the cloud server. However, this limits the search operations, as encrypted data may not be directly searchable.

To address these challenges, the searchable encryption (SE) technology is proposed to achieve keyword searching over ciphertext without revealing sensitive data and queries to the cloud server. Searching over encrypted data inevitably reduces search efficiency; hence, the SE schemes try to maintain an acceptable compromise between security and efficiency. In recent years, a series of schemes have been proposed [3], [4], [5], [6], which enable the data owner to encrypt the data and generate encrypted indices for searching. The cloud server can then retrieve the stored ciphertext based on the received search tokens.

In general, searchable encryption can be divided into two representative techniques: Symmetric searchable en-

ryption (SSE), and Public-key searchable encryption. SSE has been extensively studied, and various schemes [7], [8], [9], [10] have been presented successively. However, the majority of schemes are severely limited to support single keyword search. In realistic scenarios, conjunctive keyword search is more appropriate, such as, e-health systems [11], task recommendation systems [12], [13], and e-mail systems [14]. Hence, we focus on conjunctive keyword search in SSE in this work.

A simple solution to support conjunctive keyword search has been described in [15]. This solution returns the intersection of search results of each single keyword in the conjunction. Thus, this search method is inefficient due to repeated searches in the database. Furthermore, it will breach the searched data's secrecy. The cloud server should be allowed to acquire only the encrypted file identifiers corresponding to all keywords in the conjunctive search query $q = (w_1 \wedge \dots \wedge w_n)$. To maintain this secrecy, Cash et al. [3] proposed the first sub-linear SSE protocol supporting conjunctive keyword search, named Oblivious Cross-Tags (OXT). The earlier SSE constructions [16] scale linearly with the size of the entire database. Sub-linear means that the search complexity is independent of the total number of documents stored in the database. To reduce search overhead, the complexity of OXT is proportional to the number of matches involving the least frequent keyword as the s -term in the conjunction. Subsequently, many conjunctive SSE schemes based on OXT have been proposed [17], [18], [19].

According to Lai et al.'s scheme [5], the OXT protocol suffers Keyword Pair Result Pattern (KPRP) leakage during the search phase. This means that for an n keywords conjunctive search query q , the server can learn the encrypted file identifiers involving each pair of query

- Chang Xu, Liehuang Zhu, and Chuan Zhang are with School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China.
E-mail: {xuchang, liehuangz, chuanz}@bit.edu.cn
- Ruijuan Wang and Kashif Sharif are with School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.
E-mail: {ruijuanw, kashif}@bit.edu.cn
- Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada.
E-mail: rlu1@unb.ca.

TABLE 1
Functionality Comparison of Existing Schemes and This Work.

	Conjunctive keyword	Forward privacy	Backward privacy	Keyword pair result privacy	Non-interactive
OXT [3]	✓	✗	✗	✗	✗
HXT [5]	✓	✗	✗	✓	✗
ODXT [20]	✓	✓	✓	✗	✗
SE-EPOM [6]	✓	✗	✗	✗	✓
Ours	✓	✓	✓	✓	✓

keywords $(w_1, w_i), 2 \leq i \leq n$, where the keyword w_1 is assumed to be the s -term. The file-injection attack [21] resorts to using KPRP leakage to expose all keywords in a conjunctive query with 100% accuracy. To eliminate KPRP leakage, Lai et al. [5] proposed the Hidden Cross-Tags (HXT) protocol. Unfortunately, all of the above SSE schemes are limited to static databases.

Adding and deleting files is generally required in real-world scenarios to dynamically update the database, which raises security concerns. For instance, the server might intentionally match the previous search tokens with the newly added file indexes to infer the keywords contained in the file and infer the keywords of the query from repeated search queries. Zhang et al. [21] presented file-injection attacks. Specifically, an untrusted server first crafts a set of files and tricks the client into encrypting them. After that, the server searches for the injected files using the prior tokens. According to the known keywords, the server can infer which keyword is involved in the token. File-injection attacks are effective to break the privacy of client queries and require only a small number of injected files. This attack was also described as leakage-abuse attacks in known-document and chosen-document attack setting by Cash et al. [22]. File-injection attacks and leakage-abuse attacks are based on the prior knowledge of an adversarial server. File-injection attacks require less prior knowledge and the server must inject files. Therefore, we consider the above attacks. To deal with the attacks, forward privacy and backward privacy were first introduced informally in by Stefanov et al. [23], and later formalized by Bost et al. [24], [25]. **Forward privacy** ensures that the newly updated files cannot be linked with the previously executed search, which prevents the server from inferring the keywords. **Backward privacy** requires that deleted files cannot be retrieved in subsequent search queries. That is, search queries should not reveal information about files that have already been deleted from the database.

Most of the published forward and backward private DSSE schemes only support single keyword search. For conjunctive queries, Zuo et al. [26] proposed FBDSSE-CQ scheme using the extended bitmap index and achieving both forward and backward privacy. However, the number of keywords in a conjunction query is limited. To date, for the existing works that is not limited by the number of search keywords, only the Oblivious Dynamic Cross Tags (ODXT) scheme [20] supports conjunctive keyword search guaranteeing forward and backward privacy. However, there is still the threat of KPRP leakage.

Besides, the previous OXT-based SSE protocols require

the search user to ask the data owner during each search because the least frequent keyword in the conjunction should be known in advance. However, if a search user initiates distinct search queries multiple times, it should interact with the data owner for each query, and such frequent interactions cause additional communication overhead problems. In actuality, the data owner outsources the data to the server in the expectation of not having to do anything else but update the data and distribute secret keys. This work aims to propose a DSSE scheme that guarantees forward and backward privacy and eliminates the KPRP leakage to achieve strong privacy, and enable search users to obtain the least frequent keyword without interacting with the data owner.

Our Contributions. This work develops a solution for the privacy-preserving leakage problem of conjunctive DSSE and proposes an Efficient Strong Privacy-preserving Conjunctive Keyword Search (ESP-CKS) scheme. It ensures forward and backward privacy of DSSE and avoids keyword pair result pattern leakage. Table 1 shows a comparison of our scheme with prior representative works for conjunctive keyword search. Our contributions are listed below in detail.

- We propose the first strong privacy-preserving conjunctive keyword search scheme named ESP-CKS. It guarantees forward and backward privacy for dynamic databases. Besides, our scheme can prevent KPRP leakage in conjunctive search queries. Although the previous schemes support forward and backward privacy, they suffer from KPRP leakage.
- Our ESP-CKS scheme's search complexity is independent of the total number of documents in the database, but scales with the update frequency of the least frequent keyword in the conjunction. To address the issue of frequent interactions between the data owner and search users in previous work, we propose the first non-interactive least frequent keyword acquisition protocol.
- We conduct extensive experiments to evaluate the scheme's performance in terms of storage, computation, and communication. We prove that our scheme is more secure and efficient than the other two conjunctive keyword search approaches.

2 PROBLEM FORMULATION

This section describes the system model and then introduces the threat model and design goals.

2.1 System model

The system consists of three main entities as shown in Figure 1: a data owner, a cloud server, and search users.

- **Data Owner:** The data owner is in charge of generating the system parameters and the secret key. The data owner encrypts documents via symmetric encryption and generates related indexes according to system parameters. Then it submits both indexes and encrypted documents to the cloud server. It can also add and delete documents from the encrypted database on the cloud server.
- **Cloud Server:** The cloud server holds unlimited computation and storage capacities. It stores documents and the indexes delivered by the data owner and handles search queries.
- **Search Users:** A search user is permitted to search the documents stored on the cloud server. It computes search tokens for desired keywords and sends them to the cloud server. After receiving the search results from the cloud server, it decrypts them to obtain the target documents' indexes.

We call the data owner and search users as clients, and the cloud servers as the server.

2.2 Threat Model

In our schemes, the data owner and search users are assumed to be fully trusted. Specifically, they honestly follow the protocol and never expose its encryption keys to other entities except for permitted search users. The cloud server is considered honest but curious and always honestly executes the protocol but may attempt to infer information about data objects and the content of queries.

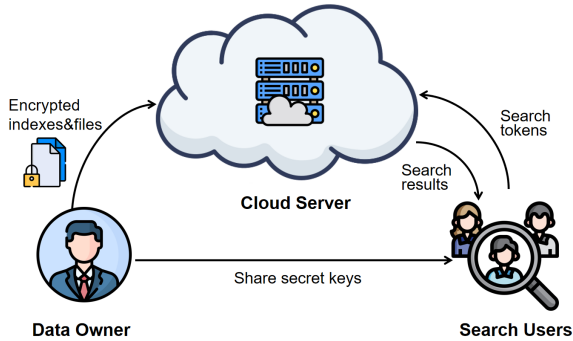


Fig. 1. System model

2.3 Design Goals

We aim to design a strong privacy-preserving conjunctive keyword DSSE scheme, which enables the cloud server to provide storage and search services in a privacy-preserving manner. Our designed scheme should meet the following security requirements and features.

- **Forward Privacy:** The cloud server should not learn whether the newly stored documents contain the previously searched keywords.

- **Backward Privacy:** The cloud server should not learn the deleted document when searching.
- **Keyword Pair Result Privacy:** The cloud server should not obtain the partial query result sets for two specific keywords, eliminating KPRP leakage.
- **Non-interactive:** The search user should not interact with the data owner for acquiring the keyword with the least frequency in each search query.
- **Search Efficiency:** The search complexity should be sub-linear. That means it is independent of the total number of stored documents but is correlated with the frequency of the least frequent keyword in the conjunction.

This paper considers the scheme with the above-mentioned privacy properties as a strong privacy-preserving scheme.

3 PRELIMINARIES

In this section, we first present notations used in this work in Table 2. Then we present the syntax of DSSE and a formal security definition. Finally, we give a brief description of Bloom filter and symmetric-key hidden vector encryption scheme, which are adopted to construct our scheme.

TABLE 2
Notations and Descriptions.

Notation	Meaning
λ	security parameter
w	a keyword
op	operation in $\{add, del\}$
id_i	identifier of i -th file where $id_i \in \{0, 1\}^\lambda$
W_i	list of keywords contained in the file id_i
M	number of files stored in DB
DB	database $\{op_i, id_i, W_i\}_{i=1}^M$
\mathcal{W}	set of keywords $\cup_{i=1}^M W_i$
$ \mathcal{W} $	number of keywords stored in DB
q	conjunctive query $(w_1 \wedge \dots \wedge w_n)$
\mathcal{Q}	list $\{w_1, \dots, w_n\}$ of keywords for q
$DB(w)$	file identifiers containing w
$DB(q)$	file identifiers $\bigcap_{i=1}^n DB(w_i)$ for q
w_s	the keyword with the least update frequency
cnt_w	the update frequency of w
$ecnt_w$	encrypted value of cnt_w
Γ	dictionary of keyword-frequency pairs (w, cnt_w)
f_{token_w}	frequency token of w
\mathcal{T}	list of frequency tokens for q
Δ	dictionary of token-value pairs $(f_{token_w}, ecnt_w)$
BF	a Bloom filter
m	size of BF
m'	number of non-wildcard elements in BF
N	number of triple (op, id, w) stored in DB
$[t]$	set of integers $\{1, 2, \dots, t\}$
$poly(\lambda)$	an unspecified polynomial in λ
$negl(\lambda)$	a negligible function in λ
\parallel	concatenation of strings
$x \xleftarrow{R} \mathcal{X}$	uniformly sampling a random x from \mathcal{X}

3.1 Syntax of Dynamic Searchable Symmetric Encryption

A Dynamic Searchable Symmetric Encryption (DSSE) scheme $\Pi = (\text{SETUP}, \text{UPDATE}, \text{SEARCH})$ consists of an al-

gorithm **SETUP** run by a client and two protocols **UPDATE** and **SEARCH** run by a client and a cloud server.

- **SETUP**(λ, DB): Given the security parameter λ and a database DB , the client executes the algorithm to generate (sk, σ, EDB) , where sk is a secret key, σ is the client's state, and EDB is an empty encrypted database that is sent to the server.
- **UPDATE**($sk, \sigma, op, id, w; \text{EDB}$): The client takes as input the secret key sk , the state σ , an operation op which can be *add* or *del*, a file identifier id and a keyword w . The server takes as input EDB . After executing the protocol, the client updates its internal state σ' and the server updates encrypted database EDB' .
- **SEARCH**($sk, \sigma, q; \text{EDB}$): The client inputs the secret key sk , the state σ , and a search query q . The server takes as input EDB . After executing the protocol, the client outputs $\text{DB}(q)$ as the search result.

3.2 Security Definition of DSSE

The security of a DSSE scheme states that the server must learn as little as possible about the content of the database and queries. We expect that the adversary cannot learn anything beyond certain obvious leakages. The security of a DSSE scheme can be parametrized by a stateful leakage function $\mathcal{L} = (\mathcal{L}_{\text{Setup}}, \mathcal{L}_{\text{Update}}, \mathcal{L}_{\text{Search}})$ that expresses the information leaked to the adversary throughout each operation. We describe two probabilistic experiments in the real world and the ideal world as follows.

- **Real** $_{\mathcal{A}}^{\Pi}(\lambda, Q)$: The adversary \mathcal{A} chooses a database DB and a query list q . The experiment runs **SETUP**(λ, DB) and returns EDB to \mathcal{A} . Then, for each $i \in Q$ ($Q = |q|$), the experiment responds to the query by running **UPDATE**($sk, \sigma_i, q_i; \text{EDB}_i$) \rightarrow ($\sigma_{i+1}, \text{EDB}_{i+1}$) or **SEARCH**($sk, \sigma_i, q_i; \text{EDB}_i$) \rightarrow $\text{DB}(q_i)$ depending on whether q_i is an update query or a search query. Eventually, \mathcal{A} outputs a bit $b \in \{0, 1\}$.
- **Ideal** $_{\mathcal{A}}^{\Pi}(\lambda, Q)$: The adversary \mathcal{A} chooses a database DB . Given the leakage function $\mathcal{L}_{\text{Setup}}$, the simulator \mathcal{S} returns EDB to \mathcal{A} . Then, \mathcal{A} adaptively chooses a query list q . For each $i \in Q$, \mathcal{S} answers the query q_i . Eventually, the adversary \mathcal{A} outputs a bit $b \in \{0, 1\}$.

Definition 1. A DSSE scheme Π with a collection of leakage function \mathcal{L} is adaptively-secure if for any probabilistic polynomial-time adversary \mathcal{A} issuing a maximum of $Q = \text{poly}(\lambda)$ queries, there exists a probabilistic polynomial-time simulator \mathcal{S} such that

$$|Pr[\text{Real}_{\mathcal{A}}^{\Pi}(\lambda, Q) = 1] - Pr[\text{Ideal}_{\mathcal{A}}^{\Pi}(\lambda, Q) = 1]| \leq \text{negl}(\lambda).$$

3.3 Bloom Filter

Bloom filter is a space-efficient probabilistic data structure used to represent a set $\mathcal{S} = \{s_1, s_2, \dots, s_N\}$ of N elements. Its main functionality is to support fast set membership verification. A Bloom filter consists of a binary array of m -bits which is initially all 0. It is associated with k independent hash functions $\{H_i\}_{1 \leq i \leq k}$. Given each element $s \in \mathcal{S}$, the bits at positions $\{H_i(s)\}_{1 \leq i \leq k}$ in the Bloom filter are set to 1. To test the existence of s , check whether all bits at

positions $\{H_i(s)\}_{1 \leq i \leq k}$ are equal to 1. If so, $s \in \mathcal{S}$ with high probability due to the false positive rate. Otherwise, $s \notin \mathcal{S}$ with the probability 1. The false positive means that $s \notin \mathcal{S}$ but membership test returns to 1. Suppose elements in \mathcal{S} are hashed into the Bloom filter, the false positive rate P_e is

$$P_e \leq (1 - e^{-kN/m})^k.$$

P_e can be negligible by choosing optimal parameters m and k . Given N, P_e , the optimal choice of k is $k \approx \log_2(1/P_e)$, while the required $m \approx 1.44 \cdot \log_2(1/P_e) \cdot N$ [27].

3.4 Symmetric-key Hidden Vector Encryption

Symmetric-key Hidden Vector Encryption (SHVE) [5] is a lightweight predicate encryption scheme that supports comparison over encrypted data. Assume that $\Sigma = \{0, 1\}$ is a finite set of attributes, and $*$ is a wildcard symbol not in Σ , $\Sigma_* = \Sigma \cup \{*\}$. For each index vector $\mathbf{x} = (x_1, \dots, x_m) \in \Sigma^m$ and predicate vector $\mathbf{v} = (v_1, \dots, v_m) \in \Sigma_*^m$, we have:

$$P_{\mathbf{v}}^{\text{SHVE}}(\mathbf{x}) = \begin{cases} 1, & \forall 1 \leq i \leq m (v_i = x_i \text{ or } v_i = *) \\ 0, & \text{otherwise.} \end{cases}$$

The predicate $P_{\mathbf{v}}^{\text{SHVE}}(\mathbf{x}) = 1$ means that the vector \mathbf{x} matches \mathbf{v} in all the locations that are non-wildcard characters. The details of the SHVE are as follows:

- **SHVE.Setup**(1^λ) \rightarrow (msk, \mathcal{M}): Given the security parameter λ , the algorithm outputs a uniformly sampled master secret key $msk \xleftarrow{R} \{0, 1\}^\lambda$, and defines the payload message space $\mathcal{M} = \{\text{"True"}\}$.
- **SHVE.Enc**($msk, \mu = \text{"True"}, \mathbf{x} \in \Sigma^m$) \rightarrow \mathbf{c} : The algorithm takes as input the master secret key msk , a message $\mu = \text{"True"}$ and an index vector $\mathbf{x} = (x_1, \dots, x_m) \in \Sigma^m$. It returns the ciphertext

$$\mathbf{c} = \{c_l\}_{l \in [m]},$$

where for each $l \in [m]$, $c_l = F(msk, x_l || l)$.

- **SHVE.KeyGen**($msk, \mathbf{v} \in \Sigma_*^m$) \rightarrow \mathbf{s} : The algorithm takes as input the master secret key msk and a predicate vector $\mathbf{v} = (v_1, \dots, v_m) \in \Sigma_*^m$. Let set $S = \{l_j \in [m] | v_{l_j} \neq *\}$ be the set of all locations in \mathbf{v} that do not contain the wildcard characters. The algorithm randomly samples to get $K_0 \xleftarrow{R} \{0, 1\}^{\lambda + \log \lambda}$, and calculates:

$$d_0 = \bigoplus_{j \in [|S|]} (F(msk, v_{l_j} || l_j)) \oplus K_0, \text{ and} \\ d_1 = \text{Sym.Enc}(K_0, 0^{\lambda + \log \lambda}).$$

Here F is a pseudorandom function, and Sym.Enc denotes a symmetric encryption algorithm. The algorithm finally returns the decryption key

$$\mathbf{s} = (d_0, d_1, S).$$

- **SHVE.Query**(\mathbf{s}, \mathbf{c}) \rightarrow $\mu(\perp)$: On input of the ciphertext \mathbf{c} and the key \mathbf{s} , the query algorithm parse $\mathbf{s} = (d_0, d_1, S)$ and $\mathbf{c} = (\{c_l\}_{l \in [m]})$, where $S = \{l_1, l_2, \dots, l_{|S|}\}$. Then, it computes

$$K_0' = (\bigoplus_{j \in [|S|]} c_{l_j}) \oplus d_0.$$

Next the decryption algorithm calculates

$$\mu' = \text{Sym.Dec}(K_0', d_1).$$

If $\mu' = 0^{\lambda+\log \lambda}$, it outputs “True”, otherwise outputs \perp .

Correctness: Given the ciphertext $\mathbf{c} = (\{c_l\}_{l \in [m]})$ related to index vector $\mathbf{x} = (x_1, \dots, x_m)$, the decryption key $\mathbf{s} = (d_0, d_1, S)$ corresponding to predicate vector $\mathbf{v} = (v_1, \dots, v_m)$, and the set of locations $S = \{l_1, l_2, \dots, l_{|S|}\}$, we analyze the correctness of aforementioned scheme in the following two scenarios:

- If $P_v^{\text{SHVE}}(\mathbf{x}) = 1$, $v_{l_j} = x_{l_j}$ holds for each $j \in [|S|]$. This means that we can calculate $c_{l_j} = F(\text{msk}, v_{l_j} || l_j)$ for each $j \in [|S|]$. Then we have

$$\begin{aligned} K_0' &= (\oplus_{j \in [|S|]} c_{l_j}) \oplus d_0 \\ &= (\oplus_{j \in [|S|]} c_{l_j}) \oplus (\oplus_{j \in [|S|]} (F(\text{msk}, v_{l_j} || l_j))) \oplus K_0 \\ &= (\oplus_{j \in [|S|]} F(\text{msk}, v_{l_j} || l_j)) \\ &\quad \oplus (\oplus_{j \in [|S|]} (F(\text{msk}, v_{l_j} || l_j))) \oplus K_0 \\ &= K_0, \text{ and} \end{aligned}$$

$$\mu' = \text{Sym.Dec}(K_0', d_1) = 0^{\lambda+\log \lambda}.$$

Finally, the query algorithm SHVE.Query returns “True”.

- If $P_v^{\text{SHVE}}(\mathbf{x}) = 0$, there exists some $j \in [|S|]$ such that $v_{l_j} \neq x_{l_j}$. This implies that $c_{l_j} \neq F(\text{msk}, v_{l_j} || l_j)$, and hence, $K_0' \neq K$, so that we have $\mu' \neq 0^{\lambda+\log \lambda}$ and the algorithm returns the failure symbol \perp .

4 CONSTRUCTION OF THE PROPOSED SCHEME

In this section, we present the detailed construction of our ESP-CKS scheme. We first introduce our least frequent keyword acquisition protocol to achieve non-interactive acquisition of the s -term w_s with the least update frequency in the conjunction. After that, we propose the algorithm for the setup, update and search phases, respectively.

4.1 Least Frequent Keyword Acquisition Protocol

The existing conjunctive keyword SSE schemes require that a search user asks for the keyword with the least frequency of updates from the data owner in each search query. However, this produces redundant communication overhead. Ideally, after the update is complete, the data owner should not be required to perform additional interactions for each search. We propose the privacy-preserving Least Frequent Keyword Acquisition (LFKA) protocol to address this problem. The protocol guarantees that a search user can obtain the least frequent keyword without frequent interactions with the data owner. Meanwhile, the privacy of update frequency is protected.

Let $F_1 : \{0, 1\}^\lambda \times \{0, 1\}^* \rightarrow \{0, 1\}^{2\lambda}$ and $F_2 : \{0, 1\}^\lambda \times \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ be pseudorandom functions (PRFs). Our LFKA protocol $\Lambda = (\text{KeyGen}, \text{FreqSetup}, \text{TokenGen}, \text{FreqFind}, \text{Compare})$ is constructed as follows.

- $\text{KeyGen}(\lambda) \rightarrow K$: Given the security parameter λ , the data owner generates the random secret key $K \in \{0, 1\}^\lambda$ for PRFs F_1, F_2 .
- $\text{FreqSetup}(\Gamma, r) \rightarrow \mathcal{C}$: The data owner selects a random integer $r \in \mathbb{N}$. Then it takes r and the

dictionary Γ as input. For each keyword $w \in \mathcal{W}$, the data owner computes the ciphertext $ecnt_w$ for the plaintext cnt_w composing the set \mathcal{C} , then it sends \mathcal{C} to the cloud server.

- $\text{TokenGen}(\mathcal{Q}, K, r) \rightarrow \mathcal{T}$: The search user inputs a query \mathcal{Q} , the secret key K and the random integer r . It generates a frequency token list \mathcal{T} , then sends \mathcal{T} to the cloud server.
- $\text{FreqFind}(\mathcal{T}, \mathcal{C}) \rightarrow \Delta$: The cloud server inputs the token list \mathcal{T} and set \mathcal{C} . It matches each token in \mathcal{T} with the elements in \mathcal{C} to get a matching result set Δ .
- $\text{Compare}(\Delta, K, r) \rightarrow (w_s, cnt_{w_s})$: Given the set Δ , the secret key K and the random integer r , the search user outputs the least frequent keyword in the conjunction with its update frequency.

In the stage FreqSetup , to reduce frequent interaction between the search user and the data owner, for each $w \in \mathcal{W}$, the client computes the ciphertext as

$$ecnt_w = F_1(K, r || w) + F_2(K, w || r) + cnt_w,$$

where F_1 generates 2λ -bits output, F_2 generates λ -bits output, and operation $+$ is a bitwise addition operation, satisfying the value of $(F_2(K, w || r) + cnt_w)$ does not exceed 2^λ , which is convenient to find $ecnt_w$ later. Therefore, when retrieving the encrypted value $ecnt_w$ corresponding to w , it is obtained that only the first λ -bits need to be matched according to $F_1(K, r || w)$. In addition, a fresh random integer r is generated in each update and owned by the client, and $ecnt_w$ for each keyword is updated, ensuring the privacy of the updated keywords.

When a client requests to search the least frequent keyword in \mathcal{Q} , it runs TokenGen to generate the frequency tokens. It computes $f_{token_w} = F_1(K, r || w)$ for each keyword $w \in \mathcal{Q}$, and then sends $\mathcal{T} = \{f_{token_{w_1}}, \dots, f_{token_{w_n}}\}$ to the cloud server.

In response to the client, the cloud server executes the algorithm FreqFind . Note that the server uses f_{token_w} to match and obtain the corresponding $ecnt_w$ in the same range according to the first λ -bits. Then the server sends the set $\Delta = \{(f_{token_{w_i}}, ecnt_{w_i})\}_{i \in [n]}$ to the client.

The client runs the algorithm Compare to find the keyword with the least update frequency. Specifically, it obtains cnt_w by decrypting $ecnt_w, w \in \mathcal{Q}$

$$cnt_w = ecnt_w - F_1(K, r || w) - F_2(K, w || r),$$

where the secret key K and r are received previously from the data owner. Finally, the client compares the frequencies of each desired keyword in list \mathcal{Q} , and returns the least frequent keyword w_s and its update frequency cnt_{w_s} .

4.2 Setup Phase

Client: The SETUP algorithm (Algorithm 1) inputs the security parameter λ and returns the $sk, st, param, \text{EDB}$. The client initializes two empty maps Cnt and TSet , where Cnt is the state parameter σ mentioned in Section 3.1 storing update frequencies of keywords, and TSet stores address-value entries. Then it generates a m -bits Bloom filter BF with k hash functions $\{H_j\}_{1 \leq j \leq k}$, which is set up for dynamic cross-tags x_{tag} introduced later. The client

randomly chooses keys for PRFs and msk . Then it executes SHVE.Setup to encrypt BF . $TSet$ and $xtagBF$ are components of the encrypted database EDB and sent to the server.

Algorithm 1 Setup

Input: a security parameter λ

Output: $sk, st, param$, EDB

Client:

- 1: Initialize $Cnt, TSet$ to empty maps
 - 2: Initialize a Bloom filter BF with element 0
 - 3: Select k hash functions $\{H_i\}_{1 \leq i \leq k}$ for BF
 - 4: Select K_T for PRF F
 - 5: Select K_X, K_Y, K_Z for PRF F_p
 - 6: Execute LFKA.KeyGen(λ) to get key for PRF F_1, F_2
 - 7: Execute SHVE.Setup(1^λ) to get secret key msk
 - 8: Compute $xtagBF = \text{SHVE.Enc}(msk, \mu = \text{"True"}, BF)$
 - 9: Set $sk = (msk, K, K_T, K_X, K_Y, K_Z)$, $st = Cnt$, $param = \{H_i\}_{1 \leq i \leq k}$
 - 10: Set $\text{EDB}(1) = TSet$, $\text{EDB}(2) = xtagBF$
 - 11: Send $\text{EDB} = (\text{EDB}(1), \text{EDB}(2))$ to the server
-

4.3 Update Phase

Client: In the UPDATE algorithm (Algorithm 2), the client takes as input the key set sk , the state parameter st , and the set L consisting of (op, w, id) . The server takes database EDB as input. The algorithm updates a batch of documents in each update. For each triple (op, id, w) in L , the corresponding entry $(addr, val, \alpha)$ and dynamic cross-tag $xtag$ are generated adopting pseudorandom functions and exponentiations, then the counter $Cnt[w] = Cnt[w] + 1$. The entry $(addr, val, \alpha)$ will be stored in the form of address-value pair in the $TSet$. The address $addr$ is generated by a pseudorandom function F for w and the current update frequency cnt_w . The corresponding stored contents of the address $addr$ are val and α , where val is the encrypted (id, op) pair related to w and α is the dynamic blinding factor that enables the server to calculate the $xtag$. Besides, the client calculates the k positions related to $xtag$ in the Bloom filter BF , and sets the corresponding elements to 1. Then it executes SHVE.Enc algorithm to encrypt BF , and gets the ciphertext $xtagBF$, and send the entries mentioned above to the server.

Note that in each update, the client is required to select a new random number $r' \in \mathbb{N}$ (to distinguish the previous r from the fresh r , we use r' to represent the latter) and re-encrypt the update frequencies of all keywords via LFKA.FreqSetup. Then it sends the fresh set \mathcal{C} to the cloud server. According to the changed entry, this approach prevents the cloud server from deducing which keyword is in the updated file. Besides, it avoids frequent interactions between the data owner and the search user. Even though the cloud server might notice that the number of elements in set \mathcal{C} changes after each update, it cannot infer the plaintext of the underlying keywords without knowing the secret key K .

Server. After receiving the encrypted entries, the server updates dictionary $TSet$ and encrypted Bloom filter $xtagBF$ accordingly.

We introduce the update counter cnt to ensure forward privacy. cnt_w is incremented when an updated file contains the keyword w . Similarly, the random integer r changes in each update. Because cnt_w and r are not the same as those

Algorithm 2 Update

Input: sk, st, L ; EDB

Output: st , EDB

Client:

- 1: Initialize a dictionary Γ
 - 2: Initialize \mathcal{C} , $addrList$, $valList$, \alphaList to empty sets
 - 3: Get $sk = (msk, K, K_T, K_X, K_Y, K_Z)$, $st = Cnt$
 - 4: **for** $i = 1 : L.size$ **do**
 - 5: $(op, id, w) = L[i]$
 - 6: **if** $Cnt[w] = NULL$ **then**
 - 7: Set $Cnt[w] = 0$
 - 8: **end if**
 - 9: Set $Cnt[w] = Cnt[w] + 1$
 - 10: Set $\Gamma[w] = Cnt[w]$
 - 11: Compute $addr = F(K_T, w || Cnt[w] || 0)$
 - 12: Set $addrList = addrList \cup \{addr\}$
 - 13: Compute $val = (id || op) \oplus F(K_T, w || Cnt[w] || 1)$
 - 14: Set $valList = valList \cup \{val\}$
 - 15: Compute $\alpha = F_p(K_Y, id || op) \cdot F_p(K_Z, w || Cnt[w])^{-1}$
 - 16: Set $\alphaList = \alphaList \cup \{\alpha\}$
 - 17: Compute $xtag = g^{F_p(K_X, w) \cdot F_p(K_Y, id || op)}$
 - 18: **for** $i = 1 : k$ **do**
 - 19: Compute $hind(op, id, w) = H_i(xtag)$
 - 20: Set $BF[hind(op, id, w)] = 1$
 - 21: **end for**
 - 22: **end for**
 - 23: Select a random number $r \in \mathbb{N}$
 - 24: Set $\mathcal{C} = \text{LFKA.FreqSetup}(\Gamma, r)$
 - 25: Compute $xtagBF = \text{SHVE.Enc}(msk, \mu = \text{"True"}, BF)$
 - 26: Send $(\mathcal{C}, addrList, valList, \alphaList, xtagBF)$ to the server
- Server:*
- 27: Get $TSet = \text{EDB}(1)$
 - 28: **for** $i = 1 : addrList.size$ **do**
 - 29: Set $TSet[addrList[i]] = (valList[i], \alphaList[i])$
 - 30: **end for**
 - 31: Set $\text{EDB} = (TSet, xtagBF)$
-

in the latest search query, it ensures there is no relation between the new update and the previous search. Moreover, since the server learns nothing about the secret key K and K_T , an update operation hides the underlying operation op , the identifier id , and the keyword w .

4.4 Search Phase

Let $q = (w_1 \wedge \dots \wedge w_n)$ be a conjunctive search query issued by the client, then the SEARCH algorithm (Algorithm 3) involves three rounds of interaction between the client and the server. The keyword with the least frequency in the conjunction is denoted as the s -term w_s , and other keywords $w_i (2 \leq i \leq n)$ as the x -term.

Round-1 allows the client to execute the LFKA protocol with the server. Then it obtain the s -term w_s and its frequency cnt_{w_s} .

In Round-2, the client reverses the positions of w_s and w_1 , resulting in w_s becoming w_1 . Subsequently, it generates all relevant addresses $saddr$ s in the $TSet$ involving w_1 from the first to the $cnt_{w_1}^{th}$ update. Meanwhile, it generates an additional set of cross-tokens $\{xtoken_{i,j}\}_{i \in [2,n], j \in [cnt_{w_1}]}$ related to the s -term w_1 and the x -term for each id_j , where id_j is the identifier of the file involving the s -term. After receiving the list of $saddr$, the server retrieves the corresponding $TSet$ entries and gets the encrypted values $sval$ of (id, op) pairs and the dynamic blinding factors α . The server computes the relevant cross-tag $xtag_{i,j}$ involving w_i and id_j via exponential operations of cross-tokens and blinding factors. Then it calculates the positions of BF which are set to 1 for storing $xtag_{i,j}$, and sends them to the client.

Round-3 allows the client to construct a Bloom filter vBF_j for id_j based on the received positions. That is, we assume that each triple $(op, id_j, w_i) (2 \leq i \leq n)$ already exists in the database, where op is the operation executed to the id_j involving the s -term. Then the client executes SHVE.KeyGen to encrypt vBF_j , and sends the ciphertext sBF_j to the server. The server performs membership test by executing SHVE.Query($sBF_j, xtagBF$) to measure whether id_j involves all keywords in the conjunction. If it returns “True”, all $n - 1$ x -terms are also in the file id_j , and the server sends the relevant $svals$ to the client.

Finally, the client decrypts $svals$ to recover $DB(w_1)$ along with corresponding operations. Then it includes the identifiers with add operation in the result set $IdRList$, and discards those with delete operation from $IdRList$.

Recall that the data owner encrypts and uploads the frequency of updates to the server in the latest update. Therefore, the search user can ask the server for the least frequent keyword, which eliminates frequent interactions between clients in searches, and handing over the corresponding task to the server with unlimited computation and storage capacities.

We now elaborate the correctness of oblivious conjunctive search. Over any number of update operations, the dynamic cross-tag can evaluate the presence or absence of any identifier-keyword pair (id, w) in a dynamic dataset. More specifically, for an update triple (op, id_j, w_i) , there exists the cross-tag

$$xtag_{op,i,j} = g^{F_p(K_X, w_i) \cdot F_p(K_Y, id_j || op)},$$

where $op \in \{add, del\}$. The $xtag$ value is stored in BF . We use a Bloom filter containing $\{xtag_{i,j}\}_{i \in [2,n]}$ to assume that file id_j involves all $n - 1$ x -terms. By performing SHVE protocol, the server checks the membership of id_j in ciphertext without the KPRP leakage on which w_i are in id_j .

To make the server obliviously compute the cross-tag, the client as a search user generates and sends an additional set of cross-tokens $\{xtoken_{i,j}\}_{i \in [2,n], j \in [cnt_{w_1}]}$ to the server which involve s -term w_1 and $n - 1$ x -terms. Then, we can obtain

$$xtoken_{i,j} = g^{F_p(K_X, w_i) \cdot F_p(K_Z, w_1 || j)}.$$

Since the $TSet$ address corresponding to the j^{th} update operation involving w_1 stores an additional pre-computed blinding factor α , the server can calculate $xtag_{op,i,j}$ using blinded exponentiations as

$$xtag_{op,i,j} = g^{F_p(K_X, w_i) \cdot F_p(K_Y, id_j || op)} = (xtoken_{i,j})^\alpha.$$

The aforementioned computation process shows that without learning underlying update triple (op, id, w) , the server can obliviously compute the relevant cross-tag. Next, if all elements in $\{xtag_{op,i,j}\}_{i \in [2,n]}$ exist in $xtagBF$, the file id_j involving w_1 also contains other $n - 1$ keywords.

5 SECURITY ANALYSIS

In this section, we evaluate the security of our scheme. Firstly, we formally describe the leakage functions for ESP-CKS. Subsequently, we demonstrate its forward, backward, and keyword pair result privacy.

5.1 Leakage Functions

We aim to guarantee that the DSSE scheme reveals as little information as possible, ensuring that it achieves a higher level of security. The leakage functions capture leakage. Similar to [20], we formally define the leakage functions as

$$\mathcal{L} = (\mathcal{L}_{Setup}, \mathcal{L}_{Update}, \mathcal{L}_{Search}),$$

where $\mathcal{L}_{Setup} = \perp$, $\mathcal{L}_{Update}(op, id, w) = \perp$, $\mathcal{L}_{Search}(q) = (\text{TimeDB}(q), \text{Upd}(q))$.

Let \mathcal{O} be a list containing the following entries:

- 1) (t, w) : search query for keyword w at timestamp t ;
- 2) (t, op, id, w) : update query for (op, id, w) at timestamp t , where $op \in \{add, del\}$.

For any conjunctive query $q = (w_1 \wedge \dots \wedge w_n)$, we define $\text{TimeDB}(q)$ as a function that returns file identifiers and insertion timestamps. Files corresponding to the identifiers contain all the keywords involved in the query q and have not yet been deleted from the database. Suppose the keyword w_1 is the s -term, namely aforementioned least frequent term w_s . Formally, it is expressed as

$$\text{TimeDB}(q) = \{(\{t_i\}_{i \in [n]}, id) \mid (t_i, add, id, w_i) \in \mathcal{O} \text{ and } \forall t' : (t', del, id, w_i) \notin \mathcal{O}\}.$$

We define $\text{Upd}(q)$ as a function that returns the timestamps of all update operations on the s -term w_1 . Formally, it is expressed as

$$\text{Upd}(q) = \{t \mid \exists (op, id) : (t, op, id, w_1) \in \mathcal{O}\}.$$

For simplicity, we assume that no Bloom filter false positives occur in our protocol. We have the following theorem for the security of our scheme.

Theorem 1. *Our ESP-CKS is adaptively-secure with respect to a leakage function \mathcal{L} defined as before, assuming that F, F_1, F_2, F_p are secure pseudorandom functions, the decisional Diffie-Hellman assumption holds over \mathbb{G} , and SHVE is a selectively simulation-secure protocol.*

Proof. The detailed proof is given in Appendix. \square

5.2 Forward Privacy

According to the forward privacy definition introduced in [25], the dynamic conjunctive SSE is adaptively forward privacy iff the update leakage function \mathcal{L}_{Update} can be written as

$$\mathcal{L}_{Update}(op, id, w) = \mathcal{L}'(op, id),$$

where \mathcal{L}' is a stateless function. Note that forward privacy guarantees that the underlying keyword w is hidden during update phase. Whereas, in our scheme $\mathcal{L}_{Update}(op, id, w) = \perp$. That is, an update operation hides the underlying keyword w , along with the identifier id and the operation op . A natural corollary of Theorem 1 is as follows.

Corollary 1 (Forward Privacy of ESP-CKS). *Our scheme is adaptively forward private if F, F_1, F_2, F_p are secure pseudorandom functions, the decisional Diffie-Hellman assumption holds over \mathbb{G} , and SHVE is a selectively simulation-secure protocol.*

Algorithm 3 Search

Input: $param, sk, r, \mathcal{Q} = \{w_1, \dots, w_n\}$; EDB
Output: $IdRList$

Client:
1: Initialize \mathcal{T} to an empty set
2: Get $sk = (msk, K, K_T, K_X, K_Y, K_Z)$
3: Set $\mathcal{T} = LFKA.TokenGen(\mathcal{Q}, K, r)$
4: Send \mathcal{T} to the server

Server:
5: Initialize Δ to empty list
6: Set $\Delta = LFKA.FreqFind(\mathcal{T}, \mathcal{C})$
7: Send set Δ to the client

Client:
8: Set $(w_s, cnt_{w_s}) = LFKA.Compare(\Delta, K, r)$
9: Replace w_s to the front of the list \mathcal{Q} as w_1
10: Initialize $saddrList$ and $xtokenList[cnt_{w_1}]$ to empty lists
11: **for** $j = 1 : cnt_{w_1}$ **do**
12: Compute $saddr_j = F_p(K_T, w_1 || j || 0)$
13: Set $saddrList = saddrList \cup \{saddr_j\}$
14: **for** $i = 2 : n$ **do**
15: Compute $xtoken_{i,j} = g^{F_p(K_Z, w_1 || j) \cdot F_p(K_X, w_i)}$
16: $xtokenList[j] = xtokenList[j] \cup \{xtoken_{i,j}\}$
17: **end for**
18: Randomly shuffle the tuple-entries of $xtokenList[j]$
19: **end for**
20: Send $saddrList, xtokenList$ to the server

Server:
21: Get $(TSet, xtagBF) = EDB$
22: Initialize $vBFind[xtokenList.size]$ to empty lists
23: **for** $j = 1 : saddrList.size$ **do**
24: Set $(sval_j, \alpha_j) = TSet[saddrList[j]]$
25: **for** $i = 2 : n$ **do**
26: Set $xtoken_{i,j} = xtokenList[j][i - 1]$
27: Compute $xtag_{i,j} = xtoken_{i,j}^{\alpha_j}$
28: **for** $t = 1 : k$ **do**
29: Compute $hind = H_t(xtag_{i,j})$
30: Set $vBFind[j] = vBFind[j] \cup \{hind\}$
31: **end for**
32: **end for**
33: **end for**
34: Send $vBFind[1], \dots, vBFind[j]$ to the client

Client:
35: Initialize cnt_{w_1} Bloom filters vBF with element *
36: Initialize $sBFList$ to an empty list
37: **for** $j = 1 : cnt_{w_1}$ **do**
38: **for** $c = 1 : vBFind[j].size$ **do**
39: Set $vBF[j][vBFind[c]] = 1$
40: **end for**
41: Compute $sBF_j = SHVE.KeyGen(msk, vBF[j])$
42: Set $sBFList = sBFList \cup \{sBF_j\}$
43: **end for**
44: Send $sBFList$ to the server

Server:
45: Initialize $sRList$ to an empty list
46: **for** $j = 1 : sBFList.size$ **do**
47: Compute $res_j = SHVE.Query(sBFList[j], xtagBF)$
48: **if** $res_j = \text{"True"}$ **then**
49: Add $(j, sval_j)$ into $sRList$
50: **end if**
51: **end for**
52: Send $sRList$ to the client

Client:Final
53: Initialize $IdRList$ to an empty list
54: **for** $i = 1 : sRList.size$ **do**
55: Set $(j, sval_j) = sRList[i]$
56: Compute $(id_j, op_j) = sval_j \oplus F(K_T, w_1 || j || 1)$
57: **if** $op_j == add$ **then**
58: Set $IdRList = IdRList \cup \{id_j\}$
59: **else if** $op_j == del$ **then**
60: Set $IdRList = IdRList \setminus \{id_j\}$
61: **end if**
62: **end for**

5.3 Backward Privacy

Backward privacy is formally defined in [25] and is ordered from the most to least secure as Type-I, Type-II, Type-III. According to this, a DSSE scheme supporting single keyword search is adaptively backward privacy iff the update leakage function \mathcal{L}_{Update} and the search leakage function $\mathcal{L}_{Search(w)}$ can be written as

$$\mathcal{L}_{Update}(op, id, w) = \mathcal{L}''(op, id), \text{ and}$$

$$\mathcal{L}_{Search(w)} = \mathcal{L}'''(\text{TimeDB}(w), \text{Upd}(w)),$$

where \mathcal{L}'' and \mathcal{L}''' are stateless functions.

For a DSSE scheme supporting conjunctive keyword search, a natural generalization of the aforementioned leakage function is defined as

$$\mathcal{L}_{Update}(op, id, w) = \perp, \text{ and}$$

$$\mathcal{L}_{Search(q)} = (\text{TimeDB}(q), \text{Upd}(q)),$$

where q is a conjunctive search query. The leakage meets the Type-II backward privacy. A natural corollary of Theorem 1 is as follows.

Corollary 2 (Backward Privacy of ESP-CKS). *Our scheme is Type-II adaptively backward private if F, F_1, F_2, F_p are secure pseudorandom functions, the decisional Diffie-Hellman assumption holds over \mathbb{G} , and SHVE is a selectively simulation-secure protocol.*

5.4 Keyword Pair Result Privacy

We compare ESP-CKS with ODXT scheme [20] to prove that our scheme guarantees keyword pair result privacy; that is, it prevents the KPRP leakage. We consider the following example to illustrate the privacy improvement of the proposed scheme.

Assuming that the database in the server stores five encrypted files, whose plaintext of indexes are $\{id_i\}_{1 \leq i \leq 5}$. The set of keywords \mathcal{W} contains six keywords expressed as $\{w_i\}_{1 \leq i \leq 6}$. Each file involves different keywords, as shown in Table 3.

TABLE 3
Document Identifier's and Keywords Contained.

identifier	keywords
1	w_1, w_2, w_3, w_4, w_6
2	w_2, w_3, w_4, w_5
3	w_1, w_2, w_4, w_5, w_6
4	w_2, w_3, w_6
5	w_1, w_3, w_4

Suppose that the client issues the search query $q = (w_1 \wedge w_2 \wedge w_3)$. Note that keyword w_1 is the least frequent term. The set of documents involving w_1 is $\text{DB}(w_1) = \{id_1, id_3, id_5\}$. Then the server determines whether the files contain w_2 and w_3 . In ESP-CKS, the server only learns the result $\text{DB}(q) = \{id_1\}$. However, in ODXT, the server can learn the partial query result that $\text{DB}(w_1) \cap \text{DB}(w_2) = \{id_1, id_3\}$

and $DB(w_1) \cap DB(w_3) = \{id_1, id_5\}$. Finally, the server sends the encrypted data of the intersection of these two sets (namely $\{id_1\}$) to the client. Although the partial query result called KPRP leakage is under the ciphertext, [21] shows that file-injection attacks can reveal all keywords of a conjunctive query with 100% accuracy via utilizing KPRP leakage. Our ESP-CKS scheme eliminates the aforementioned KPRP leakage and improves privacy.

6 PERFORMANCE ANALYSIS

In this section, we analyze the performance of the ESP-CKS scheme and compare it with existing schemes. We evaluate the performance theoretically and experimentally.

6.1 Theoretical Analysis

We now discuss the practical storage, computation, communication costs of our scheme and compare them with that in conjunctive keyword search schemes ODXT [20] and SE-EPOM [6]. Notably, there are two types of servers in SE-EPOM. Here we only focus on the performance of the Cloud Platform (CP). We give a list of notations for comparative analysis in Table 5.

Storage: Focusing on the storage size, the server in ODXT stores the dictionary $TSet$ and the Bloom filter of $XSet$. For a triple (op, id, w) , a $O(\lambda)$ -size entry of the form $(addr, \alpha, val)$ is added to $TSet$, and the size of $XSet$ is m . In comparison to ODXT, we adopt ciphertext of a Bloom filter with $m\lambda$ -size instead of $XSet$. Moreover, the server stores the ciphertext of keyword update frequency of size $|\mathcal{W}|\lambda$. If update operations contain N triples (op, id, w) , the overall storage size of our ESP-CKS scheme is $N\lambda + m\lambda + |\mathcal{W}|\lambda$. In SE-EPOM, server stores M entries and each encrypted by Enc algorithm [28] of Z_{λ^2} -size. Hence, the storage cost of SE-EPOM is more than the proposed scheme.

Update Computation Overhead: During the update phase, the computational cost is generated by the client. For a batch update containing multiple triples (op, id, w) , in our scheme, the time taken to encrypt keywords update frequencies is $uT_{PRF} + uT_{ADD}$. For the computation in $TSet$, the cost of generating N entries $(addr, val, \alpha)$ is $N(T_{PRF} + T_{XOR} + T_{MUL})$. For the generation of $xtags$ and $xtagBF$, the costs are $Ne_{pre} + NkT_{hash}$ and mT_{PRF} , respectively. Note that the ODXT shares a lot of similarities with our scheme on the stage of $TSet$ computation and the Bloom filter generation involving $xtags$, the cost is $N(T_{PRF} + T_{MUL} + T_{XOR} + e_{pre} + kT_{hash})$. Although forward and backward privacy cannot be guaranteed, the static SE-EPOM protocol can also perform adding files operation. In SE-EPOM, the client encrypts a decimal number of keywords in a file with the proposed public key encryption method, which involves exponentiations and multiplications, and the overall update cost is $m(e + T_{MUL^2})$.

Search Computation Overhead: We now investigate the search computation overhead of our ESP-CKS and contrast it with that of the ODXT and SE-EPOM. Both server-side and client-side should be considered during the search phase.

First, we focus on the computational cost of our scheme. The client spends $n(T_{PRF} + T_{ADD}) + T_{Comp}$ when it generates frequency tokens. In the subsequent interaction, the

client computes $saddrs$ and $xtokens$ costing $C(nT_{PRF} + (n-1)e_{pre})$. The server spends $C(n-1)e$ to compute $xtags$ and $C(n-1)kT_{hash}$ for membership test in Bloom filters. The client computes sBF through SHVE.KeyGen, which costs $m'T_{PRF} + m'T_{XOR} + T_{Enc}$. The server spends $C(m'T_{XOR} + T_{Dec})$ to perform SHVE.Query. Finally, the client decrypts the search result costing $C(T_{PRF} + T_{XOR})$. The search computation overhead investigated above is presented in Table 4.

In ODXT, the server spends $C(n-1)(e + kT_{hash})$ to generate $xtags$ and calculate the location of elements in the Bloom filter. The client spends $C(nT_{PRF} + (n-1)e_{pre})$ in $saddrs$ and $xtokens$ generation, and $C(T_{PRF} + T_{XOR})$ in result decryption, respectively.

We now compare the computational cost on the server-side and client-side between our ESP-CKS scheme and the ODXT scheme. We define O_s on the server-side as

$$O_s = \frac{m'T_{XOR} + T_{Dec}}{(n-1)(e + kT_{hash})}.$$

According to the relationship between execution time of different operations mentioned in [5], we conclude that additional cost on server side is only about 1.8%, while false positive rate is 10^{-6} , $n = 2$, $k \approx \log_2(1/P_e) \approx 20$, and $m' = (n-1)k$.

Similarly, we define O_c on the client side as

$$O_c = \frac{T_{freq} + T_{sBF}}{C((n+1)T_{PRF} + (n-1)e_{pre} + T_{XOR})},$$

where $T_{freq} = n(T_{PRF} + T_{ADD}) + T_{Comp}$ and $T_{sBF} = C(m'T_{PRF} + m'T_{XOR} + T_{Enc})$. Likewise, we can deduce that additional cost on server side is about 15% when $C = 10$, and it decreases as n and C increase. Therefore, our scheme achieves an enhanced security with minimum compromise in search efficiency.

In SE-EPOM, the client needs to generate a trapdoor and decrypt the results with a weak private key, which costs $M(e + T_{MUL^2})$. On the server-side, the computation overhead of protocol SBD [29], SAD, and SMD is $\theta(e + T_{MUL^2})$. Moreover, NOT-operation in ciphertext takes $e + T_{MUL^2}$. Hence, the overall search computational cost is $M(\theta + 1)(e + T_{MUL^2})$ for the server. Note that the server computation overhead is linear with C and n in our scheme, but scales with M and θ in SE-EPOM. It can be observed that the search in our scheme is more efficient because of $M \geq C$, $\theta \geq n$ and the probability of being equal is extremely low.

Communication Overhead: Focusing on communication cost in our scheme, the bandwidth of sending encrypted frequency values is $nO(\lambda)$. The interaction of transmit $saddrs$ and $xtokens$ costs $C\lambda + C(n-1)G$. The server sends the positions in BF to the client with $CO(m')$ communication overhead. Then the client sends the output of SHVE.KeyGen with $C(O(m') + 2\lambda)$ cost. Finally, the server costs $rO(\lambda)$ to transmit the final result. Note that in ODXT, there is no communication about transmitting frequency values and ciphertext of Bloom filter, and the overall communication overhead is $C\lambda + C(n-1)G + rO(\lambda)$. In SE-EPOM, the client sends the search token with $O(\lambda^2)$, and the server sends encrypted result with $MO(\lambda^2)$. The communication overhead investigated above is presented in Table 4.

TABLE 4
Comparison of Conjunctive Keyword Search Schemes.

		SE-EPOM [6]	ODXT [20]	OURS
storage	storage size(S)	MZ_{λ^2}	$N\lambda + m$	$N\lambda + m\lambda + \mathcal{W} \lambda$
computation	update cost	common	$m(e + T_{MUL^2})$	$N(T_{PRF} + T_{MUL} + T_{XOR} + e_{pre} + kT_{hash})$
		additional		N/A
	search cost (S)	common	$M(\theta + 1) \cdot (e + T_{MUL^2})$	$C(n - 1)(e + kT_{hash})$
		additional		N/A
	search cost (C)	common	$M(e + T_{MUL^2})$	$C((n + 1)T_{PRF} + (n - 1)e_{pre} + T_{XOR})$
		additional		N/A
comunication	cost	common	$C((n - 1)G + \lambda) + rO(\lambda)$	
		additional	N/A	$nO(\lambda) + CO(m' + 2\lambda)$

TABLE 5
Notations for Comparative Analysis.

Notation	Meaning
C	update frequency of the s -term
e	number of exponentiations
e_{pre}	number of preprocessed exponentiations
u	number of keywords updated in a batch
N	number of triple (op, id, w) updated in a batch
m	number of files updated in a batch
r	number of pairs (id, op) in the search result
θ	the maximum number of keywords in set \mathcal{W}
G	size of an element from \mathbb{G}
Z_{λ^2}	size of an element from $\mathbb{Z}_{\lambda^2}^*$
T_{PRF}	time taken to compute a PRF
T_{XOR}	time taken to execute an XOR operation over λ
T_{hash}	time taken to compute a hash of Bloom filter
T_{ADD}	time taken to perform an addition over λ
T_{MUL}	time taken to perform a multiplication over λ
T_{MUL^2}	time taken to perform a multiplication over λ^2
T_{Comp}	time taken to compare n values.
T_{Enc}	time taken to perform a symmetric encryption.
T_{Dec}	time taken to perform a symmetric decryption.

6.2 Experimental Evaluations

We conduct the following experiments to evaluate the efficiency of our scheme in terms of setup, update, and conjunctive search time. Each experimental result is the average execution times over 10 runs.

TABLE 6
Overhead Comparison.

	SE-EPOM	ODXT	OURS
setup(ms)	4006	416	468
update-1000(ms)	2.91	55.27	61.01
update-10000(ms)	/	42.34	73.21

Implementation details. We implement our ESP-CKS with Python. The communication is simulated in a single-threaded environment to facilitate testing time. We conduct our experiments on a PC with a 1.30 GHz eight-core processor and 16GB RAM. The security parameter is $\lambda = 256$. We deploy the scalable Bloom filter from Paulo Sérgio Almeida

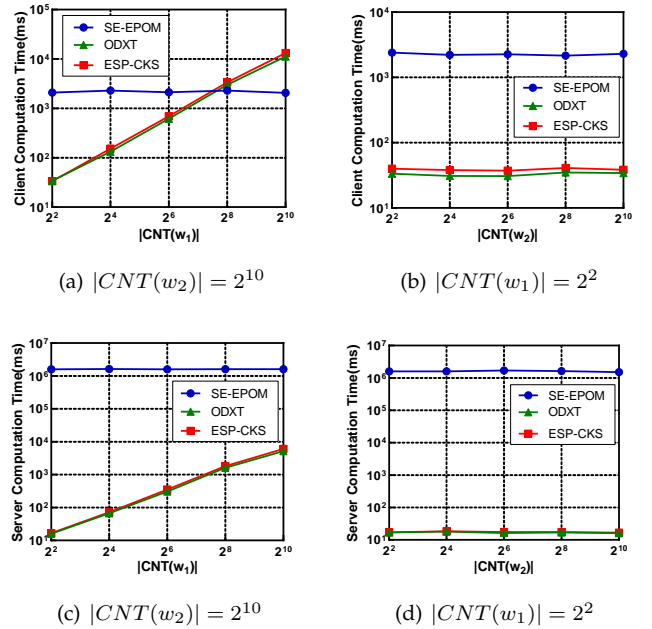


Fig. 2. Two-conjunctive search query $q = (w_1 \wedge w_2)$ computation time over client and server.

[30]. The false positive rate of the Bloom filter is set to 10^{-6} , which is negligible and enables the server to perform the membership test accurately. For comparison, we also implement conjunctive keyword search schemes ODXT [20] and SE-EPOM [6]. Notation θ denotes the maximum number of keywords in set \mathcal{W} . The outputs of F, F_p, F_2 are 256 bits long, while that of F_1 is 512 bits long.

Datasets: We employ two different sorts of datasets: one with 1,200 files and the other with 12,500 files. Because SE-EPOM runs slow in a single thread on an extensive database without supporting deleting files, the dataset with 1,200 files is exploited to compare the three works. The dataset with 12,500 files is to compare our ESP-CKS with ODXT. We ensure that each document in datasets is inserted at least once. Update operations can be additions or deletions of batch files, 20% of which are deletions.

Setup: Table 6 compares the costs of the Setup (KeyGen in

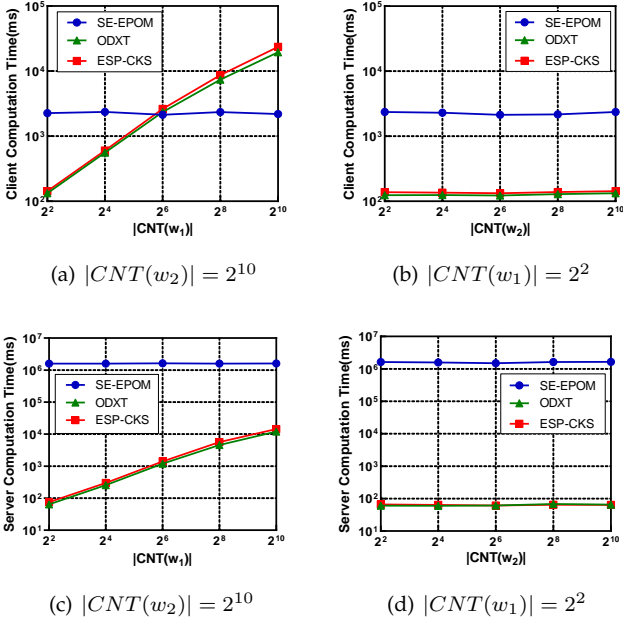


Fig. 3. Multi-conjunctive search query $q = (w_1 \wedge \dots \wedge w_5)$ computation time over client and server.

SE-EPOM) algorithm. The time taken in ODXT and our ESP-CKS scheme are similar and better than that of SE-EPOM, since SE-EPOM consumes more time assigning the secret keys under the public key system.

Update: Table 6 also compares the cost of the client in the Update(Store in SE-EPOM) algorithm, where each update involves 1000 or 10,000 documents. For SE-EPOM, although it takes less time than our ESP-CKS, the variety of keywords is limited because it must be agreed upon in advance. Moreover, update calculation overhead of SE-EPOM scales with value θ , so it is more expensive than our scheme if θ is initially set to be large and some documents have just a few keywords. Comparing ODXT with our scheme, our scheme takes longer due to keyword pair result privacy, this is because the update phase requires the Bloom filter to be encrypted, and the computational overhead of this part depends on the size of the Bloom filter. However, when we update batch files, the average cost of each file between the two schemes is not significant, because files in a batch update amortize the cost of encrypted bloom filter. Therefore, our scheme does not sacrifice significant update efficiency but offers a higher level of security.

Search: Figures 2, 3, and 4 show the comparisons of computation overheads in Search (Test in SE-EPOM) algorithm for various schemes.

Figure 2 shows the impact of the update frequency for conjunctive searches involving two keywords on query efficiency. In this implementation, we set the frequency of one term to a constant value and changed the frequency of another term from 2^2 to 2^{10} . The dataset contains 1200 files, and θ of scheme SE-EPOM is 10. Assuming w_1 is the keyword s -term with the least frequency, both the client and the server computation overhead in ESP-CKS and ODXT scale with the frequency of the s -term, while SE-EPOM has a constant but prohibitively expensive computation cost.

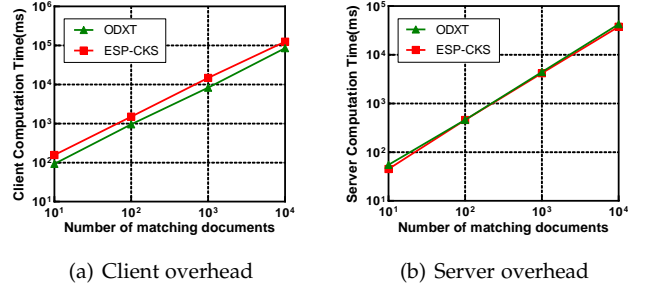


Fig. 4. Two-conjunctive search query $q = (w_1 \wedge w_2)$ computation time over client and server.

This also demonstrates the significance of selecting the least frequent keyword.

Similarly, Figure 3 shows the impact of the update frequency for conjunctive searches involving multiple keywords on query efficiency. Note that the difference in performance between ESP-CKS and ODXT is not significant, but keyword pair result privacy is protected in ESP-CKS. Therefore, our scheme is better than ODXT and SE-EPOM.

Figure 4 compares ESP-CKS and ODXT for the computation overhead. We adopt the dataset with 12,500 documents and perform conjunctive search queries with 10 to 10,000 matching documents. Note that the time spent by client and server in our scheme and ODXT is proportional to the number of matching documents, and there is not much difference between the time cost of both. However, our scheme offers a higher level of security.

Based on the evaluations above, the computation overhead of SE-EPOM is prohibitively expensive, and ESP-CKS closely matches ODXT but performs better in privacy protection. Overall, our scheme delivers superior functionality and security, as shown in Table 1.

7 RELATED WORK

SSE was first proposed by Song et al. [31], the search cost of the proposed SSE scheme is linear with the number of file-keyword pairs. To improve search efficiency, Curtmola et al. [32] presented an inverted index data structure later to achieve sub-linear search time, but focused mainly on single keyword search. Cash et al. [3] proposed the first sub-linear SSE protocol named Oblivious Cross-Tags (OXT) that supports conjunctive keyword search. The computational overhead of search scales with the update frequency for s -term, and sub-linear with the size of the database. It also uses a dictionary $TSet$ to make it possible to correlate a list of fixed-sized data tuples with each keyword in the database, and later allows the server to retrieve these lists via keyword-related tokens. Based on the above, the cloud server can first search for the encrypted data tuples associated with documents containing the s -term, then determine over encrypted data whether the documents match the other desired keywords. Lai et al. [5] improved the OXT protocol in terms of private information leakage and proposed the Hidden Cross-Tags (HXT) protocol. According to [5], the OXT protocol leads to Keyword Pair Result Pattern (KPRP) leakage during the search phase. However, the file-injection attacks [21] can exploits this leakage to reveal all keywords

in the conjunction with 100% accuracy. Hence, it is essential to eliminate KPRP leakage.

The SSE schemes above focused mainly on static database settings. To support updates of the encrypted database, Kamara et al. [33] introduced the DSSE scheme, but it is vulnerable to file-injection attacks. SSE providing forward privacy firstly defined by Stefanov et al. [23] which enables to resist the attacks, and the first efficient DSSE scheme is proposed to achieve small leakage based on oblivious RAM (ORAM). In the setting of single keyword search, the schemes [34], [35] can provide forward privacy based on ORAM. However, they suffer from prohibitively expensive computation overhead. Yang et al. [4] proposed a novel dynamic searchable symmetric encryption scheme with forward privacy by maintaining an increasing counter for each keyword. Song et al. [36] presented the efficient FAST scheme based on symmetric key cryptography, which achieves forward privacy by singly linked lists structure and a pseudorandom permutation. Li et al. [9] developed the hidden pointer technique (HPT) to further strengthen security while ensuring forward privacy. Whereas, aforementioned works focused mainly on single keyword search. In order to provide conjunctive keyword search, Kamara and Moataz [37] presented IEX-2Lev and IEX-ZMF, achieving forward security. Wu et al.'s scheme [14] exploited counters and building trees with efficient queries to guarantee forward private conjunctive search.

Backward privacy was later formalized by Bost et al. [25]. Subsequently, Javad et al. [10] achieve backward privacy based on ORAM, and Sun et al. [38] adopted symmetric puncturable encryption technology to guarantee backward privacy. He et al. [39] proposed CLOSE-FB scheme with forward and backward privacy based on fish-bone chain, and the scheme only stores a secret key and a global counter on the client achieving constant client storage cost. Xu et al. [40] presented a forward and backward secure DSSE scheme named Bestie which also attains non-interactive real deletion, and Xu et al. [41] presented the robust DSSE scheme with forward and backward security based on the key-updatable pseudorandom function. However, these schemes can only perform single keyword search. Zuo et al. [42] considered range queries, and developed a scheme called FBDSE-RQ based on their refined binary tree which can retrieve files containing keywords in a given range. For conjunctive queries that achieve forward and backward privacy, a FBDSE-CQ scheme using extended bitmap indexing was proposed by Zuo et al. [26], but it is severely limited to the number of search keywords. The most effective existing scheme that is not restricted by keywords is the ODXT scheme [20], [43], but there is still the threat of KPRP leakage, and the problem of frequent interactions between clients in the search phase.

8 CONCLUSION

This work proposes the first efficient conjunctive keyword search ESP-CKS scheme with strong privacy. It supports dynamic databases and achieves forward and backward privacy. Besides, it eliminates the threat of keyword pair result leakage, which has never been guaranteed in existing

dynamic conjunctive keyword search schemes. It is independent of the total number of documents in the database but scales with the least frequency of updates. Furthermore, the least frequent keyword acquisition protocol presented can efficiently attain the lowest frequency in a non-interactive approach, eliminating frequent communication between the data owner and search users. Compared with other keyword search approaches, our scheme offers a higher level of security, and experiment results demonstrate that it performs better.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Grant Nos. 61972037, 61872041, U1804263), the China Postdoctoral Science Foundation (Grant Nos. 2021M700435, 2021TQ0042).

REFERENCES

- [1] Q. Liu, G. Wang, F. Li, S. Yang, and J. Wu, "Preserving privacy with probabilistic indistinguishability in weighted social networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 28, no. 5, pp. 1417–1429, 2017.
- [2] S. Zhang, X. Mao, K. R. Choo, T. Peng, and G. Wang, "A trajectory privacy-preserving scheme based on a dual-k mechanism for continuous location-based services," *Inf. Sci.*, vol. 527, pp. 406–419, 2020.
- [3] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, ser. Lecture Notes in Computer Science, R. Canetti and J. A. Garay, Eds., vol. 8042. Springer, 2013, pp. 353–373.
- [4] L. Yang, Q. Zheng, and X. Fan, "RSPP: A reliable, searchable and privacy-preserving e-healthcare system for cloud-assisted body area networks," in *2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017*. IEEE, 2017, pp. 1–9.
- [5] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S. Sun, D. Liu, and C. Zuo, "Result pattern hiding searchable encryption for conjunctive queries," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 745–762.
- [6] X. Liu, G. Yang, W. Susilo, J. Tonien, X. Liu, and J. Shen, "Privacy-preserving multi-keyword searchable encryption for distributed systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 3, pp. 561–574, 2021.
- [7] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.
- [8] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.
- [9] J. Li, Y. Huang, Y. Wei, S. Lv, Z. Liu, C. Dong, and W. Lou, "Searchable symmetric encryption with forward search privacy," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 1, pp. 460–474, 2021.
- [10] J. G. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1038–1055.
- [11] R. Zhang, R. Xue, and L. Liu, "Searchable encryption for healthcare clouds: A survey," *IEEE Trans. Serv. Comput.*, vol. 11, no. 6, pp. 978–996, 2018.

- [12] J. Shu, X. Jia, K. Yang, and H. Wang, "Privacy-preserving task recommendation services for crowdsourcing," *IEEE Trans. Serv. Comput.*, vol. 14, no. 1, pp. 235–247, 2021.
- [13] C. Zhang, L. Zhu, C. Xu, J. Ni, C. Huang, and X. S. Shen, "Location privacy-preserving task recommendation with geometric range query in mobile crowdsensing," *IEEE Transactions on Mobile Computing*, 2021, doi: 10.1109/TMC.2021.3080714.
- [14] Z. Wu and K. Li, "Vbtree: forward secure conjunctive queries over encrypted data for cloud computing," *VLDB J.*, vol. 28, no. 1, pp. 25–46, 2019.
- [15] N. Jho and D. Hong, "Symmetric searchable encryption with efficient conjunctive keyword search," *KSII Trans. Internet Inf. Syst.*, vol. 7, no. 5, pp. 1328–1342, 2013.
- [16] P. Golle, J. Staddon, and B. R. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings*, ser. Lecture Notes in Computer Science, M. Jakobsson, M. Yung, and J. Zhou, Eds., vol. 3089. Springer, 2004, pp. 31–45.
- [17] J. Wang, X. Chen, S. Sun, J. K. Liu, M. H. Au, and Z. Zhan, "Towards efficient verifiable conjunctive keyword search for large encrypted database," in *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. López, J. Zhou, and M. Soriano, Eds., vol. 11099. Springer, 2018, pp. 83–100.
- [18] S. K. Kermanshahi, J. K. Liu, R. Steinfeld, S. Nepal, S. Lai, R. Loh, and C. Zuo, "Multi-client cloud-based symmetric searchable encryption," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 5, pp. 2419–2437, 2021. [Online]. Available: <https://doi.org/10.1109/TDSC.2019.2950934>
- [19] Q. Gan, J. K. Liu, X. Wang, X. Yuan, S. Sun, D. Huang, C. Zuo, and J. Wang, "Verifiable searchable symmetric encryption for conjunctive keyword queries in cloud storage," *Frontiers Comput. Sci.*, vol. 16, no. 6, p. 166820, 2022. [Online]. Available: <https://doi.org/10.1007/s11704-021-0601-8>
- [20] S. Patranabis and D. Mukhopadhyay, "Forward and backward private conjunctive searchable symmetric encryption," in *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.
- [21] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 707–720.
- [22] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 668–679.
- [23] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.
- [24] R. Bost, "Σοφος: Forward secure searchable encryption," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1143–1154.
- [25] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1465–1482.
- [26] C. Zuo, S. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and G. Wei, "Forward and backward private dynamic searchable symmetric encryption for conjunctive queries," *IACR Cryptol. ePrint Arch.*, p. 1357, 2020.
- [27] A. Z. Broder and M. Mitzenmacher, "Survey: Network applications of bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2003.
- [28] X. Liu, R. H. Deng, K. R. Choo, and J. Weng, "An efficient privacy-preserving outsourced calculation toolkit with multiple keys," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 11, pp. 2401–2414, 2016.
- [29] B. K. Samanthula, H. Chun, and W. Jiang, "An efficient and probabilistic secure bit-decomposition," in *8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13, Hangzhou, China - May 08 - 10, 2013*, K. Chen, Q. Xie, W. Qiu, N. Li, and W. Tzeng, Eds. ACM, 2013, pp. 541–546.
- [30] P. S. Almeida, C. Baquero, N. M. Pregoça, and D. Hutchison, "Scalable bloom filters," *Inf. Process. Lett.*, vol. 101, no. 6, pp. 255–261, 2007.
- [31] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*. IEEE Computer Society, 2000, pp. 44–55.
- [32] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds. ACM, 2006, pp. 79–88.
- [33] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 965–976.
- [34] S. Garg, P. Mohassel, and C. Papamanthou, "TWRAM: efficient oblivious RAM in two rounds with applications to searchable encryption," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, ser. Lecture Notes in Computer Science, M. Robshaw and J. Katz, Eds., vol. 9816. Springer, 2016, pp. 563–592.
- [35] I. Demertzis, J. G. Chamani, D. Papadopoulos, and C. Papamanthou, "Dynamic searchable encryption with small client storage," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.
- [36] X. Song, C. Dong, D. Yuan, Q. Xu, and M. Zhao, "Forward private searchable symmetric encryption with optimized I/O efficiency," *IEEE Trans. Dependable Secur. Comput.*, vol. 17, no. 5, pp. 912–927, 2020.
- [37] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10212, 2017, pp. 94–124.
- [38] S. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, and S. Nepal, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 763–780.
- [39] K. He, J. Chen, Q. Zhou, R. Du, and Y. Xiang, "Secure dynamic searchable symmetric encryption with constant client storage cost," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1538–1549, 2021.
- [40] T. Chen, P. Xu, W. Wang, Y. Zheng, W. Susilo, and H. Jin, "Bestie: Very practical searchable encryption with forward and backward security," in *Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, E. Bertino, H. Shulman, and M. Waidner, Eds., vol. 12973. Springer, 2021, pp. 3–23.
- [41] P. Xu, W. Susilo, W. Wang, T. Chen, Q. Wu, K. Liang, and H. Jin, "Rose: Robust searchable encryption with forward and backward security," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1115–1130, 2022.
- [42] C. Zuo, S. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and L. Xu, "Forward and backward private DSSE for range queries," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 1, pp. 328–338, 2022.
- [43] S. Patranabis and D. Mukhopadhyay, "Forward and backward private conjunctive searchable symmetric encryption," *IACR Cryptol. ePrint Arch.*, p. 1342, 2020.

APPENDIX A PROOF OF THEOREM 1

We prove through a sequence of games between a challenger and an adversary. We start from $\text{Real}_A^\Pi(\lambda)$ and construct a

sequence of games where each game is designed slightly differently from the previous one, and show that they are computationally indistinguishable for the adversary \mathcal{A} . The final construction is a simulation game $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\lambda)$. Due to the transitive property of each two successive games' indistinguishability, $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\lambda)$ is computationally indistinguishable from the $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda)$. Eventually, we conclude that the views of \mathcal{A} in $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\lambda)$ are computationally indistinguishable, thus completing the proof of the Theorem 1.

Game₀: Game₀ is the same as the real game $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\lambda)$ of our scheme.

Game₁: Game₁ is the same as Game₀, except that we replace the PRFs $F(K_T, \cdot)$, $F_1(K, \cdot)$, $F_2(K, \cdot)$, $F_p(K_X, \cdot)$, $F_p(K_Y, \cdot)$, and $F_p(K_Z, \cdot)$ with random functions $G_T(\cdot)$, $G_1(\cdot)$, $G_2(\cdot)$, $G_X(\cdot)$, $G_Y(\cdot)$, and $G_Z(\cdot)$, respectively. Specifically, the function $G_T(\cdot)$ is uniformly randomly sampled from the set of all functions that map λ -bit string onto λ -bit string, $G_1(\cdot)$ is uniformly randomly sampled from the set of all functions that map λ -bit string onto 2λ -bit string, $G_2(\cdot)$ is uniformly randomly sampled from the set of all functions that map λ -bit string onto λ -bit string, while $G_X(\cdot)$, $G_Y(\cdot)$ and $G_Z(\cdot)$ are uniformly randomly sampled from the set of all functions that map λ -bit string onto components in \mathbb{Z}_p^* .

Lemma 1. *Suppose that F, F_1, F_2 , and F_p are secure PRFs, the views of \mathcal{A} in Game₁ and Game₀ are computationally indistinguishable.*

Proof. Assuming that there exists a probabilistic polynomial-time adversary \mathcal{B}_1 that can distinguish between the views of adversary \mathcal{A} in Game₀ and Game₁. This implies that \mathcal{B}_1 can be used to construct a probabilistic polynomial-time adversary \mathcal{B}'_1 that can distinguish pseudorandom functions from random functions. Obviously, this is contrary to the pseudorandomness of the pseudorandom function. \square

Game₂: Game₂ is the same as Game₁, except that we regenerate $xtoken$ in the search phase. Specifically, for a query $q = (w_1 \wedge w_2 \cdots \wedge w_n)$, the challenger initially looks for the adversary \mathcal{A} 's history of update queries to get the set of update operations (op_j, id_j, w_1) involving the s -term w_1 . Then, for each x -term w_i , it computes $\alpha_{i,j}$ and $xtag_{i,j}$, where $xtag_{i,j}$ is stored in an array A. The array A is used to generate BF and the element $A[op, id, w]$ is added to BF . Next, it computes $xtoken_{i,j} = xtag_{i,j}^{1/\alpha_{i,j}}$. For an impossible matched $xtoken$, the challenger computes $xtoken_{i,j} = g^{G_X(w_i) \cdot G_Z(w_1 || j)}$ and stores it in an array B.

It's trivial to see that the distribution of each $xtoken$ value in Game₂ is the same as that of in Game₁. Therefore, the view of the adversary \mathcal{A} in Game₂ is indistinguishable from the view of the adversary \mathcal{A} in Game₁.

Game₃: Game₃ is the same as Game₂, except that we generate α in an alternative but equivalent way during each update phase. We replace computing α with sampling randomly $\alpha \xleftarrow{R} \mathbb{Z}_p^*$.

Lemma 2. *The views of \mathcal{A} in Game₃ and Game₂ are statistically indistinguishable.*

Proof. Note that the previous α is the product of an evaluation of $G_Y(\cdot)$ with the inverse of an evaluation of $G_Z(\cdot)$,

where $G_Y(\cdot)$ and $G_Z(\cdot)$ are uniformly randomly sampled from the set of all functions that map λ -bit string onto components in \mathbb{Z}_p^* . Moreover, the value of α is also uniform and independent of the rest of the randomness in Game₂. Consequently, the distribution of each α value in Game₃ is statistically indistinguishable from that in Game₂. \square

Game₄: Game₄ is the same as Game₃, except that we regenerate $xtag$ in array A and $xtoken$ in array B. We replace $xtag$ in array A with random sample $A[op, id, w] = g^\gamma$, where $\gamma \xleftarrow{R} \mathbb{Z}_p^*$. Moreover, we replace $xtoken$ in array B with random sample $B[w_1, w_i, j] = g^t$, where $t \xleftarrow{R} \mathbb{Z}_p^*$. That is, all of the values in arrays A and B are randomly picked from \mathbb{G} .

Lemma 3. *Suppose that the decisional Diffie-Hellman (DDH) assumption holds in the group \mathbb{G} , the views of \mathcal{A} in Game₄ and Game₃ are computationally indistinguishable.*

We prove Lemma 3 by the equivalent Lemma 4, where the equivalence is derived from the (polynomial) equivalence of the DDH assumption and the extended DDH assumption [43] over any group \mathbb{G} . (The paper [43] has already proved that the extended DDH assumption holds over a group \mathbb{G} iff the DDH assumption holds over the same group \mathbb{G} .)

Lemma 4. *Suppose that the extended DDH assumption holds in the group \mathbb{G} , the views of \mathcal{A} in Game₄ and Game₃ are computationally indistinguishable.*

Proof. Assuming that there exists a probabilistic polynomial-time adversary \mathcal{B}_2 that can distinguish between the views of adversary \mathcal{A} in Game₄ and Game₃. \mathcal{B}_2 can easily be used to construct a probabilistic polynomial-time adversary \mathcal{B}'_2 that can distinguish between $g^{G_X(w) \cdot G_Y(id || op)}$ and g^γ . Similarly, $xtoken$ in array B is the same. Obviously, \mathcal{B}'_2 breaks the extended DDH assumption over the group \mathbb{G} , hence there is no such aforementioned probabilistic polynomial-time adversary \mathcal{B}_2 . \square

Game₅: Game₅ is the same as Game₄, except that we regenerate $addr$ and val in the update phase. The function evaluation of the form $G_T(w || cnt || b) (b \in \{0, 1\})$ is replaced with a function evaluation of the form $G_T(t)$, where t is the timestamp when the update operation is executed. Similarly, we regenerate $saddr$ in the search phase, namely a function evaluation of the form $G_T(t)$ is substituted for the function evaluation of the form $G_T(w || cnt || 0)$, where t is the timestamp associated with the corresponding update operation.

Lemma 5. *The views of \mathcal{A} in Game₅ and Game₄ are computationally indistinguishable.*

Proof. Note that the counter increases monotonically, and the uniform random function G_T never takes a value on the same input at two different timestamps. Therefore, the values of G_T in Game₅ and Game₄ are computationally indistinguishable from the point of view of the adversary \mathcal{A} . \square

Game₆: Game₆ is the same as Game₅, except that we replace the challenger with a simulator \mathcal{S} that could only

access the leakage function $\mathcal{L} = (\mathcal{L}_{Setup}, \mathcal{L}_{Update}, \mathcal{L}_{Search})$ for each update and search query.

Lemma 6. *The views of \mathcal{A} in $Game_6$ and $Game_5$ are computationally indistinguishable.*

Proof. The Simulator \mathcal{S} can access an empty update leakage function $\mathcal{L}_{Update}(op, id, w) = \perp$ and search leakage function $\mathcal{L}_{Search}(q) = (\text{TimeDB}(q), \text{Upd}(q))$, ensuring that \mathcal{S} does not have access to the actual queries issued by \mathcal{A} . The rest of the variables are generated by \mathcal{S} as done by the challenger in $Game_5$. For the search phase, \mathcal{S} can learn the update frequency involving the s -term and the timestamp of each operation, which is expressed as $\text{Upd}(q)$. It can also learn the final result of conjunctive search queries, that is, a set of document identifiers together with operation timestamps expressed as $\text{TimeDB}(q)$. In addition, \mathcal{S} can infer that the two conjunctive search queries q_1 and q_2 have the same s -term, which is subsumed by $\text{Upd}(q_1)$ and $\text{Upd}(q_2)$. Note that from the perspective of adversary \mathcal{A} , the transcripts generated by \mathcal{S} are identical to the corresponding transcripts generated by the challenger in $Game_5$. \square

The proof of Lemma 6 is complete, that is, Theorem 1 is proved.