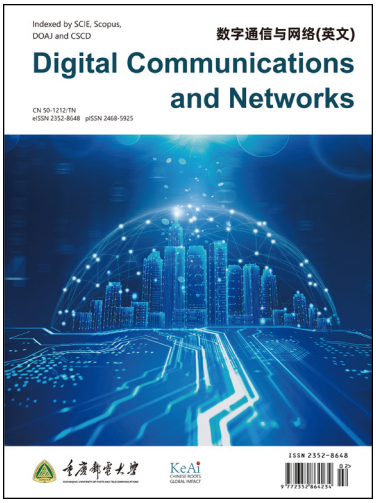
Journal Pre-proof



An efficient conjunctive keyword searchable encryption for cloud-based IoT systems

Tianqi Peng, Bei Gong, Chong Guo, Akhtar Badshah, Muhammad Waqas et al.

PII: S2352-8648(25)00024-0
DOI: https://doi.org/10.1016/j.dcan.2025.03.002
Reference: DCAN 847

To appear in: *Digital Communications and Networks*

Received date: 2 August 2024
Revised date: 21 February 2025
Accepted date: 6 March 2025

Please cite this article as: T. Peng, B. Gong, C. Guo et al., An efficient conjunctive keyword searchable encryption for cloud-based IoT systems, *Digital Communications and Networks*, doi: https://doi.org/10.1016/j.dcan.2025.03.002.

Digital Communications and Networks(DCN)

HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/dcan

An efficient conjunctive keyword searchable encryption for cloud-based IoT systems

Tianqi Peng [a], Bei Gong [a], Chong Guo [a,*], Akhtar Badshah [b], Muhammad Waqas [c,d,*], Hisham Alasmary [e], Sheng Chen [f,g]

[a] *Faculty of Information Technology, Beijing University of Technology, Beijing 100124 China*
[b] *Department of Software Engineering, University of Malakand, Dir Lower 18800, Pakistan*
[c] *School of Computing and Mathematical Sciences, Faculty of Engineering and Science, University of Greenwich, U.K.*
[d] *School of Engineering, Edith Cowan University, Perth, 6027 WA, Australia*
[e] *Department of Computer Science, College of Computer Science, King Khalid University, 61421, Abha, Saudi Arabia*
[f] *School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, U.K.*
[g] *Faculty of Information Science and Engineering, Ocean University of China, Qingdao 266100 China*

Abstract

Data privacy leakage has always been a critical concern in cloud-based Internet of Things (IoT) systems. Dynamic Symmetric Searchable Encryption (DSSE) with forward and backward privacy aims to address this issue by enabling updates and retrievals of ciphertext on untrusted cloud server while ensuring data privacy. However, previous research on DSSE mostly focused on single keyword search, which limits its practical application in cloud-based IoT systems. Recently, Patranabis (NDSS 2021) [1] proposed a groundbreaking DSSE scheme for conjunctive keyword search. However, this scheme fails to effectively handle deletion operations in certain circumstances, resulting in inaccurate query results. Additionally, the scheme introduces unnecessary search overhead. To overcome these problems, we present CKSE, an efficient conjunctive keyword DSSE scheme. Our scheme improves the oblivious shared computation protocol used in the scheme of Patranabis, thus enabling a more comprehensive deletion functionality. Furthermore, we introduce a state chain structure to reduce the search overhead. Through security analysis and experimental evaluation, we demonstrate that our CKSE achieves more comprehensive deletion functionality while maintaining comparable search performance and security, compared to the oblivious dynamic cross-tags protocol of Patranabis. The combination of comprehensive functionality, high efficiency, and security makes our CKSE an ideal choice for deployment in cloud-based IoT systems.

KEYWORDS:
Symmetric searchable encryption, Conjunctive keyword search, Forward and backward privacy, Cloud server.

1. Introduction

With the advancement of cloud computing and the Internet of Things (IoT), various cloud-based IoT systems, including intelligent logistics, smart homes, and intelligent healthcare, are experiencing widespread adoption [2–5]. During the deployment of these systems, individuals and organizations often choose to outsource large volumes of data to cloud servers for storage and processing. However, ensuring data privacy becomes a critical concern due to the potential lack of trustworthiness of cloud servers. Unauthorized disclosure of sensitive information can result in adverse consequences such as reputational damage, unnecessary discrimination, and location leakage [6–8]. To protect data privacy, a common approach is to encrypt the data before uploading it to the cloud server. However, traditional encryption algorithms render the data indistinguishable from random values, posing a challenge when performing effective retrieval on ciphertext.

To address the aforementioned challenge, researchers have proposed Symmetric Searchable Encryption (SSE) as a solution. SSE aims to enable the retrieval of ciphertext without revealing sensitive information. An important further advancement in this field is Dynamic SSE (DSSE), which not only allows retrieval but also supports dynamic updates on the encrypted database by revealing limited additional privacy information. Unfortunately, adversaries can exploit this leaked information to launch attacks, such as file-injection attacks [9]. In order to mitigate the security concerns arising from this information leakage, Stefanov *et al.* [10] introduced two new concepts into the DSSE: forward privacy and backward privacy. Forward privacy ensures that newly added files to the database do not disclose whether they contain keywords that have been previously queried. Conversely, backward privacy guarantees that current search queries do not disclose any information about previously deleted files. Bost *et al.* [11, 12] provided formal definitions for forward and backward privacy, and three types of backward privacy were defined in [12], with the security strength gradually weakening from Type-I to

*Tianqi Peng (tianqi_peng@emails.bjut.edu.cn), Bei Gong (gongbei@bjut.edu.cn), Akhtar Badshah (akhtarbadshah@uom.edu.pk), M. Waqas (engr.waqas2079@gmail.com), G. Chong (chongguo@emails.bjut.edu.cn), H. Alasmary (alasmary@kku.edu.sa), Sheng Chen (sqc@soton.ac.uk).

Type-III. Building upon these concepts, several dynamic searchable encryption schemes with forward and backward privacy were proposed in the literature [13–20].

Nevertheless, most existing DSSE schemes with forward and backward privacy only support single keyword search, which limits the expressive efficiency of the scheme. In cloud-based IoT systems, it is essential for DSSE to support conjunctive keyword search to enhance the practicality of the scheme. Although running single-keyword DSSE schemes multiple times can achieve conjunctive search, this approach has efficiency and security issues. Specifically, the computational and communication overhead is related to the update frequency of each keyword, and when multiple keywords have high update frequencies, the search performance is poor. In terms of security, the adversary may learn the update count and timestamps of each keyword, as well as files unrelated to the query. These additional leakages are undesirable. Recent works [1, 21] have proposed dynamic conjunctive keyword search schemes with forward and backward privacy. In particular, the Oblivious Dynamic Cross-tags (ODXT) protocol, introduced in [1], is the first efficient forward and Type-II backward private conjunctive keyword DSSE scheme known to us. ODXT [1] builds upon the static scheme of Oblivious Cross-tags (OXT) [22] and the single keyword DSSE scheme known as Mitra [14]. The concept of ODXT [1] is similar to OXT [22], where the server maintains two encrypted databases named 'TSet' and 'XSet'. When a client issues a conjunctive search query, the server first matches files containing the least frequent keyword in TSet. It then uses cross-token and dynamic cross-tag techniques to determine whether these files contain the remaining queried keywords in XSet. The server returns the results to the client for further filtering. The key to enabling dynamic conjunctive keyword search in ODXT [1] lies in the on-the-fly computation of cross-tags and cross-tokens with each update operation.

Although ODXT [1] is more efficient than the early conjunctive keyword DSSE schemes, it still has two drawbacks. First, it fails to adequately consider the update type of a keyword when computing the cross-tag, which results in the inability to perform valid delete operations in certain circumstances. For instance, consider the following interaction between the client and server: $[1, add, (w_1, ind_1)]$, $[2, add, (w_2, ind_1)]$, $[3, search, (w_1 \wedge w_2)]$, $[4, del, (w_2, ind_1)]$, $[5, search, (w_1 \wedge w_2)]$. After the delete operation at time 4, if the server performs a search operation $(w_1 \wedge w_2)$, the results still contain ind_1. It is worth noting that similar delete operations are common in real-world scenarios, and a scheme that cannot effectively handle delete operations will yield inaccurate query results. Second, consider that the client runs an instance of Mitra within ODXT [1]. In order to achieve retrieval in TSet, the client needs to compute all the locations involving the least frequent keyword in advance and send them to the server. This increases the computational and communication overhead on the client side, particularly when the least frequent keyword still has a high update frequency. Consequently, the search performance degrades significantly. To sum up, the lack of effective delete operations and the computational and communication overheads hinder the practicality of existing conjunctive keyword DSSE schemes in cloud-IoT systems [23, 24].

In this paper, our objective is to design a conjunctive keyword search scheme with forward and backward privacy, incorporating a complete deletion function, while minimizing search overheads. We anticipate two significant challenges that need to be addressed. First, we need to achieve efficient deletion operations within the scheme while ensuring that the search complexity remains proportional to the number of documents containing the least frequent keywords in conjunction. This scenario represents the optimal achievable search complexity among conjunctive SSE schemes, as noted in [1]. Second, supporting more comprehensive functions inherently results in increased overheads and may introduce additional security concerns. Striking a right balance between functionality, efficiency, and security simultaneously within a scheme poses a challenging task. Before proceeding to our contributions, we provide a comprehensive review of the existing literature.

1.1. Related Works

Song *et al.* [25] proposed the first practical SSE scheme having linear search time with the size of database. Subsequently, Curtmola *et al.* [26] considered leakage and constructed the first reversed-index based scheme with sub-linear efficiency. Chase and Kamara [27] traded higher storage complexity for the similar scheme. However, these works focus on static settings and are not suitable for many scenarios that require real-time data updates. To support data update on SSE and mitigate the leakage, the forward and backward private DSSE has become an important branch in this research area.

1.1.1. Single Keyword DSSE

The first sub-linear-complexity DSSE scheme was proposed by Karama *et al.* [28]. Subsequent work has focused on the security, efficiency, and expressiveness of DSSE. Liu et al. [29] and Yu et al. [30] leveraged the flexibility of attribute-based encryption (ABE) combined with blockchain to enable fine-grained search and revocable functionalities. Yin et al. [31] introduced a secure index based on access policies and an attribute-based search token, which supports fine-grained search with integrated access control. While these methods reduce decryption and revocation overhead, DSSE schemes relying on ABE still face challenges in resource-constrained environments.

Forward and backward privacy can address the additional privacy leakage issues introduced by dynamic updates. The notion of forward privacy was first proposed in [32]. Since then, several forward private DSSE schemes supporting single keyword search have been proposed [11, 33, 34]. Among them, Bost [11] proposed a pioneering forward privacy scheme called Sophos, which uses the state-based approach to reduce the overhead of the scheme. Although the trapdoor permutation structure of Sophos is limited by public key operations, it provides new ideas for subsequent research. Backward privacy was first introduced by Stefanov *et al.* [10]. Later, Bost *et al.* [12] formally defined three types of backward privacy, called Type-I, Type-II and Type-III with progressively weakening security, and a number of schemes with various types of backward privacy were proposed in [12–17]. Chamani *et al.* [14] presented three schemes, Mitra, Orion and Hours. Mitra, a Type-II scheme, obtains better performance by using symmetric key encryption. Orion is a Type-I scheme based on oblivious random-access memory, and Hours, a Type-III scheme, optimizes Orion's performance at the cost of leaking more information. In order to reduce the client-side storage, Demertzis *et al.* [16] proposed three schemes called SDa, SDd and Qos. SDa [16] and SDd [16] use static-to-dynamic techniques to achieve Type-II backward privacy. Qos [16] is a quasi-optimal Type-III backward privacy scheme. Dou et al. [35] introduced a robust scheme that ensures both forward and backward privacy, designed to handle more complex update and query processes. Chen et al. [36] leveraged blockchain and hash-proof chain technologies to create a publicly verifiable DSSE scheme, incorporating a novel data hiding structure that offers both forward and backward privacy.

To the best of our knowledge, majority of the existing forward and backward private schemes primarily focus on supporting single keyword search, with limited attention given to conjunctive keyword search.

1.1.2. Conjunctive Keyword DSSE

The inclusion of conjunctive keyword search functionality greatly enhances the practicality of SSE schemes. The first efficient conjunctive keyword SSE scheme, OXT, was proposed by Cash *et al.* [22]. However, its construction lacks the capability for data updates. To address this limitation, Lai *et al.* [37] compensated for the leakage in OXT, while the works [38] and [39] focused on forward privacy in conjunctive keyword searches on dynamic databases. Zuo *et al.* [21] introduced FBDSSE-CQ, a forward and backward private conjunctive keyword search scheme that trades linear search overhead for efficiency. Chen et al. [40] developed DSSE-DC, a conjunctive search DSSE scheme featuring a revocation mechanism based on inner product matching. Guo

et al. [41] created a forward index using a t-puncturable pseudorandom function, combining it with an inverted index to support conjunctive keyword searches. Li et al. [42] introduced an update counter to build a bi-directional index structure that supports conjunctive queries over bipartite graphs. Their scheme also employed a new oblivious data structure for storing the bi-directional index and used semantically secure encryption to protect node information, achieving forward privacy and backward privacy. Yuan et al. [43] presented the first sub-linear KPRP-hiding conjunctive DSSE scheme with forward and backward privacy, utilizing a novel cryptographic primitive called Attribute Updatable Hidden Map Encryption (AUHME). Li et al. [44] introduced the Indistinguishable Binary Tree (IBtree), a highly balanced binary tree structure designed to support conjunctive keyword searches. Additionally, numerous studies have expanded DSSE to incorporate various advanced query functionalities [45–47]. More recently, Mitra$_{\text{CONJ}}$, BDXT and ODXT were proposed in [1]. Mitra$_{\text{CONJ}}$ [1] is a straightforward extension of Mitra from single keyword queries to conjunctive queries, but the computational and communication complexity of this scheme is proportional to the sum of the update frequency of each keyword in conjunction. BDXT [1] is the first forward and backward private conjunctive keyword SSE scheme with the computational and communication complexity proportional to the least frequent keywords in conjunction. ODXT [1] further optimizes the performance of BDXT, which is the most effective forward and backward conjunctive keyword scheme at present. However, as mentioned before, ODXT [1] suffers from ineffective deletion and high search overhead.

1.2. Our Contributions

Against the above background and to achieve more efficient conjunctive keyword search in real-world scenarios, We proposed a conjunctive keyword DSSE scheme, named CKSE. In particular, CKSE achieves more comprehensive deletion operations than ODXT [1], while balancing the efficiency and security. Table 1 shows the comparison of our scheme with other state-of-the-art schemes.Our contributions can be summarized as follows.

- We propose an efficient conjunctive keyword DSSE scheme called CKSE based on ODXT [1]. By constructing a new cross-tag that combines the update type with the keyword and modifying the elements involving the oblivious shared computation, our CKSE is capable of performing effective deletion operations and providing the client with accurate query results. Moreover, the search complexity of CKSE is related to the least update frequency keyword.
- We introduce a state chain structure to instantiate our scheme. This structure is a symmetric cryptographic primitive version of the public key-based trapdoor permutation structure [11]. With this state-chain structure, the client only needs to obtain the latest state of the keyword to execute subsequent operations, which greatly reduces the computational and communication overheads of the scheme.
- We conduct a comprehensive analysis to prove that CKSE maintains forward and Type-II backward privacy. Additionally, we implement CKSE and perform experiments to evaluate its performance in comparison with ODXT [1]. The results demonstrate that our scheme achieves search performance comparable to ODXT [1].

2. Preliminaries

This section presents the notations used in the paper as well as the cryptographic background and definitions for SSE.

2.1. Notations

We use $x \xrightarrow{\$} X$ to denote that an element x is uniformly and randomly sampled from the set X. For a security parameter $\lambda \in \mathbb{N}$, we refer to $poly(\lambda)$ and $negl(\lambda)$ as the unspecified polynomial and negligible functions of λ, respectively. We store all documents and their respective contained keywords $w \in \mathcal{W}$ in the database **DB** as keyword/document identifier pairs (w, ind), where $\mathcal{W}$ is the set of all keywords in **DB**, and ind is the file identifier. We also denote by $|\mathcal{W}|$ the number of distinct keywords, and by $\mathbf{DB}(w)$ the set of all documents containing the keyword w.

In addition, we use $q = (w_1 \wedge w_2 \cdots \wedge w_n)$ to denote a conjunctive query, and assume that w_1 is the least frequent term in the conjunctive, called s-term, while the remaining keywords in q are called x-terms. The result of a conjunctive query is expressed as $\mathbf{DB}(q) = \cap_{i=1}^{n} \mathbf{DB}(w_i)$, i.e., the intersection of the search results for all keywords $w_i, i \in \{1, \cdots, n\}$.

2.2. Decisional Diffie-Hellman Assumption

Let g be a uniformly sampled generator for the $p = p(\lambda)$ order cyclic group $\mathbb{G}$. The Decisional Diffie-Hellman (DDH) assumption is that for any Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$, the probability of distinguishing (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^c) is negligible, formally defined as:

$$\left| \Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1] \right| \leq negl(\lambda) \tag{1}$$

2.3. Dynamic Searchable Encryption

A dynamic searchable encryption scheme consists of three parts: one $Setup$ algorithm, two protocols $Update$ and $Search$ that are run by client and server, as follows.

$Setup(\lambda, \mathbf{DB})$ is executed unilaterally by the client. This algorithm takes the security parameter λ and the database **DB** as input and outputs the tuple $(sk, \sigma; \mathbf{EDB})$, where sk denotes the client's key, σ is the client's local state, both of which are stored locally by the client, and **EDB** is an empty encrypted database stored by the server.

$Update(sk, \sigma, w, ind, op; \mathbf{EDB})$ is executed jointly by the client and server. This protocol takes (sk, σ, w, ind, op) as input for the client, where $op \in (add, del)$ indicates the update type, and takes the encrypted database **EDB** as input for the server. Eventually, the client gets a modified local state σ' and the server gets a modified encrypted database $\mathbf{EDB}'$.

$Search(sk, \sigma, q; \mathbf{EDB})$ is executed jointly by the client and server. This protocol takes (sk, σ, q) as input for the client, and takes **EDB** as input for the client. At the end of the protocol, the client outputs the result of query $\mathbf{DB}(q)$.

Note that there are two definitions of dynamic searchable encryption [48, 49]. One is the above definition adopted in this paper. Another definition is to take the addition/deletion of the entire file as an update operation, which is functionally equivalent to performing multiple add/delete operations on keyword-document identifier pairs in our definition.

Finally, we default that after receiving the file identifier contained in $\mathbf{DB}(q)$, the client still needs to generate additional interaction with the server to obtain the actual file.

2.4. Definitions of Correctness and Security

Correctness. The correctness of a dynamic searchable encryption scheme $\Sigma = (Setup, Update, Search)$ means that for any conjunctive query q, the search protocol can always return the correct result $\mathbf{DB}(q)$.

Security. The security of a dynamic searchable encryption scheme is described by a leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Updt}, \mathcal{L}^{Srch})$, where $\mathcal{L}^{Stp}$, $\mathcal{L}^{Updt}$ and $\mathcal{L}^{Srch}$ represent the leakage information captured in the setup, update and search, respectively, which can be learned by an adversary.

If an adversary server cannot learn any private information except those contained in leakage function, the dynamic searchable encryption scheme is secure. Formally, the security of a dynamic searchable encryption scheme can be proven by two games of I_{DEAL} and R_{EAL}.

Table 1
Comparison of existing schemes with proposed scheme.

Scheme	Query Type	Update cost	Search cost	Forward Privacy	Backward Privacy
[1]	Conjunctive	$O(1)$	$O(na_{w_{min}})$	✓	II
[16]	Single	$O(\log P)$	$O(a_w + \log P)$	✓	II
[16]	Single	$O(\log^3 P)$	$O(n_w \log i_w + \log^2 \lvert\mathcal{W}\rvert)$	✓	III
[35]	Single	$O(\lvert D\rvert)$	$O(a_w \lvert D\rvert)$	✓	I^-
[40]	Conjunctive	$O(a_f)$	$O(a_{w_{min}})$	✓	I^-
[41]	Conjunctive	$O(1)$	$O(na_{w_{min}})$	✓	×
[42]	Conjunctive	$O(\log N \log_k P)$	$O(na_{w_{min}} \log N \log_k P)$	✓	I
Ours	Conjunctive	$O(1)$	$O(na_{w_{min}})$	✓	II

P is the number of keyword/document pairs, $|D|$ is the number of total files, and $|s|$ is the number of authorized user. For keyword w, a_w is the total number of keyword updates, i_w is the total number of Add queries, n_w is the number of files currently containing w, $a_{w_{min}}$ is the number of the least update keyword in $q = (w_1 \wedge w_2 \wedge, \cdots, \wedge w_n)$, and n is the number of keyword in conjunctive query q. a_f is the number of updates for f.

- $\mathrm{R_{EAL}}^{\Sigma}_{\mathcal{A}}(\lambda)$: This game first runs $Setup(\lambda, \mathbf{DB})$ algorithm against database **DB** chosen by adversary $\mathcal{A}$ to obtain an encrypted database **EDB**. Then, the game executes $Search(sk, \sigma, q; \mathbf{EDB})$ or $Update(sk, \sigma, w, ind, op; \mathbf{EDB})$ protocol based on a series of queries q_i performed by $\mathcal{A}$. Finally, according to the returned result, adversary $\mathcal{A}$ outputs bit $b \in \{0, 1\}$.
- $\mathrm{I_{DEAL}}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$: After adversary $\mathcal{A}$ selects database **DB**, simulator $\mathcal{S}$ returns encrypted database $\mathbf{EDB} \leftarrow \mathcal{S}(\mathcal{L}^{Stp}(\mathbf{DB}))$ to $\mathcal{A}$ by the leakage function. Then, simulator $\mathcal{S}$ executes $\mathcal{S}(\mathcal{L}^{Srch}(q_i))$ or $\mathcal{S}(\mathcal{L}^{Updt}(q_i))$ according to a series of queries performed by $\mathcal{A}$. Finally, according to the returned result, adversary $\mathcal{A}$ outputs bit $b \in \{0, 1\}$.

Definition 1 ($\mathcal{L}$-adaptive Security)**.** *A DSSE scheme is $\mathcal{L}$-adaptively secure if for any PPT adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ such that:*

$$\left|\Pr\left(\mathrm{R_{EAL}}^{\Sigma}_{\mathcal{A}}(\lambda) = 1\right) - \Pr\left(\mathrm{I_{DEAL}}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda) = 1\right)\right| \leq negl(\lambda) \tag{2}$$

2.5. Forward and Backward Privacy

Forward privacy. For any adversary who can observe the interaction between the client and server, forward privacy can ensure that the update does not leak information about the latest addition operation, which prevents the server from matching to the new update using previous queries. The formal definition of forward privacy is as follows [12].

Definition 2 (Forward Privacy)**.** *An $\mathcal{L}$-adaptively-secure SSE scheme is forward private, if leakage function $\mathcal{L}^{Updt}$ can be expressed in the following form:*

$$\mathcal{L}^{Updt}(op, (w, ind)) = \mathcal{L}'(op, ind) \tag{3}$$

where $\mathcal{L}'$ is stateless function.

Backward privacy. Backward privacy ensures that the server cannot match files that were added and then removed. Bost at al. [12] formally defines three types of backward privacy: Type-I, Type-II, and Type-III. The security decreases from Type-I to Type-III. Type-I and Type-II are considered to be strong backward private, while Type-III is a weaker backward private. Note that Type-III backward privacy leaks the timestamp of when the files were deleted. For cloud-IoT system, time is a critical piece of information that many attacks [50, 51] can exploit to break the security of the system. For DSSE, the adversary can correlate the information of subsequent queries or make statistical inferences based on when the file was deleted. Therefore, for the DSSE scheme deployed in cloud-IoT system, it is highly desire to reach higher level of backward privacy. We focus on **Type-II** backward privacy involved in the subsequent content. This kind of backward privacy allows the scheme to leak the file identifier containing the keyword w and the timestamp, when keyword-identifier pair (w, ind) was inserted into the database, and the number of keyword updates. Before formally defining **Type-II** backward privacy, we introduce two related functions **TimeDB**(w) and **Updt**(w).

Let **Q** be a list maintaining all the search queries (u, w) and the update queries $(t, op, (ind, w))$, where t denotes the timestamp of query. **TimeDB**(w) consists of the files containing the keywords w that have not yet been deleted, along with their timestamps of insertion, that is,

$$\begin{aligned}\mathbf{TimeDB}(w) = &\{(t, ind) | (t, add, (w, ind)) \in \mathbf{Q} \text{ and} \\ &\forall t', (t', del, (w, ind)) \notin \mathbf{Q}\}\end{aligned} \tag{4}$$

Updt(w) is the function that contain the timestamp of each update of the keyword, which is defined as:

$$\begin{aligned}\mathbf{Updt}(w) = &\{t | (t, add, (w, ind)) \in \mathbf{Q} \text{ or} \\ &(t, del, (w, ind)) \in \mathbf{Q}\}\end{aligned} \tag{5}$$

Definition 3 (Backward Privacy)**.** *An $\mathcal{L}$-adaptively-secure SSE scheme is **Type-II** backward private, if leakage function $\mathcal{L}^{Updt}$ and $\mathcal{L}^{Srch}$ can be written as following form:*

$$\begin{aligned}&\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op, w) \text{ and} \\ &\mathcal{L}^{Srch}(w) = \mathcal{L}''(\mathbf{TimeDB}(w), \mathbf{Updt}(w))\end{aligned} \tag{6}$$

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

3. CKSE: An Efficient Conjunctive Keyword Searchable Encryption Scheme

In this section, we present our CKSE, an efficient conjunctive keyword searchable symmetric encryption scheme. This scheme achieves more efficient deletion operations than ODXT [1], while reducing communication and computational overheads as much as possible. Moreover, CKSE maintains the forward and **Type-II** backward private.

3.1. System Model of CKSE

Our scheme is designed to achieve more comprehensive, efficient and secure conjunctive keyword search in cloud-IoT systems. Smart home is a typical cloud-IoT system, which includes the collection and transmission of data, e.g., temperature, health and air quality, etc., by smart devices and the query of various data by the host. Taking the smart home as an example, Fig. 1 shows the system model of CKSE for smart home. It consists of three kinds of entities as follows.

Data Owner (DO). DO encrypts the collected data and uploads it to the cloud server, and performs real-time updates on the data.

Data User (DU). DU initiates search queries to the cloud server and decrypts the obtained encrypted results.

Algorithm 1 CKSE.Setup(λ)

Client:

1: $K_S \xleftarrow{\$} \{0,1\}^\lambda$ for PRF F
2: $K_X, K_Y, K_Z \xleftarrow{\$} \{0,1\}^\lambda$ for PRF F_p
3: $\mathbf{W} \leftarrow$ empty map

Server:

4: $\mathbf{T}, \mathbf{X} \leftarrow$ empty map

Algorithm 2 CKSE.Update($K_S, K_X, K_Y, K_Z, \mathbf{W}, w, ind, op; \mathbf{T}, \mathbf{X}$)

Client:

1: $K_w \,||\, K_w^* \leftarrow F(K_S, w)$
2: $(ST_c, c) \leftarrow \mathbf{W}[w]$
3: **if** $(ST_c, c) = \perp$ **then**
4: $\quad ST_0 \xleftarrow{\$} \{0,1\}^\lambda, c \leftarrow 0$
5: **end if**
6: $ST_{c+1} \xleftarrow{\$} \{0,1\}^\lambda$
7: $\mathbf{W}[w] \leftarrow (ST_{c+1}, c+1)$
8: $u_{c+1} \leftarrow H_1(K_w, ST_{c+1})$
9: $e_{c+1} \leftarrow (ind||op) \oplus H_2(K_w^*, c+1)$
10: $C_{ST_c} \leftarrow ST_c \oplus H_3(K_w, ST_{c+1})$
11: $\alpha \leftarrow F_p(K_Y, ind) \cdot (F_p(K_Z, w||c+1))^{-1}$
12: $xtag \leftarrow g^{F_p(K_X, w||op) \cdot F_p(K_I, ind)}$
13: $Send(u_{c+1}, e_{c+1}, C_{ST_c}, \alpha, xtag) to server$

Server:

14: $\mathbf{T}[u_{c+1}] = (e_{c+1}, C_{ST_c}, \alpha)$
15: $\mathbf{X}[xtag] = 1$

Cloud Server (CS). CS stores the encrypted data uploaded by DO and performs ciphertext retrieval on the search queries issued by DU. CS is a semi-honest entity provided by a third party. It strictly follows the steps of the protocol but may curiously learn information during the execution process.

Like all existing DSSE schemes, the security of our scheme relies on a "strong" assumption that the client's key can always be protected and will not be compromised [52]. Therefore, the issue of key sharing and distribution is not our concern. Since both DO and DU are legal, and DO can also act as DU, we refer DO and DU collectively as Clients in the following text.

3.2. Construction of CKSE

Although the basic idea of CKSE is similar to ODXT [1], the key difference between them is that CKSE achieves effective deletion by designing a new oblivious dynamic cross-tags. Also unlike ODXT [1] which uses Mitra for the instantiation, causing unnecessary computational and communication overhead, CKSE uses a state chain structure to complete the instantiation, which reduces the computation and communication overhead. To elucidate the core idea behind CKSE, we go back to why ODXT [1] lacks the above features.

First, ODXT [1] cannot delete a file in certain circumstances. In order to execute effective retrieval in XSet, the cross-tag known as *xtag* constructed by ODXT [1] is split into two parts, one part is related to the pair (ind_j, op) involving the *s*-term w_1 and the other is related to the *x*-terms w_i, $i \in 2, \cdots, n$. It can be seen that for the same file, the server can only recognize the update of w_1 according to the *xtag*. Consider the case given in the introduction section again: $[1, add, (w_1, ind_1)]$, $[2, add, (w_2, ind_1)]$, $[3, search, (w_1 \wedge w_2)]$, $[4, del, (w_2, ind_1)]$, $[5, search, (w_1 \wedge w_2)]$. Since the update type in the cross-tag only corresponds to the files containing w_1, the cross-tag will not be affected by the deletion operation at time 4. For the search query at time 5, the final retrieval result will still contain what has been removed. This is clearly unreasonable (w_2 is not included in ind_1 at this time). In addition, ODXT [1] achieves conjunctive search based on Mitra's framework. During the search, the client needs to pre-compute all the locations of update involving the *s*-term w_1 and send them to the server. Afterward, subsequent search operations can be performed. However, the computational and communication overhead on the client side scales with the frequency of w_1. Consequently, when the *s*-term has very high frequency of updates, the scheme will generate lots of unnecessary search overhead.

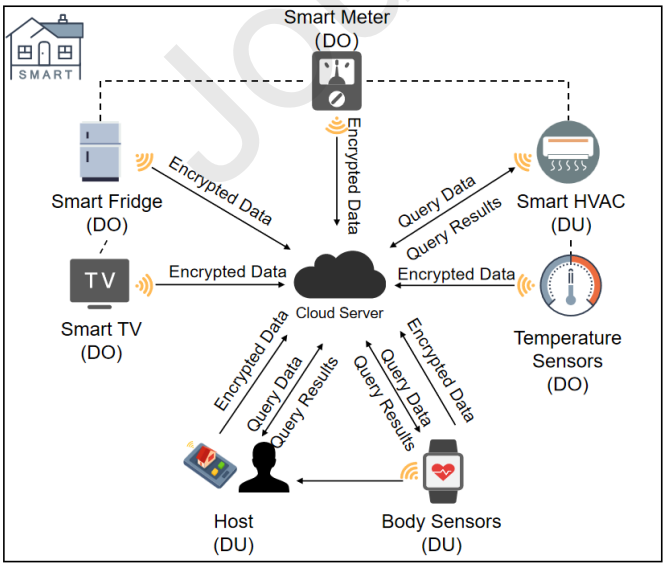


Fig. 1. The system model of CKSE for smart home.

A goal of CKSE is to incorporate the update operations of *x*-terms w_i, $i \in 2, \cdots, n$, into the oblivious cross-tag computation and make it unnecessary for clients to obtain all the locations of keywords during the search, thereby avoiding the ineffective deletion and reducing the overhead. Algorithms 1 to 3 summarize the ***Setup***, ***Update*** and ***Search*** procedures of our CKSE, respectively.

Setup. In this algorithm, the client generates λ-bit keys K_S, K_X, K_Y and K_Z for the Pseudo-Random Functions (PRFs) F and F_p respectively and holds an empty set $\mathbf{W}$ to store the state of each keyword. On the server side, two empty sets $\mathbf{T}$ and $\mathbf{X}$ are generated to store encrypted indexes.

Update. When updating a keyword/file identifier pair $[add/del, (w, ind)]$, the client first queries the state ST_c and update times c of the keyword w according to $\mathbf{W}[w]$, and then randomly generates a new state ST_{c+1} and updates $\mathbf{W}[w]$ (lines 1-7). Next, the client uses the new state ST_{c+1} and several hash functions to compute encrypted entries e_{c+1} along with location u_{c+1}, as well as a state token C_{ST_c} to trace the previous state of the keyword w (lines 8-10). More importantly, in order to achieve subsequent conjunctive queries, the client needs to obtain a blinding factor α and a cross-tag *xtag* so that the server can learn whether a fixed file contains all the keywords in the conjunctive query (lines 11-12). Finally, the client sends $(u_{c+1}, e_{c+1}, C_{ST_c}, \alpha, xtag)$ to the server, who stores $(e_{c+1}, C_{ST_c}, \alpha)$ in $\mathbf{T}[u_{c+1}]$ and sets the corresponding $\mathbf{X}[xtag]$ to 1 (lines 13-15).

Search. Assume that w_1 is the keyword with the least update frequency. In order to achieve the search query for $(w_1 \wedge w_2 \wedge \cdots \wedge w_n)$, the client uses the currently state of the *s*-term w_1 to get the cross-token pair $(xtoken_{add}, xtoken_{del})$ of $w_i, i \in 2, \cdots, n$, which involves each update of w_1. The search token $(K_w, ST_{c_1}, c_1, \mathbf{xtoken})$ is then send to the server (lines 20-29). After receiving the search token, the server first computes the location u_j according to the current state of w_1, and takes out the current encrypted entry e_j, state token $C_{ST_{j-1}}$ and blinding factor α_j from $\mathbf{T}[u_j]$ (lines 32-34). Upon using $(xtoken_{add}, xtoken_{del})$ and α_j to obtain

Algorithm 3 CKSE.Search($K_S, K_X, K_Y, K_Z, \mathbf{W}, w; \mathbf{T}, \mathbf{X}$)

Client:
1: Assume w_1 is the keyword with least updates
2: $K_w \| K_w^* \leftarrow F(K_S, w_1)$
3: $(ST_{c_1}, c_1) \leftarrow \mathbf{W}[w_1]$
4: **for** $j = 1$ to c_1 **do**
5: **for** $i = 2$ to n **do**
6: $\mathbf{xtoken}[i, j] \leftarrow g^{F_p(K_X, w_i \| add/del) \cdot F_p(K_Z, w_1 \| j)}$
7: **end for**
8: **end for**
9: Send $(K_w, ST_{c_1}, c_1, \mathbf{xtoken}[1], \cdots, \mathbf{xtoken}[c_1])$ to server
Server:
10: $\mathbf{Val} \leftarrow \emptyset$
11: **for** j = c_1 to 1 **do**
12: $cnt_j = 1$
13: $u_j \leftarrow H_1(K_w, ST_j)$
14: $(e_j, C_{ST_{j-1}}, \alpha_j) \leftarrow \mathbf{T}[u_j]$
15: **for** i = 2 to n **do**
16: $(xtoken_{add}, xtoken_{del}) \leftarrow \mathbf{xtoken}[i, j]$
17: **if** $\mathbf{X}[(xtoken_{add})^{\alpha_j}] = 1, \mathbf{X}[(xtoken_{del})^{\alpha_j}] = \perp$ **then**
18: $cnt_j = cnt_j + 1$
19: **end if**
20: **end for**
21: $\mathbf{Val} \leftarrow (cnt_j, e_j, j)$
22: $ST_{j-1} \leftarrow C_{ST_{j-1}} \oplus H_3(K_w, ST_j)$
23: **end for**
24: Send **Val** to client
Client:
25: $\mathbf{Res} \leftarrow \emptyset$
26: **for** each $(cnt_j, e_j, j) \in \mathbf{Val}$ **do**
27: $(ind_j \| op_j) \leftarrow e_j \oplus H_2(K_w^*, j)$
28: **if** $op_j = add, cnt_j = n$ **then**
29: $\mathbf{Res} \leftarrow \mathbf{Res} \cup ind_j$
30: **else if** $op_j = add, cnt_j > 0$ **then**
31: $\mathbf{Res} \leftarrow \mathbf{Res} \setminus ind_j$
32: **end if**
33: **end for**
34: **return Res**

the cross-tag *xtag*, the server exploits it to learn whether the current update contains $w_i, i \in 2, \cdots, n$ (lines 35-40). Similar to locations, the server also observes *xtag* during the update. The encrypted entry and counter are then stored in the list **Val**, and the state token and the current state are used to infer a previous state by the server (lines 41-42). Iteratively (lines 31-43), the server obtains all the states and corresponding encrypted entries about $(w_1 \wedge w_2 \wedge \cdots \wedge w_n)$, and sends **Val** to the client (line 44). Finally, the client decrypts the encrypted entries locally and further filters them (lines 45-55).

We now further explain the important features of our CKSE.

3.2.1. New Cross-Tag

To support conjunctive keyword search while achieving efficient deletion operations, the first important feature of CKSE is to construct a new cross-tag as:

$$xtag_{i,j,op} \leftarrow g^{F_p(K_X, w_i \| op) \cdot F_p(K_Y, ind_j)} \tag{7}$$

Conceptually, our cross-tag is also divided into two parts. But unlike ODXT [1], we integrate the update types of w_i, $i \in 2, \cdots, n$, into the generation of cross-tag. The first part of our cross tag contains w_i along with update type, and the second part is only related to the file identifier containing w_1. Next, to perform effective conjunctive retrieval with new cross-tag, we modify the cross-token to add/delete token pairs as:

$$\begin{aligned} xtoken_{i,j,add} &\leftarrow g^{F_p(K_X, w_i \| add) \cdot F_p(K_Z, w_1 \| j)} \\ xtoken_{i,j,del} &\leftarrow g^{F_p(K_X, w_i \| del) \cdot F_p(K_Z, w_1 \| j)} \end{aligned} \tag{8}$$

Upon receiving the add/delete token pairs, the server obtains a blind factor α_j involving the j^{th} update of w_1 from TSet:

$$\alpha_j \leftarrow F_p(K_I, ind_j) \cdot (F_p(K_Z, w_1 \| j))^{-1} \tag{9}$$

Given the add/delete token pairs $xtoken_{i,j,op}, op = \{add, del\}$ and a blind factor α_j, the sever can easily carries out the oblivious cross-tag computation as:

$$xtag_{i,j} = (xtoken_{i,j,op})^{\alpha_j} \tag{10}$$

thereby detecting whether each update of w_1 contains the *x*-term w_i.

To see why this is useful, recall that in our CKSE, we bundle the update types with keywords and introduce them into the oblivious cross-tag computation. After the delete operation on *x*-term is completed, the cross-tag calculated by the server using the add/delete token pair during the search can implicitly identify the deletion update, while the delete operations related to *s*-term are filtered out locally by the client, thereby avoiding the ineffective deletion.

3.2.2. State Chain Structure

CKSE achieves comprehensive deletion operations based on ODXT [1] but it would need more overheads to do so (e.g., additional group exponentiation operations). It is known that some existing schemes using a state-based approach impose less overhead [11, 33, 34]. Inspired by the hash chain, we exploit a state chain structure to reduce the computational and communication overhead on the client side. This structure enables the client to perform subsequent conjunctive queries only by obtaining the latest state of *s*-term w_1 during the search. Concretely, in the update protocol, the client randomly generates a state ST for each update of the keyword and stores it in $\mathbf{W}$. More importantly, this state is connected by a state token C_{ST}. Therefore, the client only takes out the current state of the keyword w_1 from $\mathbf{W}$ and sends it along with the latest state token to the server during the search. The server uses the state token to trace back to the previous state of w_1, i.e., each update of w_1. Fig. 2 depicts this state chain structure.

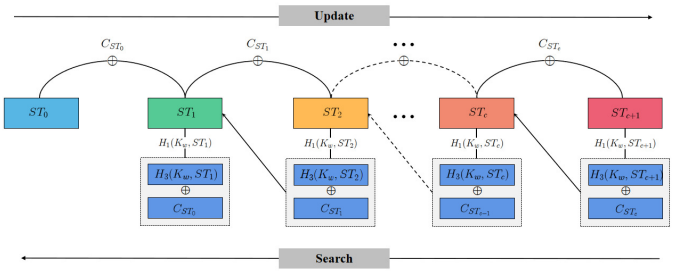


Fig. 2. The state chain structure.

Note that this structure can also support CKSE to achieve forward privacy. Previously, many forward privacy schemes were constructed based on Oblivious RAM (ORAM) structures. The main issue with this approach is the high communication cost caused by using ORAM-like structures. Only a few DSSE schemes avoid using ORAM. Among them, Bost et al. [11] proposed a one-way trapdoor permutation, which effectively reduces communication costs but is still limited by public-key cryptographic primitives. The advantage of the state chain structure lies in its use of symmetric cryptographic primitives, ensuring forward privacy based on the same principle as [11].

3.3. Efficiency of CKSE

3.3.1. Client and Server Storage

In CKSE, the client stores four λ-bit secret keys, K_S, K_X, K_Y and K_Z, for PRFs F and F_p, respectively, and a map $\mathbf{W}$ containing the states of all the keywords. Initially, $\mathbf{W}$ is the empty map, and after N updates, the size of $\mathbf{W}$ grows to $O(|\mathcal{W}| \cdot \log N)$, where $|\mathcal{W}|$ is the total number of keywords. On the server side, it needs to store the maps $\mathbf{T}$ and $\mathbf{X}$, and they are both initialized to be empty. After N updates, the server storage grows to $O(N)$. It is clear that the client storage grows logarithmically with the number of update operations and the server storage grows linearly with the number of update operations.

3.3.2. Update and Search Overhead

During the update, since the numbers of operations for client and server are constant, the computational overheads of both scale with $O(1)$, and the same is true for communication overheads of updating a single keyword-file identifier pair (w, ind). In the search protocol, for each update of w_1, the client needs to execute only $n-1$ exponential operations, compared to the overhead of computing all the locations of updates involving w_1 required by ODXT [1]. Therefore, the computational complexity is $O((n-1) \cdot a_{w_1})$ and the same is true for the communication overhead, where a_w is the total number of updates for w_1. For the server, the computational and communication complexity are $O(n \cdot a_{w_1})$ and $O(a_{w_1})$, respectively, which are the same as ODXT [1].

3.4. Security analysis

We now show that CKSE achieves forward privacy and Type-II backward privacy. The evolution of each state of keywords plays an important role in a search query, and the states of keywords are randomly generated during the update. For the server, the value of each state is indistinguishable from a random value, and the server cannot infer the future state and state token of the keyword using the current search token (containing the current state and state token of the keyword). Therefore, CKSE leaks no information during the update, and the forward privacy is guaranteed.

In the search protocol, the server obtains a series of locations for s-term w_1 which have been observed previously in the update protocol. This leakage helps the server to learn the timestamp of each update for w_1. In addition, for each update $(op_j, (w_1, ind_j))$ of w_1, the server learns the number of updates of the form $(op_j, (w_i, ind_j))$ for each x-term w_i, $i \in 2, \cdots, n$, along with the corresponding timestamp for each update. The definition of the above leakage is as follows:

$$\mathbf{Updt}(q) = \mathbf{Updt}(w_1) \bigcup \Big(\bigcup_{i=2}^{n} \mathbf{Updt}(w_1, w_i) \Big) \tag{11}$$

where q is a conjunctive search query and

$$\begin{aligned} \mathbf{Updt}(w_1, w_i) = &\{(t_1, t_i) | (t_1, op, (w_1, ind)) \in \mathbf{Q} \text{ and} \\ &(t_i, op, (w_i, ind)) \in \mathbf{Q}\} \end{aligned} \tag{12}$$

Except for the above leakage, the server cannot obtain any information that breaks the backward privacy of CKSE. We use the leakage function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Updt}, \mathcal{L}^{Srch})$ defined in section 2.5 to describe the leakages as mentioned above. After the leakage is captured, the formal definition of our scheme's leakage functions are as follows:

$$\begin{aligned} \mathcal{L}^{Updt}(op, (w, ind)) &= (\perp) \\ \mathcal{L}^{Srch}(q) &= (\mathbf{TimeDB}(q), \mathbf{Updt}(q)) \end{aligned} \tag{13}$$

According to *Definition 2* of forward privacy and *Definition 3* of backward privacy, our scheme achieves forward privacy and Type-II backward privacy.

Formally, the forward privacy and Type-II backward privacy of CKSE is summarized in the following theorem.

Theorem 1. *(Security of CKSE) Assume that F and F_p are secure pseudorandom function, the DDH assumption holds over the group $\mathcal{G}$, and H_1, H_2 and H_3 are hash functions modeled as random oracles. CKSE is an $\mathcal{L}$-adaptively secure SSE scheme with the leakage functions $\mathcal{L}^{Updt}(op, (w, ind)) = \perp$ and $\mathcal{L}^{Srch}(q) = (\mathbf{TimeDB}(q), \mathbf{Updt}(q))$.*

Proof. See Appendix. □

4. Performance Evaluation

In this section, we implement CKSE, and compare it with the two existing schemes, $\text{Mitra}_{\text{CONJ}}$ [1] is an instantiation of the naïve solution, and comparison with it can show the advantages of the conjunctive keyword search scheme, and ODXT [1] is the baseline of our scheme, and comparison with it can best verify our goal of achieving comprehensive deletion operations without compromising efficiency and security. Additionally, in order to clearly demonstrate the performance improvement of the state chain structure for CKSE, we utilize ODXT [1] to instantiate CKSE under the Mitra framework, named $\text{CKSE}_{\text{Mitra}}$, and compare its performance with CKSE.

4.1. Implementation and Settings

We implement CKSE and $\text{CKSE}_{\text{Mitra}}$ in Python 3.10 and use PyCrypto library and Sagemath library to achieve symmetric cryptographic operations and group-based operations, respectively. Specifically, we use AES-256 to realize PRFs F and F_p, SHA-256 for all hash operations H_1, H_2 and H_3, and the elliptic curve Curve25519 [53] for group operations in CKSE. Our scheme aims to provide efficient, privacy-preserving ciphertext retrieval and data updates in a cloud environment. IoT devices serve as auxiliary components responsible for collecting and uploading data, without impacting the core search performance of the scheme. All experiments were conducted on workstations equipped with an Intel(R) Core(TM) i7-14700K CPU (3.40 GHz), 32GB and 16GB RAM, running the Windows 11 (64-bit) operating system. When evaluating $\text{Mitra}_{\text{CONJ}}$ [1] and ODXT [1], we use the Python code released by Patranabis and Mukhopadhyay [1]. To provide a fair comparison, the specific implementations of CKSE and $\text{CKSE}_{\text{Mitra}}$ are the same as ODXT [1].

We test the performance of the schemes compared using the data from Enron email dataset[1], which is derived from the real world and consists of multiple folders containing email messages from about 150 different users. We choose 30109 emails in the sent-email folder as the file set, and apply the keyword extraction process of [54, 55] to obtain 77,000 unique keywords, which exclude some stopwords like 'a', 'the' and 'so'. All the experiments are repeated 10 times and the results are averaged over the ten runs.

Table 2
Comparison of Update Computational Overheads [ms].

Scheme / |**DB**|	$\text{Mitra}_{\text{CONJ}}$ [1]	ODXT [1]	$\text{CKSE}_{\text{Mitra}}$	CKSE
10	2.9	12.5	12.5	13.4
100	35.3	131.3	131.3	137.4
1000	242.3	1202.3	1202.3	1213.6
10000	1934.2	11534.1	11534.1	11640.3

4.2. Update Time Performance

We first compare the computational overheads of the four schemes in the update operation. We generate variable update entries with sizes $|\mathbf{DB}| = 10 \sim 10000$, where each entry consists of a keyword and a file identifier, and test each scheme's overall update computational overhead. Table 2 shows that the update computational overhead of

[1] Enron Email Dataset: available online at https://www.cs.cmu.edu/~enron/.

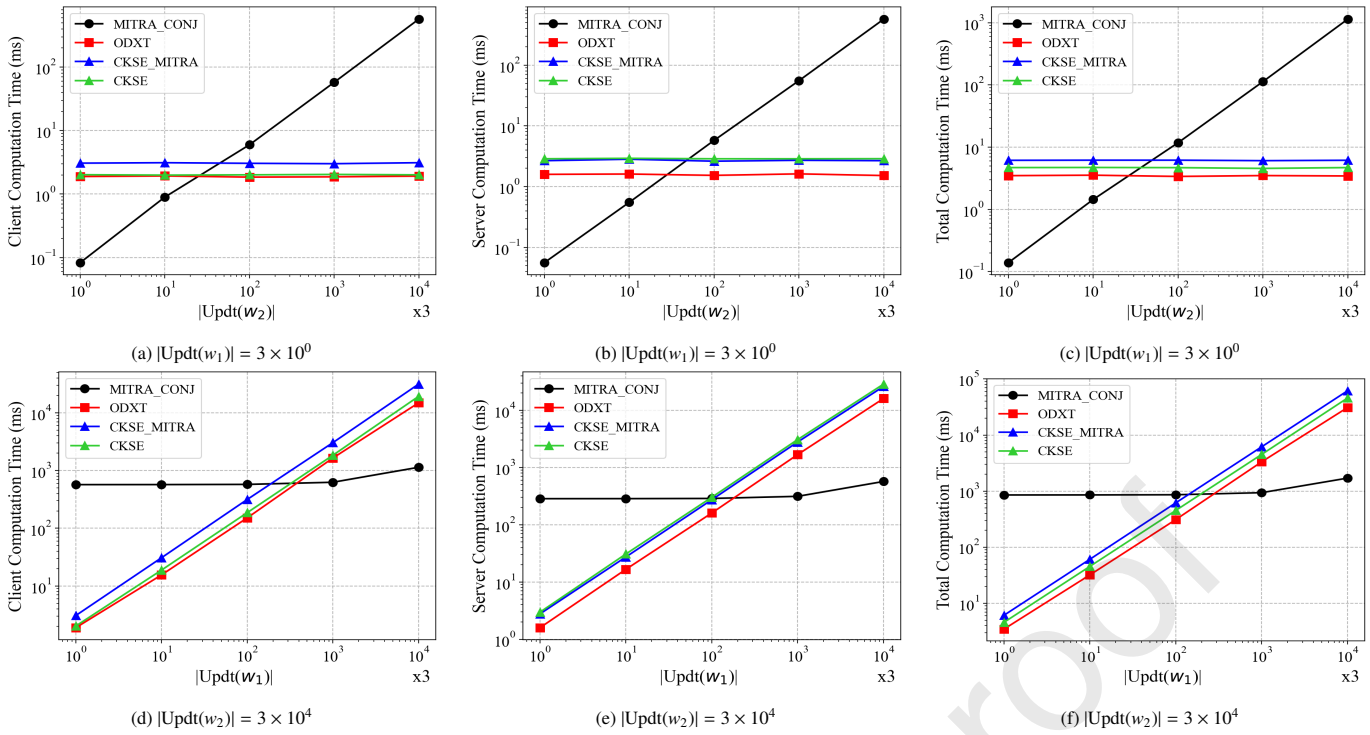


Fig. 3. Client and server computation times for two-keyword conjunctive search query $q = (w_1 \wedge w_2)$.

$\text{Mitra}_{\text{CONJ}}$ [1] is the least, about one order of magnitude smaller than those of the other schemes. This is because $\text{Mitra}_{\text{CONJ}}$ [1] only needs to consider update process for single keyword. It does not need to prepare for the subsequent conjunctive keyword search like ODXT [1], $\text{CKSE}_{\text{Mitra}}$ and CKSE, which greatly reduces the computational overhead of the update. However, the excellent update performance of $\text{Mitra}_{\text{CONJ}}$ [1] is at the cost of higher overhead in the search, as will be shown later. Since the operations of ODXT [1] and $\text{CKSE}_{\text{Mitra}}$ during the update are the same, they maintain the same computational overheads. The computational overhead of our CKSE is slightly higher than ODXT [1] and $\text{CKSE}_{\text{Mitra}}$. This is because unlike ODXT [1] and $\text{CKSE}_{\text{Mitra}}$, CKSE executes additional hash and *XOR* operations in the update. However, these additional operations will help CKSE to perform better during the search.

4.3. Search Time Performance

We next compare the computational overheads of the client and server for the four schemes in the cases of two-keyword conjunctive queries $q = (w_1 \wedge w_2)$ and four-keyword conjunctive queries $q = (w_1 \wedge \cdots \wedge w_4)$. We execute two types of experiments in each case. In the first type, we set the update frequency of the s-term w_1 to constant $|\text{Updt}(w_1)| = 3$ and the update frequency of the x-term w_2 to $|\text{Updt}(w_2)| = 3 \times 10^0 \sim 3 \times 10^4$, while in the second type, we set $|\text{Updt}(w_1)|$ to $3 \times 10^0 \sim 3 \times 10^4$ and fix $|\text{Updt}(w_2)| = 3 \times 10^4$. Additionally, in the case of four-keyword conjunctive queries $q = (w_1 \wedge \cdots \wedge w_4)$, the values of $|\text{Updt}(w_3)|$ and $|\text{Updt}(w_4)|$ remain constant at 3×10^4 for both the experimental types.

4.3.1. Two-keyword Conjunctions

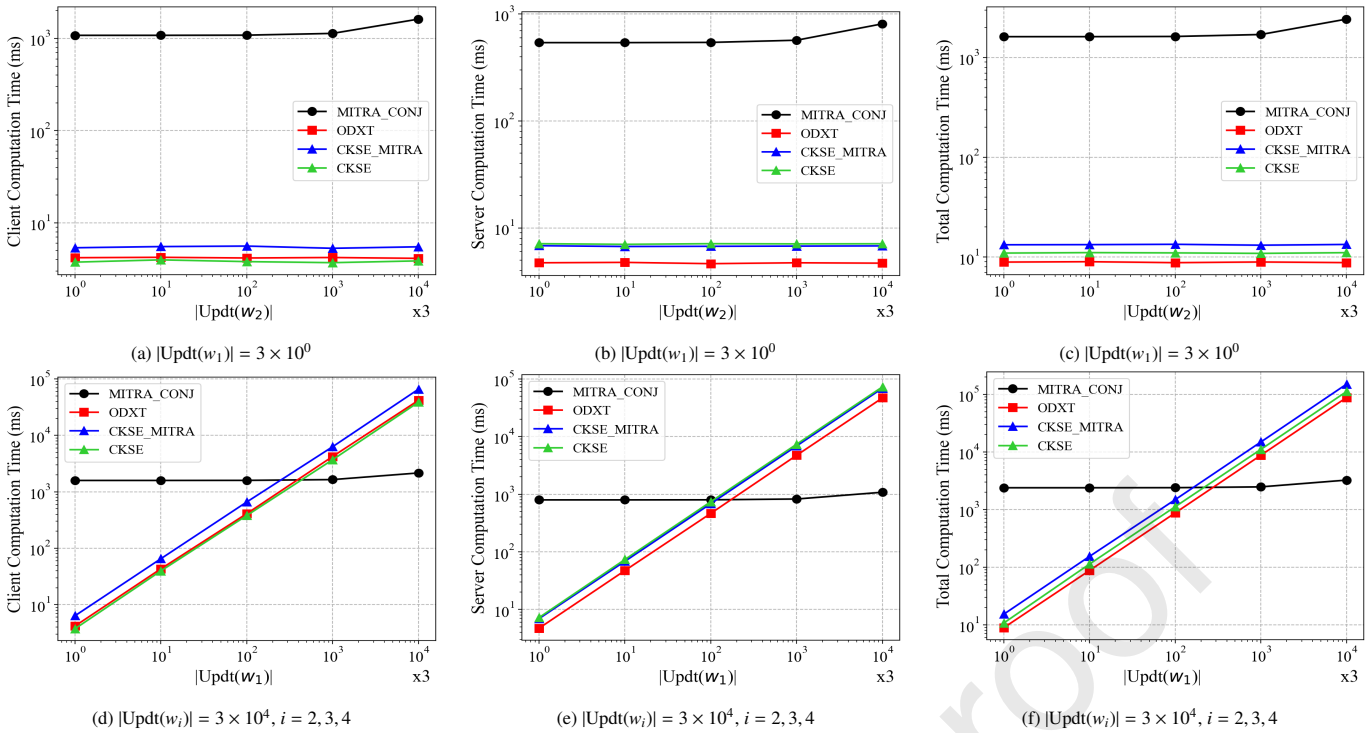
Fig. 3 compares the computational overheads of the client and server for the four schemes in the two-keyword conjunctive search. The first thing to note is that the computational overheads of ODXT [1], $\text{CKSE}_{\text{Mitra}}$ and CKSE are proportional to the update frequency of s-term w_1, and they are independent of the update frequency of x-term w_2, which is consistent with our analysis of CKSE in Subsection 3.2. By contrast, the computational overhead of $\text{Mitra}_{\text{CONJ}}$ [1] is mainly proportional to the update frequency of x-term w_2, and its computational overhead is higher than the other schemes in most cases (when $|\text{Updt}(w_1)| = 3 \times 10^1$, $|\text{Updt}(w_2)| = 3 \times 10^4$, CKSE takes 45.8ms, $\text{CKSE}_{\text{Mitra}}$ takes 61.1ms, ODXT [1] takes 35.2ms, and $\text{Mitra}_{\text{CONJ}}$ [1] takes 857.3ms, which is more than 20 times the cost of other schemes). On client side, CKSE outperforms $\text{CKSE}_{\text{Mitra}}$, and it matches ODXT [1]. This is due to the fact that compared to $\text{CKSE}_{\text{Mitra}}$, CKSE does not need to compute all the locations of the keywords, and compared to ODXT [1], CKSE does not need to compute all the locations of the keywords but requires an additional cross-token $xtoken_{\{add/del\}}$ (when $|\text{Updt}(w_1)| = 3 \times 10^4$, $|\text{Updt}(w_2)| = 3 \times 10^4$, CKSE takes 19.3s, $\text{CKSE}_{\text{Mitra}}$ takes 32.5s). On server side, CKSE has slightly higher computational overhead than ODXT [1], and it matches $\text{CKSE}_{\text{Mitra}}$. Although the search computational overhead of CKSE is slightly higher than ODXT [1], CKSE supports more efficient deletion function, which is critical in practice.

4.3.2. Multi-keyword Conjunctions

In Fig. 4, we compare the computational overheads of the client and server for the four schemes in the four-keyword conjunctive queries $q = (w_1 \wedge \cdots \wedge w_4)$. The trends of the four schemes in Fig. 4 are similar to those shown in Fig. 3, with one obvious exception. Specifically, in Fig. 4 (a)-(c), the computational overhead of $\text{Mitra}_{\text{CONJ}}$ [1] is no longer proportional to the update frequency of w_2. This is because the keywords w_3 and w_4 have the higher update frequency $|\text{Updt}(w_3)| = |\text{Updt}(w_4)| = 3 \times 10^4$ in x-terms, which increases the computational overhead of $\text{Mitra}_{\text{CONJ}}$ [1] to an extremely high level (when $|\text{Updt}(w_1)| = 3 \times 10^0$, $|\text{Updt}(w_2)| = |\text{Updt}(w_3)| = |\text{Updt}(w_4)| = 3 \times 10^4$, CKSE takes 11.1ms, $\text{CKSE}_{\text{Mitra}}$ takes 13.4ms, ODXT [1] takes 9.4ms, and $\text{Mitra}_{\text{CONJ}}$ [1] takes 2430.3ms, which is much more expensive than other schemes).

4.4. Communication Performance

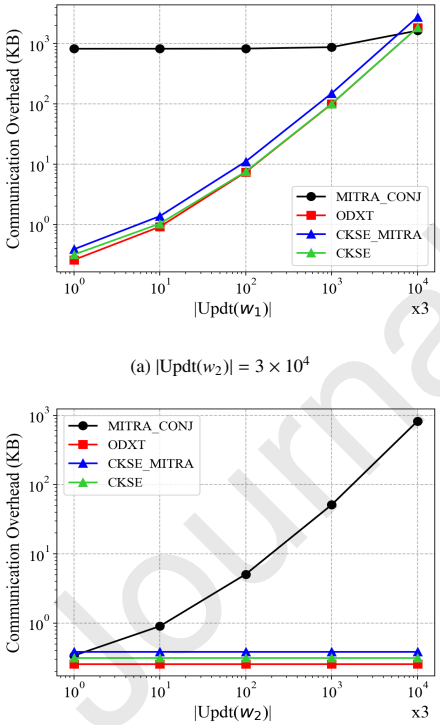
In Fig. 5, we compare the communication overheads for the four schemes in two-keyword conjunctive search. For ODXT [1], $\text{CKSE}_{\text{Mitra}}$ and CKSE, the communication overheads increase with the update frequency of s-term w_1, while the communication overhead of $\text{Mitra}_{\text{CONJ}}$ [1] increases with the update frequency of x-term w_2. Moreover, the communication overhead of $\text{Mitra}_{\text{CONJ}}$ [1] is much higher than the other schemes. In addition, CKSE slightly outperforms $\text{CKSE}_{\text{Mitra}}$ and is very



(a) $|\text{Updt}(w_1)| = 3 \times 10^0$ (b) $|\text{Updt}(w_1)| = 3 \times 10^0$ (c) $|\text{Updt}(w_1)| = 3 \times 10^0$

(d) $|\text{Updt}(w_i)| = 3 \times 10^4$, $i = 2, 3, 4$ (e) $|\text{Updt}(w_i)| = 3 \times 10^4$, $i = 2, 3, 4$ (f) $|\text{Updt}(w_i)| = 3 \times 10^4$, $i = 2, 3, 4$

Fig. 4. Computation times for client and server in a multi-keyword conjunctive search query $q = (w_1 \wedge \cdots \wedge w_4)$. In (a)-(c), $|\text{Updt}(w_3)| = |\text{Updt}(w_4)| = 3 \times 10^4$.



(a) $|\text{Updt}(w_2)| = 3 \times 10^4$

(b) $|\text{Updt}(w_1)| = 3 \times 10^0$

Fig. 5. Communication overheads in two-keyword conjunctive search query $q = (w_1 \wedge w_2)$.

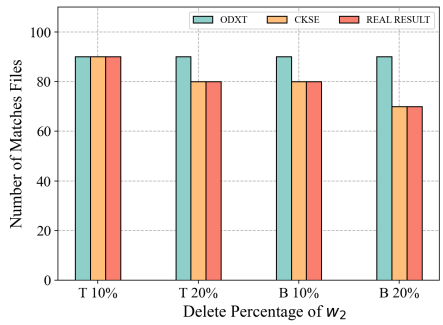
close to ODXT [1]. This is due to the fact that compared to $\text{CKSE}_{\text{Mitra}}$, CKSE omits the transmission of each location of s-term w_1, and compared to ODXT [1], CKSE omits the transmission of each location of s-term w_1 but needs to transmit additional cross-token.

To sum up, the performance of $\text{Mitra}_{\text{CONJ}}$ [1] is generally worse than the other three schemes in terms of both computational overhead and communication overhead. For our approach, CKSE outperforms $\text{CKSE}_{\text{Mitra}}$, which shows the effectiveness of the state chain structure. More importantly, compared to the state-of-the-art ODXT [1], CKSE has a similar performance. It is worth recapping that our CKSE maintains the same security level as ODXT [1], i.e., forward and Type-II backward privacy, and unlike ODXT [1], our CKSE can achieve efficient deletion operation in any case.

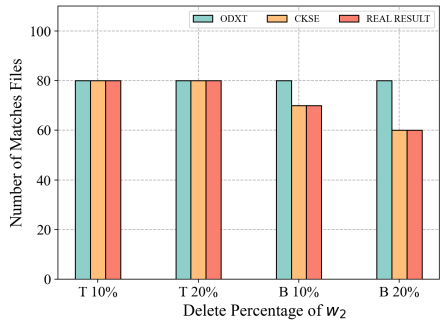
4.5. Query Result Performance

After performing different deletion operations, we compare the query results of ODXT [1] and CKSE involving search query $q = (w_1 \wedge w_2)$. We first select 100 files containing the keyword w_1 and w_2, and then delete w_1 and w_2 from the Top (T) 10%, T 20%, Bottom (B) 10%, and B 20% of the file set, respectively. Finally, we collect the query results of each scheme under different deletion scenarios. Note that w_1 is the s-term.

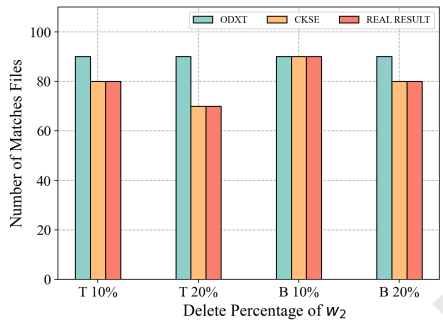
After performing deletion updates, the comparison between the query results of ODXT [1] and CKSE and the ground truth is illustrated in Fig. 6. It can be seen that CKSE consistently produces the same query results as the real results in all the 16 cases, whereas ODXT [1] fails to achieve accurate query results in 10 out of 16 cases. Ideally, deletion operations involving either w_1 or w_2 should have an impact on the final query result. However, the query result of ODXT [1] only changes with deletion operations involving w_1, and it remains unaffected by deletion operations involving w_2, which results in ODXT [1] being able to obtain accurate query results only when the set of documents with deletions of w_1 includes the documents with deletions of w_2. This limitation arises from the fact that the cross-tag *xtag* computed by ODXT [1] during the search query can only respond to the update type corresponding to s-term w_1, but ignores the update type of other keywords in conjunction. Consequently, ODXT [1] fails to detect deletion operations on keywords other than the s-term w_1, resulting in discrepancy between the query results and the actual results. In contrast, CKSE incorporates the update types of each keyword into the design of the new cross-tag, enabling it to effectively respond to deletion operations for every keyword. As a result, CKSE consistently achieves accurate query results across various scenarios.



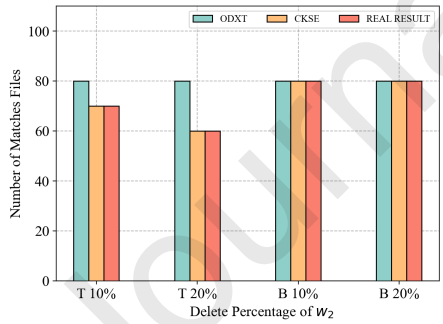
(a) Delete Percentage of w_1 = T 10%



(b) Delete Percentage of w_1 = T 20%



(c) Delete Percentage of w_1 = B 10%



(d) Delete Percentage of w_1 = B 20%

Fig. 6. The query results of search query $q = (w_1 \wedge w_2)$.

5. Conclusions and Future Work

In this work, we have designed an effective and efficient conjunctive keyword DSSE scheme called CKSE based on the state-of-the-art ODXT [1]. However, unlike ODXT [1], our scheme supports update operations in any scenario, especially for robust deletion operations, which enable the client to obtain accurate query results. Additionally, we have adopted a state chain structure to save unnecessary ODXT [1] operations during the search and achieve efficient search performance. In terms of security, our CKSE leaks no information in update and moderates leakage during the search to achieve forward privacy and Type-II backward privacy. In summary, our CKSE design comprehensively considers functionality, efficiency and security, and it offers an ideal scheme for cloud-IoT systems. In the future, extending CKSE to support more expressive queries, e.g., boolean queries, is meaningful, which will strengthen the practical applications of our scheme.

Moreover, the proposed scheme operate under the assumption that the client's key remains secure at all times, without addressing the potential risks associated with key sharing. This oversight presents challenges in real-world applications, where key exposure could allow an adversary to compromise the encrypted database and monitor update and search activities. Moreover, our scheme primarily focuses on minimizing access pattern leakage but fails to consider the vulnerability to keyword guessing attacks, which are a common concern in public key searchable encryption schemes. In future work, we aim to explore and address these two issues in DSSE, which will enable us to develop a more secure scheme for cloud-IoT systems.

6. Acknowledgements

This work was supported in part by the Major Science and Technology Projects in Yunnan Province (202202AD080013). The authors also extend their appreciation at King Khalid University for funding this work through Large Group Project under grant number RGP.2/373/45.

Appendix A. Proof of Theorem 1

We use the $\mathrm{R_{EAL}}$-$\mathrm{I_{DEAL}}$ model mentioned in Subsection 2.4 to prove the security of CKSE. Specifically, a sequence of games are constructed from $\mathrm{R_{EAL}}^{\Sigma}_{\mathcal{A}}(\lambda)$ and reached to $\mathrm{I_{DEAL}}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$. We prove that $\mathrm{R_{EAL}}^{\Sigma}_{\mathcal{A}}(\lambda)$ and $\mathrm{I_{DEAL}}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$ are indistinguishable by proving the indistinguishability between two adjacent games.

Game G_0: G_0 is the real world game $\mathrm{R_{EAL}}^{\Sigma}_{\mathcal{A}}(\lambda)$.

Game G_1: The difference between G_1 and G_0 is that G_1 replaces PRFs $F(K_S, \cdot)$, $F_p(K_X, \cdot)$, $F_p(K_Y, \cdot)$ and $F_p(K_Z, \cdot)$ with random functions $G_S(\cdot)$, $G_X(\cdot)$, $G_Y(\cdot)$ and $G_Z(\cdot)$, respectively. Specifically, $G_S(\cdot)$ is uniformly sampled from the set of all random functions on $(0,1)^{\lambda}$, while $G_X(\cdot)$, $G_Y(\cdot)$ and $G_Z(\cdot)$ are uniformly sampled from the set of all random functions on $\mathbb{Z}_p^*$. Since we cannot distinguish a pseudo-random function from a truly random function, G_1 and G_0 are indistinguishable.

Game G_2: The difference between G_2 and G_1 is that G_2 no longer calls H_1, H_2 and H_3 to generate location u, encrypted entry e and state token C_{ST} in the update protocol, but uses random numbers instead. Taking H_1 and u as an example, it replaces $u \leftarrow H_1(K_w, ST_{c+1})$ with $u \xleftarrow{\$} \{0,1\}^{\lambda}$ and executes $\mathbf{L}[K_w \| ST_{c+1}] \leftarrow u$, where $\mathbf{L}$ is a mapping maintained by G_2. Afterward, $\mathbf{H_1}[K_w \| ST_{c+1}] \leftarrow \mathbf{L}[K_w \| ST_{c+1}]$ is executed in the search protocol, where $\mathbf{H}_1$ is the table of the random oracles H_1. Thus, $\mathbf{H}_1$ is not updated immediately, and when an adversary accesses $\mathbf{H}_1[K_w \| ST_{c+1}]$ before a search query is issued, $\mathbf{H}_1[K_w \| ST_{c+1}]$ will randomly generate a value u^* that is not equal to u. If the adversary queries $\mathbf{H}_1[K_w \| ST_{c+1}]$ again after next search query, it will get the value u that has been updated to $\mathbf{H}_1$. By observing the difference between the two queries, the adversary knows that it is in game G_2. We now show that the probability of this case is negligible. Based on the above discussion, it is clear that this case will only take place if the adversary uses $K_w \| ST_{c+1}$ to query $\mathbf{H}_1$. Since ST_{c+1} is randomly generated, the adversary chooses ST_{c+1} with probability $\frac{1}{2^{\lambda}} + negl(\lambda)$. Assuming that a PPT adversary makes at most $p = poly(\lambda)$ guesses, the probability of adversary choosing ST_{c+1} is $\frac{p}{2^{\lambda}} + p \cdot negl(\lambda)$, which is negligible. H_2 and H_3 are processed in the same way as H_1 in G_2. Therefore, G_2 and G_1 are indistinguishable.

Game G_3: The difference between G_3 and G_2 is that in the search protocol of G_3, the manner of generating *xtoken* is changed. Specifically, for a conjunctive query $q = (w_1 \wedge w_2 \wedge \cdots \wedge w_n)$, G_3 first looks up the update query history of adversary to obtain the set of update operations involving s-term w_1. Then, it computes α and *xtag* involving each x-term w_i in conjunction q, and obtains *xtoken* as $xtoken = xtag^{1/\alpha}$. It

is clear that the distribution of each *xtoken* value in G_3 is the same as the its distribution in G_2. Therefore, G_3 and G_2 are indistinguishable.

Game G_4: The difference between G_4 and G_3 is that the manner of generating α is changed in the update protocol of G_4. Specifically, G_4 replace computing α in G_3 with random sampling $\alpha \xrightarrow{\$} \mathbb{Z}_p^*$. Note that α in G_3 is computed by $G_Y(\cdot)$ and the inverse of $G_Z(\cdot)$, where $G_Y(\cdot)$ and $G_Z(\cdot)$ are uniformly sampled from the set of all random functions on $\mathbb{Z}_p^*$, and the value of α in G_4 is also uniform and independent random distribution on $\mathbb{Z}_p^*$. Therefore, G_4 and G_3 are indistinguishable.

Game G_5: The difference between G_5 and G_4 is that the manner of generating *xtag* is changed in the update protocol of G_5. Specifically, G_5 replace computing *xtag* in G_4 with random sampling g^γ, where g is an uniformly sampled generator for the group $\mathbb{G}$ and $\gamma \xrightarrow{\$} \mathbb{Z}_p^*$. Since the DDH assumption holds in the group $\mathbb{G}$, the probability of a PPT adversary distinguishing $xtag = g^{G_X(\cdot)\cdot G_Y(\cdot)}$ in G_4 from $xtag = g^\gamma$ in G_5 is negligible. Therefore, G_5 and G_4 are indistinguishable.

Game G_6: The difference between G_6 and G_5 is that the manner of computing the maps involving u, e and C_{ST} are changed in the update and search protocol of G_6. Specifically, still taking u as an example, G_6 replace $\mathbf{L}[K_w \| ST_{c+1}] \xleftarrow{\$} \{0,1\}^\lambda$ with $\mathbf{L}[t] \xleftarrow{\$} \{0,1\}^\lambda$, where t is the timestamp for each update operation. Note that each state of keyword is different in G_5, and the values sampled uniformly randomly are never the same when input two different timestamps in G_6. The map involving e and C_{ST} are processed in the same way as $\mathbf{L}[\cdot]$ in G_5. This implies that G_6 and G_5 are indistinguishable.

Simulator: In $\mathrm{I}_{\mathrm{DEAL}}^{\Sigma}{}_{\mathcal{A},\mathcal{S}}(\lambda)$, the simulator $\mathcal{S}$ generates a view according to the given leakage function

$$\mathcal{L}^{Updt}(op,(w,ind)) = \perp$$
$$\mathcal{L}^{Srch}(q) = (\mathbf{TimeDB}(q), \mathbf{Updt}(q))$$

where $\mathbf{TimeDB}(q)$ and $\mathbf{Updt}(q)$ are defined in (4) and (5) of Subsection 2.5. Specifically, from $\mathcal{L}^{Updt}$, $\mathcal{S}$ gains no information about update operations, and a series of variables are generated by $\mathcal{S}$ as done by G_6. In the search protocol, $\mathcal{S}$ uses $\mathbf{Updt}(q)$ to learn the number of updates involving the s-term w_1, as well as the corresponding timestamp and x-term leakage for each update operation. It can also learn the final set of file identifiers in the conjunction by using $\mathbf{TimeDB}(q)$. Moreover, $\mathcal{S}$ can learn whether two (or more) conjunctive queries contain the same s-term w_1 by using $\mathbf{Updt}(q_1)$ and $\mathbf{Updt}(q_2)$. Note that the view generated by $\mathcal{S}$ using the above information are identical to the view in G_6.

This completes the proof of Theorem 1.

References

[1] S. Patranabis, D. Mukhopadhyay, Forward and backward private conjunctive searchable symmetric encryption, in: 28th Annual Network and Distributed System Security Symposium (NDSS 2021), The Internet Society, 2021.

[2] L. Chen, J. Li, J. Li, Toward forward and backward private dynamic searchable symmetric encryption supporting data deduplication and conjunctive queries, IEEE Internet of Things Journal 10 (19) (2023) 17408–17423.

[3] G. S. Poh, P. Gope, J. Ning, Privhome: Privacy-preserving authenticated communication in smart home environment, IEEE Transactions on Dependable and Secure Computing 18 (3) (2019) 1095–1107.

[4] Y. Li, B. Cao, M. Peng, L. Zhang, L. Zhang, D. Feng, J. Yu, Direct acyclic graph-based ledger for internet of things: Performance and security analysis, IEEE/ACM Transactions on Networking 28 (4) (2020) 1643–1656 .

[5] W. Liu, B. Cao, M. Peng, Web3 technologies: Challenges and opportunities, IEEE Network 38 (3) (2024) 187–193.

[6] J. Shu, X. Jia, K. Yang, H. Wang, Privacy-preserving task recommendation services for crowdsourcing, IEEE Transactions on Services Computing 14 (1) (2018) 235–247.

[7] C. Zhang, L. Zhu, C. Xu, J. Ni, C. Huang, X. Shen, Location privacy-preserving task recommendation with geometric range query in mobile crowdsensing, IEEE Transactions on Mobile Computing 21 (12) (2021) 4410–4425.

[8] B. Cao, Z. Wang, L. Zhang, D. Feng, M. Peng, L. Zhang, Z. Han, Blockchain systems, technologies, and applications: A methodology perspective, IEEE Communications Surveys & Tutorials 25 (1) (2022) 353–385.

[9] Y. Zhang, J. Katz, 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), 2016, pp. 707–720.

[10] C. B. Papamanthou, E. Stefanov, E. Shi, Practical dynamic searchable encryption with small leakage, in: Proc. Netw. Distrib. Syst. Secur. Symp, 2014, pp. 23–26.

[11] R. Bost, $\sum o\phi o\xi$: Forward secure searchable encryption, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1143–1154.

[12] R. Bost, B. Minaud, 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, 2017, pp. 1465–1482.

[13] S.-F. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, S. Nepal, Practical backward-secure searchable encryption from symmetric puncturable encryption, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 763–780.

[14] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, 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, 2018, pp. 1038–1055.

[15] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, J. Pieprzyk, Dynamic searchable symmetric encryption with forward and stronger backward privacy, in: European symposium on research in computer security, Springer, 2019, pp. 283–303.

[16] I. Demertzis, J. G. Chamani, D. Papadopoulos, C. Papamanthou, Dynamic searchable encryption with small client storage, Cryptology ePrint Archive.

[17] S.-F. Sun, R. Steinfeld, S. Lai, X. Yuan, A. Sakzad, J. K. Liu, S. Nepal, D. Gu, Practical non-interactive searchable encryption with forward and backward privacy, in: Usenix Network and Distributed System Security Symposium 2021, The Internet Society, 2021.

[18] P. Xu, W. Susilo, W. Wang, T. Chen, Q. Wu, K. Liang, H. Jin, ROSE: Robust searchable encryption with forward and backward security, IEEE transactions on information forensics and security 17 (2022) 1115–1130.

[19] P. Zhang, Y. Chui, H. Liu, Z. Yang, D. Wu, R. Wang, Efficient and privacy-preserving search over edge–cloud collaborative entity in IoT, IEEE Internet of Things Journal 10 (4) (2021) 3192–3205.

[20] R. Zhou, X. Zhang, X. Wang, G. Yang, H.-N. Dai, M. Liu, Device-oriented keyword-searchable encryption scheme for cloud-assisted industrial IoT, IEEE Internet of Things Journal 9 (18) (2021) 17098–17109.

[21] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, J. Pieprzyk, G. Wei, Forward and backward private dynamic searchable symmetric encryption for conjunctive queries, Cryptology ePrint Archive.

[22] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, 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, Springer, 2013, pp. 353–373.

[23] M. Haus, M. Waqas, A. Y. Ding, Y. Li, S. Tarkoma, J. Ott, Security and privacy in device-to-device (d2d) communication: A review, IEEE Communications Surveys & Tutorials 19 (2) (2017) 1054–1079.

[24] M. Waqas, S. Tu, Z. Halim, S. U. Rehman, G. Abbas, Z. H. Abbas, The role of artificial intelligence and machine learning in wireless networks security: Principle, practice and challenges, Artificial Intelligence Review 55 (7) (2022) 5215–5261.

[25] D. X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceeding 2000 IEEE symposium on security and privacy. S&P 2000, IEEE, 2000, pp. 44–55.

[26] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: Improved definitions and efficient constructions, Journal of Computer Security 19 (5) (2011) 895–934.

[27] M. Chase, S. Kamara, Structured encryption and controlled disclosure, in: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16, Springer, 2010, pp. 577–594.

[28] S. Kamara, C. Papamanthou, T. Roeder, Dynamic searchable symmetric encryption, in: Proceedings of the 2012 ACM conference on Computer and communications security, 2012, pp. 965–976.

[29] S. Liu, J. Yu, Y. Xiao, Z. Wan, S. Wang, B. Yan, BC-SABE: Blockchain-aided searchable attribute-based encryption for cloud-IoT, IEEE Internet of Things Journal 7 (9) (2020) 7851–7867.

[30] J. Yu, S. Liu, M. Xu, H. Guo, F. Zhong, W. Cheng, An efficient revocable and searchable MA-ABE scheme with blockchain assistance for C-IoT, IEEE Internet of Things Journal 10 (3) (2022) 2754–2766.

[31] H. Yin, W. Zhang, H. Deng, Z. Qin, K. Li, An attribute-based searchable encryption scheme for cloud-assisted IIoT, IEEE Internet of Things Journal 10 (12) (2023) 11014–11023.
[32] Y.-C. Chang, M. Mitzenmacher, Privacy preserving keyword searches on remote encrypted data, in: International conference on applied cryptography and network security, Springer, 2005, pp. 442–455.
[33] Y. Wei, S. Lv, X. Guo, Z. Liu, Y. Huang, B. Li, FSSE: Forward secure searchable encryption with keyed-block chains, Information Sciences 500 (2019) 113–126.
[34] X. Song, C. Dong, D. Yuan, Q. Xu, M. Zhao, Forward private searchable symmetric encryption with optimized I/O efficiency, IEEE Transactions on Dependable and Secure Computing 17 (5) (2018) 912–927.
[35] H. Dou, Z. Dan, P. Xu, W. Wang, S. Xu, T. Chen, H. Jin, Dynamic searchable symmetric encryption with strong security and robustness, IEEE Transactions on Information Forensics and Security .
[36] B. Chen, T. Xiang, D. He, H. Li, K.-K. R. Choo, BPVSE: Publicly verifiable searchable encryption for cloud-assisted electronic health records, IEEE Transactions on Information Forensics and Security 18 (2023) 3171–3184.
[37] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S.-F. Sun, D. Liu, C. Zuo, Result pattern hiding searchable encryption for conjunctive queries, in: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, 2018, pp. 745–762.
[38] S. Kamara, 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 36, Springer, 2017, pp. 94–124.
[39] Z. Wu, K. Li, VBTree: Forward secure conjunctive queries over encrypted data for cloud computing, The VLDB journal 28 (1) (2019) 25–46.
[40] L. Chen, J. Li, J. Li, Toward forward and backward private dynamic searchable symmetric encryption supporting data deduplication and conjunctive queries, IEEE Internet of Things Journal 10 (19) (2023) 17408–17423.
[41] C. Guo, W. Li, X. Tang, K.-K. R. Choo, Y. Liu, Forward private verifiable dynamic searchable symmetric encryption with efficient conjunctive query, IEEE Transactions on Dependable and Secure Computing 21 (2) (2023) 746–763.
[42] M. Li, C. Jia, R. Du, W. Shao, Forward and backward secure searchable encryption scheme supporting conjunctive queries over bipartite graphs, IEEE Transactions on Cloud Computing 11 (1) (2021) 1091–1102.
[43] D. Yuan, C. Zuo, S. Cui, G. Russello, Result-pattern-hiding conjunctive searchable symmetric encryption with forward and backward privacy, Proceedings on Privacy Enhancing Technologies.
[44] R. Li, A. X. Liu, Adaptively secure and fast processing of conjunctive queries over encrypted data, IEEE Transactions on Knowledge and Data Engineering 34 (4) (2020) 1588–1602.
[45] Y. Li, J. Ning, J. Chen, Secure and practical wildcard searchable encryption system based on inner product, IEEE Transactions on Services Computing 16 (3) (2022) 2178–2190.
[46] F. Liu, K. Xue, J. Yang, J. Zhang, Z. Huang, J. Li, D. S. Wei, Volume-hiding range searchable symmetric encryption for large-scale datasets, IEEE Transactions on Dependable and Secure Computing.
[47] M. Xie, X. Yang, H. Hong, G. Wei, Z. Zhang, A novel verifiable chinese multi-keyword fuzzy rank searchable encryption scheme in cloud environments, Future Generation Computer Systems 153 (2024) 287–300.
[48] B. Gong, G. Zheng, M. Waqas, S. Tu, S. Chen, Lcdma: Lightweight cross-domain mutual identity authentication scheme for internet of things, IEEE Internet of Things Journal 10 (14) (2023) 12590–12602.
[49] B. Gong, C. Guo, C. Guo, Y. Sun, M. Waqas, S. Chen, Slim: A secure and lightweight multi-authority attribute-based signcryption scheme for iot, IEEE Transactions on Information Forensics and Security 19 (2023) 1299–1312.
[50] S. Feghhi, D. J. Leith, A web traffic analysis attack using only timing information, IEEE Transactions on Information Forensics and Security 11 (8) (2016) 1747–1759.
[51] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, T. H. Lai, Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution, in: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2019, pp. 142–157.
[52] T. Chen, P. Xu, S. Picek, B. Luo, W. Susilo, H. Jin, K. Liang, The power of bamboo: On the post-compromise security for searchable symmetric encryption, in: 30th Annual Network and Distributed System Security Symposium, NDSS 2023, 2023.
[53] D. J. Bernstein, Curve25519: new diffie-hellman speed records, in: Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9, Springer, 2006, pp. 207–228.
[54] Z. Shang, S. Oya, A. Peter, F. Kerschbaum, Obfuscated access and search patterns in searchable encryption, arXiv preprint arXiv:2102.09651.
[55] M. S. Islam, M. Kuzu, M. Kantarcioglu, Access pattern disclosure on searchable encryption: ramification, attack and mitigation., in: Ndss, Vol. 20, Citeseer, 2012, p. 12.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: