A Survey on Searchable Symmetric Encryption

FENG LI, JIANFENG MA, and YINBIN MIAO, School of Cyber Engineering, Xidian University, China
XIMENG LIU, School of Mathematics and Computer Science, Fuzhou University, China
JIANTING NING, Fujian Provincial Key Laboratory of Network Security and Cryptology, School of Mathematics and Informatics, Fujian Normal University, China
ROBERT H. DENG, School of Information Systems, Singapore Management University, Singapore

Outsourcing data to the cloud has become prevalent, so Searchable Symmetric Encryption (SSE), one of the methods for protecting outsourced data, has arisen widespread interest. Moreover, many novel technologies and theories have emerged, especially for the attacks on SSE and privacy-preserving. But most surveys related to SSE concentrate on one aspect (e.g., single keyword search, fuzzy keyword search) or lack in-depth analysis. Therefore, we revisit the existing work and conduct a comprehensive analysis and summary. We provide an overview of state-of-the-art in SSE and focus on the privacy it can protect. Generally, (1) we study the work of the past few decades and classify SSE based on query expressiveness. Meanwhile, we summarize the existing schemes and analyze their performance on efficiency, storage space, index structures, and so on.; (2) we complement the gap in the privacy of SSE and introduce in detail the attacks and the related defenses; (3) we discuss the open issues and challenges in existing schemes and future research directions. We desire that our work will help novices to grasp and understand SSE comprehensively. We expect it can inspire the SSE community to discover more crucial leakages and design more efficient and secure constructions.

CCS Concepts: • **Security and privacy** → **Privacy-preserving protocols**; Hash functions and message authentication codes; *Cryptanalysis and other attacks*; Block and stream ciphers;

Additional Key Words and Phrases: Searchable encryption, privacy-preserving, cloud security

ACM Reference format:
Feng Li, Jianfeng Ma, Yinbin Miao, Ximeng Liu, Jianting Ning, and Robert H. Deng. 2023. A Survey on Searchable Symmetric Encryption. *ACM Comput. Surv.* 56, 5, Article 119 (November 2023), 42 pages.
https://doi.org/10.1145/3617991

This work was supported in part by the National Key Research and Development Program of China under Grant No. 2021YFB3101100; in part by China Scholarship Council; in part by the National Natural Science Foundation of China under Grant No. 61902290, 62072352 and 62202364; and in part by the Key Research and Development Program of Shaanxi under Grant No. 2020ZDLGY09-06.

Authors' addresses: F. Li, J. Ma, Y. Miao (Corresponding author), School of Cyber Engineering, Xidian University, No. 2 South Taibai Road, Xi'an, Shaanxi 710071, China; e-mails: feng.li@stu.xidian.edu.cn, jfma@mail.xidian.edu.cn, ybmiao@xidian.edu.cn; X. Liu, College of Computer and Data Science/College of Software, Fuzhou University, No. 2, Wulongjiang North Avenue, Fuzhou, Fujian 350108, China; e-mail: snbnix@gmail.com; J. Ning, College of Computer and Cyber Security, Fujian Normal University, No. 8 Xuefu South Road, Shangjie, Minhou, Fuzhou, Fujian 350117, China; e-mail: jtning88@gmail.com; R. H. Deng, School of Computing and Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902, Singapore; e-mail: robertdeng@smu.edu.sg.

0360-0300/2023/11-ART119 $15.00
https://doi.org/10.1145/3617991

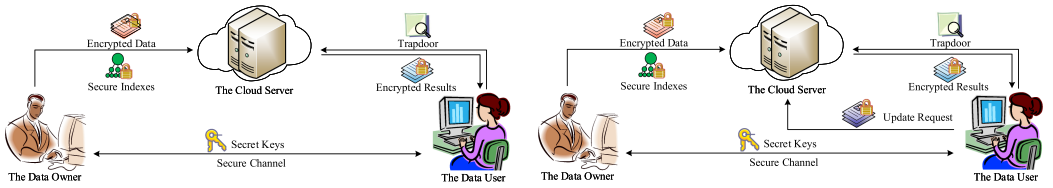
1 INTRODUCTION

The pay-as-you-demand and powerful capabilities of cloud computing prompt individuals and organizations to outsource data storage and computing tasks to the cloud server [52, 92, 182]. This has been widely accepted and applied due to the drawbacks of local storage (such as geographical restrictions, expensive labor costs, machine maintenance costs, etc.) and the advantages of cloud computing (such as flexibility, unlimited resources, affordability, etc.). However, storing data in the cloud results in a loss of user control over the data, making the privacy of outsourced data a vital concern. While it is possible to encrypt data before being outsourced using traditional encryption algorithms, this makes the usability of the data impossible or complicated. The trivial idea is to authorize the secret key to the cloud server to decrypt all the data before executing the search, which exposes the privacy of the data. Alternatively, the cloud server returns all data. The user decrypts the data prior to searching for documents of interest, which suffers from expensive communication and computation overheads. As a consequence, an approach that can perform a search on encrypted data with privacy-preserving is desired.

Searchable Encryption (**SE**) is a mechanism that allows users to search over encrypted data while preserving data privacy. It mainly consists of **Asymmetric Searchable Encryption** (**ASE**) [8] and **Searchable Symmetric Encryption** (**SSE**) [129]. This article focuses on discussing SSE, and the system model (static and dynamic) is shown in Figure 1. The data owner has a collection of documents and encrypts documents using a symmetric encryption algorithm before outsourcing them to the cloud server. Meanwhile, the data owner generates secure indexes based on the predefined keyword set, then outsources them with encrypted documents to the cloud server. To perform a search on encrypted data, the data owner needs to share secret keys with the data user over a secure channel. The data user then generates a query trapdoor that should not reveal the queried information and sends it to the cloud server. Using the trapdoor, the cloud server performs the search with secure indexes and returns matching documents to the data user.

Since the seminal article was proposed by [129], SSE has attracted extensive attention and research. Several search functionalities (such as multi-keyword search [5, 55, 152], Boolean search [1, 19, 110, 154], ranked search [16, 139, 144], etc.) have been proposed to enrich the SSE, and the security of the SSE (such as forward privacy [12, 13, 22, 155], backward privacy [13, 123, 135], access pattern [27, 34, 80], etc.) has been studied. While there has been a lot of work [11, 124] surveying SSE as comprehensively as possible, these efforts have failed to cover the progress SSE has made in recent years, especially regarding privacy-preserving and leakage-abuse attacks. Bösch et al. [11] overviewed techniques of SSE and ASE for the nonspecialist, but the survey failed to provide in-depth analysis in terms of efficiency, functionality, security, and so on. Furthermore, since the survey was published very early, it does not include the latest attacks and defenses such as access patterns and search patterns. Poh et al. [124] studied SSE from an in-depth perspective from the aspects of existing schemes' underlying designs and structures and identified open issues and future directions. However, the survey leaves out some important search functionalities (such as substring search, semantic search, etc.) and rarely analyzes the current threats SSE faces and corresponding defenses. Motivated by this, we attempt to complement the gap by providing an in-depth and comprehensive survey of SSE. Specifically, we systematically review current work on search functionalities and privacy-preserving. The specific contributions of this article are as follows:

— We study the work of the past few decades and classify SSE based on query expressiveness. At the same time, we summarize the methods used in the literature and analyze the performance on efficiency, storage space, index structures, and so on.



(a) System Model of Static SSE (b) System Model of Dynamic SSE

Fig. 1. System model.

— We complement the gap in the privacy of SSE, which has received great concerns in recent years, and introduce in detail the attacks and the related defenses.
— We discuss the open issues and challenges in existing SSE and future research directions.

The organization of this article is as follows: In Section 2, we present a general model to describe the operations of SSE and provide partial commonly used techniques. Then the search functionalities are described in detail by taxonomy in Section 3. Before discussing attacks and defenses, we introduce the formal definitions of the leakages that we study in Section 4, and then we present the attacks that exploit these leakages and the corresponding defenses. We discuss the open issues and future research directions in Section 5. Finally, we conclude the article in Section 6.

2 GENERAL FRAMEWORK AND PRELIMINARIES

2.1 Searchable Symmetric Encryption

We abstract a generic model to describe the operations of SSE over secure outsourced database systems and capture the potential leakage information. While the model ignores the concrete functionalities (e.g., multi-keyword query, range query, verification), it is general enough to help us grasp SSE.

Without loss of generality, we assume $\mathsf{DB} = \{(\mathsf{ind}_i, \mathsf{w}_i)_{i=1}^{d}\}$ is the database that the user wants to outsource, where ind_i is the identifier of the ith document and w_i is the set of keywords contained in the corresponding document. A generic model of SSE consists of three protocols $\Pi = (\textbf{Setup}, \textbf{Search}, \textbf{Update})$ between a user $\mathcal{U}$ (the data owner can also be considered as a user) and a server $\mathcal{S}$. Figure 1 shows the models of two types of systems, static SSE and dynamic SSE, where the static SSE does not include the **Update** protocol, i.e., it does not support the operation of updating data.

— $\textbf{Setup}(1^\lambda, \mathsf{DB}) \rightarrow (K, \sigma; \mathcal{I}, \mathsf{EDB})$. $\mathcal{U}$ takes a security parameter λ and a database DB as input, and $\mathcal{S}$ has no input. The protocol outputs $(K, \sigma; \mathcal{I}, \mathsf{EDB})$, where K and σ are the secret key and the client's state, stored in the $\mathcal{U}$, and $\mathcal{I}$ and EDB are the secure indexes and the encrypted database, stored in the $\mathcal{S}$.
— $\textbf{Search}(K, q, \sigma; \mathcal{I}, \mathsf{EDB}) \rightarrow \mathcal{R}$. $\mathcal{U}$ takes (K, q, σ) as input, and $\mathcal{S}$ takes $\mathcal{I}$ and EDB generated in **Setup** protocol as input. The protocol searches over secure indexes $\mathcal{I}$ and picks up the satisfied results $\mathcal{R}$ from EDB. Note that the query $q : \mathcal{X} \rightarrow \{0, 1\}$ is a predicate, so the results can be denoted by $\mathcal{R} = \{\mathsf{ind}_i | q(\mathsf{w}_i) = 1\}$.
— $\textbf{Update}(K, \sigma, \mathsf{in}, \mathsf{op}; \mathcal{I}, \mathsf{EDB}) \rightarrow (\mathcal{I}', \mathsf{EDB}')$. $\mathcal{U}$ takes $(K, \sigma, \mathsf{in}, \mathsf{op})$ as input, where in is an updated document and op (i.e., add or delete) is the corresponding operation, and $\mathcal{S}$ takes the secure indexes $\mathcal{I}$ and the encrypted database EDB as input. The output for $\mathcal{S}$ is the updated secure indexes $\mathcal{I}'$ and the updated encrypted database EDB′; $\mathcal{U}$ has no formal output.

2.2 Security Definition

Intuitively, secure searchable symmetric encryption should not reveal any information about both the queried keywords and document content. However, achieving this level of security is difficult, and most existing schemes [23, 53] reveal the search and access patterns unless oblivious RAMs [61, 99, 160] techniques are used. Therefore, a more relaxed definition of SSE security is necessary. Curtmola et al. [31] first formally defined the security of SSE that reveals nothing other than the search and access patterns. They also characterized two adversary models, *non-adaptive* and *adaptive*, with the former indicating that the generated queries do not depend on previous search results, while the latter is the opposite.

Following Curtmola et al.'s work, Kamara et al. [69] extended the SSE security definition to a dynamic setting with a set of leakage functions. In the following, we refer to Kamara et al.'s work to provide a formal definition of security for **dynamic SSE** (**DSSE**). The primary difference from static SSE is that there is no **Update** protocol.

We present the following probabilistic experiments with a set of leakage functions $\mathcal{L} = \{\mathcal{L}_{\text{Setup}}, \mathcal{L}_{\text{Search}}, \mathcal{L}_{\text{Update}}\}$, where $\mathcal{A}$ is a stateful adversary and $\mathcal{S}$ is a stateful simulator.

- $\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda)$: $\mathcal{A}$ outputs DB and receives $(\mathcal{I}, \text{EDB})$ such that $\mathbf{Setup}(1^{\lambda}, \text{DB}) \rightarrow (K, \sigma; \mathcal{I}, \text{EDB})$ from the challenger. Then the adversary adaptively generates a series of queries $(q_i, \text{in}_i, \text{op}_i)$. The challenger performs $\mathbf{Search}(K, q_i, \sigma; \mathcal{I}, \text{EDB}) \rightarrow \mathcal{R}$ if q_i is a search query or performs $\mathbf{Update}(K, \sigma, \text{in}, \text{op}; \mathcal{I}, \text{EDB}) \rightarrow (\mathcal{I}', \text{EDB}')$ if q_i is an update query. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.
- $\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}$ outputs DB. Given $\mathcal{L}_{\text{Setup}}(\text{DB})$, the simulator $\mathcal{S}$ runs $\mathbf{Setup}(\mathcal{L}_{\text{Setup}}(\text{DB})) \rightarrow (K, \sigma; \mathcal{I}, \text{EDB})$ and sends $(\mathcal{I}, \text{EDB})$ to $\mathcal{A}$. Then the adversary adaptively generates a series of queries $(q_i, \text{in}_i, \text{op}_i)$. The simulator performs $\mathbf{Search}(\mathcal{L}_{\text{Search}}(q_i))$ if q_i is a search query or performs $\mathbf{Update}(\mathcal{L}_{\text{Update}}(q_i))$ if q_i is an update query. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

We say that Π is $(\mathcal{L}_{\text{Setup}}, \mathcal{L}_{\text{Search}}, \mathcal{L}_{\text{Update}})$-secure against adaptive dynamic chosen-keyword attacks if for all **probabilistic polynomial-time** (**PPT**) adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$|\Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leqslant \text{negl}(\lambda), \tag{1}$$

where $\text{negl}(\lambda)$ is negligible in security parameter λ.

2.3 Preliminaries

In this subsection, we overview some commonly used techniques.

2.3.1 Pseudorandom Function. Let $F : \mathcal{K} \times \mathcal{Y} \rightarrow \mathcal{Z}$ be a function that mapped $\mathcal{Y}$ with $\mathcal{K}$ to $\mathcal{Z}$. If it can be computed in polynomial time and for all PPT adversaries $\mathcal{A}$ satisfy the equation

$$|\Pr[\mathcal{A}^{F(K,\cdot)}(1^{\lambda}) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^{\lambda}) = 1]| \leqslant \text{negl}(\lambda), \tag{2}$$

where $\text{negl}(\lambda)$ is negligible in security parameter λ, $K \xleftarrow{\$} \mathcal{K}$ and f is a random function from $\mathcal{Y}$ to $\mathcal{Z}$, then it is called **Pseudorandom Function** (**PRF**).

2.3.2 Constrained Pseudorandom Function. A constrained pseudorandom function [9, 73] is a pseudorandom function that works in a specific range, whose constrained key evaluates the elements out of the range is impossible. A PRF $F : \mathcal{K} \times \mathcal{Y} \rightarrow \mathcal{Z}$ uses the master key $K \xleftarrow{\$} \mathcal{K}$ to generate a constrained key K_c acting on a subset $C \subseteq \mathcal{Y}$, which allows evaluation of F for each $x \in C$. The constrained pseudorandom function consists of two algorithms as follows:

— F.Constrain (K, C) takes the master key K of F and a subset C as input, then outputs a constrained key K_c acting on the C.
— F.Eval (K_c, x) takes the constrained key K_c and $x \in C$ as input, and outputs $z \in \mathcal{Z}$. Otherwise, outputs $\perp$, if $x \notin C$, where $\perp \notin \mathcal{Z}$. This algorithm can be formulated as

$$F.\text{Eval}(K_c, x) = \begin{cases} F(K, x), & \text{if } x \in C; \\ \perp, & \text{otherwise.} \end{cases} \tag{3}$$

2.3.3 Puncturable Pseudorandom Function. The puncturable pseudorandom function [62] prevents the punctured element from being evaluated. A PRF $F : \mathcal{K} \times \mathcal{Y} \to \mathcal{Z}$ is a puncturable PRF if the secret key $K \in \mathcal{K}$ is allowed to be punctured on an element $x \in \mathcal{Y}$, and then the element x cannot be evaluated using the punctured key. The algorithms are defined as follows

— F.Punc (K, x) takes the secret key $K \in \mathcal{K}$ and an element $x \in \mathcal{Y}$ as input and outputs a punctured key K_x on x.
— F.Eval (K_x, x') takes the punctured key K_x and an element $x' \in \mathcal{Y}$ as input and outputs the evaluation $z \in \mathcal{Z}$ if $x \neq x'$. Otherwise, outputs $\perp$, where $\perp \notin \mathcal{Z}$. This algorithm can be formulated as

$$F.\text{Eval}(k_x, x') = \begin{cases} F(k, x'), & \text{if } x \neq x'; \\ \perp, & \text{otherwise.} \end{cases} \tag{4}$$

3 TAXONOMY OF SSE FUNCTIONALITIES

Search functionality is the essential ingredient of SSE, which meets the various retrieval requirements and promotes the deployment of SSE. In this section, we classify the search functionalities of SSE and introduce them, including index structures, search protocols, efficiency, and so on. Furthermore, we summarize and analyze the complexity (such as storage complexity, search complexity, computational complexity, etc.) of partial schemes, as shown in Tables 1 and 2. Security is another essential ingredient of SSE, which we will cover in detail in the next section. Figure 2 shows the timeline of partial articles on search functionalities.

3.1 Single Keyword Search

To enable the searching of encrypted data (Figure 3 shows the example of a single keyword search), the seminal article proposed by Song et al. [129] is a single keyword search scheme without an index, which achieves optimal storage but increases the burden of the search. In general, a document is divided into a sequence of words, each of which is encrypted independently by using a deterministic encryption algorithm. The encrypted word is then divided into two parts, L_i and R_i, and XOR with pseudorandom strings S_i and $F_{k_i}(S_i)$, respectively, where $k_i = f_{k'}(L_i)$. When searching for documents containing the keyword ω, the user sends the encrypted keyword ω and k_q to the server, and then the server traverses each word in documents to check whether the XOR result satisfies the form $(S_i, F_{k_q}(S_i))$. Nevertheless, such a sequential scan scheme suffers from low efficiency and leaks more information. Therefore, there are many works to optimize efficiency and improve security [104, 117, 167]. Goh et al. [53] formally defined a Semantic Security Against Adaptive Chosen Keyword Attack (IND-CKA) secure index scheme and formulated a security index model, and then they proposed an efficient IND-CKA secure scheme that allows a user to test whether a document contains a specified keyword in $O(1)$ time while revealing no information about its content. The main idea of their scheme is to store keywords of each document in a bloom filter and to use pseudorandom functions to construct the IND-CKA secure indexes. Chang et al. [23] created a string of the same size as the universal keyword set to represent a document, set the corresponding position of the string to 1 if the document contains the keyword, 0 otherwise, and

Table 1. A summary of partial SSE schemes

References	Search Type	Index Type	Index Structure	Server Storage	Trapdoor Size & Round	Search Complexity	Search Computation	Primitives	Adversarial Model	Security	Other Characteristic
Song et al. [129]	Single	Wb	–	$O(d \cdot l)$	$O(1), O(1)$	$O(d \cdot l)$	$O(d \cdot l)$	–	Non-Adaptive	CPA	Dynamic
Goh et al. [53]	Single	FI	Bloom Filter	$O(d \cdot a_m \log \frac{1}{\epsilon})$	$O(s), O(1)$	$O(d)$	$O(d \cdot s)$	–	Non-Adaptive	IND-CKA Secure Index	Dynamic
Chang et al. [23]	Single	FI	Dictionary, String	$O(m + d \cdot m)$	$O(1), O(2)$	$O(d)$	$O(d)$	–	Non-Adaptive	IND2-CKA Secure Index	Dynamic
Curtmola et al.-1 [31]	Single	II	Array, Look-up Table	$O(d \cdot M + m)$	$O(1), O(1)$	$O(r)$	$O(r)$	–	Non-Adaptive	CKA1	Static, Multi-user
Curtmola et al.-2 [31]	Single	KV	Look-up Table	$O(d \cdot M)$	$O(M), O(1)$	$O(M)$	$O(1)$	–	Adaptive	CKA2	Static
Yang et al. [167]	Single	KV	List, Table	$O(d \cdot a_m)$	$O(r), O(1)$	$O(d + \log d)$	$O(d + \log d)$	Two-Step Map	Non-Adaptive	CKA1	Static
Kamara et al. [69]	Single	II	Dictionary, Array	$O(n + m + d + v)$	$O(1), O(1)$	$O(r)$	$O(r)$	–	Adaptive	CKA2	Dynamic
Ogata et al. [117]	Single	II	List	$O(n)$	$O(1), O(1)$	$O(r)$	$O(1)$	–	Non-Adaptive	Semantically Secure	Dynamic
Asharov et al. [4]	Single	II	Dictionary, Array	$O(n \log n + m)$	$O(1), O(1)$	$O(1)$	$O(1)$	–	Adaptive	$\mathcal{L}$-Secure	Static
Cash et al. [20]	Single	II	Hash Table, List	$O(n \cdot \log n)$	$O(1), O(1)$	$O(\log n)$	$O(r)$	–	Adaptive	$\mathcal{L}$-Secure	Static
Cash et al. [18]	Single	II	Dictionary, Array	$O(n + v)$	$O(1), O(1)$	$O(r)$	$O(v)$	–	Adaptive	$\mathcal{L}$-Secure	Static
Demertzis et al. [36]	Single	II	Hash Table, Array	$O(n \cdot v + n)$	$O(1), O(1)$	$O(r)$	$O(r)$	–	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q)$-Secure	Dynamic
Golle et al. [55]	Multiple	FI	Vector	$O(d \cdot m)$	$O(\|q\|), O(1)$	$O(d)$	$O(d)$	Bilinear Map	Non-Adaptive	CKA1	Dynamic
Ballard et al. [5]	Multiple	FI	Vector	$O(d \cdot m)$	$O(\|q\|), O(1)$	$O(d)$	$O(d)$	Bilinear Map	Non-Adaptive	CKA1	Dynamic
Byun et al. [15]	Multiple	FI	Vector	$O(d \cdot m)$	$O(\|q\|), O(1)$	$O(d)$	$O(d)$	Bilinear Map	Adaptive	SS-CTA	Dynamic
Ryu et al. [127]	Multiple	FI	Vector	$O(d \cdot m)$	$O(\|q\|), O(1)$	$O(d)$	$O(d)$	Bilinear Map	Adaptive	CKA2	Dynamic
Wang et al. [153]	Multiple	FI	Polynomial	$O(d \cdot a_m)$	$O(a_m), O(1)$	$O(d)$	$O(d)$	Bilinear Map	Non-Adaptive	CKA1	Dynamic, Dynamic-Group
Wang et al. [152]	Multiple	FI	Vector	$O(d \cdot v)$	$O(v), O(1)$	$O(d)$	$O(d \cdot v)$	Accumulator	Non-Adaptive	CKA1	Dynamic, Dynamic-Group
Cash et al. [19]	Boolean	II	Hash table, Bloom Filter	$O(n + n \log \frac{1}{\epsilon})$	$O(r \cdot \|q\|), O(1)$	$O(r \cdot \|q\| \cdot s)$	$O(r \cdot \|q\| \cdot s)$	Diffie-Hellman	Adaptive	$\mathcal{L}$-Secure	Static
Moataz et al. [110]	Boolean	FI	Vector	$O(d \cdot (m + v))$	$O(1), O(1)$	$O(d)$	$O(d)$	Gram-Schmidt	Adaptive	Semantically Secure	Dynamic
Abdelraheem et al. [1]	Boolean	II	Bitmap	$O(m \cdot d + m)$	$O(\|q\|), O(1)$	$O(r \cdot \|q\|)$	$O(r \cdot \|q\|)$	–	Adaptive	$\mathcal{L}$-Secure	Static
Zuo et al. [179]	Boolean	II	Hash table, Bloom Filter	$O(n + n \log \frac{1}{\epsilon})$	$O(\|q\| + v), O(1)$	$O(v \cdot r \cdot \log \|q\|)$	$O(v \cdot r \cdot \log \|q\|)$	Diffie-Hellman, Tree	Non-Adaptive	$\mathcal{L}$-Semantically Secure	Static
Kamara et al.-1 [66]	Disjunctive	II	Dictionary, Multi-Map	$O(n + m \cdot v)$	$O(\|q\|), O(1)$	$O(\|q\|^2 \cdot M)$	$O(\|q\|^2 \cdot M)$	Set-Theoretic	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U)$-Secure	Dynamic
Kamara et al.-2 [66]	Boolean	II	Dictionary, Multi-Map	$O(n + m \cdot v)$	$O(\|q\|), O(1)$	$O(\|q\|^2 \cdot (M + v))$	$O(\|q\|^2 \cdot (M + v))$	Set-Theoretic	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U)$-Secure	Dynamic
Kamara et al.-3 [66]	Boolean	II	MF, Multi-Map	$O(n + n \log \frac{1}{\epsilon})$	$O(\|q\|), O(1)$	$O(\|q\|^2 \cdot (M + v))$	$O(\|q\|^2 \cdot (M + v))$	Set-Theoretic	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U)$-Secure	Dynamic
Patel et al.-1 [121]	Conjunctive	II	Set, Multi-Map	$O(m^2 \cdot v)$	$O(\|q\|), O(1)$	$O(v \cdot \|q\|)$	$O(v \cdot \|q\|)$	Set-Theoretic	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q)$-Secure	Static

(Countinued)

Table 1. Countinued

References	Search Type	Index Type	Index Structure	Server Storage	Trapdoor Size & Round	Search Complexity	Search Computation	Primitives	Adversarial Model	Security	Other Characteristic
Patel et al.-2 [121]	Boolean	II	Set, Multi-Map	$O(m^2 \cdot v)$	$O(\|q\|^2), O(1)$	$O(v \cdot \|q\|^2)$	$O(v \cdot \|q\|^2)$	Set-Theoretic	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q)$-Secure	Static
Li et al. [88]	Fuzzy, Single	II	Table	$O(m \cdot \tau^v)$	$O(\tau^v), O(1)$	$O(\tau^v)$	$O(1)$	Edit Distance, Wildcard	Non-Adaptive	CKA1	Static
Wang et al. [146]	Fuzzy, Single	Tb	Trie-Tree	$O(m \cdot \tau_m^v)$	$O(\tau_m^v), O(1)$	$O(\tau_m^v)$	$O(1)$	Edit Distance, Wildcard	Non-Adaptive	Semantically Secure	Static
Wang et al. [151]	Fuzzy, Single	Tb	Trie-Tree	$O(m \cdot \tau_m^v)$	$O(\tau_m^v), O(1)$	$O(\tau_m^v)$	$O(1)$	Edit Distance, Wildcard	Non-Adaptive	Semantically Secure	Static, Verifiability
Zhu et al. [178]	Fuzzy, Single	II	Matrix	$O(m \cdot \tau^v \cdot d)$	$O(\tau^v \cdot d), O(1)$	$O(\tau^v)$	$O(\tau^v)$	Edit Distance, Wildcard, RSAA	Non-Adaptive	UC Secure	Dynamic, Verifiability
Ahsan et al. [2]	Fuzzy, Single	II	List	$O(n)$	$O(1), O(2)$	$O(1)$	$O(1)$	Monogram, JS	Non-Adaptive	CKA1	Static
Kuzu et al. [79]	Fuzzy, Ranked, Single	II	Table, Bloom Filter	$O(m \cdot s \cdot d)$	$O(s), O(2)$	$O(s)$	$O(s)$	LSH, Bigram, JS	Adaptive	Semantically Secure	Static
Wang et al. [147]	Fuzzy, Ranked, Single	II	Table, Vector	$O(n + m \cdot v)$	$O(v), O(1)$	$O(m)$	$O(m)$	Edit Distance, Secure kNN, Bigram	Non-Adaptive	KPA-Secure	Static
Wang et al. [142]	Fuzzy, Multiple	FI	Bloom Filter	$O(d \cdot a_m \log \frac{1}{\epsilon})$	$O(a_m \log \frac{1}{\epsilon}), O(1)$	$O(d)$	$O(d)$	LSH, Bigram, Secure kNN	KBM	KBM-Secure	Dynamic
Fu et al. [47]	Fuzzy, Multiple, Ranked	FI	Bloom Filter	$O(d \cdot a_m \log \frac{1}{\epsilon})$	$O(a_m \log \frac{1}{\epsilon}), O(1)$	$O(d)$	$O(d)$	LSH, Stem, Uni-gram, Secure kNN	KBM	KBM-Secure	Dynamic
Chen et al. [28]	Fuzzy, Multiple	FI, II	Bloom Filter, Array	$O(n + d \cdot a_m \log \frac{1}{\epsilon})$	$O(a_m \log \frac{1}{\epsilon}), O(1)$	$O(r)$	$O(r)$	Bigram, LSH, Secure kNN	KBM	KBM-Secure	Static
Zhong et al. [175]	Fuzzy, Multiple, Ranked	Tb	Bloom Filter, Binary Tree	$O(d \cdot a_m \log \frac{1}{\epsilon})$	$O(a_m \log \frac{1}{\epsilon}), O(1)$	$O(c \cdot \log d)$	$O(c \log d)$	Uni-gram, LSH, Secure kNN	KBM	KBM-Secure	Dynamic
Tong et al. [140]	Fuzzy, Multiple	Tb	Twin Bloom Filter, Binary Tree	$O(d \cdot v)$	$O(\|q\|), O(1)$	$O(\|q\| \cdot c \cdot \log d)$	$O(\|q\| \cdot c \cdot \log d)$	Uni-gram, LSH, MSA, MHT, Stem	Adaptive	CKA2	Static, Verifiability
Nayak et al. [114]	Fuzzy, Multiple, Ranked	FI	Quotient Filter	$O(d \cdot v)$	$O(\|q\| \cdot \tau), O(1)$	$O(d \cdot \|q\| \cdot \tau)$	$O(1)$	Trigram, Wildcard	KPA	KPA-Secure	Dynamic
Liu et al. [97]	Fuzzy, Multiple, Disjunctive	FI	Vector	$O(d \cdot a \cdot v)$	$O(\|q\| \cdot v), O(1)$	$O(d)$	$O(d)$	Secure kNN	KBM	KBM-Secure	Static
Liu et al. [98]	Fuzzy, Boolean	Tb	Binary Tree, Vector	$O(d)$	$O(\|q\| \cdot v), O(1)$	$O(c \log d)$	$O(c \log d)$	Secure kNN	KBM	KBM-Secure	Dynamic

Table 2. A summary of partial SSE schemes

References	Search Type	Index Type	Index Structure	Server Storage	Trapdoor Size & Round	Search Complexity	Search Computation	Primitives	Adversarial Model	Security	Other Characteristic
Wang et al. [145]	Single, Ranked	II	List	$O(d \cdot a_m)$	$O(1)$, $O(1)$	$O(1)$	$O(c)$	OPM, TF-IDF	Adaptive	CKA2	Dynamic, Verifiability
Cao et al. [16]	Multiple, Ranked	FI	Vector	$O(d \cdot (m+v))$	$O(m+v)$, $O(1)$	$O(d)$	$O(d)$	Secure kNN, TF-IDF	KBM	KBM-Secure	Dynamic
Li et al. [87]	Boolean, Ranked	FI	Vector	$O(d \cdot m)$	$O(m)$, $O(1)$	$O(d)$	$O(d)$	Secure kNN, TF-IDF	KBM	KBM-Secure	Dynamic
Yu et al. [169]	Multiple, Ranked	FI	Vector	$O(d \cdot m)$	$O(m)$, $O(1)$	$O(d)$	$O(d+c)$	TF-IDF, Homomorphic Encryption	KBM	KBM-Secure	Dynamic
Li et al. [86]	Multiple, Ranked	II	Vector	$O(n \cdot m)$	$O(m)$, $O(1)$	$O(r)$	$O(r)$	TF-IDF, Secure kNN, CP-ABE	KBM	KBM-Secure	Dynamic
Jiang et al. [65]	Multiple, Ranked	II	Loop-up Table, Array	$O(2^{\lceil \log m \rceil} \cdot M + n)$	$O(r \cdot \lvert q \rvert)$, $O(1)$	$O(r \cdot \lvert q \rvert)$	$O(r \cdot \lvert q \rvert)$	TF-IDF, DH, MAC, Paillier	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q)$-Secure	Static, Verifiability
Fu et al. [46]	Multiple, Ranked, Semantic	Tb	Binary Tree, Vector, Hash Table	$O(d \cdot (m+v))$	$O(m+v+\lvert q \rvert)$, $O(1)$	$O(c \cdot \log d)$	$O(\lvert q \rvert \cdot c \cdot \log d)$	TF-IDF, Secure kNN	KBM	KBM-Secure	Static
Chen et al. [25]	Multiple, Ranked	HC	Vector	$O(d \cdot (m+v_1))$	$O(m+v_1)$, $O(1)$	$O(v_2+v_3)$	$O(v_2+v_3)$	Coordinate Matching, Dymanic K-means	KCM	KCM-Secure	Dynamic, Verifiability
Miao et al. [106]	Multiple, Ranked	Tb	Binary Tree, Vector	$O(d \cdot (m+v_1))$	$O(m+v_1)$, $O(1)$	$O(v_2 + c \log v_3)$	$O(v_2 + c \log v_3)$	TF-IDF, K-means, Secure kNN	KBM	KBM-Secure, Forward Security	Dynamic
Faber et al. [40]	Range	II	Hash Table, Bloom Filter	$O(n + n \log \frac{1}{\epsilon})$	$O(\lceil \log d \rceil)$, $O(1)$	$O(\lceil \log d \rceil)$	$O(c)$	Diffie-Hellman	Adaptive	$\mathcal{L}$-Secure	Static
Li et al. [89]	Range	Tb	Bloom Filter, Binary Tree	$O(d \cdot v \log \frac{1}{\epsilon})$	$O(\lvert q \rvert \cdot s)$, $O(1)$	$O(c \cdot \log d)$	$O(\lvert q \rvert \cdot c \cdot \log d)$	Prefix Encoding	Non-Adaptive	IND-CKA Secure Index	Static
Demertzis et al. [35]-Quadratic	Range	II	List	$O(n)$	$O(1)$, $O(1)$	$O(r)$	$O(1)$	–	Adaptive	$\mathcal{L}$-Secure, Forward Security	Dynamic
Demertzis et al. [35]-Constant	Range	II	List, Binary Tree	$O(d)$	$O(\log \lvert q \rvert)$, $O(1)$	$O(c)$	$O(\lvert q \rvert + c)$	Delegatable PRF, Range Covering	Adaptive	$\mathcal{L}$-Secure, Forward Security	Dynamic
Demertzis et al. [35]-Logarithmic	Range	II	List, Binary Tree	$O(d \cdot \log m)$	$O(\log \lvert q \rvert)$, $O(1)$	$O(c)$	$O(1)$	Range Covering	Adaptive	$\mathcal{L}$-Secure, Forward Security	Dynamic
Fu et al. [48]	Semantic, Multiple, Ranked	FI	Vector	$O(d \cdot (m+v))$	$O(m+v)$, $O(1)$	$O(d)$	$O(d)$	Stanford Parser, TF-IDF, Secure kNN	KBM	KBM-Secure	Dynamic
Fu et al. [44]	Semantic, Multiple, Ranked	FI	Vector	$O(d \cdot v_1)$	$O(v_1)$, $O(1)$	$O(d+v_2)$	$O(d+v_2)$	Conceptual Graph, Text Summarization, Tregex, TF-IDF, Secure kNN, OPSE	KBM	KBM-Secure	Dynamic
Fu et al. [43]	Semantic, Multiple, Ranked	FI	Vector	$O(d \cdot (m+v))$	$O(m+v)$, $O(1)$	$O(d)$	$O(d)$	Conceptual Graph, Text Summarization, Tregex, TF-IDF, Secure kNN	KBM	KBM-Secure	Dynamic
Fu et al. [49]	Semantic, Multiple, Ranked	Tb	Vector, k-d Tree	$O(d \cdot (m+v))$	$O(m+v)$, $O(1)$	$O(c \cdot \log d)$	$O(c \cdot \log d)$	Concept Hierarchy, Secure kNN	KBM	KBM-Secure	Static

(Countinued)

Table 2. Countinued

References	Search Type	Index Type	Index Structure	Server Storage	Trapdoor Size & Round	Search Complexity	Search Computation	Primitives	Adversarial Model	Security	Other Characteristic
Chase et al. [24]	Substring	Tb	Suffix Tree, Dictionary, Array	$O(l)$	$O(\|q\|)$, $O(3)$	$O(\|q\|+p)$	$O(\|q\|+p)$	–	Adaptive	$(\mathcal{L}_S, \mathcal{L}_Q)$-CQA2 Secure	Static
Hahn et al. [60]	Substring	KV	Map, List	$O(l+v)$	$O(\|q\|)$, $O(2)$	$O(r \cdot \|q\|)$	$O(r \cdot \|q\|)$	FHOPE, k-Gram	Non-Adaptive	IND-CPA-IOQ	Static
Faber et al. [40]	Substring	II	Hash Table, Bloom Filter	$O(n + n\log\frac{1}{\epsilon})$	$O(r \cdot \|q\|)$, $O(1)$	$O(r \cdot \|q\|)$	$O(r \cdot \|q\|)$	Diffie-Hellman, k-Gram	Adaptive	$\mathcal{L}$-Secure	Static

Notes. FI: Forward Index; II: Inverted Index; Wb: Word-based; Tb: Tree-based; KV: (Key, Value); HC: Hierarchical Clustering. d: Number of documents; m: Number of keywords; n: Number of keyword-document pairs; r: Number of documents matching a keyword; M: Maximum number of documents matching a keyword; l: Length of a document; τ: Length of a keyword; τ_m: Maximum length of a keyword; s: Number of hash functions; a: Number of keywords a document contains; a_m: Maximum number of keywords a document contains; c: Number of results, i.e., $c = \mathsf{DB}(q)$; v: Variable (subscripts are used to distinguish between different values); ϵ: False positive rate; p: Number of occurrences of q as a substring of s. In substring search, the meaning of some symbols is slightly changed depending on the scheme.

Primitives. The data structures or cryptographic tools used in the scheme, common cryptographic primitives (such as PRF, PRP, hash function, encryption, etc.) and data structures (such as array, list, dictionary, hash table, etc.) will not be listed. MF: Matryoshka Filter; RSAA: RSA Accumulator; JS: Jaccard Similarity; MSA: Multiset Accumulator; MHT: Merkle Hash Tree; OPM: Order Preserving Mapping; CP-ABE: Ciphertext Policy Attribute-Based Encryption; TDAG: Tree-like Directed Acyclic Graph; FHOPE: Frequency-Hiding Order-Preserving Encryption.

Security. CPA: chosen plaintext attack; IND-CKA: semantic security against adaptive chosen keyword attack; IND2-CKA: A more robust security model than IND-CKA, where the document size and the number of keywords are possibly unequal; CKA1: non-adaptive security against chosen-keyword attack; CKA2: adaptive security against chosen-keyword attack; SS-CTA: semantic security against chosen trapdoor attack; UC: Universally Composable; KPA: know plaintext attack; KPA: Know Plaintext Attack; KBM: Know Background Model; KCM: Know Ciphertext Model; CQA2: adaptive security against chosen query attack; IOQ: Identically Ordered Queries.

Leakage profiles. $\mathcal{L}_S$, $\mathcal{L}_Q$, $\mathcal{L}_U$, $\mathcal{L}$. The scheme guarantees that the setup algorithm does not reveal information about the underlying structure, except for the setup leakage $\mathcal{L}_S$, the query algorithm does not reveal information about the structure and the queries, except for the query leakage $\mathcal{L}_Q$, and the update algorithm does not reveal information about the structure and the updates, except for the update leakage $\mathcal{L}_U$. The definitions of different schemes for $\mathcal{L}_S$, $\mathcal{L}_Q$ and $\mathcal{L}_U$ depend on the adopted structures, search protocols and update protocols, respectively. The leakage profile $\mathcal{L}$ is defined by the scheme to represent all leakages that are allowed to be revealed.

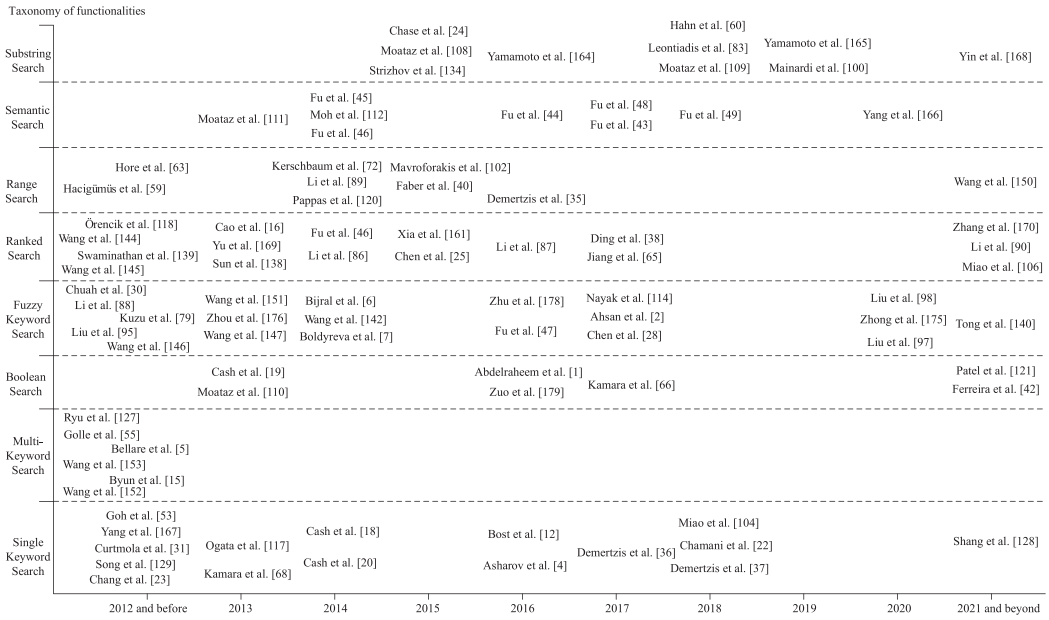


Fig. 2. Timeline of SSE Search Function

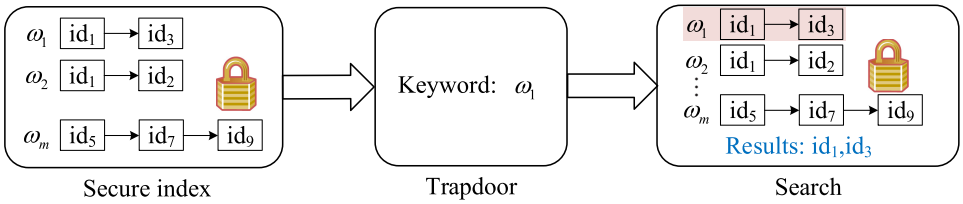


Fig. 3. Example of Single Keyword Search

then masked the string with pseudorandom bits. Curtmola et al. [31] further improved the security definition and defined two adversarial models, *non-adaptive* and *adaptive*, respectively. Then they proposed an efficient construction against adaptive adversaries. This construction uses inverted lists as index structures, and then randomly stores the list nodes into an array. Meanwhile, a look-up table needs to be maintained to store the first element of each keyword list.

In recent years, very large datasets have attracted more attention [18, 20, 36, 37], and the previous in-memory schemes are not fit in such scenarios. Storing extensive indexes in memory may frequently cause random access, which is an impractical and expensive solution. Therefore, optimizing the locality of the schemes has become a new direction of research, in which the goal is to reduce the number of non-continuous reads as much as possible. Asharov et al. [4] investigated and proposed a scheme with optimal locality, which creates $l = \log n + 1$ arrays of size n for storing all keyword lists, where n denotes the total number of keyword-document pairs. Each array A_i is divided into $n/2^i$ chunks and stores all keyword lists of size 2^i (assuming that both n and keyword list sizes are powers of two). However, this scheme requires $O(n \log n)$ storage space, so Demertzis et al. [36] improved it while maintaining optimal locality. They choose s arrays out of l arrays to store all keyword lists, each separated by a distance $p = \lceil l/s \rceil$ (i.e., $\{ A_l, A_{l-p}, \ldots, A_{l-(s-1)\cdot p} \}$), where s is a small constant. Moreover, the size of the chunk in each

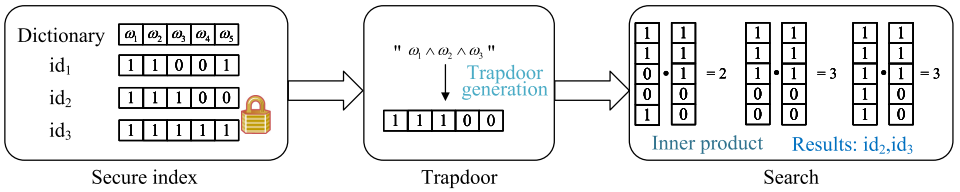


Fig. 4. Example of Multiple Keyword Search

array is doubled and increased by one more chunk, namely, the size of each array A_i is expanded to $2n+2^{i+1}$. For each keyword list $\mathrm{DB}(\omega)$, $2^j < |\mathrm{DB}(\omega)| \leq 2^i$, where $i, j \in \{l, l-p, \ldots, l-(s-1)\cdot p\}$ and $j < i$, it will be stored in array A_i. Thus, the storage space is reduced to $O(s \cdot n)$. In addition, Demertzis et al. also proposed an extended scheme with tunable locality, which increases the locality slightly to gain read efficiency. Furthermore, Demertzis et al. [37] compressed the plaintext indexes before generating the encrypted indexes to further improve search efficiency.

As an underlying search function, single keyword search has been widely studied not only in search efficiency and storage space but also in security [12, 22, 128]. More details about security will be introduced in Section 4.

3.2 Multi-Keyword Search

Multi-keyword search (Figure 4) allows users to retrieve interested documents containing several keywords, which is a practical and desirable search function to return more exact results and save communication costs. A naive approach to the problem of multi-keyword search is based on a single keyword search. Given several interested keywords, the server performs a single keyword search for each keyword and then returns the intersection of these document sets. Obviously, such an approach is inefficient and allows the server to know extra information other than the results matching queries. Golle et al. [55] first considered the multi-keyword search over encrypted data and proposed schemes for performing such searches securely. They used vectors to represent documents and assumed each document had m fields. Each vector consists of three parts, namely, m modular exponentiations that are computed from keywords and random values R_i, m modular exponentiations that are computed from random values R_i and an intermediate value. When performing a search, the user sends the product of all interested keywords along with their corresponding field positions to the server. Then the server traverses all vectors to find matching results relying on the bilinear pairings according to the received field positions. Ballard et al. [5] proposed two schemes for multi-keyword search using Shamir Secret Sharing and bilinear pairings, respectively. Byun et al. [15] and Ryu et al. [127] focused on improving the security of the schemes under different assumptions.

However, all of the above schemes require the users to send the field positions of the interested keywords in the indexes, which is impractical for many scenarios. Out of this consideration, Wang et al. [153] proposed a keyword field-free multi-keyword search scheme by constructing an l-degree polynomial as the index of a document, where l is the number of keywords contained in the document and all keywords are the roots of the equation. It calculates the polynomial value of all interested keywords to determine whether a document satisfies the query, thus getting rid of the dependence on the keyword position. Another approach proposed by Wang et al. [152] is constructing a bit string as the index of a document, which is computed from the bitwise product of all keywords contained in the document, and the trapdoor is generated in the same way. For searching matching documents, the server checks whether the positions of 0 in the trapdoor are 0 in the index.

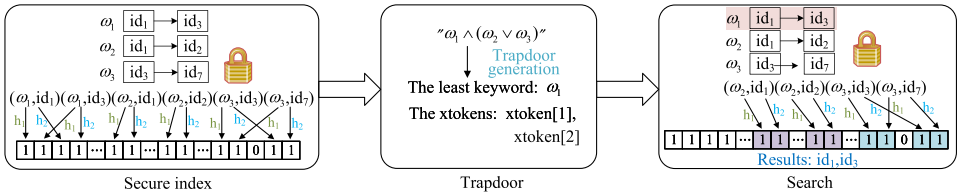


Fig. 5. Example of Boolean search.

Therearemanyschemes[19,30,46]thatsupportmulti-keywordsearchwithlesscomputa-tionalandstorageoverhea

3.3 Boolean Search

Boolean search (Figure 5) is an extension of multi-keyword search, which supports regular Boolean operations s
contains. Trapdoors are generated in a similar way as document labels, but each vector is multiplied by a random
XSetisasetfortestingmembership.Eachinvertedliststoresalldocumentsthatcontainthekeywordωinencryptedform

Fromthenon-Booleansearchhasbeenextensivelystudied[42,120]. Abdelraheemetal.[1]pointedoutthat[19]suffe
"*Male*$\wedge$(*Australian* $\vee$ *Chinese*)" sincemanydocumentscontain "*Male*",thustheyproposedanal- $\Delta_1 \vee \cdots \vee \Delta_n$,whichrequireslin- ternativeofsendingmu
ciency.Kamaraetal.[66]consideredtheworst-casequeriessuchas earsearchcomplex

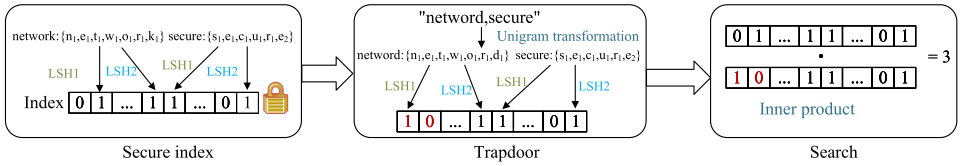


Fig. 6. Example of fuzzy search.

support arbitrary Boolean queries with optimal communication and worst-case sub-linear search complexity, and used matryoshka filters to reduce storage overhead. However, these constructions [66] reveal the result size of each keyword. Patel et al. [121] noticed the information leakage and proposed constructions with significantly less leakage than previous works while maintaining efficient search time and optimal communication overheads. The core idea behind their constructions is the filtering algorithm, which performs set intersections with less leakage. Specifically, they designed a construction ConjFilter to support conjunctive queries based on the filtering algorithm, then used it as a building block to propose a construction CNFFilter to support arbitrary Boolean queries.

3.4 Fuzzy Keyword Search

Fuzzy keyword search (Figure 6) enhances system usability by tolerating keywords with minor typos sent from users, which is able to handle typos and return exact or the closest matching results. A naive way to deal with the challenge is to add a validation mechanism that determines the validity of the input before searching. However, it requires an extra interaction to verify the validity of the input, and it may fail to catch typos, such as typing "cat" into "hat".

Li et al. [88] first considered the fuzzy keyword search over encrypted data and then proposed a straightforward construction based on edit distance. In their construction, each keyword is encoded to a wildcard-based fuzzy keyword set with edit distance d, and then the fuzzy keyword set indexes all documents that contain the exact keyword. To search a keyword with pre-specified edit distance k, where $k \leq d$, the user generates trapdoors of the fuzzy keyword set with edit distance k, then the server returns all possible documents that are indexed by the fuzzy keywords. Wang et al. [146, 151, 178] also used a wildcard-based fuzzy keyword set to achieve a fuzzy keyword search. Expanding keywords to cover as many typos of keywords as possible is a common idea of implementing fuzzy keyword search, which enables the system to handle keywords with typos. Liu et al. [95] and Bijral et al. [6] presented a dictionary-based fuzzy set construction whose expanded fuzzy set is composed of the keywords in the dictionary. Zhou et al. [176] presented a k-gram based fuzzy set construction, which composes a fuzzy set of keywords containing the same k-gram. Boldyreva et al. [7] improved the security and efficiency of construction [88] by exploiting efficient searchable encryption [3], a closeness-preserving tagging function, and a batch-encoding family. Besides, Ahsan et al. [2] implemented a fuzzy keyword search by transforming the keyword into a monogram set, which is based on the assumption that a word will not have multiple errors.

Unfortunately, these constructions are not scalable as the storage increases with the increase of typos. To deal with an arbitrary number of typographical errors in words without increasing the storage complexity, Kuzu et al. [79] proposed an efficient scheme for fuzzy keyword search by utilizing locality sensitive hashing and bloom filters. In their scheme, each keyword is represented as k-grams and encoded into a bloom filter, and then the bloom filter is hashed to multiple buckets by a locality sensitive function family. When a search for related documents, the same way is performed on the queried keyword and the resulting buckets are returned. Wang et al. [147]

implemented fuzzy keyword search by transforming keywords into fingerprint vectors, which extracted the features of words to evaluate the similarity between different words.

To complement the limitation of single keyword fuzzy search, Chuah et al. [30] first considered multi-keyword fuzzy search based on the bedtree and the gram counting order [173], but it could only handle pre-defined phrases, such as "information retrieval" as a single keyword. Subsequently, Wang et al. [142] studied and proposed a general construction that supports multi-keyword fuzzy search. It is based on the techniques of locality sensitive hashing and bloom filters as [79], but there are differences in both the index construction and the search process. The construction represents a document as a bloom filter and maps all keywords contained in the document into the bloom filter using a locality sensitive function family, where each keyword is encoded as a bigram vector of size 26^2-bit. The search process can then be done by an inner product between the document vector and the query vector as the relevance score of the document and query, where the query vector is generated as the document vector, mapping all searched keywords into a bloom filter. Motivated by Wang et al.'s scheme, Fu et al. [47] presented a more accurate and efficient multi-keyword fuzzy search scheme. Instead of using the bigram vector to represent the keyword, this scheme uses the uni-gram. It is a major change compared with [142], enabling the scheme to tolerate more typos and return more accurate results.

However, the search complexities for these schemes are linear with the size of the database. In order to improve the search efficiency, Chen et al. [28] designed a novel two-stage index structure to speed up the search efficiency, which consists of inverted indexes and forward indexes. Then the server first uses the inverted indexes to find all documents containing a certain keyword, and then determines whether these documents satisfy the query based on the forward indexes. Zhong et al. [175] introduced a balanced binary tree as an index structure, in which each leaf node corresponds to a bloom filter of a document, and the parent node is generated based on the child nodes. Tong et al. [140] also designed a balanced binary tree using the graph-based keyword partition algorithm, but each document is represented by a twin bloom filter. These schemes implement the feature of multi-keyword fuzzy search are mainly based on the k-gram vector representation and the locality sensitive hashing. Beyond these, there are many works [97, 98, 114] based on different techniques to achieve multi-keyword fuzzy search for accuracy, security, dynamic, and so on. Nayak et al. [114] used the quotient filter, the trigram set, and the modified quotient function to construct a dynamic fuzzy multi-keyword search scheme. Liu et al. [97] encoded each keyword as a vector, where each element of the vector is filled with a prime or the reciprocal of a prime. Thus each document (or query) is represented by a matrix consisting of multiple vectors corresponding to the keywords it contains and the search operation is based on the inner product between the indexes and the trapdoor. Further, Liu et al. [98] improved the search efficiency by constructing a balanced binary tree and enhanced the security by adding random noises to the query matrix.

3.5 Ranked Search

Although Boolean search enriches the query expressiveness and improves the accuracy of results, it fails to return the most relevant documents and incurs unnecessary bandwidth as it returns undifferentiated documents that satisfy the query. Ranked keyword search (Figure 7) may be a desirable approach for the problem by returning matching documents in ranked order with regard to relevance criteria (e.g., frequency or weight of keyword).

Assigning relevance scores to documents based on the frequency of keyword occurrences is a common method used to implement ranking [139, 144, 145]. The first attempt [139] to secure ranked keyword search over encrypted data was implemented by integrating scores and cryptographic primitives. Following that, Wang et al. [144, 145] formulated the secure ranked keyword search over encrypted data, and then proposed constructions that assisted in the relevance scores.

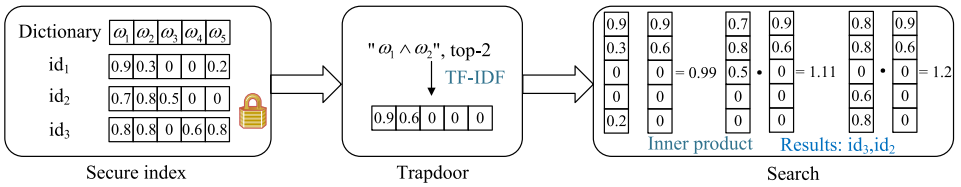


Fig. 7. Example of ranked search.

They adopted the existing SSE scheme [31] as the framework and attached a relevance score to each entry for ranking, where the score is calculated according to the TF-IDF[1] rule. OPSE[2] is then used to encrypt the scores so that the server can return the results without knowing the scores.

For a single keyword ranked search, the relevance score reflects the relevance between a keyword and a document so that all documents containing a certain keyword can be easily ranked by attached scores. However, multi-keyword ranking needs to consider the impact of multiple keywords on documents comprehensively, so that assigning a relevance score to each document is impractical. Cao et al. [16] quantified the relevance of a document to a query by counting the number of searched keywords contained in the document for supporting the multi-keyword ranked search. Further, they calculated the relevance scores between queries and documents by replacing the binary value 1 on the document vector and query vector with the TF and IDF, respectively, and using the secure inner product. Örencik et al. [118] adopted the scheme [152] as the base for multi-keyword search, then added additional structure to rank the documents. Specifically, they introduced relevance levels based on the keyword frequency. Each level stores an index for the document's frequent keywords in a descending cumulative fashion for each document. Therefore, the higher the level of the matching document is located, the more relevant the document is. In addition to relying on relevance scores, Li et al. [87] introduced the preference factors of keywords (i.e., weights of keywords) to both improve the search accuracy and support Boolean search. Yu et al. [169] proposed a top-k retrieval construction requiring two roundtrips and homomorphic encryption for a security guarantee.

With regard to efficiency, Li et al. [86] used inverted indexes to improve search efficiency. They represented documents and queries as vectors, in which each dimension is set to TF or IDF value. For performing a search, the user chooses the least frequent keyword in the query, then the server finds the corresponding inverted list and ranks the documents in the list using the secure inner product technique. Jiang et al. [65] modified the scheme [19] by attaching a relevance score between a keyword and a document to each entry, and then summed the scores of each matching document for ranking. Fu et al. [46] constructed a balanced binary tree whose leaf nodes are index vectors of documents as the index structure and used a secure inner product technique to evaluate the relevance scores for ranking. Furthermore, Sun et al. [138] utilized a multi-dimensional algorithm to find the best top-k matching results with multidimensional b-tree-based indexes. Chen et al. [25] proposed a hierarchical clustering method to cluster the documents based on the relevance score. Therefore, the server only needs to look for the top-k documents in a few categories instead of traversing the entire database. Miao et al. [106] proposed another clustering

[1]Term Frequency - Inverse Document Frequency (TF-IDF), which is used to quantify words in a set of document. TF denotes the frequency of a word appearing in a document, and IDF describes the importance of a word in the whole document collection, which is obtained by dividing the total number of documents by the number of documents containing the word.
[2]Order Preserving Symmetric Encryption (OPSE) is an encryption algorithm that allows comparison of actual encrypted form without decryption.

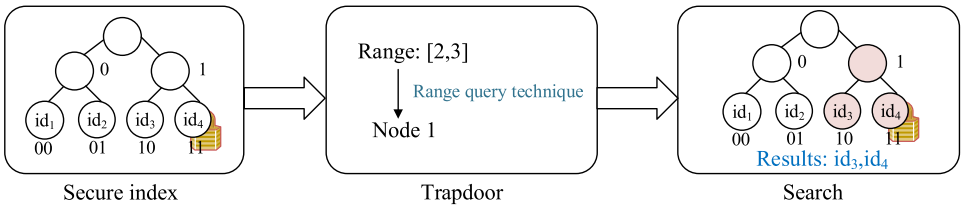


Fig. 8. Example of range search.

method using a k-means algorithm, and then constructed indexes on a balanced binary tree for each cluster, which achieves an efficient and accurate search.

Ranked search has attracted concerns extensively and is studied in many aspects for more excellent performance [38, 161]. In addition to ranking returned results on an exact search, there are also some works [90, 170] that focus on fuzzy searches. The main idea is to replace the values in the bloom filter with the weights (or frequency of keywords) of the corresponding keywords. Inspired by this, other retrieval schemes can also attach relevance scores to documents to achieve ranked search by modifying indexes and search algorithms.

3.6 Range Search

Range search (Figure 8) allows users to send an interval $[a, b]$ as a query and returns all documents within that interval, where a and b are the lower and upper bounds, respectively. Typically, the range search is appropriate for numeric datasets or documents with numeric attributes.[3]

Reducing range search to existing retrieval protocol is a common way to realize range search while inheriting extra properties. Faber et al. [40] extended the scheme of [19] to support range search while preserving the security guarantee and scalability. They constructed a full binary tree with l+1 levels and mapped attribute values of documents into leaves, where leaves are represented as binary strings between 0 and $2^l - 1$ in ascending order. Each query range $[a, b]$ is transformed into a set of nodes that cover the required range and then reduced to a disjunctive search on nodes. The search process follows the OXT protocol. Pappas et al. [120] constructed a bloom filter-based balanced b-ary tree as an index structure supporting the Boolean search. And the range search is reduced to a disjunctive search building on the technique of [126]. Demertzis et al. [35] proposed several schemes to support range search by reducing it to keyword-based search. The basic scheme enumerates all possible query ranges in the domain and assigns a unique keyword to each sub-range. Then each document is associated with a set of keywords whose corresponding sub-ranges cover the attribute value of the document. Therefore, the resulting document-keyword dataset can be used to implement a range search based on existing SSE schemes. Other enhanced schemes use range covering techniques and tree-based structures to reduce range search to keyword-based search, especially for building a binary tree over the domain and assigning each node a unique keyword. Each document is then associated with a set of keywords that correspond to all nodes on the path from the root to the leaf. The query range is represented as a set of nodes that cover the range using range covering techniques. In addition, a line of schemes is derived based on the scheme that aims at optimizing security, storage, efficiency, and false positive. Li et al. [89] used the prefix encoding scheme [94] to convert values into prefix families and constructed a bloom filter-based complete binary tree as the index structure. The query range $[a, b]$ is then converted to a minimum set of prefixes. Therefore, the range search is reduced to testing whether two sets have elements in common.

[3]Without loss of generality, an arbitrary domain can be converted to a numeric domain.

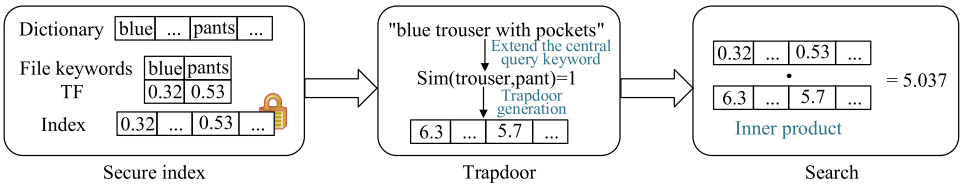


Fig. 9. Example of semantic search.

Utilizing cryptographic primitives [72, 102] is another powerful approach to implementing range search. **Order Preserving Encryption** (**OPE**) has a property that allows the comparison of actual order in ciphertext without decryption. As a result, existing efficient schemes can directly utilize OPE to encrypt data and perform range search as in plaintext. Another line of work is based on bucketization techniques [59, 63], which map documents with similar attribute values into the same bucket. In the bucketing scheme, the data owner divides the entire data domain into several buckets of different sizes and maps documents into corresponding buckets. The query range is represented as a bucket (or several buckets) that covers the range of the query. For example, the data owner divides $[0, 100]$ into $[0, 10]$, $[11, 43]$, $[44, 76]$ and $[77, 100]$, and then maps documents into the appropriate buckets according to the attribute values. If the query range is $[40, 50]$, then the server returns documents in buckets $[11, 43]$ and $[44, 76]$.

3.7 Semantic Search

Keyword-based search has been extensively studied by the SSE community and has been equipped with many desirable features as aforementioned (e.g., multi-keyword search, fuzzy search, Boolean search), but they cannot perfectly satisfy users' search intentions since the semantic information between documents and queries is ignored. In other words, the keyword-based search can only return documents that actually contain the searched keywords but not documents with similar meanings. Consequently, to further improve the search accuracy and experience, the ability to go beyond searching exactly on keywords should be taken into account. Semantic search (Figure 9) is one such design that compensates for the bottleneck by supporting content-aware search and returns documents with similar meanings to the queries.

Synonym-based expansion [46, 48, 112] is an effective approach to support semantic search, which expands synonyms for each keyword to capture the similar semantic information of distinct keywords. [46, 112] used various synonym expansion tools (New American Roget's College Thesaurus, Wikipedia Similarity Matching, and so on) to construct the synonym set as the initial keyword set and build indexes as most existing schemes, but the synonym set incurs heavy storage and complex indexes. Instead of expanding all keywords, [48] expands the central keyword of a query to balance the semantic search and efficiency. Besides, [45, 111] exploited the stemming algorithms to extract the same stem (the stem may not make sense) from different keyword forms to achieve semantic search.

In addition to the synonyms of keywords, the relationships between keywords are also important semantic information. Fu et al. [43, 44] took the relationships between keywords extracted from the documents into account and proposed schemes to support content-aware search. These schemes are based on the conceptual graph, which is a knowledge representation model. They first used Text Summarization and Tregex techniques to extract sentences that represented documents. Then they constructed conceptual graphs from these sentences and converted them into vectors. With these vectors, the implementation of the constructions is similar to the scheme [16]. Yang et al. [166] transformed semantic matching between queries and documents into random linear programming problems, and then used existing optimizers to solve the problems. In their

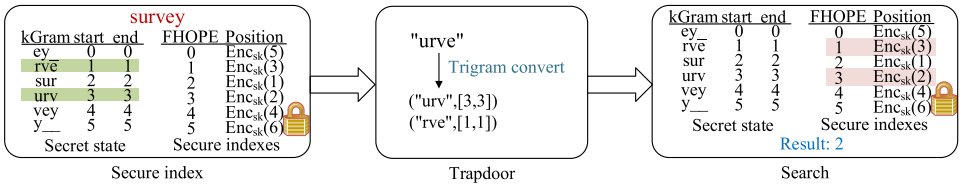


Fig. 10. Example of substring search.

scheme, each pair of query and document is transformed into a word transportation problem, which is then transformed into a random linear programming problem. The solution of the random linear programming problems serves as an evaluation of the similarity between queries and documents. Fu et al. [49] extended the concept hierarchy to design a novel semantic search scheme. They introduced attributes for each concept in the concept hierarchy, which can be assigned values. Then they built two index vectors for each document based on the extended concept hierarchy, one for matching concepts and the other for matching attributes. The index vectors of the query are also constructed in this way.

Most existing schemes for implementing semantic search can be classified into two categories, one based on synonym expansion techniques and the other on relationships between keywords. Exploring effective representation models for semantic information of documents and queries is the key to implementing semantic search, and combining it with ranked search can also further improve the accuracy of retrieval.

3.8 Substring Search

In addition to returning records that contain the substring q, the substring search (Figure 10) returns all positions where q occurs as a substring of s. It is rarely researched but does serve as an essential search function. For example, a medical researcher wants to find out the positions of all occurrences of a given gene fragment q from a gene sequence s while preserving the privacy of s and q.

A straightforward approach is that it considers each substring as a separate keyword and then performs keyword-based searches, but it incurs high storage costs due to the exponential inflation of the indexes. To achieve comparable retrieval and storage overheads as in the plaintext setting, various data structures have been adopted to support substring search. Chase et al. [24] proposed a substring search scheme based on a suffix tree with an interactive protocol, a data structure widely used to solve string matching and queries. Moataz et al. [108] proposed two substring search schemes based on suffix arrays and suffix trees, respectively, and combined them with the hierarchy ORAM tree structure for oblivious search. Strizhov et al. [134, 168] used a position heap tree [39] to support the substring search. Yamamoto et al. [164] constructed a substring search scheme on a **directed acyclic word graph** (**DAWG**) and a **hierarchical bloom filter** (**HBF**).

Although auxiliary data structures facilitate substring searches, efficiency is not comprehensively considered. Leontiadis et al. [83] revisited current substring search schemes and proposed a storage-efficient substring search scheme. They pointed out that the scheme [24] that adding dummy nodes to protect information incurs a surge in storage overhead is unacceptable. To decrease the size of the index, they used suffix arrays instead of suffix trees, which have a constant size compared to suffix trees. And proposed a dummy node policy with bucketization techniques to secure the index. Yamamoto et al. [165] also proposed a storage-efficient substring search scheme by designing an **augmented DAWG** (**ADAWG**), whose all transitions have distinct strings (meta-symbol). With the properties of ADAWG, the scheme does not require dummy data to hide the frequency of letters. Considering the communication cost between the user side and the server side,

Mainardi et al. [100] proposed a substring search scheme with sub-linear search time by combining the **Burrows Wheeler Transform** (**BWT**) [14] technique and **private information retrieval** (**PIR**) [93] protocol. Moreover, they protected the search pattern and access pattern privacy.

Another way to implement substring search is to integrate it into existing systems. Faber et al. [40] reduced substring searches to conjunctive searches based on the extended OXT protocol [19]. They represented a substring query as a series of k-grams with relative distances and divided documents into a set of k-grams. Then each substring query is executed as a conjunctive search of such k-grams. Hahn et al. [60] transformed substring searches into range searches for efficiency and security guarantee without significant modifications. They adopted the **Frequency-Hiding Order-Preserving Encryption** (**FHOPE**) [71] as a building block and divided a string into a set of overlapping k-grams. Each k-gram is associated with a list that contains all positions where the k-gram occurs. All positions of each k-gram are then iterated to create a consistent ciphertext range for k-grams. As a consequence, range searches can be applied to implement substring searches while tokenizing a query into a form compatible with the index. Moataz et al. [109] represented each letter as an orthogonal vector, such that each keyword or substring is defined as a matrix, where each column associates with a letter of the keyword. Then the search phase is performed on an inner product between a substring and keywords. In the case that a keyword contains a query substring, the diagonal elements of the resulting matrix are all different from zero.

3.9 Other Characteristics

In addition to the search above functionalities, a few notable features cannot be ignored, such as updating encrypted data, verifying results, and so on. These features facilitate the deployment and practical application of SSE schemes. And we will cover them in the following.

Dynamic SSE (**DSSE**) allows users to update (add or delete documents) remote encrypted data while preserving privacy, which enhances system usability and data freshness. As a system property, dynamism is usually related to the underlying index structure. In other words, implementing a dynamic SSE scheme depends on the underlying index structure, and different index structures adopt different update methods. Generally, schemes without indexes [3, 129] to add/delete a document are straightforward. The corresponding encrypted document can be added to or deleted from the outsourced data directly since the retrieval is performed on documents. Schemes with forward indexes [23, 47, 53] are updated similarly to schemes without indexes, which add document-keyword pairs to indexes or delete document-keyword pairs from indexes. For schemes with inverted indexes [69, 84, 144], where the indexes consist of keyword-document pairs. Unlike the previous two approaches, adding or deleting a document requires finding all entries in the inverted lists that contain that document. Kamara et al. [69] created two extra data structures, deletion array and deletion table, respectively, to assist with updates. The deletion array stores all addresses of each document in the inverted lists in the form of a linked list, and the deletion table stores the address of the head node of each linked list. Tree-based schemes [35, 68, 106, 161] could be extended to support updates if the underlying tree data structures adopted provide update operations.

Verifying the integrity of the returned documents is necessary for a setting where the cloud server is untrusted, which is a practical assumption that the cloud server may return partial documents or execute a fraction search for economic considerations. Chai et al. [21] first considered verifying the returned results under SSE and proposed a verification algorithm to detect the results' correctness and completeness using keyed hash functions. In general, the integrity of the results involves three aspects:

— *Completeness.* For a given query q, if the result set $\mathcal{R}$ satisfies that each document $D_i \in \mathrm{DB}(q)$ exists in $\mathcal{R}$ and $|\mathcal{R}| = |\mathrm{DB}(q)|$, we say that $\mathcal{R}$ is completeness.

— *Correctness.* For a given query q, if each document $R_i \in \mathcal{R}$ is outsourced from the data owner and has not been tampered with, we say that $\mathcal{R}$ is correctness.
— *Soundness.* For a given query q, if each document $R_i \in \mathcal{R}$ satisfies the query q, we say that $\mathcal{R}$ is soundness.

In summary, most existing schemes that support verification are constructed on hash functions or some of their derivatives. [132] used **Message Authentication Code** (**MAC**) to verify the correctness of the results. [105] verified the correctness of the results by appending signatures to the documents. [91] verified the correctness of the results by constructing a Merkle tree. Multiset hash functions [74, 174] are used to verify both correctness and completeness. Accumulators are usually used to verify the completeness of the results, such as the accumulator tree [137], the bilinear-map accumulator [149], the RSA accumulator [78], and so on. In addition to the common techniques described above, many extended algorithms [51, 177] are used for integrity verification. For soundness, users can locally verify it against the received results.

4 ATTACKS AND DEFENSES

Searchable Symmetric Encryption (**SSE**), as one type of protecting data privacy, enables searches over encrypted data stored in an untrusted server. However, it is far from enough to simply focus on data confidentiality. In practical applications, there is still a lot of valuable leakage information, such as the frequency of queries, the number of returned records for each query, and so on. Malicious adversaries or servers are able to use these leaks to infer information of interest to them.

In this section, we discuss the attacks based on the various leakage information against existing SSE schemes and describe a sequence of leakage profiles. Then we show, due to the emergence of new attacks, a plethora of schemes that protect information that might be exploited by adversaries. SSE has now toward fewer "leakage information" implementations motivated by existing threats and concerns.

4.1 Leakage Profiles

Before we enumerate the attacks SSE is facing, we discuss several widely studied leakages and give formal definitions of these based on existing literatures [12, 13, 80, 119]. Besides, we also introduce a new leakage, *update pattern*, that may compromise sensitive information, and give an informal definition.

Forward Privacy. Forward privacy [12, 75, 132, 174] is powerful privacy protection, which has been proved that disclosure of this kind of information will make the privacy of the queries subject to fatal adaptative attack [172]. Forward privacy aims at hiding what DSSE leaks about the relationship between newly added documents and the previous search tokens (or trapdoors).[4] For example, an adversary is able to infer that the updated document matches a previous query if DSSE does not take forward privacy into account. In other words, the adversary learns that the newly added document contains a particular keyword, even if he/she does not know what the keyword is. The substantial reason lies in that the search token for the same keyword is the same no matter how many times it is updated.

The leakage profile can be described as follows:

$$\{i|tok_1(\mathsf{ind}_i) = 1\}, \ldots, \{i|tok_n(\mathsf{ind}_i) = 1\}. \tag{5}$$

[4]Without loss of generality, the user generates token or trapdoor and sends it to the server as a query request. Therefore, we do not make a further distinction between tokens, trapdoors, and queries in this article; they all mean the same thing.

Intuitively, this leakage profile reveals the pattern of the updated documents match a previous token tok. Therefore, the first set includes all updated documents that match tok_1. Then we refer to [136] for a formal definition.

Definition 1 (Forward Privacy). A $\mathcal{L}$-adaptively-secure DSSE is called forward privacy iff the update leakage function can be written as

$$\mathcal{L}^{\text{Updt}}(\text{in}, \text{op}) = \mathcal{L}'(\text{op}, \text{ind}), \tag{6}$$

where $\mathcal{L}'(\text{op}, \text{ind})$ captures the operation op and the identifier ind of the updated document, but fails to have knowledge that the document matches the previous token. And $\mathcal{L}'$ is stateless.

Backward Privacy. Backward privacy was first mentioned by Stefanov et al. [132] and formalized by Bost et al. [13] with three different types of leakage of decreasing strength:
Backward Privacy with Insertion Pattern (Type-I): Type-I reveals the documents currently matching a search for a keyword ω, the timestamps they were inserted, and the total number of updates on ω.
Backward Privacy with Update Pattern (Type-II): In addition to leaking Type-I, Type-II also reveals the timestamps of all updates that occur on ω.
Weak Backward Privacy (Type-III): Besides revealing the Type-II, Type-III also leaks which deletion update canceled which addition update.
Subsequently, a line of work [22, 123, 135] focusing on the leakage proposed schemes with protecting backward privacy under various leakage profiles. However, although backward privacy has been proposed for a long time, it is almost not discussed the impact of such leakage or studied attacks against them. [135] considers that timestamp is crucial information (Type-III will leak when the deleted documents occur) and can be exploited to compromise security in many fields, such as timing analysis on network traffic [41] and side-channel attacks against hardware enclaves [26]. In any case, our goal is to minimize the leakage, even if there is no attack against such information.

Informally, backward privacy aims at preventing adversaries from learning which document has been deleted after searching on ω. In other words, if one adds a pair of document/keyword (ind, ω) and then deletes it, the adversary does not know the deleted document ind. It is worth noting that the addition and deletion operations are successive.[5] Next, we introduce the formal definition of backward privacy based on some leakage functions.

TimeDB(ω) captures all documents currently matching ω and the insertion times of those documents. And Q is a query list that records all operations, both addition and deletion. Thus, **TimeDB**(ω) can be written as follows:

$$\textbf{TimeDB}(\omega) = \{(u, \text{ind}) | (u, \text{add}, (\omega, \text{ind})) \in Q \text{ and } \forall (u, \text{del}, (\omega, \text{ind})) \notin Q\}. \tag{7}$$

UpHist(ω) is an update history that stores all update operations both addition and deletion on ω. **UpHist**(ω) is formalized as follows:

$$\textbf{UpHist}(\omega) = \{u | (u, \text{add}, (\omega, \text{ind})) \in Q \text{ or } (u, \text{del}, (\omega, \text{ind})) \in Q\}. \tag{8}$$

DelHist(ω), a deletion history on ω, records a list of timestamps for all deletion operations and the timestamps when the documents corresponding to these deletion operations are inserted. **DelHist**(ω) can be constructed as

$$\textbf{DelHist}(\omega) = \{(u^{\text{add}}, u^{\text{del}}) | \exists \text{ind s.t. } (u^{\text{del}}, \text{del}, (\omega, \text{ind})) \in Q \text{ and } (u^{\text{add}}, \text{add}, (\omega, \text{ind})) \in Q\}. \tag{9}$$

With these leakage functions in place, we formally define the three types of backward privacy.

[5] Inevitably, if a search is performed before a deletion operation and a subsequent search is performed, an adversary can compare the results of the two searches and infer the deleted document easily.

Definition 2 (Backward Privacy). A $\mathcal{L}$-adaptively-secure DSSE is called backward privacy with insertion pattern iff the leakage functions of $\mathcal{L}^{\text{Updt}}$ and $\mathcal{L}^{\text{Srch}}$ can be written as follows, where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless, and a_ω is the total number of updates on ω.

$$\mathcal{L}^{\text{Updt}}(\text{op}, \omega, \text{ind}) = \mathcal{L}'(\text{op}), \tag{10}$$

$$\mathcal{L}^{\text{Srch}}(\omega) = \mathcal{L}''(\mathbf{TimeDB}(\omega), a_\omega). \tag{11}$$

A $\mathcal{L}$-adaptively-secure DSSE is called backward privacy with an update pattern iff the leakage functions of $\mathcal{L}^{\text{Updt}}$ and $\mathcal{L}^{\text{Srch}}$ can be written as follows, where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

$$\mathcal{L}^{\text{Updt}}(\text{op}, \omega, \text{ind}) = \mathcal{L}'(\text{op}, \omega), \tag{12}$$

$$\mathcal{L}^{\text{Srch}}(\omega) = \mathcal{L}''(\mathbf{TimeDB}(\omega), \mathbf{UpHist}(\omega)). \tag{13}$$

A $\mathcal{L}$-adaptively-secure DSSE is called weak backward privacy iff the leakage functions of $\mathcal{L}^{\text{Updt}}$ and $\mathcal{L}^{\text{Srch}}$ can be written as follows, where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

$$\mathcal{L}^{\text{Updt}}(\text{op}, \omega, \text{ind}) = \mathcal{L}'(\text{op}, \omega), \tag{14}$$

$$\mathcal{L}^{\text{Srch}}(\omega) = \mathcal{L}''(\mathbf{TimeDB}(\omega), \mathbf{DelHist}(\omega)). \tag{15}$$

Search Pattern. Search pattern [17, 96, 128, 159] is a common leakage that exists in many constructions [89, 103, 120] and is ignored for efficiency considerations. It reveals which queries are performed on the same keyword ω. Formally, it can be defined as

$$\mathbf{sp}(\omega) = \{u | (u, \omega) \in Q\}, \tag{16}$$

where $\mathbf{sp}(\omega)$ captures all queries performed on ω with timestamps and thus is able to recognize which queries are repeated.

Access Pattern. Like the search pattern, access pattern [17, 27, 64, 130] is also a common leakage that captures all documents currently matching ω, which is defined as

$$\mathbf{ap}(\omega) = \{\text{DB}(\omega)\}, \tag{17}$$

where $\text{DB}(\omega)$ denotes all documents that contain the keyword ω. With $\text{DB}(\omega)$, the adversary learns not only the documents (not the content of the document) that currently match ω, but also the number of results.

Update Pattern. Update pattern is an emerging leakage pattern, which has not received much attention. [143] introduced the potential threat of such leakage for the first time, and proposed a new construction to protect such privacy. In a growing database, the update pattern records the entire update history on the database and it may record the time and the number of updates. With the leakage pattern, the adversary can infer some valuable information with the assistance of auxiliary information. Therefore, the update pattern leakage can be defined as

$$\mathbf{up}(\text{DB}) = \{(t_i, \mu_i), i \in N^*\}, \tag{18}$$

where t_i denotes the time when the update occurred and μ_i denotes the number of records updated.

Besides the leakages mentioned above, we introduce the leakages of the *result pattern* [82] and the *volume pattern* [56, 58, 70] additionally. The result pattern is a broader but rarely discussed leakage that reveals the results of queries, where the query can be of any type (such as single-keyword query, conjunctive query, etc.). But the access pattern in most existing literature [32, 57, 116, 125] usually refers to single-keyword query results. And the volume pattern represents the number of results returned by the query without the knowledge of the returned document identifiers. Therefore, in order to better characterize the different focuses of existing attacks and defenses, we have adopted a more detailed classification of these leakages.

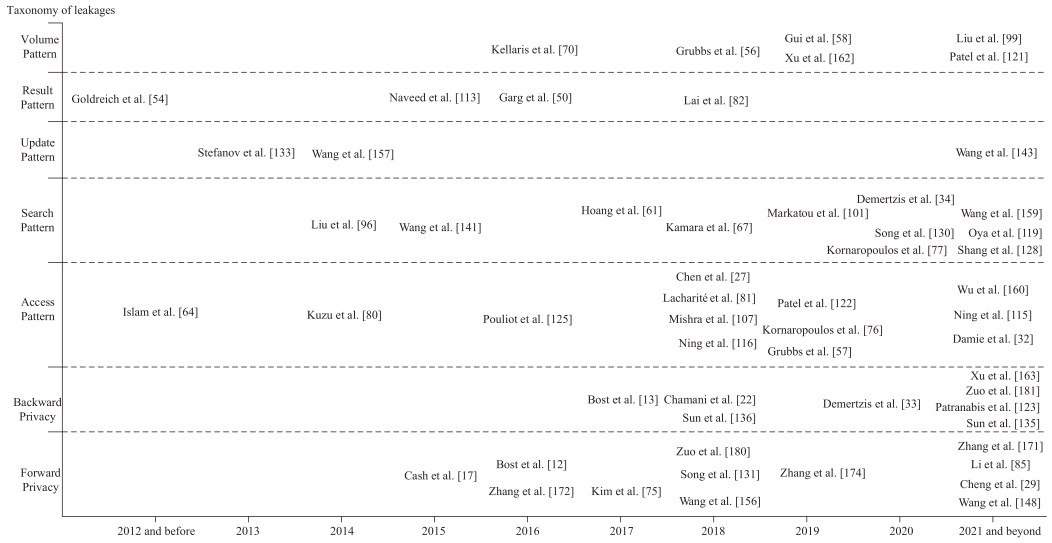


Fig. 11. Timeline of SSE attacks and defenses

In the following subsections, we will discuss the attacks based on these leakages and the defenses to prevent them from leaking. Figure 11 shows the timeline of partial articles on attacks and defenses. It can be seen that in recent years, the privacy of SSE has received a great deal of attention.

4.2 Attacks

In this subsection, we will discuss the attacks that exploit the leakages mentioned in Section 4.1. Table 3 summarizes these attacks. The goals of these attacks are mainly classified as *database recovery* and *query recovery*. Query recovery aims at recovering the keyword of a query, which facilitates the reconstruction of the database (since the keywords are part of the database), thus most schemes aim at recovering the query. And database recovery typically occurs on *range query* constructions [56, 58, 81].

4.2.1 Forward Privacy Leakage. Schemes without forward privacy disclose whether the newly added document matches the previous query, a feature that can be exploited by an active adversary for query recovery. *File-injection attack* [148, 156, 171, 172] is such an attack that exploits this kind of leakage to compromise the privacy of queries (Figure 12 shows the example of a file injection attack). The adversary injects the rigorous documents into the database, and then observes whether the token sent by the client matches the injected documents. If the returned results contain the injected documents, it means that the token is included in the keywords of the documents. Then, the keyword of the query can be inferred from the keywords contained in the injected documents (since each injected document contains the token, it is enough to calculate the intersection of the keywords of all injected documents). This kind of attack does not require the adversary to have much prior knowledge about the documents, and recovers the queries with 100% accuracy.

File-injection attacks are possible and devastating in email systems. Adversaries send rigorous emails to a client, who then encrypts them and hosts them on the cloud server (known as an email system with searchable encryption). This kind of attack was first introduced by Cash et al. [17], named *known-document attack*. Subsequently, Zhang et al. [172] further investigated the leakage of SSE, and proposed both adaptive and non-adaptive attacks. The key idea is to inject $\lceil \log |K| \rceil$

Table 3. A summary of the attacks

References	Leakage	Query Type	Prior Knowledge	Techniques Exploited	Attack Type	Objective	Countermeasures
Zhang et al. [172]	Forward Privacy, Access Pattern	Single, Conjunctive	Keyword universe, Partially known files	File Injection, Binary Search	Active Attack	Query Recovery	Limiting the number of keywords per indexed file, Padding files with random keywords
Wang et al. [148]	Forward Privacy, Access Pattern	Single, Conjunctive	Keyword universe, Partially known files	File Injection, Finite Set Theory	Active Attack	Query Recovery	–
Wang et al. [156]	Forward Privacy, Access Pattern	Order by, Range	Ideal leakage of OPE/ORE, Plaintext space	File Injection, Binary Search	Active Attack	Database Recovery	Trapdoor Permutation, Protecting forward privacy
Cash et al. [17]-1	Access Pattern	Single	Full known files	Count Strategy	Passive Attack	Query Recovery	Padding additional entries
Cash et al. [17]-2	Forward Privacy, Access Pattern, Search Pattern	Single	Distributional document and query knowledge	File Injection, Frequency Analysis	Active Attack	Plaintext Recovery	–
Islam et al. [64]	Access Pattern	Single	Underlying keywords for k queries, Fully (Partially) known files	Simulated Annealing, Frequency Analysis	Passive Attack	Query Recovery	Noise addition (adding fake files)
Pouliot et al. [125]	Access Pattern	Single	Plaintext corpus	Weighted Graph Matching, Labeled Graph Matching, Maximum Likelihood Estimation	Passive Attack	Query Recovery	Tuning of the Bloom filter parameters
Ning et al. [116]	Access Pattern	Single	Partially known files and the corresponding file identifiers	Count Strategy, Count Track	Passive Attack	Query Recovery	Blurring the (file identifier, token) relationship, Hiding the count/frequency of tokens
Ning et al. [115]	Access Pattern	Single	Partially known files	Count Strategy, Count Track	Passive Attack	Query and Plaintext Recovery	Padding files with dummy keywords
Damie et al. [32]	Access Pattern	Single	Distributionally similar files, Underlying keywords for k queries	Refine Score	Passive Attack	Query Recovery	Padding and obfuscation
Kellaris et al. [70]	Access Pattern, Volume Pattern	Range	Query distribution (uniform)	Frequency Analysis, Statistical Analysis, Polynomial Factorization	Passive Attack	Database Recovery	–
Grubbs et al. [56]	Volume Pattern	Range	Total number of records, Total number of possible data values	Graph-Theoretic Approach, Graph Pre-processing, Clique-finding Algorithm	Passive Attack	Database Recovery	Issuing batches several queries together, Putting a lower limit of a range query, Adding dummy records
Gui et al. [58]	Volume Pattern	Range	Maximum label N	Breadth-first Search, Clique-finding Algorithm	Passive Attack	Database Recovery	Padding the returned answers, Containing as many as possible volumes
Grubbs et al. [57]	Access Pattern	Range	Total number of possible data values, Set of all possible queries	Statistical Learning Theory, PQ-tree	Passive Attack	Database Recovery	–
Kornaropoulos et al. [76]	Access Pattern	k-NN query	All possible responses to k-NN queries, Exact length of the Voronoi segment	Voronoi Diagram	Passive Attack	Database Recovery	–
Kornaropoulos et al. [77]	Access Pattern, Search Pattern	Range, k-NN query	Number of encrypted values, Size of the universe of database values, Endpoints of the universe	Support Size Estimation, Statistics Learning Theory, Convex Optimization, Voronoi Diagram	Passive Attack	Database Recovery	Hiding the overlap of records between responses
Liu et al. [96]	Search Pattern	Single	Users' search habits	Frequency Analysis, Euclidean Distance	Passive Attack	Query Recovery	Launching several fake queries along with the real query
Markatou et al. [101]	Access Pattern, Search Pattern	Range	Size of the universe of database values	PQ Tree	Passive Attack	Database Recovery	–
Oya et al. [119]	Access Pattern, Search Pattern	Range	A training set	Maximum Likelihood Estimation	Passive Attack	Query Recovery	Hiding the search pattern with collisions, Hiding the search pattern with fresh randomness, Hiding the query frequencies with dummy queries

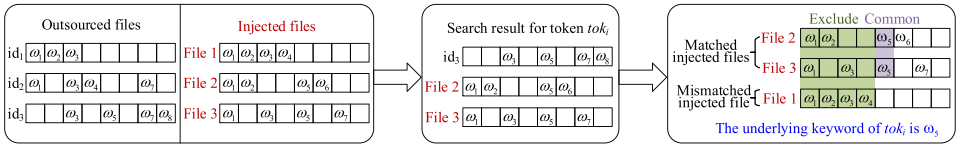


Fig. 12. Example of a file injection.

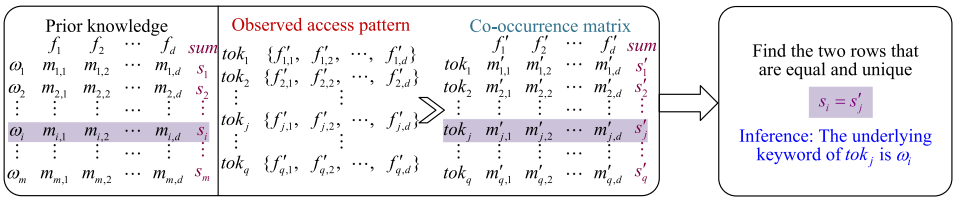


Fig. 13. Example of a simplified version of access pattern attack.

documents into the database, each containing $\lceil |K|/2 \rceil$ different keywords, where $K = \mathsf{w}_1 \cup \mathsf{w}_2 \cup \cdots \cup \mathsf{w}_d$ is the universe keyword set, and it is assumed that the adversary knows. The adversary can successfully recover the query by observing the keywords contained in the documents matching the query. Based on this, Zhang et al. have proposed a more efficient method by utilizing the frequency of occurrence of the queries and keywords in the client's documents, which decreases the number of injected documents. Wang et al. [148] further reduced the number of injected documents by dividing the universe keyword set into several subsets based on the method of constructing a uniform (k, n)-set of a finite set.

4.2.2 Access Pattern Leakage. For tradeoff security and efficiency, many existing schemes do not take protecting the access pattern into account. From the perspective of the user, meanwhile, it seems to be a common leakage since the user sends a query and expects the server to return results that satisfy the query. However, it is such a seemingly trivial and inevitable leakage that poses a great risk. Intuitively, an adversary observes a list of encrypted documents that match the trapdoor, $(tok, \{d_1, d_2, \ldots, d_l\})$. Based on the observations, the adversary can generate valuable information (e.g., *trapdoor-identifier matrix, trapdoor-trapdoor matrix*) and exploit this information to reveal the relationship between the trapdoor and the underlying keyword or reconstruct the database, thus, compromising the scheme privacy (Figure 13 shows a simplified version of such an attack).

Islam et al. [64] were the first to investigate the practical impact of the access pattern leakage and proposed a formal inference attack model based on the access pattern leakage and some prior knowledge. Their attack assumes that an adversary knows the underlying keywords for partial trapdoors and a matrix M that each cell (i, j) represents the probability of both ith and jth keywords appearing in any document. Under these assumptions, the adversary exploits the access pattern leakage to generate the *trapdoor-trapdoor* co-occurrence matrix M_1, which records the probability of any two trapdoors appearing in any document, the same as M. For each cell in M_1, the adversary finds the closest cell in M, then recovers the underlying keyword related to the trapdoor. Pouliot et al. [125] proposed attacks using combinatorial optimization problems based on graph matching: weighted graph matching and labeled graph matching. The attacks assume that an adversary knows a plaintext corpus and a ciphertext corpus, and then generates graphs G and H, respectively, based on prior knowledge, and finally reveals the underlying keywords for trapdoors by solving the graph matching problem. Ning et al. [116] proposed attacks under different types of assumptions, revealing the relationship between trapdoors and keywords by mapping

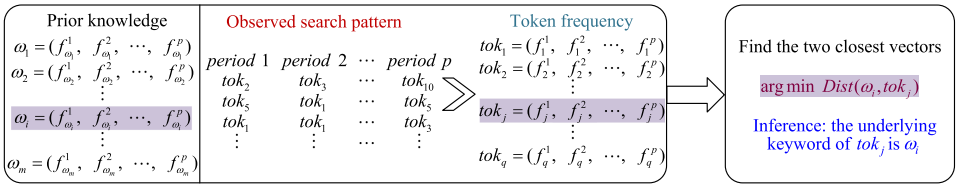


Fig. 14. Example of a simplified version of search pattern attack.

trapdoors to known keywords. Extendedly, they proposed a new leakage-abuse attack [115] based on the L2 leakage and partially known documents, which is able to accurately recover the underlying keywords of trapdoors. Damie et al. [32] designed a novel attack based on a similar document set instead of a real document set, named *refined score attack*, which reveals underlying keywords for trapdoors by scoring trapdoor-keyword pairs, with the highest score being the matching result. The refined score attack also requires information about some trapdoors and their related keywords, though less than Islam et al.'s [64] attack.

Besides exploiting the access pattern to infer the underlying keywords for queries, an adversary also exploits it to reconstruct the range query database [57, 76, 81]. Kellaris et al. [70] designed attacks to reconstruct a database using an access pattern and demonstrated for the first time that it was possible to reconstruct a database using only communication volume. Although the attacks assume that an adversary needs to know the distribution of queries and observe $O(N^4)$ queries, where N is the domain size. Grubbs et al. [56] further investigated the practical database reconstruction from volume leakage on range queries and improved the attacks of Kellaris et al. in multiple dimensions, including the assumptions and leakage information. Their attacks only assume that an adversary knows volume leakage but not the distribution of queries, and then introduce the *elementary range* and the *elementary volume* to implement reconstruction attacks based on a novel graph-theoretic approach. Gui et al. [58] continued previous work [56, 70] on volume leakage attacks and further weakened the prior knowledge required for the success of the attacks. Inspired by the attacks of Grubbs et al., their attack goal is to recover the elementary volumes by expanding partial solutions iteratively. Kornaropoulos et al. [76] first studied data recovery attacks against schemes supporting ***k*-nearest-neighbor** (***k*-NN**) queries and demonstrated that the attacks could be extended to multidimensional spatial data by Hilbert curves. They analyzed the feasibility of exact reconstruction theoretically under certain assumptions and then proposed practical attacks for approximate reconstruction without unrealistic assumptions.

4.2.3 Search Pattern Leakage. Similar to the access pattern, the search pattern is also a common and overlooked leakage, and the vast majority of SSE schemes suffer from it. Nevertheless, its practical implications have not been fully studied so far. Conceptually, the search pattern reveals which queries are derived from the same keyword, but it is hard to hide if the access pattern is still accessible. Because even if the probabilistic algorithm is used to generate the queries, the matching results are the same, so the adversary is able to get the search pattern by observing the results returned by the previous queries. From the perspective of an adversary, the search pattern discloses the frequency of distinct queries, which can be exploited by statistical and inference attacks to deduce sensitive information (Figure 14 shows a simplified version of such an attack).

Liu et al. [96] explored the possibility of disclosing the underlying keywords for queries based on the search pattern leakage and some auxiliary information. They mapped queries to keywords based on the similarity in frequency under the knowledge of the user's search habits. Furthermore, they presented an extended attack, an *adaptive attack*, without knowledge of specific background at the beginning, which adaptively adjusts the background of the keywords after each attack to

improve the recovery accuracy. Instead of relying on the distribution of queries [57, 70], dense databases [81], or certain prior knowledge [96], Markatou et al. [101] exploited access and search pattern leakage to reconstruct the full database for range queries. Their attack first recovered the order of the database records by value via PQ trees [10], which relies on the access pattern only. Then, based on the ordered records, the adversary reconstructs the full database by using the number of distinct range queries that return a specific response to calculate the distances between consecutive records after observing all possible queries.

Recently, some works [77, 119] have investigated the potential risks of SSE leaking the search pattern under weaker assumptions. Kornaropoulos et al. [77] proposed the value reconstruction attacks for both range queries and k-NN queries beyond the distributions of queries. Their attacks take the same idea as [101], where the distances between consecutive records are inferred by the number of specific responses returned. However, instead of requiring the entire number of all possible queries, the attacks approximately reconstruct an encrypted database by estimating the number of specific responses returned using search pattern leakage. Therefore, the core problem of the attacks is to estimate the number of unseen range queries from the frequency of the observed range queries. And their experimental evaluations demonstrate the attacks are useful under different query distributions and various database densities. Unlike aforementioned attacks [77, 101] that require the assistance of access pattern leakage, Oya et al. [119] developed a kind of attack against SSE hiding the access pattern while leaking the search pattern, aiming at recovering the underlying keywords of the queries. Basically, their attack relies on the frequency and volume of queries by using a MLE approach to find the most likely underlying keyword behind a query. Furthermore, they modified the attack to target particular privacy-preserving SSE schemes (such as obfuscated access pattern [27, 34], volume-hiding [122], etc.) based on the knowledge of the parameters used by these defenses, which works well in query recovery.

4.3 Defenses

In this subsection, we will discuss the existing schemes with leakage suppression, which mainly protect the leakages mentioned above. Table 4 summarizes the privacy protected by these schemes.

4.3.1 Forward Privacy Protection. The forward privacy aims at preventing newly inserted files from being matched by previous queries, thus mitigating file-injection attacks. From the perspective of file-injection attacks, the reason why the query recovery can be successful is that the previous queries can match the files inserted (injected) later. In other words, if one wants to retrieve the newly inserted files, he/she does not need to produce a new query. Intuitively, to thwart file-injection attacks, SSE can require a new search token for retrieval after inserting a new file. More specifically, it requires that after inserting a new file, only the reproduced query can find the newly inserted file, and that the previous queries match only files that were already in the database before the queries were generated, not files that are newly inserted later. This straightforward method cuts off the connection between the newly inserted files and the previous queries, realizing forward privacy.

Oblivious RAM (**ORAM**) is one of the effective ways to protect forward privacy, which is a compiler that protects the input-output behavior of the algorithm by continuously shuffling and re-encrypting the accessed data. The concept of ORAM was formulated by Goldreich [54] and has been improved in algorithms and data structures [50, 133, 157, 158]. But this is not a wise choice nor a practical method, since it achieves security at the expense of efficiency and storage space. Subsequently, Bost et al. [12] proposed a forward privacy scheme with optimal search and less storage, named $\Sigma o\varphi o\varsigma$, which can be transplanted to the existing SSE schemes without forward privacy with minor changes. Their scheme adopts an inverted index structure and a trapdoor

Table 4. A summary of the privacy protected by the schemes

References	Privacy Protected	Client Storage	Server Storage	Search Complexity	Search Computation	Communication Complexity	Round	Update Communication	Update Computation	Primitives
Bost et al. [12]	Forward Privacy	$O(m \cdot \log d)$	$O(\sum_{\omega \in W} a_\omega)$	$O(a_\omega)$	$O(a_\omega)$	$O(c)$	$O(1)$	$O(1)$	$O(1)$	Trapdoor Permutation
Bost et al. [13]-Diana	Forward Privacy	$O(m \cdot \log d)$	$O(\sum_{\omega \in W} a_\omega)$	$O(a_\omega)$	$O(a_\omega)$	$O(c + \log a_\omega)$	$O(1)$	$O(1)$	$O(\log a_\omega)$	Range Constrained PRF
Bost et al. [13]-Fides	Forward Privacy, Type-II	$O(m \cdot \log d)$	$O(\sum_{\omega \in W} a_\omega)$	$O(a_\omega)$	$O(a_\omega)$	$O(a_\omega)$	$O(2)$	$O(1)$	$O(1)$	Trapdoor Permutation
Bost et al. [13]-Diana$_{\text{del}}$	Forward Privacy, Type-III	$O(m \cdot \log d)$	$O(\sum_{\omega \in W}(a_\omega + d_\omega))$	$O(a_\omega + d_\omega)$	$O(a_\omega)$	$O(c + d_\omega \log a_\omega)$	$O(2)$	$O(1)$	$O(\log a_\omega)$	Range Constrained PRF, Puncturable PRF
Bost et al. [13]-Janus	Forward Privacy, Type-III	$O(m \cdot \log d)$	$O(\sum_{\omega \in W}(a_\omega + d_\omega))$	$O(a_\omega + d_\omega)$	$O(c \cdot d_\omega)$	$O(c)$	$O(1)$	$O(1)$	$O(1)$	Puncturable Encryption
Sun et al. [136]	Forward Privacy, Type-III	$O(m \cdot \log d)$	$O(\sum_{\omega \in W}(a_\omega + d_\omega))$	$O(a_\omega + d_\omega)$	$O(c \cdot v)$	$O(c)$	$O(1)$	$O(1)$	$O(v)$	Symmetric Puncturable Encryption
Chamani et al. [22]-Mitra	Forward Privacy, Type-II	$O(m \cdot \log d)$	$O(\sum_{\omega \in W} a_\omega)$	$O(a_\omega)$	$O(a_\omega)$	$O(a_\omega)$	$O(2)$	$O(1)$	$O(1)$	–
Chamani et al. [22]-Orion	Forward Privacy, Type-I	$O(1)$	$O(N)$	$O(c \log^2 N)$	$O(c \log^2 N)$	$O(c \log^2 N)$	$O(\log N)$	$O(\log^2 N)$	$O(\log^2 N)$	Oblivious Map
Sun et al. [135]	Forward Privacy, Type-II	$O(m \cdot v)$	$O(\sum_{\omega \in W} a_\omega)$	$O(a_\omega)$	$O(c)$	$O(c)$	$O(1)$	$O(1)$	$O(1)$	Symmetric Revocable Encryption
Demertzis et al. [33]-SD$_a$	Forward Privacy, Type-II	$O(1)$	$O(N)$	$O(a_\omega + \log N)$	$O(a_\omega + \log N)$	$O(a_\omega + \log N)$	$O(1)$	$O(\log N)(am.)$	$O(\log N)(am.)$	–
Demertzis et al. [33]-SD$_d$	Forward Privacy, Type-II	$O(1)$	$O(N)$	$O(a_\omega + \log N)$	$O(a_\omega + \log N)$	$O(a_\omega + \log N)$	$O(1)$	$O(\log^3 N)$	$O(\log^3 N)$	Oblivious Map
Demertzis et al. [33]-QOS	Forward Privacy, Type-III	$O(1)$	$O(N + m)$	$O(c \log a_\omega + \log^2 m)$	$O(c \log a_\omega + \log^2 m)$	$O(c \log a_\omega + \log^2 m)$	$O(1)$	$O(\log^3 N)$	$O(\log^3 N)$	Oblivious Map
Chen et al. [27]	Access Pattern	$O(1)$	$O(n \cdot v)$	$O(c \cdot v)$	$O(c \cdot v)$	$O(c)$	$O(1)$	–	–	Differential Privacy, Erasure Coding
Patel et al. [122]-dprfMM	Volume Pattern	$O(v)$	$O(n)$	$O(M + v)$	$O(M)$	$O(M)$	$O(1)$	–	–	Cuckoo Hashing, DPRF
Patel et al. [122]-dpMM	Volume Pattern	$O(v_1 + v_2)$	$O(n + m)$	$O(c + \text{Lap}(\frac{2}{\epsilon}) + v)$	$O(c + v_3 + \text{Lap}(\frac{2}{\epsilon}))$	$O(c + v_3 + \text{Lap}(\frac{2}{\epsilon}))$	$O(2)$	–	–	Cuckoo Hashing, DPRF, Differential Privacy
Shang et al. [128]	Access Pattern Search Pattern	$O(1)$	$O(d \cdot a_m)$	$O(d \cdot ctr_{max})$	$O(d \cdot ctr_{max})$	$O(M \cdot ctr_{max} + c)$	$O(1)$	–	–	Inner Product Predicate Encryption
Xu et al. [162]	Access Pattern	$O(1)$	$O(m \cdot M)$	$O(M)$	$O(1)$	$O(M)$	$O(1)$	–	–	Padding
Liu et al. [96]	Search Pattern	$O(1)$	$O(n)$	$O(v \cdot r)$	$O(1)$	$O(v \cdot r)$	$O(1)$	–	–	Grouping
Song et al. [130]	Access Pattern, Search Pattern, Forward Privacy	$O(1)$	$O(m + v_1 \log \frac{1}{\epsilon})$	$O(v_2 \cdot s)$	$O(v_2 \cdot s)$	$O(c + v_2)$	$O(2)$	$O(1)$	$O(m + v_3)$	Re-encryption Cryptosystem
Wang et al. [143]-DP-Timer	Update Pattern	$O(v)$	$O(\sum_{t \in [1,2,\cdots]}(v_t + \text{Lap}(\frac{1}{\epsilon})))$	$O(a_\omega)$	$O(a_\omega)$	$O(c)$	$O(1)$	$O(v + \text{Lap}(\frac{1}{\epsilon}))$	$O(1)$	Differential Privacy
Lai et al. [82]	Result Pattern	$O(1)$	$O(n + n \log \frac{1}{\epsilon})$	$O(r)$	$O(r \cdot \lvert q \rvert \cdot s)$	$O(c + rn \log \frac{1}{\epsilon} + r\lvert q \rvert)$	$O(3)$	–	–	Hidden Vector Encryption

Notes. Type-I (resp. Type-II, Type-III) refers to the strength level of backward privacy. Update Complexity (resp. Update Communication) refers to the cost (resp. communication) per the updated document/keyword pair. a_ω refers to the number of times involving keyword ω is added. d_ω refers to the number of times involving keyword ω is deleted. W refers to universe keyword set. N refers to the maximum number of records that can be inserted. $am.$ refers to amortized efficiency. ε refers to privacy budget. $ctr_{max} = 2 \ln d / \ln \ln M$. DPRF refers to Delegatable PRF. For those schemes that introduce false positives and false negatives to protect privacy, the analysis of their complexity is approximate.

permutation function to generate search token ST. Each pair of keyword/document (ind, ω) will be stored at a location (Update Token, UT) derived from ST using a keyed hash function. Specifically, the client will generate the search token $ST_{i+1}(\omega)$ from the $ST_i(\omega)$ for the inserted pair $(\mathsf{ind}_{i+1}, \omega)$ applying the trapdoor permutation function, and then generates UT_{i+1} from ST_{i+1} and the keyed hash function. The server stores an inserted element in the location of UT_{i+1} (without the knowledge of ST_{i+1}). When the client retrieves documents containing ω, he/she will send the latest ST_{i+1} to the server. After receiving ST_{i+1}, the server depends on the public key to recover all $ST_j, (0 \leqslant j \leqslant i)$, and generates the corresponding $UT_j, (0 \leqslant j \leqslant i+1)$. Finally, the server returns all the documents on UT_j.

The key idea of [12] is that before the client issues the search token ST_{i+1}, the server can neither generate ST_{i+1} from ST_i nor infer other update tokens $(UT_j, 0 \leqslant j \leqslant i)$ based on UT_{i+1}. Therefore, the newly inserted document is unlinkable with previous search tokens. This kind of method of protecting forward privacy has been widely borrowed and developed, such as improving the search efficiency [75, 131], providing verifiability of results [85, 174], and supporting range query [180, 181]. Besides, Bost et al. [13] developed another SSE scheme with forward privacy based on the constrained pseudorandom function, which allows the server to find all matching documents, including newly inserted documents after receiving the constrained key allowing the evaluation of all existing documents. Otherwise, the server can not use the previous constrained keys to predict the evaluation for the newly inserted document. Chen et al. [29] maintained a structure **Count** to store the number of times a keyword was updated after the most recent search, as well as the total number of times the keyword was searched. With each update, the client generates a new key based on the update count and determines the corresponding storage location. The underlying idea of these approaches is the same as before, that newly inserted documents must be searched using the new search token, and that it is unlinkable between the location of the inserted document and the location of the previously searched documents.

4.3.2 Backward Privacy Protection. The backward privacy aims at preventing deleted documents from being leaked during searches, which has been introduced recently. Bost et al. [13] formally defined three types of backward privacy of decreasing strength for the first time and gave instantiations of different types. The first construction, named Moneta, is backward privacy with insertion pattern which leaks the least information, and it is realized based on ORAM at the cost of bandwidth, storage, and computation. The second construction, named Fides, is backward privacy with update pattern, which is a generic two-roundtrip scheme from an arbitrary scheme. In this construction, the client uploads a ciphertext $E_{k_\omega}(\mathsf{ind},\mathsf{op})$ to the server, where k_ω is an encryption key of keyword ω and $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$, so the server cannot distinguish whether the received ciphertext is added or deleted. When the client performs a search, the server will return all ciphertexts including the op is del. Then the client decrypts all ciphertexts and removes the deleted documents, and obtains all documents matching the search. In the end, the client returns all ciphertexts that have not been deleted to the server for storage. Additionally, they proposed weak backward privacy in a single roundtrip by using puncturable encryption with incremental punctures, named Janus. The construction initializes two forward-secure SSE instances, one for storing documents encrypted with incremental puncture encryption, and the other for storing punctured keys. When the client performs a search, the secret key sk_0 and the search token are sent to the server for searching on both instances. After the server finds all the ciphertexts and the punctured keys, it decrypts and obtains all the documents that have not been deleted. However, the server learns which deletion operation canceled which insertion operation during the decryption, so the construction achieves weak backward privacy.

Since then, a body of literature [33, 150, 181] on how to improve the efficiency and rich functionalities of schemes with backward privacy has been emerging. Instead of using a public-key cryptosystem, Sun et al. [136] proposed a more efficient backward-secure construction from symmetric puncturable encryption, which is based on simple cryptographic tools (such as AES). The construction preserves the privacy property of weak backward privacy and speeds up the search latency. Chamani et al. [22] proposed three novel constructions with different levels of backward privacy that improved previous results in multiple ways. The first scheme takes the same idea as Fides, uploading the triplet $(\mathsf{op}, \omega, \mathsf{ind})$ to the server and storing it in a dictionary, which achieves type-II backward privacy and forward privacy. The second scheme relies on an oblivious map [158] to achieve type-I backward privacy, which has quasi-optimal search time and non-trivial interaction. The final scheme modifies the second one by reducing the number of roundtrips to improve the search time at the cost of leaking more information (weak backward privacy). Sun et al. [135] proposed a generic forward and backward privacy scheme relying on symmetric revocable encryption, which is non-interactive type-II backward privacy. Xu et al. [163] explored scenarios where clients may issue duplicate update queries or delete queries to remove entries that do not exist. To address this issue, they proposed the use of key-updatable PRF cryptographic primitives in order to achieve a more robust DSSE with both forward and backward privacy. In addition to focusing on the backward privacy of single keyword search, the SSE community develops the expressiveness of search. Patranabis et al. [123] proposed a type-II backward privacy and forward privacy scheme for conjunctive search, called **Oblivious Dynamic Cross Tags** (**ODXT**), which is derived from [19]. Zuo et al. [180] proposed a forward and backward privacy scheme supporting range queries by leveraging the Paillier cryptosystem and binary tree.

4.3.3 Access Pattern Protection. Leaking access pattern incurs severe catastrophe for the confidentiality of the outsourced databases, which has been exploited by many attacks to recover the queries or reconstruct the databases as aforementioned [64, 70, 81, 125]. Conceptually, the access pattern discloses the exact results matching the queries, and the adversaries have developed a variety of attacks based on this leakage. Intuitively, the straightforward method to prevent it from leaking is to return ambiguous results with matched and mismatched records (i.e., false positive and false negative). In general, the state-of-the-art protections for access pattern are mainly categorized into access-pattern obfuscation, result padding, and oblivious access.

Differential privacy is a frequently used technique for access-pattern obfuscation, where the client introduces noise (or dummy records) when generating indexes or queries, and then the server returns an approximate result set with ε-differentially private. Kuzu et al. [80] first considered a privacy-aware SSE scheme leaking obfuscated access pattern in a differentially private way, which keeps some records locally and injects some dummy records into the outsourced database. Subsequently, Chen et al. [27] proposed a novel framework to protect the access pattern by borrowing the differential privacy. They implemented the obfuscation mechanism in the index construction, so that the keyword list corresponding to each document contains some keywords that do not belong to it and some keywords that belong to it. Moreover, to guarantee that the client can obtain all matching documents, the framework uses erasure coding to partition a document into multiple shards, so the client is able to restore the original document based on the partial shards received. However, this framework introduces differential privacy in the setup phase so that the same query will get the same result, and this repetition reveals the queries frequency of the keywords. Shang et al. [128] considered this drawback and proposed an **Obfuscated SSE** (**OSSE**) that obfuscates the access pattern independently for each query performed. The core idea of their construction is to introduce random false positives and false negatives into the generation of a query, and generate multiple tokens per query to avoid a single token matching multiple documents.

Therefore, OSSE achieves a high level of security but at the cost of heavy computation. Similarly, Patel et al. [122] used differential privacy to hide the volume pattern by introducing noise during the query, so the number of records returned by each query is distinct, even for the same query.

Result padding is another technique that is often used to protect the access pattern [17, 64, 122, 162]. Specifically, it mainly aims at hiding the volume of any query from an adversary. A naive method is padding the number of records for each query to M, where M is the maximum records of any query. Unfortunately, the simplest way burdens the storage cost of the service provider. Therefore, Patel et al. [122] proposed storage-friendly encrypted multi-maps based on cuckoo hashing, which achieves the optimal storage overhead of $(2+\alpha)$ times the original cost and returns $2M$ records for any query. Furthermore, Xu et al. [162] theoretically analyzed the relationship between security strength and padding overhead based on information theory, and then proposed an extended approach based on the existing works [17, 64] with the main contribution being that the inserted dummy records are carefully crafted and indistinguishable from the real records.

In addition to relying on differential privacy or result padding, ORAM techniques have recently been increasingly used to protect against access pattern leakage [34, 107]. Mishra et al. [107] combined ORAM techniques (such as Path ORAM [133]) and hardware enclaves (such as Intel SGX) to propose an efficient oblivious search index, Oblix, which preserves the privacy of both the access pattern and the result size. Specifically, they designed a novel implementation of *doubly-oblivious data structures* to obliviously access the external memory at the server and the client's internal memory, and returned a fixed number of records to hide the result size. Demertzis et al. [34] proposed a family of SE schemes combining an adjustable ORAM and an adjustable padding algorithm, named **Searchable Encryption with Adjustable Leakage** (**SEAL**), that leaks a few bits of search or access pattern. On the one hand, the data is partitioned into 2^{α} regions and stored in separate ORAMs, and the operations in each ORAM are not modified. On the other hand, each keyword list size is padded to the closet power of x. Therefore, the parameters α and x are the leaked bits of search pattern and access pattern, respectively, and they can be adjusted adaptively for different efficiency and security considerations.

4.3.4 Search Pattern Protection. To eliminate the potential risks caused by the search pattern leakage, some literature [17, 96, 119] has proposed heuristic protections to mitigate the risks while proposing the attacks. Liu et al. [96] suggested dividing the universal keywords into multiple subsets, and each time a query is made, entire keywords in the subset containing the real keyword are queried. Cash et al. [17] pointed out that the ORAM might work to hide the leakage but, unfortunately, is significantly computational. Oya et al. [119] considered multiple defenses against their proposed attacks relying on the search pattern. One option to hide the search pattern is to force the same documents to be returned for different keywords. And another option is to add fresh randomness to the returned documents.

However, the above countermeasures have limitations because they are all possible effective for specific attacks, and the search pattern may also be induced through other leakages (such as the access pattern or the volume pattern). To be more general, Kamara et al. [67] proposed a suppressing search pattern leakage construction, which results from a basic SSE scheme (the Piggyback Scheme, PBS) with response-hiding and two compilers, **Rebuild Compiler** (**RBC**) and **Cache-Based Compiler** (**CBC**), respectively. The construction first applies RBC to PBS to make it rebuildable and then transforms it into a scheme that leaks no search pattern by using CBC. Furthermore, Song et al. [130] considered protecting both the access pattern and the search pattern under a dynamic SSE scheme supporting multi-user. They designed an *index shuffle protocol* and an *index redistribution protocol* using a proxy re-encryption cryptosystem between two cloud servers

and then shuffled entire index entries after leaking the maximum allowable information. Therefore, it is impossible for the non-colluding cloud servers to learn both the access pattern and the search pattern. [141] and [159] used a polynomial representation of multiset to construct the index structure and introduced random values each time a trapdoor was generated. Therefore, even for the same keyword, the trapdoor generated by each query is different. Similarly, Shang et al. [128] introduced random false positives and false negatives in each trapdoor generation, making it possible to get different trapdoors even when the same keyword is queried, thus hiding the search pattern. Moreover, to prevent an adversary from inferring the search pattern from the response length, [141] and [159] padded the response to the same size, and Shang et al. [128] hid the access pattern by returning different records.

4.3.5 Update Pattern Protection. Different from the aforementioned forward privacy and backward privacy, the update pattern considers the privacy of *when* updated dataset, which may breach the privacy of an outsourced database if leaked. Wang et al. [143] first considered the problem and posed a simple example to clarify the potential risk of leaking such information, and pointed out that the leakage is common in any event-driven update. And then, they proposed a framework, DP-Sync, to prevent the update pattern from leaking. Specifically, two differentially private synchronization strategies, DP-Timer and DP-ANT, are provided for obtaining different 3-way tradeoffs between privacy, accuracy, and performance. The Timer-based synchronization strategy (DP-Timer) synchronizes an update every T moments with a varying number of records. For each update, the client uploads c records (may contain dummy records) to the server, where c is the number of all newly received records within T moments with Laplace noise $\mathsf{Lap}(\epsilon)$. The second strategy, Above Noisy Threshold (DP-ANT), synchronizes an update after the client receives approximately θ records. For each update, the client will generate a threshold, $\tilde{\theta} = \theta + \mathsf{Lap}(\epsilon')$, which is regenerated in each round. And then if the client receives $\tilde{\theta}$ records, he/she will signal synchronization.

Both of the above two strategies hide the number and the inserted time of records in each update, and ensure update privacy. To achieve different tradeoffs between privacy, accuracy, and performance, the parameters, T and θ, of the strategies can be adjusted adaptively. For instance, to achieve high accuracy, parameter T can be set close to a single time unit, or parameter θ can be set close to a single record.

4.3.6 Result Pattern Protection. The recently proposed **Oblivious Cross-Tags** (**OXT**) protocol [19] supports efficiently conjunctive keyword search at the cost of leaking "partial" database information to the server. Specifically, OXT reveals to the server which documents contain both keyword pairs (ω_1, ω_i) when performing a search, where ω_1 is the least frequent keyword and ω_i is the other keywords in the conjunctive query. This leakage is referred to as the **Keyword Pair Result Pattern** (**KPRP**) and has been exploited by attacks [17, 64, 113, 172] to compromise the database confidentiality. Therefore, there arises a concern about the security-efficiency tradeoffs for the OXT protocol.

Motivated by the particular leakage of OXT, Lai et al. [82] attempted to eliminate the KPRP leakage while preserving the performance of OXT. They proposed a new SSE protocol, **Hidden Cross-Tags** (**HXT**), extended on OXT that reveals only the documents matching all query keywords but removes KPRP. The HXT additionally relies on the **Symmetric-key Hidden Vector Encryption** (**SHVE**) and the **Bloom Filters** (**BF**) compared to OXT. It inserts the elements stored in XSet into BF, and encrypts BF with SHVE. When executing the query, the XSet membership test for conjunctions in OXT is replaced by an SHVE token generation and query. Thus, HXT hides the KPRP information about whether the document containing ω_1 contains other ω_i, but introduces the extra interactions.

5 OPEN ISSUES AND FUTURE INSIGHTS

In this section, we discuss the open issues in existing SSE and envision research directions subsequently.

5.1 Open Issues in Existing SSE

— **Functionalities.** Most existing schemes adopt inverted lists as the index structure and perform the search in memory. However, these in-memory schemes are not suitable for extensive databases and indexes because they cause substantial random accesses and the indexes need to be stored on disk. Despite a few works on this issue, it is still in the early stages of research, and there are many problems to be solved (such as update, query expressiveness, storage efficiency, etc.). How to design an efficient and practical scheme to support very large databases is of great significance in promoting the application of SSE. Besides, the relationship between storage, efficiency, and security is an open issue that needs to be considered. The current schemes usually sacrifice the performance of other aspects to optimize one. How to strike the balance of these three metrics remains an open issue to be addressed.

— **Attacks.** Analyzing the information leakages in the search and update and designing effective attacks can help discover deficiencies in the SSE schemes and thus remedy them. Therefore, it is of great significance to design an efficient, convenient, and accurate attack to reinforce the robustness of the schemes. However, many existing attacks rely on assumptions, such as the knowledge of the query distributions, all possible tokens, or the underlying keywords for partial trapdoors, and so on, some of which seem unrealistic. Few studies aim at compromising privacy by inherent leakage without other prior assumptions. It is critical to design attacks that rely as little as possible on extra assumptions.

— **Defenses.** Providing whole-process security for data is challenging but vital for users to protect data privacy. There have been countermeasures for existing known information leakages that may compromise data confidentiality in recent years. However, these countermeasures are aimed at a single leakage and cannot effectively prevent data privacy from various attacks. Integrating defenses against multiple attacks can be helpful but may cause unnecessary computational costs and utility loss. A practical SSE scheme should satisfy the users' requirements concerning security, efficiency, and functionality. On the search side, protection against the search pattern relies on computationally expensive cryptographic tools or increased communication overhead, so it would be desired to develop efficiency-oriented protection techniques for search patterns. Meanwhile, it is necessary to protect both the access pattern and the search pattern, since the search pattern can be inferred from the access pattern. On the update side, achieving efficient backward privacy is challenging, especially considering designing efficient Type-I backward privacy in a single roundtrip. In addition, most of the existing schemes that support forward and backward private allow limited query expressiveness, so it is still open to constructing secure SSE schemes and supporting as rich query expressiveness as possible.

5.2 Research Directions

Bridging the gap between encrypted data searches and plaintext searches is an ongoing research direction, including efficiency, query expressiveness, multi-cloud collaboration, scalability, and so on. Combining memory and disk to design cross-hierarchy indexes and retrieval protocols may be a potential idea for efficiently searching very large databases. Meanwhile, research on multiple cloud servers to cooperate to perform storage, search, computing, and other operations. Enriching query expressiveness (such as regular matching) or incorporating multiple query expressiveness can be considered to improve the accuracy further.

Current schemes focus on a single data type (e.g., textual, images, spatial text), and few schemes consider the search involves multiple data types. With the diversification of data, supporting various types of data search has become a future requirement. For example, someone wants to search for images containing "school" within a specific time frame, images of landscapes taken at "school", and so on. Therefore, considering the cross-data and cross-index schemes is a promising direction in the future. At the same time, the balance between storage, efficiency, and security should also be considered.

Concerning attacks, exploring attacks that rely on as few assumptions as possible is the key to improving realism. On this basis, in terms of search, potential directions are to consider transforming the problem of trapdoor recovery into an optimization problem or design attack by incorporating novel data structures and statistical theories. In terms of updates, more exploration of the impact of information leakage on data privacy is needed, especially the potential risks of backward privacy. Moreover, existing attacks are designed based on well-studied leakages, and it is meaningful to explore whether there are other information leaks during the operations of SSE schemes. Regarding defenses, whole-process security is always a direction worthy of continuous exploration and improvement. Specifically, designing and optimizing encryption algorithms and index structures is a compelling research topic.

Another exciting research direction is integration with other technologies. For example, blockchain can be used to verify the integrity of the results or to record the behaviors of the users and the cloud servers for traceability of destructive operations. Combine deep learning with conceptual graphs to improve the accuracy of semantic search.

6 CONCLUSIONS

SSE has been widely studied and has promising application prospects in many fields. However, due to the rapid development of SSE, the existing SSE surveys have been unable to conform to the current trend. This survey systematically reviews current work on search functionalities and privacy-preserving of SSE. First, we survey the work of the past few decades and classify SSE based on query expressiveness. At the same time, we summarize the methods used in the literature and illustrate their contributions to efficiency, storage space, index structures, and so on. Then we complement the gap in the privacy of SSE, which has received significant concerns in recent years, and introduce in detail the attacks and the related defenses. Finally, we discuss the open issues and challenges in existing SSE and future research directions.

REFERENCES

[1] Mohamed Ahmed Abdelraheem, Christian Gehrmann, Malin Lindström, and Christian Nordahl. 2016. Executing boolean queries on an encrypted bitmap index. In *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*. ACM, 11–22.

[2] MA Manazir Ahsan, Fahad Zaman Chowdhury, Musarat Sabilah, Ainuddin Wahid Bin Abdul Wahab, and Mohd Yamani Idna Bin Idris. 2017. An efficient fuzzy keyword matching technique for searching through encrypted cloud data. In *Proceedings of the 2017 International Conference on Research and Innovation in Information Systems*. IEEE, 1–5.

[3] Georgios Amanatidis, Alexandra Boldyreva, and Adam O'Neill. 2007. Provably-secure schemes for basic query support in outsourced databases. In *Proceedings of the International Conference on Data and Applications Security*. Springer, 14–30.

[4] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. 2016. Searchable symmetric encryption: Optimal locality in linear space via two-dimensional balanced allocations. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 1101–1114.

[5] Lucas Ballard, Seny Kamara, and Fabian Monrose. 2005. Achieving efficient conjunctive keyword searches over encrypted data. In *Proceedings of the 7th International Conference on Information and Communications Security*. Springer, 414–426.

[6] Simran Bijral and Debajyoti Mukhopadhyay. 2014. Efficient fuzzy search engine with B -tree search mechanism. In *Proceedings of the 2014 International Conference on Information Technology*. IEEE, 118–122.
[7] Alexandra Boldyreva and Nathan Chenette. 2014. Efficient fuzzy search on encrypted data. In *Proceedings of the 21st International Workshop on Fast Software Encryption*. Springer, 613–633.
[8] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. 2004. Public key encryption with keyword search. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 506–522.
[9] Dan Boneh and Brent Waters. 2013. Constrained pseudorandom functions and their applications. In *Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 280–300.
[10] Kellogg S. Booth and George S. Lueker. 1976. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences* 13, 3 (1976), 335–379.
[11] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. 2014. A survey of provably secure searchable encryption. *ACM Computing Surveys* 47, 2 (2014), 1–51.
[12] Raphael Bost. 2016. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1143–1154.
[13] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. 2017. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1465–1482.
[14] Michael Burrows and David Wheeler. 1994. A block-sorting lossless data compression algorithm. In *Proceedings of the Digital SRC Research Report*. Citeseer.
[15] Jin Wook Byun, Dong Hoon Lee, and Jongin Lim. 2006. Efficient conjunctive keyword search on encrypted data storage system. In *Proceedings of the European Public Key Infrastructure Workshop*. Springer, 184–196.
[16] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. 2013. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems* 25, 1 (2013), 222–233.
[17] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 668–679.
[18] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*. The Internet Society.
[19] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Proceedings of the 33rd Annual Cryptology Conference*. Springer, 353–373.
[20] David Cash and Stefano Tessaro. 2014. The locality of searchable symmetric encryption. In *Proceedings of the 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 351–368.
[21] Qi Chai and Guang Gong. 2012. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Proceedings of the IEEE International Conference on Communications*. IEEE, 917–922.
[22] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New constructions for forward and backward private symmetric searchable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1038–1055.
[23] Yan-Cheng Chang and Michael Mitzenmacher. 2005. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of the Applied Cryptography and Network Security*. Springer, 442–455.
[24] Melissa Chase and Emily Shen. 2015. Substring-searchable symmetric encryption. *Proc. Priv. Enhancing Technol.* 2015, 2 (2015), 263–281.
[25] Chi Chen, Xiaojie Zhu, Peisong Shen, Jiankun Hu, Song Guo, Zahir Tari, and Albert Y. Zomaya. 2015. An efficient privacy-preserving ranked keyword search method. *IEEE Transactions on Parallel and Distributed Systems* 27, 4 (2015), 951–963.
[26] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. 2019. SgxPectre: Stealing intel secrets from sgx enclaves via speculative execution. In *Proceedings of the IEEE European Symposium on Security and Privacy*. IEEE, 142–157.
[27] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. 2018. Differentially private access patterns for searchable symmetric encryption. In *Proceedings of the 2018 IEEE Conference on Computer Communications*. IEEE, 810–818.
[28] Jing Chen, Kun He, Lan Deng, Quan Yuan, Ruiying Du, Yang Xiang, and Jie Wu. 2017. EliMFS: Achieving efficient, leakage-resilient, and multi-keyword fuzzy search on encrypted cloud data. *IEEE Transactions on Services Computing* 13, 6 (2017), 1072–1085.

[29] Tianyang Chen, Peng Xu, Wei Wang, Yubo Zheng, Willy Susilo, and Hai Jin. 2021. Bestie: Very practical searchable encryption with forward and backward security. In *Proceedings of the 26th European Symposium on Research in Computer Security*. Springer, 3–23.

[30] M. Chuah and W. Hu. 2011. Privacy-aware BedTree based solution for fuzzy multi-keyword search over encrypted data. In *Proceedings of the 31st IEEE International Conference on Distributed Computing Systems Workshops*. IEEE Computer Society, 273–281.

[31] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*. ACM, 79–88.

[32] Marc Damie, Florian Hahn, and Andreas Peter. 2021. A highly accurate query-recovery attack against searchable encryption using non-indexed documents. In *Proceedings of the 30th USENIX Security Symposium*. USENIX Association, 143–160.

[33] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2020. Dynamic searchable encryption with small client storage. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium*. The Internet Society.

[34] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. 2020. SEAL: Attack mitigation for encrypted databases via adjustable leakage. In *Proceedings of the 29th USENIX Security Symposium*. USENIX Association, 2433–2450.

[35] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos N. Garofalakis. 2016. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 185–198.

[36] Ioannis Demertzis and Charalampos Papamanthou. 2017. Fast searchable encryption with tunable locality. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1053–1067.

[37] Ioannis Demertzis, Rajdeep Talapatra, and Charalampos Papamanthou. 2018. Efficient searchable encryption through compression. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1729–1741.

[38] Xiaofeng Ding, Peng Liu, and Hai Jin. 2017. Privacy-preserving multi-keyword top-k k similarity search over encrypted data. *IEEE Transactions on Dependable and Secure Computing* 16, 2 (2017), 344–357.

[39] Andrzej Ehrenfeucht, Ross M. McConnell, Nissa Osheim, and Sung-Whan Woo. 2011. Position heaps: A simple and dynamic text indexing data structure. *Journal of Discrete Algorithms* 9, 1 (2011), 100–121.

[40] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. 2015. Rich queries on encrypted data: Beyond exact matches. In *Proceedings of the 20th European Symposium on Research in Computer Security*. Springer, 123–145.

[41] Saman Feghhi and Douglas J Leith. 2016. A web traffic analysis attack using only timing information. *IEEE Transactions on Information Forensics and Security* 11, 8 (2016), 1747–1759.

[42] Bernardo Ferreira, Bernardo Portela, Tiago Oliveira, Guilherme Borges, Henrique Domingos, and João Leitão. 2022. Boolean searchable symmetric encryption with filters on trusted hardware. *IEEE Transactions on Dependable and Secure Computing* 19, 2 (2022), 1307–1319.

[43] Zhangjie Fu, Fengxiao Huang, Kui Ren, Jian Weng, and Cong Wang. 2017. Privacy-preserving smart semantic search based on conceptual graphs over encrypted outsourced data. *IEEE Transactions on Information Forensics and Security* 12, 8 (2017), 1874–1884.

[44] Zhangjie Fu, Fengxiao Huang, Xingming Sun, Athanasios V Vasilakos, and Ching-Nung Yang. 2016. Enabling semantic search based on conceptual graphs over encrypted outsourced data. *IEEE Transactions on Services Computing* 12, 5 (2016), 813–823.

[45] Zhangjie Fu, Jiangang Shu, Xingming Sun, and Daxing Zhang. 2014. Semantic keyword search based on trie over encrypted cloud data. In *Proceedings of the 2nd International Workshop on Security in Cloud Computing*. ACM, 59–62.

[46] Zhangjie Fu, Xingming Sun, Nigel Linge, and Lu Zhou. 2014. Achieving effective cloud search services: multi-keyword ranked search over encrypted cloud data supporting synonym query. *IEEE Transactions on Consumer Electronics* 60, 1 (2014), 164–172.

[47] Zhangjie Fu, Xinle Wu, Chaowen Guan, Xingming Sun, and Kui Ren. 2016. Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. *IEEE Transactions on Information Forensics and Security* 11, 12 (2016), 2706–2716.

[48] Zhangjie Fu, Xinle Wu, Qian Wang, and Kui Ren. 2017. Enabling central keyword-based semantic extension search over encrypted outsourced data. *IEEE Transactions on Information Forensics and Security* 12, 12 (2017), 2986–2997.

[49] Zhangjie Fu, Lili Xia, Xingming Sun, Alex X Liu, and Guowu Xie. 2018. Semantic-aware searching over encrypted data for cloud computing. *IEEE Transactions on Information Forensics and Security* 13, 9 (2018), 2359–2371.

[50] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. 2016. TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption. In *Proceedings of the 36th Annual International Cryptology Conference*. Springer, 563–592.

[51] Xinrui Ge, Jia Yu, Hanlin Zhang, Chengyu Hu, Zengpeng Li, Zhan Qin, and Rong Hao. 2019. Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification. *IEEE Transactions on Dependable and Secure computing* 18, 1 (2019), 490–504.

[52] Dilxat Ghopur, Jianfeng Ma, Xindi Ma, Yinbin Miao, Jialu Hao, and Tao Jiang. 2023. Puncturable ciphertext-policy attribute-based encryption scheme for efficient and flexible user revocation. *Science China Information Sciences* 66, 7 (2023), 1–17.

[53] Eu-Jin Goh. 2003. Secure indexes. *IACR Cryptol. ePrint Arch.* (2003), 216. Retrieved from http://eprint.iacr.org/2003/216

[54] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. ACM, 182–194.

[55] Philippe Golle, Jessica Staddon, and Brent R. Waters. 2004. Secure conjunctive keyword search over encrypted data. In *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security*. Springer, 31–45.

[56] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 315–331.

[57] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2019. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*. IEEE, 1067–1083.

[58] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted databases: New volume attacks against range queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 361–378.

[59] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 International Conference on Management of Data*. ACM, 216–227.

[60] Florian Hahn, Nicolas Loza, and Florian Kerschbaum. 2018. Practical and secure substring search. In *Proceedings of the 2018 International Conference on Management of Data*. Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.), ACM, 163–176.

[61] Thang Hoang, Attila A. Yavuz, F. Betül Durak, and Jorge Guajardo. 2017. Oblivious dynamic searchable encryption via distributed PIR and ORAM. *IACR Cryptol. ePrint Arch.* (2017), 1158. Retrieved 9, August, 2022 from http://eprint.iacr.org/2017/1158

[62] Susan Hohenberger, Venkata Koppula, and Brent Waters. 2015. Adaptively secure puncturable pseudorandom functions in the standard model. In *Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 79–102.

[63] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. 2012. Secure multidimensional range queries over outsourced data. *The VLDB Journal* 21, 3 (2012), 333–358.

[64] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*. The Internet Society.

[65] Xiuxiu Jiang, Jia Yu, Jingbo Yan, and Rong Hao. 2017. Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. *Inf. Sci.* 403 (2017), 22–41. DOI: https://doi.org/10.1016/J.INS.2017.03.037

[66] Seny Kamara and Tarik Moataz. 2017. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 94–124.

[67] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. 2018. Structured encryption and leakage suppression. In *Proceedings of the 38th Annual International Cryptology Conference*. Springer, 339–370.

[68] Seny Kamara and Charalampos Papamanthou. 2013. Parallel and dynamic searchable symmetric encryption. In *Proceedings of the 17th International Conference on Financial Cryptography and Data Security*. Springer, 258–274.

[69] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *Proceedings of the ACM Conference on Computer and Communications Security*. ACM, 965–976.

[70] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1329–1340.

[71] Florian Kerschbaum. 2015. Frequency-hiding order-preserving encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 656–667.

[72] Florian Kerschbaum and Axel Schröpfer. 2014. Optimal average-complexity ideal-security order-preserving encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 275–286.

[73] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. 2013. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 669–684.

[74] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. 2017. Forward secure dynamic searchable symmetric encryption with efficient updates. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1449–1463.

[75] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. 2017. Forward secure dynamic searchable symmetric encryption with efficient updates. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1449–1463.

[76] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2019. Data recovery on encrypted databases with k-nearest neighbor query leakage. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*. IEEE, 1033–1050.

[77] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2020. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. IEEE, 1223–1240.

[78] Kaoru Kurosawa and Yasuhiro Ohtaki. 2013. How to update documents verifiably in searchable symmetric encryption. In *Proceedings of the 12th International Conference on Cryptology and Network Security*. Springer, 309–328.

[79] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. 2012. Efficient similarity search over encrypted data. In *Proceedings of the IEEE 28th International Conference on Data Engineering*. IEEE Computer Society, 1156–1167.

[80] Mehmet Kuzu, Mohammad Saiful Islam, and Murat Kantarcioglu. 2014. Efficient privacy-aware search over encrypted databases. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*. ACM, 249–256.

[81] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. 2018. Improved reconstruction attacks on encrypted data using range query leakage. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 297–314.

[82] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. 2018. Result pattern hiding searchable encryption for conjunctive queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 745–762.

[83] Iraklis Leontiadis and Ming Li. 2018. Storage efficient substring searchable symmetric encryption. In *Proceedings of the 6th International Workshop on Security in Cloud Computing*. ACM, 3–13.

[84] Feng Li, Jianfeng Ma, Yinbin Miao, Qi Jiang, Ximeng Liu, and Kim-Kwang Raymond Choo. 2023. Verifiable and dynamic multi-keyword search over encrypted cloud data using bitmap. *IEEE Transactions on Cloud Computing* 11, 1 (2023), 336–348.

[85] Feng Li, Jianfeng Ma, Yinbin Miao, Zhiquan Liu, Kim-Kwang Raymond Choo, Ximeng Liu, and Robert H. Deng. 2023. Towards efficient verifiable boolean search over encrypted cloud data. *IEEE Transactions on Cloud Computing* 11, 1 (2023), 839–853.

[86] Hongwei Li, Dongxiao Liu, Yuanshun Dai, Tom H Luan, and Xuemin Sherman Shen. 2014. Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage. *IEEE Transactions on Emerging Topics in Computing* 3, 1 (2014), 127–138.

[87] Hongwei Li, Yi Yang, Tom H. Luan, Xiaohui Liang, Liang Zhou, and Xuemin Sherman Shen. 2016. Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data. *IEEE Transactions on Dependable and Secure Computing* 13, 3 (2016), 312–325.

[88] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. 2010. Fuzzy keyword search over encrypted data in cloud computing. In *Proceedings of the 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*. IEEE, 441–445.

[89] Rui Li, Alex X Liu, Ann L Wang, and Bezawada Bruhadeshwar. 2014. Fast range query processing with strong privacy protection for cloud computing. *Proceedings of the VLDB Endowment* 7, 14 (2014), 1953–1964.

[90] Xinghua Li, Qiuyun Tong, Jinwei Zhao, Yinbin Miao, Siqi Ma, Jian Weng, Jianfeng Ma, and Kim-Kwang Raymond Choo. 2022. VRFMS: Verifiable ranked fuzzy multi-keyword search over encrypted data. *IEEE Transactions on Services Computing* 16, 1 (2022), 698–710.

[91] Yingying Li, Jianfeng Ma, Yinbin Miao, Liming Liu, Ximeng Liu, and Kim-Kwang Raymond Choo. 2020. Secure and verifiable multikey image search in cloud-assisted edge computing. *IEEE Transactions on Industrial Informatics* 17, 8 (2020), 5348–5359.

[92] Yingying Li, Jianfeng Ma, Yinbin Miao, Yue Wang, Tengfei Yang, Ximeng Liu, and Kim-Kwang Raymond Choo. 2022. Traceable and controllable encrypted cloud image search in multi-user settings. *IEEE Transactions on Cloud Computing* 10, 4 (2022), 2936–2948.

[93] Helger Lipmaa. 2005. An oblivious transfer protocol with log-squared communication. In *Proceedings of the 8th International Conference on Information Security*. Springer, 314–328.

[94] Alex X. Liu and Fei Chen. 2008. Collaborative enforcement of firewall policies in virtual private networks. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing*. ACM, 95–104.

[95] Chang Liu, Liehuang Zhu, Longyijia Li, and Yu-an Tan. 2011. Fuzzy keyword search on encrypted cloud storage data with small index. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems*. IEEE, 269–273.

[96] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-an Tan. 2014. Search pattern leakage in searchable encryption: Attacks and new construction. *Inf. Sci.* 265 (2014), 176–188. DOI: https://doi.org/10.1016/J.INS.2013.11.021

[97] Qin Liu, Yu Peng, Shuyu Pei, Jie Wu, Tao Peng, and Guojun Wang. 2020. Prime inner product encoding for effective wildcard-based multi-keyword fuzzy search. *IEEE Transactions on Services Computing* 15, 4 (2020), 1799–1812.

[98] Qin Liu, Yu Peng, Jie Wu, Tian Wang, and Guojun Wang. 2020. Secure multi-keyword fuzzy searches with enhanced service quality in cloud computing. *IEEE Transactions on Network and Service Management* 18, 2 (2020), 2046–2062.

[99] Zheli Liu, Yanyu Huang, Xiangfu Song, Bo Li, Jin Li, Yali Yuan, and Changyu Dong. 2022. Eurus: Towards an efficient searchable symmetric encryption with size pattern protection. *IEEE Transactions on Dependable and Secure Computing* 19, 3 (2022), 2023–2037.

[100] Nicholas Mainardi, Alessandro Barenghi, and Gerardo Pelosi. 2019. Privacy preserving substring search protocol with polylogarithmic communication cost. In *Proceedings of the 35th Annual Computer Security Applications Conference*. ACM, 297–312.

[101] Evangelia Anna Markatou and Roberto Tamassia. 2019. Full database reconstruction with access and search pattern leakage. In *Proceedings of the 22nd International Conference*. Springer, 25–43.

[102] Charalampos Mavroforakis, Nathan Chenette, Adam O'Neill, George Kollios, and Ran Canetti. 2015. Modular order-preserving encryption, revisited. In *Proceedings of the 2015 International Conference on Management of Data*. ACM, 763–777.

[103] Yinbin Miao, Ximeng Liu, Kim-Kwang Raymond Choo, Robert H Deng, Jiguo Li, Hongwei Li, and Jianfeng Ma. 2019. Privacy-preserving attribute-based keyword search in shared multi-owner setting. *IEEE Transactions on Dependable and Secure Computing* 18, 3 (2019), 1080–1094.

[104] Yinbin Miao, Jianfeng Ma, Ximeng Liu, Jian Weng, Hongwei Li, and Hui Li. 2018. Lightweight fine-grained search over encrypted data in fog computing. *IEEE Transactions on Services Computing* 12, 5 (2018), 772–785.

[105] Yinbin Miao, Jian Weng, Ximeng Liu, Kim-Kwang Raymond Choo, Zhiquan Liu, and Hongwei Li. 2018. Enabling verifiable multiple keywords search over encrypted cloud data. *Information Sciences* 465 (2018), 21–37.

[106] Yinbin Miao, Wei Zheng, Xiaohua Jia, Ximeng Liu, Kim-Kwang Raymond Choo, and Robert Deng. 2022. Ranked keyword search over encrypted cloud data through machine learning method. *IEEE Transactions on Services Computing* 16, 1 (2022), 525–536.

[107] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Oblix: An efficient oblivious search index. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 279–296.

[108] Tarik Moataz and Erik-Oliver Blass. 2015. Oblivious substring search with updates. *IACR Cryptol. ePrint Arch.* (2015), 722. Retrieved 17, August, 2022 from http://eprint.iacr.org/2015/722

[109] Tarik Moataz, Indrajit Ray, Indrakshi Ray, Abdullatif Shikfa, Frédéric Cuppens, and Nora Cuppens. 2018. Substring search over encrypted data. *Journal of Computer Security* 26, 1 (2018), 1–30.

[110] Tarik Moataz and Abdullatif Shikfa. 2013. Boolean symmetric searchable encryption. In *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security*. ACM, 265–276.

[111] Tarik Moataz, Abdullatif Shikfa, Nora Cuppens-Boulahia, and Frédéric Cuppens. 2013. Semantic search over encrypted data. In *Proceedings of the 20st International Conference on Telecommunications*. IEEE, 1–5.

[112] Teng-Sheng Moh and Kam Ho Ho. 2014. Efficient semantic search over encrypted data in cloud computing. In *Proceedings of the International Conference on High Performance Computing & Simulation*. IEEE, 382–390.

[113] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 644–655.

[114] Sanjeet Kumar Nayak and Somanath Tripathy. 2017. SEMFS: Secure and efficient multi-keyword fuzzy search for cloud storage. In *Proceedings of the 13th International Conference on Information Systems Security*. Springer, 50–67.

[115] Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H. Deng. 2021. LEAP: Leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2307–2320.

[116] Jianting Ning, Jia Xu, Kaitai Liang, Fan Zhang, and Ee-Chien Chang. 2018. Passive attacks against searchable encryption. *IEEE Transactions on Information Forensics and Security* 14, 3 (2018), 789–802.

[117] Wakaha Ogata, Keita Koiwa, Akira Kanaoka, and Shin'ichiro Matsuo. 2013. Toward practical searchable symmetric encryption. In *Proceedings of the Advances in Information and Computer Security - 8th International Workshop on Security*. Springer, 151–167.

[118] Cengiz Örencik and Erkay Savas. 2012. Efficient and secure ranked multi-keyword search on encrypted cloud data. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. ACM, 186–195.

[119] Simon Oya and Florian Kerschbaum. 2021. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *Proceedings of the 30th USENIX Security Symposium*. USENIX Association, 127–142.

[120] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steven M. Bellovin. 2014. Blind seer: A scalable private DBMS. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 359–374.

[121] Sarvar Patel, Giuseppe Persiano, Joon Young Seo, and Kevin Yeo. 2021. Efficient boolean search over encrypted data with reduced leakage. In *Proceedings of the 27th International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 577–607.

[122] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2019. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 79–93.

[123] Sikhar Patranabis and Debdeep Mukhopadhyay. 2021. Forward and backward private conjunctive searchable symmetric encryption. In *Proceedings of the 28th Annual Network and Distributed System Security Symposium*. The Internet Society.

[124] Geong Sen Poh, Ji-Jian Chin, Wei-Chuen Yau, Kim-Kwang Raymond Choo, and Moesfa Soeheila Mohamad. 2017. Searchable symmetric encryption: Designs and challenges. *ACM Computing Surveys* 50, 3 (2017), 1–37.

[125] David Pouliot and Charles V. Wright. 2016. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1341–1352.

[126] Mariana Raykova, Binh Vo, Steven M. Bellovin, and Tal Malkin. 2009. Secure anonymous database search. In *Proceedings of the 1st ACM Cloud Computing Security Workshop*. ACM, 115–126.

[127] Eun-Kyung Ryu and Tsuyoshi Takagi. 2007. Efficient conjunctive keyword-searchable encryption. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications*. IEEE Computer Society, 409–414.

[128] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. 2021. Obfuscated access and search patterns in searchable encryption. In *Proceedings of the 28th Annual Network and Distributed System Security Symposium*. The Internet Society.

[129] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. 2000. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy, Berkeley*. IEEE Computer Society, 44–55.

[130] Qiyang Song, Zhuotao Liu, Jiahao Cao, Kun Sun, Qi Li, and Cong Wang. 2021. SAP-SSE: protecting search patterns and access patterns in searchable symmetric encryption. *IEEE Trans. Inf. Forensics Secur.* 16, (2021), 1795–1809. DOI: https://doi.org/10.1109/TIFS.2020.3042058

[131] Xiangfu Song, Changyu Dong, Dandan Yuan, Qiuliang Xu, and Minghao Zhao. 2018. Forward private searchable symmetric encryption with optimized I/O efficiency. *IEEE Transactions on Dependable and Secure Computing* 17, 5 (2018), 912–927.

[132] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2014. Practical dynamic searchable encryption with small leakage. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium*. The Internet Society, 72–75.

[133] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: An extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 299–310.

[134] Mikhail Strizhov and Indrajit Ray. 2015. Substring position search over encrypted cloud data using tree-based index. In *Proceedings of the 2015 IEEE International Conference on Cloud Engineering*. IEEE, 165–174.

[135] Shi-Feng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. 2021. Practical non-interactive searchable encryption with forward and backward privacy. In *Proceedings of the 28th Annual Network and Distributed System Security Symposium*. The Internet Society.

[136] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. 2018. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 763–780.

[137] Wenhai Sun, Xuefeng Liu, Wenjing Lou, Y. Thomas Hou, and Hui Li. 2015. Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In *Proceedings of the 2015 IEEE Conference on Computer Communications*. IEEE, 2110–2118.

[138] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y. Thomas Hou, and Hui Li. 2013. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security*. ACM, 71–82.

[139] Ashwin Swaminathan, Yinian Mao, Guan-Ming Su, Hongmei Gou, Avinash L. Varna, Shan He, Min Wu, and Douglas W. Oard. 2007. Confidentiality-preserving rank-ordered search. In *Proceedings of the 2007 ACM Workshop On Storage Security And Survivability*. ACM, 7–12.

[140] Qiuyun Tong, Yinbin Miao, Jian Weng, Ximeng Liu, Kim-Kwang Raymond Choo, and Robert Deng. 2022. Verifiable fuzzy multi-keyword search over encrypted data with adaptive security. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 5386–5399.

[141] Bing Wang, Wei Song, Wenjing Lou, and Y. Thomas Hou. 2015. Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee. In *Proceedings of the 2015 IEEE Conference on Computer Communications*. IEEE, 2092–2100.

[142] Bing Wang, Shucheng Yu, Wenjing Lou, and Y. Thomas Hou. 2014. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In *Proceedings of the 2014 IEEE Conference on Computer Communications*. IEEE, 2112–2120.

[143] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2021. DP-Sync: Hiding update patterns in secure outsourced databases with differential privacy. In *Proceedings of the International Conference on Management of Data*. ACM, 1892–1905.

[144] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. 2010. Secure ranked keyword search over encrypted cloud data. In *Proceedings of the 2010 International Conference on Distributed Computing Systems*. IEEE Computer Society, 253–262.

[145] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. 2011. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions on Parallel and Distributed Systems* 23, 8 (2011), 1467–1479.

[146] Cong Wang, Kui Ren, Shucheng Yu, and Karthik Mahendra Raje Urs. 2012. Achieving usable and privacy-assured similarity search over outsourced cloud data. In *Proceedings of the 31st IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*. IEEE, 451–459.

[147] Dongsheng Wang, Shaojing Fu, and Ming Xu. 2013. A privacy-preserving fuzzy keyword search scheme over encrypted cloud data. In *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science*. IEEE Computer Society, 663–670.

[148] Gaoli Wang, Zhenfu Cao, and Xiaolei Dong. 2021. Improved file-injection attacks on searchable encryption using finite set theory. *The Computer Journal* 64, 8 (2021), 1264–1276.

[149] Jianfeng Wang, Xiaofeng Chen, Shifeng Sun, Joseph K. Liu, Man Ho Au, and Zhi-Hui Zhan. 2018. Towards efficient verifiable conjunctive keyword search for large encrypted database. In *Proceedings of the 23rd European Symposium on Research in Computer Security*. Springer, 83–100.

[150] Jiafan Wang and Sherman S. M. Chow. 2022. Forward and backward-secure range-searchable symmetric encryption. *Proc. Priv. Enhancing Technol.* 2022, 1 (2022), 28–48. DOI: https://doi.org/10.2478/popets-2022-0003

[151] Jianfeng Wang, Hua Ma, Qiang Tang, Jin Li, Hui Zhu, Siqi Ma, and Xiaofeng Chen. 2013. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer Science and Information Systems* 10, 2 (2013), 667–684.

[152] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. 2008. An efficient scheme of common secure indices for conjunctive keyword-based retrieval on encrypted data. In *Proceedings of the 9th International Workshop on Information Security Applications*. Springer, 145–159.

[153] Peishun Wang, Huaxiong Wang, and Josef Pieprzyk. 2008. Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups. In *Proceedings of the 7th International Conference on Cryptology and Network Security*. Springer, 178–195.

[154] Xiangyu Wang, Jianfeng Ma, Ximeng Liu, Robert H. Deng, Yinbin Miao, Dan Zhu, and Zhuoran Ma. 2020. Search me in the dark: Privacy-preserving boolean range query over encrypted spatial data. In *Proceedings of the 2020 IEEE Conference on Computer Communications*. IEEE, 2253–2262.

[155] Xiangyu Wang, Jianfeng Ma, Ximeng Liu, Yinbin Miao, Yang Liu, and Robert H. Deng. 2023. Forward/backward and content private DSSE for spatial keyword queries. *IEEE Transactions on Dependable and Secure Computing* 20, 4 (2023), 3358–3370.

[156] Xingchen Wang and Yunlei Zhao. 2018. Order-revealing encryption: File-injection attack and forward security. In *Proceedings of the 23rd European Symposium on Research in Computer Security*. Springer, 101–121.

[157] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. 2014. SCORAM: Oblivious RAM for secure computation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 191–202.

[158] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. 2014. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 215–226.

[159] Yunling Wang, Shi-Feng Sun, Jianfeng Wang, Joseph K. Liu, and Xiaofeng Chen. 2022. Achieving searchable encryption scheme with search pattern hidden. *IEEE Transactions on Services Computing* 15, 2 (2022), 1012–1025.

[160] Zhiqiang Wu and Rui Li. 2023. OBI: A multi-path oblivious RAM for forward-and-backward-secure searchable encryption. In *Proceedings of the 30th Annual Network and Distributed System Security Symposium*. The Internet Society.

[161] Zhihua Xia, Xinhui Wang, Xingming Sun, and Qian Wang. 2015. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (2015), 340–352.

[162] Lei Xu, Xingliang Yuan, Cong Wang, Qian Wang, and Chungen Xu. 2019. Hardening database padding for searchable encryption. In *Proceedings of the 2019 IEEE Conference on Computer Communications*. IEEE, 2503–2511.

[163] Peng Xu, Willy Susilo, Wei Wang, Tianyang Chen, Qianhong Wu, Kaitai Liang, and Hai Jin. 2022. ROSE: robust searchable encryption with forward and backward security. *IEEE Trans. Inf. Forensics Secur.* 17 (2022), 1115–1130. DOI: https://doi.org/10.1109/TIFS.2022.3155977

[164] Hiroaki Yamamoto. 2016. Secure automata-based substring search scheme on encrypted data. In *Proceedings of the Advances in Information and Computer Security - 11th International Workshop on Security*. Springer, 111–131.

[165] Hiroaki Yamamoto, Yoshihiro Wachi, and Hiroshi Fujiwara. 2019. Space-efficient and secure substring searchable symmetric encryption using an improved DAWG. In *Proceedings of the 13th International Conference on Provable Security*. Springer, 130–148.

[166] Wenyuan Yang and Yuesheng Zhu. 2021. A verifiable semantic searching scheme by optimal matching over encrypted data in public cloud. *IEEE Trans. Inf. Forensics Secur.* 16 (2021), 100–115. DOI: https://doi.org/10.1109/TIFS.2020.3001728

[167] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. 2006. Privacy-preserving queries on encrypted data. In *Proceedings of the 11th European Symposium on Research in Computer Security*. Springer, 479–495.

[168] Fan Yin, Rongxing Lu, Yandong Zheng, Jun Shao, Xue Yang, and Xiaohu Tang. 2021. Achieve efficient position-heap-based privacy-preserving substring-of-keyword query over cloud. *Comput. Secur.* 110 (2021), 102432. DOI: https://doi.org/10.1016/J.COSE.2021.102432

[169] Jiadi Yu, Peng Lu, Yanmin Zhu, Guangtao Xue, and Minglu Li. 2013. Toward secure multikeyword top-k retrieval over encrypted cloud data. *IEEE Transactions on Dependable and Secure Computing* 10, 4 (2013), 239–250.

[170] Hua Zhang, Shaohua Zhao, Ziqing Guo, Qiaoyan Wen, Wenmin Li, and Fei Gao. 2021. Scalable fuzzy keyword ranked search over encrypted data on hybrid clouds. *IEEE Transactions on Cloud Computing* 11, 1 (2021), 308–323. DOI: https://doi.org/10.1109/TCC.2021.3092358

[171] Xianglong Zhang, Wei Wang, Peng Xu, Laurence T. Yang, and Kaitai Liang. 2023. High recovery with fewer injections: practical binary volumetric injection attacks against dynamic searchable encryption. In *32nd USENIX Security Symposium (USENIX Security 23)*, USENIX Association, Anaheim, CA, 5953–5970. Retrieved from https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-xianglong

[172] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *Proceedings of the 25th USENIX Security Symposium*. USENIX Association, 707–720.

[173] Zhenjie Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, and Divesh Srivastava. 2010. Bed-tree: An all-purpose index structure for string similarity search based on edit distance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 915–926.

[174] Zhongjun Zhang, Jianfeng Wang, Yunling Wang, Yaping Su, and Xiaofeng Chen. 2019. Towards efficient verifiable forward secure searchable symmetric encryption. In *Proceedings of the 24th European Symposium on Research in Computer Security*. Springer, 304–321.

[175] Hong Zhong, Zhanfei Li, Jie Cui, Yue Sun, and Lu Liu. 2020. Efficient dynamic multi-keyword fuzzy search over encrypted cloud data. *J. Netw. Comput. Appl.* 149 (2020). DOI: https://doi.org/10.1016/J.JNCA.2019.102469

[176] W. Zhou, L. Liu, H. Jing, C. Zhang, S. Yao, and S. Wang. 2013. K-Gram based fuzzy keyword search over encrypted cloud computing. *Journal of Software Engineering and Applications*, 6, 1 (2013), 29–32. DOI: 10.4236/jsea.2013.61004

[177] Jie Zhu, Qi Li, Cong Wang, Xingliang Yuan, Qian Wang, and Kui Ren. 2018. Enabling generic, verifiable, and secure data search in cloud services. *IEEE Transactions on Parallel and Distributed Systems* 29, 8 (2018), 1721–1735.

[178] Xiaoyu Zhu, Qin Liu, and Guojun Wang. 2016. A novel verifiable and dynamic fuzzy keyword search scheme over encrypted data in cloud computing. In *Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 845–851.

[179] Cong Zuo, James Macindoe, Siyin Yang, Ron Steinfeld, and Joseph K. Liu. 2016. Trusted boolean search on cloud using searchable symmetric encryption. In *Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 113–120.

[180] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. 2018. Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security. In *Proceedings of the 23rd European Symposium on Research in Computer Security*. Springer, 228–246.

[181] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, Josef Pieprzyk, and Lei Xu. 2022. Forward and backward private DSSE for range queries. *IEEE Transactions on Dependable and Secure Computing* 19, 1 (2022), 328–338.

[182] Ying Zuobin, SI Yuanping, MA Jianfeng, Jiang Wenjie, XU Shengmin, and Liu Ximeng. 2021. P2HBT: Partially policy hidden e-healthcare system with black-box traceability. *Chinese Journal of Electronics* 30, 2 (2021), 219–231.