Beyond Volume Pattern: Storage-Efficient Boolean Searchable Symmetric Encryption with Suppressed Leakage

Feng Li[1], Jianfeng Ma[1], Yinbin Miao[1], Pengfei Wu[2], and Xiangfu Song[2](✉)

[1] Xidian University, Xi'an 710071, China
feng.li@stu.xidian.edu.cn {jfma,ybmiao}@xidian.edu.cn
[2] School of Computing, National University of Singapore, 119391, Singapore
{wupf,songxf}@comp.nus.edu.sg

Abstract. Boolean Searchable Symmetric Encryption (BSSE) enables users to perform retrieval operations on the encrypted data while supporting complex query capabilities. This paper focuses on addressing the storage overhead and privacy concerns associated with existing BSSE schemes. While Patel *et al.* (ASIACRYPT'21) and Bag *et al.* (PETS'23) introduced BSSE schemes that conceal the number of single keyword results, both of them suffer from quadratic storage overhead and neglect the privacy of search and access patterns. Consequently, an open question arises: Can we design a storage-efficient Boolean query scheme that effectively suppresses leakage, covering not only the volume pattern for singleton keywords, but also search and access patterns?

In light of the limitations of existing schemes in terms of storage overhead and privacy protection, this work presents a novel solution called SESAME. It realizes efficient storage and privacy preserving based on Bloom filter and functional encryption. Moreover, we propose an enhanced version, SESAME+, which offers improved search performance. By rigorous security analysis on the leakage functions of our schemes, we provide a formal security proof. Finally, we implement our schemes and demonstrate that SESAME+ achieves superior search efficiency and reduced storage overhead.

Keywords: Searchable symmetric encryption · Boolean search · Volume pattern · Search pattern.

1 Introduction

Amidst the current explosive growth of data, outsourcing data to a cloud server is considered as a judicious choice for resource-constrained individuals or organizations. It provides them access to professional, efficient, reliable, and cost-effective computing and storage services, while also providing ubiquitous data accessibility. However, a crucial concern is how to effectively protect sensitive information while maintaining the utility.

Searchable Symmetric Encryption (SSE) [14,20,32] plays a vital role in secure search over encrypted data and facilitates data outsourcing by individuals and organizations. It allows users to retrieve interested documents stored on the cloud server while preserving the privacy of both queries and document contents. Thus far, the SSE research community has proposed many practical approaches, ranging from efficient and expressive query functionality [6,12,13,15] to secure searching using privacy-preserving methods capable of withstanding security threats [9,10,25,31].

One of the most attractive features of SSE functionality is Boolean query. A naive Boolean query construction can be derived from a single keyword scheme, where the user receives all single keyword results in a Boolean formula Φ and evaluates Φ locally using union and intersection operations. However, such a scheme has the worst performance in terms of efficiency and leakage. That is, it requires returning all query results for each single keyword and revealing the result sizes for all keywords. The Boolean query scheme with sub-linear search complexity was originally proposed by Cash *et al.* [12], however, it requires the Boolean formula to be in a searchable normal form $(w_1 \wedge \Phi(w_2, \cdots, w_q))$. Kamara *et al.* [21] proposed a non-interactive SSE scheme that enables the processing of arbitrary Boolean queries with worst-case sub-linear search complexity. Unfortunately, these schemes failed to consider the leakage of some sensitive information, including the disclosure of volume pattern for some keywords.

Recently, Patel *et al.* [29] made advancements regarding the security of Boolean queries by introducing a novel construction that specifically addresses the protection of the volume pattern for any singleton keywords. Bag *et al.* [6] developed a general Boolean query scheme from any conjunctive schemes. But both of them come with significant storage overhead and do not consider the potential leakage of search and access patterns.

Leakage-abuse Attacks. Numerous studies have extensively investigated leakage abuse attacks in SSE. For instance, access pattern leakage [19,27,30] or search pattern leakage [24,26,28] has been shown to enable adversaries to infer the underlying keyword based on prior knowledge. Furthermore, when equipped with knowledge of volume pattern, adversaries can even reconstruct the range query database [17,18,22]. Although these works primarily concentrate on single keyword or range queries, it is possible to apply them to Boolean queries as well.

In scenarios where a Boolean query reveals search pattern for certain keywords, adversaries can potentially employ inference attacks to recover the underlying keywords. For example, in the case of BIEX [21], the search pattern for each singleton keyword in the first clause can be exposed. Similarly, even in the case of CNFFilter [29], where tokens are constructed using keyword pairs, the access pattern could still be exploited to compromise the confidentiality of the underlying keyword pairs. Furthermore, existing attacks targeting exact or range queries can also potentially exploit the leakage of access and volume patterns to infer sensitive information or even the underlying keywords.

This naturally leads us to pose the following question: *Can we design a storage-efficient Boolean query scheme that effectively suppresses leakage, cov-*

ering not only the volume pattern for singleton keywords, but also search and access patterns?

Challenges. This paper focuses on addressing privacy concerns and storage overhead related to Boolean queries. Specifically, the proposed construction aims to prevent the leakage of the result size (*i.e.*, volume pattern) of any single keyword in a Boolean formula. For example, in the case of the Boolean formula $\Phi = (w_1 \wedge w_2) \vee (w_3 \wedge w_4)$, the volume pattern of any keyword w_i in Φ is protected. We are also concerned about the leakage of access and search patterns, which are often neglected by existing schemes, yet they pose comparable threats. Furthermore, the construction should exhibit linear growth in storage overhead instead of quadratic growth.

Solutions Overview. In order to conceal the volume pattern associated with any single keyword in a Boolean query, we have to avoid operations that reveal information about a single keyword within a Boolean query. To accomplish this, we utilize a forward index based on a vector representation. In particular, each document is encoded as a Bloom filter, encompassing all the keywords it contains. Each plaintext Boolean query is represented as a Disjunctive Normal Form (DNF). Such a DNF query consists of a disjunction of several conjunctive queries, where each conjunctive query can be represented as a Bloom filter as well. In doing this, Boolean query can be divided into several conjunctive queries where each conjunctive query can be done by computing the inner product between two Bloom filters and checking if the result is over a threshold.

To protect the forward indexes, we leverage inner product functional encryption (IPFE). An IPFE scheme enables a party, who holds a decryption key $\mathsf{sk}_{\boldsymbol{x}}$ corresponding to a vector $\boldsymbol{x}$, to decrypt a ciphertext $\mathsf{Enc}(\boldsymbol{y})$ encrypted from a vector $\boldsymbol{y}$ and learn the inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$. We use IPFE to encrypt the forward indexes and the server stores the encrypted ciphertexts of all documents. During a search, the client generates an IPFE decryption key for the Bloom filter associated with a conjunctive query and sends the decryption key to the server. Using the key, the server, for each encrypted forward index, computes the inner product by decryption and compares it with a threshold to find matches.

It is possible to use a *function hiding* IPFE to protect the query further, which reveals no information about the query $\boldsymbol{x}$ at the cost of heavy computing overhead. In our constructions, we adopt a more practical approach. Our idea is that the client can add dummy keywords when generating a Bloom filter associated with a conjunctive query. This approach not only fulfills the aforementioned security requirements but also circumvents the use of function hiding functional encryption for the inner product computation.

Our Contribution. We present a novel storage-efficient Boolean searchable symmetric encryption scheme that effectively mitigates the leakage of volume, search, and access patterns. Meanwhile, it incurs small communication and linear storage overheads. Compared with prior works, our scheme demonstrates a smaller *base query set of leakage*, which refers to the disclosure of the result set of Boolean queries, as introduced by Patel *et al.* [29]. This leakage only includes the result set of each clause within the Boolean formula, rather than keywords or keyword pairs. Specifically, our contributions can be summarized as follows:

- We propose a basic Boolean SSE scheme based on forward indexing structure of vector representation and inner product functional encryption, which restricts the *base query set of leakage* to the clauses within the Boolean formula, and improves the security by introducing dummy keywords.
- We enhance the basic scheme with optimizations. Typically, queries involve a small number of keywords, but the requirement for token length to match the index length during computations can introduce substantial computational overhead without meaningful contributions. To mitigate this issue, we employ a token pruning technique, improving efficiency by over tenfold.
- We provide a formal security analysis of our proposed scheme and substantiate its superior security compared to existing schemes that support Boolean queries. Additionally, we implement a series of experiments to empirically demonstrate the enhanced efficiency of our scheme in terms of search and storage capabilities.

1.1 Related Works

Curtmola *et al.* [14] were the first to provide a formal definition of SSE and establish *indistinguishability* and *simulation-based* security definitions in the static setting. Subsequently, Kamara *et al.* [20] extended the work of [14] by introducing the capability of efficient addition and deletion of files, commonly referred to as dynamic SSE (DSSE). To enhance the query function of SSE, Cao *et al.* [11] proposed a scheme based on TF-IDF to support multi-keyword ranking. Wang *et al.* [33] introduced multi-keyword fuzzy search that can tolerate minor typos in keywords. Fu *et al.* [16] presented a scheme to enable content-aware search by constructing conceptual graphs. Moreover, Cash *et al.* [12] designed a general Boolean query scheme with sub-linear search time complexity.

While [12] efficiently handles queries in searchable normal form, it exhibits linear time complexity for processing arbitrary Boolean queries. In response to this limitation, Kamara *et al.* [21] proposed a generic Boolean query scheme with worst-case sub-linear search. This scheme constructs a global multi-map and a dictionary as an index structure, where each multi-map maps each keyword v that co-occurs with a given keyword $w \in W$ to a tuple of $\mathsf{DB}(w) \cap \mathsf{DB}(v)$. However, this scheme inadvertently reveals the result size for each singleton keyword in the first clause of the Boolean formula. To address this vulnerability, Patel *et al.* [29] presented an improved construction with significantly reduced leakage by building indexes using any combination of two keywords as meta-keywords. Bag *et al.* [6] also employed the construction of meta-keywords to build indexes and allowed any scheme supporting conjunctive queries could be smoothly scaled to support any Boolean queries. Regrettably, these schemes entail substantial storage overhead and expose noteworthy information leakages.

The study of access pattern leakage was first initiated by Islam *et al.* [19] who proposed inference attacks for recovering the underlying keywords given prior knowledge. Pouliot *et al.* [30] presented a combinatorial optimization problem based on graph matching to attack access pattern leakage. Ning *et al.* [27] further designed attacks under different types of assumptions. Grubbs *et al.* [17] exploited the leakage of volume pattern in range queries to reconstruct the database. Gui *et al.* [18] further investigated attacks on volume pattern leakage and reduced the required prior knowledge. Kornaropoulos *et al.* [24] exploited search pattern leakage to develop value reconstruction attacks that succeeded without any knowledge about the query or data distribution. Oya *et al.* [28] proposed an attack on SSE against hidden access pattern and leaked search pattern, which successfully recovered the underlying keywords.

1.2 Organization

This paper is organized as follows. In §2, we introduce the cryptographic primitives that underpin our construction. §3 provides definitions for Boolean searchable symmetric encryption and security notions. In §4, we present the details of our constructions, SESAME and SESAME+. Security analysis and experimental analysis are presented in §5 and §6, respectively. Finally, §7 concludes this paper.

2 Preliminaries

This section presents cryptographic primitives utilized in our constructions. Table 1 summarizes commonly used symbols.

2.1 Bloom Filter

Bloom filter is a data structure used to represent a set, which is a bit vector of length l with a family of hash functions $\mathcal{H} = \{h_i \,|\, h_i : \{0,1\}^* \rightarrow [l], 1 \leq i \leq s\}$. Specifically, given a set $S = \{a_1, \cdots, a_n\}$ of elements, initialize a bit vector of length l and set all positions in the vector to 0. Use s independent hash functions h_i to map each element in the set S to the vector by setting the corresponding positions to 1. To verify if a given element a exists in the set S, compute the mapping positions of a using the s hash functions h_i. If all corresponding positions in the vector are 1, then a is possibly in the set (with some false positive probability), otherwise a is definitely not in the set. The false positive rate for an l-bit Bloom filter is approximately $(1 - e^{-\frac{sn}{l}})^s$.

2.2 Functional Encryption for Inner Product

Functional Encryption (FE) [8] extends traditional public key encryption, enabling the retrieval of partial information from ciphertexts without the need to decrypt them entirely. Specifically, by leveraging a decryption key associated with a designated function F and a ciphertext $\mathsf{Enc}(x)$, an authorized user can retrieve the value of $F(x)$ using a decryption key corresponding to F, without revealing the underlying message x itself.

Table 1: Summary of Notations

Notation	Description
λ	The computational security parameter
l	The length of a vector (*i.e.*, Bloom filter)
ind_i	The identifier of the i-th document
W_i	The list of keywords for the i-th document
α	The number of non-zero elements in the token $\boldsymbol{q}$
β	The positions of all non-zero elements in the token $\boldsymbol{q}$
w'	Dummy keyword, which satisfies $w' \notin \bigcup_{i=1}^{d} W_i$
$\mathsf{sk}_{\boldsymbol{q}}$	The decryption key of vector $\boldsymbol{q}$ for functional encryption
$\mathcal{R}$	The result set
Q	A Boolean query in the disjunctive normal form
$\boldsymbol{v}_i$	The Bloom filter (or vector) corresponding to the i-th document
$\boldsymbol{ev}_i$	The encrypted Bloom filter (or vector) corresponding to the $\boldsymbol{v}_i$
$\boldsymbol{A}$	A matrix consisting of encrypted vectors
$\boldsymbol{q}$	The search token corresponding to the conjunctive query q
$\boldsymbol{r}$	The result vector
$\boldsymbol{R}$	The result matrix
$\boldsymbol{Q}$	A matrix consisting of tokens
$\boldsymbol{U}$	A token set that has been pruned

Functional encryption for inner product [4,5] is a form of functional encryption that restricts F to the inner product operation, enabling the decryption key holder with a vector $\boldsymbol{x}$ to decrypt the ciphertext vector $\mathsf{Enc}(\boldsymbol{y})$ and obtain $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ without revealing any other information about $\boldsymbol{y}$. Next, we introduce a functional encryption for inner product based on the Decisional Diffie-Hellman (DDH) assumption, which serves as a fundamental building block in our construction. Formally, the cryptographic scheme [5] consists of four algorithms, denoted as $\mathsf{IPFE} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, formally defined as follows:

- $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^l)$: Choose a cyclic group $\mathbb{G}$ with a prime order $p > 2^\lambda$ and generate two generators $g, h \leftarrow \mathbb{G}$. Then randomly sample $s_i, t_i \leftarrow \mathbb{Z}_p$ for each $i \in \{1, \cdots, l\}$, and compute $h_i = g^{s_i} \cdot h^{t_i}$. The msk and the mpk are defined as, $\mathsf{msk} := \{(s_i, t_i)\}_{i=1}^{l}$ and $\mathsf{mpk} := (\mathbb{G}, g, h, \{h_i\}_{i=1}^{l})$, respectively.
- $\mathsf{sk}_{\boldsymbol{x}} \leftarrow \mathsf{Keygen}(\mathsf{msk}, \boldsymbol{x})$: Take the msk and the vector $\boldsymbol{x} = (x_1, \cdots, x_l)$ as input, where $x_i \in \mathbb{Z}_q$, compute the decryption key $\mathsf{sk}_{\boldsymbol{x}} = (s_{\boldsymbol{x}}, t_{\boldsymbol{x}}) = (\sum_{i=1}^{l} s_i \cdot x_i, \sum_{i=1}^{l} t_i \cdot x_i) = (\langle \boldsymbol{s}, \boldsymbol{x} \rangle, \langle \boldsymbol{t}, \boldsymbol{x} \rangle)$.
- $C_{\boldsymbol{y}} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \boldsymbol{y})$: Given the mpk and a vector $\boldsymbol{y} = (y_1, \cdots, y_l)$ as input, where $y_i \in \mathbb{Z}_q$, the algorithm randomly samples $r \leftarrow \mathbb{Z}_p$ and encrypts the vector $\boldsymbol{y}$ as $C = g^r, D = h^r, \{E_i = g^{y_i} \cdot h_i^r\}_{i=1}^{l}$. The resulting ciphertext is denoted as $C_{\boldsymbol{y}} = (C, D, \{E_i\}_{i=1}^{l})$.
- $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \leftarrow \mathsf{Decrypt}(\mathsf{mpk}, \mathsf{sk}_{\boldsymbol{x}}, C_{\boldsymbol{y}})$: Given the input of mpk, the decryption key $\mathsf{sk}_{\boldsymbol{x}}$, and the ciphertext $C_{\boldsymbol{y}}$, the algorithm proceeds to compute $E_{\boldsymbol{x}} = (\prod_{i=1}^{l} E_i^{x_i})/(C^{s_{\boldsymbol{x}}} \cdot D^{t_{\boldsymbol{x}}})$. The inner product of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ can be recovered from computing the discrete logarithm of $E_{\boldsymbol{x}}$ as regards the base g.

2.3 Pseudorandom Function

A keyed function $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Z}$ is a two-input function, where the first input is referred to as the *key*. If there exists a polynomial time algorithm that can compute $F(k, x)$ for any given $k \in \mathcal{K}$ and $x \in \mathcal{X}$, and for all probabilistic polynomial time adversaries $\mathcal{A}$ satisfy $|\Pr[\mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$, where $\mathsf{negl}(\lambda)$ is negligible in the security parameter λ, $k \xleftarrow{\$} \mathcal{K}$ and f is a random function from $\mathcal{X}$ to $\mathcal{Z}$, then it is called Pseudorandom Function (PRF).

3 Boolean Searchable Symmetric Encryption

Boolean Searchable Symmetric Encryption (BSSE) supports arbitrary Boolean queries on encrypted data. Typically, BSSE involves three entities: the Data Owner (DO), the Data User (DU)[3], and the Cloud Service Provider (CSP). The DO encrypts the database $\mathsf{DB} = \{(\mathsf{ind}_i, W_i)\}_{i=1}^{d}$ and generates the corresponding encrypted index. The CSP stores the encrypted data and index and handles query requests. The DU generates a query request and transmits it to the CSP. A generic BSSE scheme can be outlined with three algorithms:

– $(\mathsf{msk}, \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{DB})$: The Setup algorithm takes a security parameter 1^λ and a database DB as input and produces the master secret key msk as well as the encrypted database EDB, which encompasses both encrypted data and index.
– $\mathsf{tok}_Q \leftarrow \mathsf{Token}(Q, \mathsf{msk})$: The Token algorithm receives the master secret key msk and a Boolean query Q as input and generates the search token tok_Q.
– $\mathcal{R} \leftarrow \mathsf{Search}(\mathsf{tok}_Q, \mathsf{EDB})$: This algorithm takes the search token tok_Q and the encrypted database EDB as input. It performs a search on the encrypted index and retrieves the documents that satisfy the given Boolean query Q. The results are stored in the result set $\mathcal{R}$ and returned as the output.

3.1 Security Notions

We provide a security model for BSSE following the definition of Curtmola *et al.* [14]. The adversary's knowledge of leakage is defined as $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Token}}, \mathcal{L}_{\mathsf{Search}})$, where the leakage function of $\mathcal{L}_{\mathsf{Setup}}$ captures the leakage information of BSSE in the Setup stage, the leakage function of $\mathcal{L}_{\mathsf{Token}}$ captures the leakage information from the token learned by the adversary (*i.e.*, the server), and the leakage function $\mathcal{L}_{\mathsf{Search}}$ captures the leakage in the Search stage.

To formally describe the security notion of BSSE, we present a simulation-based Real-Ideal game against adversaries. In this game, $\mathcal{A}$ represents the adversary and $\mathcal{S}$ represents the simulator.

– $\mathbf{Real}_{\mathcal{A}}^{\mathrm{BSSE}}(\lambda)$: The stateful adversary $\mathcal{A}$ chooses a database DB and sends it to the challenger $\mathcal{C}$. $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda, \mathsf{DB}; \mathsf{msk}, \mathsf{EDB})$ and returns EDB to $\mathcal{A}$.

[3] The data owner and the data user can be the same entity.

Then $\mathcal{A}$ randomly selects a series of Boolean queries $\{Q_1, \cdots\}$ at once and sends them to $\mathcal{C}$. For each Q_i, $\mathcal{C}$ runs the $\mathsf{Token}(Q_i, \mathsf{msk}; \mathsf{tok}_{Q_i})$ and returns tok_{Q_i} to $\mathcal{A}$. $\mathcal{A}$ sends tok_{Q_i} to $\mathcal{C}$, who performs the $\mathsf{Search}(\mathsf{tok}_{Q_i}, \mathsf{EDB}; \mathcal{R}_i)$ and returns the result set $\mathcal{R}_i$ to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

– $\mathbf{Ideal}^{\mathrm{BSSE}}_{\mathcal{A},\mathcal{S}}(\lambda)$: The stateful adversary $\mathcal{A}$ chooses a database DB and sends it to the challenger $\mathcal{C}$. The simulator $\mathcal{S}$ runs $\mathsf{Sim}_{\mathsf{Setup}}(\mathcal{L}; \mathsf{EDB})$ based on the leakage information $\mathcal{L}$ and returns EDB to $\mathcal{A}$. Subsequently, $\mathcal{A}$ randomly selects a series of Boolean queries $\{Q_1, \cdots\}$ at once and sends them to $\mathcal{C}$, which $\mathcal{S}$ processes through $\mathsf{Sim}_{\mathsf{Token}}(\mathcal{L}; \mathsf{tok}_{Q_i})$, returning the corresponding token tok_{Q_i} to $\mathcal{A}$. After $\mathcal{A}$ forwards the token tok_{Q_i} to $\mathcal{C}$, $\mathcal{S}$ executes $\mathsf{Sim}_{\mathsf{Search}}(\mathcal{L}; \mathcal{R}_i)$ to obtain the result set $\mathcal{R}_i$, which is returned to $\mathcal{A}$. Finally, $\mathcal{A}$ produces a bit $b \in \{0, 1\}$ to complete the experiment.

The security of BSSE is defined as the advantage of $\mathcal{A}$ in distinguishing between two worlds: in the real world, $\mathcal{A}$ interacts with a real BSSE system, and in the ideal world, $\mathcal{A}$ interacts with a stateful simulator $\mathcal{S}$ that receives the same input as $\mathcal{A}$ and simulates the response of the ideal functionality. BSSE is $\mathcal{L}$-secure if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that $|\Pr[\mathbf{Real}^{\mathrm{BSSE}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\mathrm{BSSE}}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \mathsf{negl}(\lambda)$.

Definition 1. (Search Pattern) *The search pattern is a sequence over n queries $\mathcal{Q}$ that can be inferred whether two queries are the same, and is defined as $\mathit{sp}(Q_i) = \{u_i \mid (u_i, Q_i) \in \mathcal{Q}\}$.*

Definition 2. (Access Pattern) *The access pattern is a sequence over n queries $\mathcal{Q}$ that reveals the results of the queries (including the number of results, named volume pattern) and is defined as $\mathit{ap}(Q_i) = \{\mathsf{DB}(Q_i)\}$.*

4 Constructions

In this section, we present our fundamental scheme, SESAME (Storage-Efficient Boolean SeArchable SyMmetric Encryption with Suppressed Leakage)[4], as well as its enhancement SESAME+. We first introduce a construction that facilitates conjunctive queries, and subsequently extend it to support arbitrary Boolean queries. Finally, we propose an enhanced construction with improved efficiency.

4.1 Overview

Before presenting our constructions, we provide an overview of the core ideas behind them. As mentioned, the focus of this paper is to balance storage overhead and privacy protection while maintaining search efficiency and single-keyword search ability. To achieve linear storage overhead, we utilize the Bloom filter to represent the forward index structure of the document and protect the number of single keyword results. To address functional encryption leakage from queries

[4] SESAME implies a mystical code that unlocks the treasure.

and protect the search pattern, we introduce dummy keywords in the token generation process, which also safeguards the access pattern of the query. Lastly, to improve search efficiency, we prune tokens as the number of keywords in a query is usually small.

4.2 Basic Construction

Conjunctive Protocol. We start by describing our building block for conjunctive queries, which comprises a tuple of algorithms, denoted as $\Sigma_{\mathsf{Conj}} = (\mathsf{Setup}, \mathsf{Token}, \mathsf{Search})$. The formal description of Σ_{Conj} is presented in Alg. 1.
Setup. Given an input database DB, the Setup algorithm initializes and generates master public key mpk and the master secret key msk, with inputs of the security parameter λ and the predefined vector length l. Then the algorithm parses the DB as $\{(\mathsf{ind}_i, W_i)\}_{i=1}^{d}$.

Algorithm 1: Conjunctive Protocol Σ_{Conj}

```
  @ Setup($1^\lambda$, $1^l$, $\mathsf{DB}$; $\mathsf{mpk}, \mathsf{msk}, \mathsf{EDB}$)
1 Choose a cyclic group $\mathbb{G}$ with a prime order $p > 2^\lambda$ and parse DB as
    $\{(\mathsf{ind}_i, W_i)\}_{i=1}^{d}$;
2 Generate two generators $g, h \leftarrow \mathbb{G}$ and randomly sample $s_i, t_i \leftarrow \mathbb{Z}_p$ for each
    $i \in \{1, \cdots, l\}$ and $k \leftarrow \{0,1\}^\lambda$, then compute $h_i = g^{s_i} \cdot h^{t_i}$. Finally, let
    $\mathsf{msk} := (\mathsf{sk}_{\mathsf{IPFE}} = \{(s_i, t_i)\}_{i=1}^{l}, k)$ and $\mathsf{mpk} := (\mathbb{G}, g, h, \{h_i\}_{i=1}^{l})$;
3 for $i \in \{1, \cdots, d\}$ do
4     Construct a Bloom filter $\boldsymbol{v}_i$ by mapping each $kw_j$ into $\boldsymbol{v}_i$, where
        $kw_j \leftarrow F(k, w_j)$ and $w_j \in W_i$;
5     Encrypt the Bloom filter $\boldsymbol{v}_i$ by using functional encryption for inner
        product, $\boldsymbol{ev_i} = \mathsf{IPFE.Encrypt}(\mathsf{mpk}, \boldsymbol{v_i})$;
6 Combine all encrypted Bloom filters into a matrix $\boldsymbol{A} = \{\boldsymbol{ev}_1, \cdots, \boldsymbol{ev}_d\}$;
7 Define $\mathsf{EDB} = \boldsymbol{A}$, then output $(\mathsf{mpk}, \mathsf{msk}, \mathsf{EDB})$.
  @ Token($\mathsf{msk}, q$; $\boldsymbol{q}, \mathsf{sk}_{\boldsymbol{q}}, \alpha$)
1 Construct a Bloom filter $\boldsymbol{q}$ by mapping each $kw_j$ into $\boldsymbol{q}$ and count the number
    of non-zero elements $\alpha$, where $kw_j \leftarrow F(k, w_j)$ and $w_j \in q$;
2 Add an extra $kw'$ into $\boldsymbol{q}$, where $kw' \leftarrow F(k, w')$ and $w'$ is a dummy keyword;
3 Generate a key for the vector $\boldsymbol{q}$, $\mathsf{sk}_{\boldsymbol{q}} = \mathsf{IPFE.Keygen}(\mathsf{sk}_{\mathsf{IPFE}}, \boldsymbol{q})$;
4 Send $(\boldsymbol{q}, \mathsf{sk}_{\boldsymbol{q}}, \alpha)$ to the server.
  @ Search($\mathsf{EDB}, \mathsf{mpk}, \boldsymbol{q}, \mathsf{sk}_{\boldsymbol{q}}, \alpha$; $\mathcal{R}$)
1 Compute the inner product between the vector $\boldsymbol{q}$ and the matrix $\boldsymbol{A}$,
    $\boldsymbol{r} = \mathsf{IPFE.Decrypt}(\mathsf{mpk}, \mathsf{sk}_{\boldsymbol{q}}, \boldsymbol{A})$;
2 for $e_i \in \boldsymbol{r}$ do
      if $e_i \geq \alpha$ then put corresponding document identifier $\mathsf{ind}_i$ into the result
        set $\mathcal{R}$;
3 return query result set $\mathcal{R}$.
```

For each document ind_i, the algorithm initializes a Bloom filter $\boldsymbol{v}_i$ of length l by mapping each masked keyword kw_j into corresponding bit positions in the filter, and then encrypts it using functional encryption. All encrypted Bloom filters are combined into a matrix $\boldsymbol{A}$ to form the encrypted database EDB. The output of the algorithm is denoted as $(\mathsf{mpk}, \mathsf{msk}, \mathsf{EDB})$.

Token. The Token algorithm takes the master secret key msk and a conjunctive query $q = (w_1 \wedge \cdots \wedge w_q)$ as input. To compute the inner product with $\boldsymbol{ev}_i$, the length of the token needs to match that of $\boldsymbol{v}_i$. The algorithm initializes a Bloom filter of length l, maps each keyword in q to the Bloom filter using the same method as in the Setup algorithm, and records the number of non-zero elements in the Bloom filter $\boldsymbol{q}$. Essentially, the client can generate the decryption key $\mathsf{sk}_{\boldsymbol{q}}$ from the vector $\boldsymbol{q}$ and then send it to the server for retrieval.

However, it exposes the $\boldsymbol{q}$ in plaintext during the decryption of the functional encryption, which requires both the decryption key $\mathsf{sk}_{\boldsymbol{q}}$ and $\boldsymbol{q}$. Even though $\boldsymbol{q}$ is a Bloom filter that does not reveal the underlying keywords to an adversary, it still leaks the search pattern, allowing the adversary to distinguish whether the same query is repeated or not. To protect the search pattern, we incorporate the addition of random dummy keywords during token generation that do not correspond to any document. It is worth noting that the client has the flexibility to choose the dummy keyword and its quantity randomly, or has them randomly generated by the protocol when generating the token, so the protocol does not explicitly take the dummy keyword as input. Adding more dummy keywords increases query obfuscation but also raises the false positive matching probability; so, we note there is a trade-off between security and accuracy.

Search. The Search algorithm takes as input the encrypted database EDB, the master public key mpk, the search token $\boldsymbol{q}$, the decryption key $\mathsf{sk}_{\boldsymbol{q}}$ and the number of non-zero elements α. The server begins by computing the inner product between the vector $\boldsymbol{q}$ and the matrix $\boldsymbol{A}$[5], resulting in the vector $\boldsymbol{r}$. It then scans each element of $\boldsymbol{r}$ and checks whether the value exceeds or equals to a threshold α. If the condition is satisfied, the corresponding document identifier is added to the result set $\mathcal{R}$.

Boolean Protocol. We extend conjunctive construction to support arbitrary Boolean queries. We now give a description of an extended variant that supports arbitrary Boolean queries and refer to it as SESAME. Alg. 2 provides a more detailed illustration of the extended version.

Recall that any Boolean query can be written as a DNF query $Q = q_1 \vee \cdots \vee q_m$, where each $q_i = w_{i,1} \wedge \cdots \wedge w_{i,m_i}$ is a conjunction. Therefore, for any Boolean query, the client first parses it as disjunctive normal form, and then uses $\Sigma_{\mathsf{Conj}}.\mathsf{Token}$ to generate the token and the decryption key for each q_i. To obtain the resulting matrix $\boldsymbol{R}$, we treat all tokens as a matrix $\boldsymbol{Q}$ and multiply them by $\boldsymbol{A}$[6]. For each column $\boldsymbol{r}_i$ in $\boldsymbol{R}$, if any element is greater than or equal to the threshold α_j, the corresponding document ind_i is added to the result set $\mathcal{R}$.

[5] Representing all encrypted vectors as a matrix is a matter of convenience for notation purposes, and the actual computation still relies on the inner product operation of vectors.

[6] Similarly, it is represented as a matrix solely for descriptive purposes.

Algorithm 2: SESAME

```
  @ Token($\mathsf{msk}, Q$; $\boldsymbol{Q}, \mathsf{sk}_{\boldsymbol{Q}}, \boldsymbol{\alpha}$)
1 for $q_i \in Q$ do
2     Construct a Bloom filter $\boldsymbol{q}_i$ by mapping each $kw_j$ into $\boldsymbol{q}_i$ and count the
        number of non-zero elements $\alpha_i$, where $kw_j \leftarrow F(k, w_j)$ and $w_j \in q_i$;
3     Add an extra $kw'$ into $\boldsymbol{q}_i$, where $kw' \leftarrow F(k, w')$ and $w'$ is a dummy
        keyword;
4     Generate a key for the vector $\boldsymbol{q}_i$, $\mathsf{sk}_{\boldsymbol{q}_i} = \mathsf{IPFE.Keygen}(\mathsf{sk}_{\mathsf{IPFE}}, \boldsymbol{q}_i)$;
5 Define $\boldsymbol{Q} = \{\boldsymbol{q}_1, \cdots \boldsymbol{q}_m\}$, $\mathsf{sk}_{\boldsymbol{Q}} = \{\mathsf{sk}_{\boldsymbol{q}_1}, \cdots, \mathsf{sk}_{\boldsymbol{q}_m}\}$, $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_m\}$;
6 Send $(\boldsymbol{Q}, \mathsf{sk}_{\boldsymbol{Q}}, \boldsymbol{\alpha})$ to the server.
  @ Search($\mathsf{EDB}, \mathsf{mpk}, \boldsymbol{Q}, \mathsf{sk}_{\boldsymbol{Q}}, \boldsymbol{\alpha}$; $\mathcal{R}$)
1 Compute the matrix multiplication between the matrix $\boldsymbol{Q}$ and the matrix $\boldsymbol{A}$,
    $\boldsymbol{R} = \mathsf{IPFE.Decrypt}(\mathsf{mpk}, \mathsf{sk}_{\boldsymbol{Q}}, \boldsymbol{A})$;
2 for $\boldsymbol{r}_i \in \boldsymbol{R}$ do
      if $\exists e_j \in \boldsymbol{r}_i, e_j \geq \alpha_j$ then put document identifier $\mathsf{ind}_i$ into the result
        set $\mathcal{R}$;
3 return query result set $\mathcal{R}$.
```

4.3 Enhanced Construction

We observe that the number of queried keywords is typically much smaller than the total number of keywords in the universal keyword set, *i.e.*, $|q_i| \ll |\cup_{i=1}^{d} W_i|$. Consequently, a significant amount of unnecessary computational overhead arises since the 0 elements in the query vector are meaningless for the computation. Therefore, the primary objective of the SESAME+ enhancement construction is to improve query efficiency. The extended version of the proposed scheme is illustrated in more detail in Alg. 3, which depicts the various components and operations involved in supporting arbitrary Boolean queries.

To improve the efficiency of the scheme, SESAME+ eliminates all the 0 elements in the vector $\boldsymbol{q}_i$, which is a straightforward yet effective approach. However, directly removing the 0 elements from the vector $\boldsymbol{q}_i$ would render encryption and decryption infeasible. SESAME+ makes changes to the Token and Search algorithms, where the setup phase remains the same as that of SESAME. In the Token algorithm, it is necessary to record the number of non-zero elements, denoted as α_i, for clause $\boldsymbol{q}_i$ before adding the dummy keyword w' and the position set β_i of the non-zero elements for the modified query $\boldsymbol{q}'_i = \boldsymbol{q}_i \wedge w'$ after adding the dummy keyword (line 3, Alg. 3); suppose there are α'_i non-zero elements after adding the dummy keyword and $\alpha'_i \geq \alpha_i$, then the server doesn't know which α_i non-zero elements out of α'_i are introduced by non-dummy keywords, hence reducing leakage. Then, the 0 elements in $\boldsymbol{q}_i$ are removed to obtain the pruned vector $\boldsymbol{u}_i$ and generate the corresponding decryption key. Similarly, in the Search algorithm, the server needs to prune $\boldsymbol{A}$ according to the received β_i to obtain the matrix $\boldsymbol{A}'$ corresponding to $\boldsymbol{u}_i$. The inner product $\boldsymbol{r}_i$ from $\boldsymbol{u}_i$ and $\boldsymbol{A}'$ is then computed to filter the documents that satisfy the query condition.

Algorithm 3: SESAME+

```
  @ Token($\mathsf{msk}, Q; \boldsymbol{U}, \mathsf{sk}_{\boldsymbol{U}}, \boldsymbol{\alpha}, \boldsymbol{\beta}$)
1 for $q_i \in Q$ do
2     Construct a Bloom filter $\boldsymbol{q}_i$ by mapping each $kw_j$ into $\boldsymbol{q}_i$ and count the
        number of non-zero elements $\alpha_i$, where $kw_j \leftarrow F(k, w_j)$ and $w_j \in q_i$;
3     Add an extra $kw'$ into $\boldsymbol{q}_i$, where $kw' \leftarrow F(k, w')$ and $w'$ is a dummy
        keyword, record the positions of all non-zero elements in $\boldsymbol{q}_i$, denoted as
        $\beta_i$, and then remove the 0$s$ in $\boldsymbol{q}_i$ to get a new vector $\boldsymbol{u}_i$ with all 1$s$;
4     Generate a key for the vector $\boldsymbol{u}_i$,
        $\mathsf{sk}_{\boldsymbol{u}_i} = (s_{\boldsymbol{u}_i}, t_{\boldsymbol{u}_i}) = (\Sigma_{j=1}^{|\boldsymbol{u}_i|} s_{\beta_{i,j}} \cdot \boldsymbol{u}_{i,j}, \Sigma_{j=1}^{|\boldsymbol{u}_i|} t_{\beta_{i,j}} \cdot \boldsymbol{u}_{i,j})$;
5 Define $\boldsymbol{U} = \{\boldsymbol{u}_1, \cdots \boldsymbol{u}_m\}$, $\mathsf{sk}_{\boldsymbol{U}} = \{\mathsf{sk}_{\boldsymbol{u}_1}, \cdots, \mathsf{sk}_{\boldsymbol{u}_m}\}$, $\boldsymbol{\alpha} = \{\alpha_1, \cdots, \alpha_m\}$,
    $\boldsymbol{\beta} = \{\beta_1, \cdots, \beta_m\}$;
6 Send $(\boldsymbol{U}, \mathsf{sk}_{\boldsymbol{U}}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ to the server.
  @ Search($\mathsf{EDB}, \mathsf{mpk}, \boldsymbol{U}, \mathsf{sk}_{\boldsymbol{U}}, \boldsymbol{\alpha}, \boldsymbol{\beta}; \mathcal{R}$)
1 for $\boldsymbol{u}_i \in \boldsymbol{U}$ do
2     Select the corresponding rows from matrix $\boldsymbol{A}$ according to $\beta_i$ to form a
        new matrix $\boldsymbol{A}'$;
3     Compute the inner product between the vector $\boldsymbol{u}_i$ and the matrix $\boldsymbol{A}'$,
        $\boldsymbol{r}_i = \mathsf{IPFE.Decrypt}(\mathsf{mpk}, \mathsf{sk}_{\boldsymbol{u}_i}, \boldsymbol{A}')$;
4     for $e_j \in \boldsymbol{r}_i$ do
          if $e_j \geq \alpha_i$ then put corresponding document identifier $\mathsf{ind}_j$ into the
            result set $\mathcal{R}$;
5 return query result set $\mathcal{R}$.
```

5 Security Analysis

We overview the security of our enhanced construction SESAME+. We only provide the security of SESAME+ since all optimizations in SESAME+ do not downgrade the security of SESAME. We first present an informal discussion of the leakage functions, and then show the security of SESAME+ in Theorem 1, with the proof from Appendix A.

The Setup protocol securely encrypts the input database DB and subsequently outsources it to the server for storage. As the adversary only has access to the stored data, the leakage function $\mathcal{L}_{\mathsf{Setup}}$ is defined as $\mathcal{L}_{\mathsf{Setup}} = (d, l)$, where d denotes the number of vectors and l denotes the length of each vector.

For the Token protocol, the input Boolean query Q is converted into a token that can be computed on the encrypted database and sent to the server. Hence, the adversary's view includes $\boldsymbol{U}$, $\mathsf{sk}_{\boldsymbol{U}}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$. However, since $\boldsymbol{\beta}$ reveals the positions of all non-zero elements and the number of Boolean query clauses, the leakage function can be defined as $\mathcal{L}_{\mathsf{Token}} = (m, \boldsymbol{\alpha}, \boldsymbol{\beta})$, where m represents the number of clauses in the query Q.

For Search protocol, the server computes the inner product between the token and the encrypted database to determine whether a document satisfies the search criteria, enabling the server to learn this information. The leakage function can be defined as $\mathcal{L}_{\mathsf{Search}} = (\{\boldsymbol{r}_1, \cdots, \boldsymbol{r}_m\}, \mathcal{R})$, which contains the inner-product result $\boldsymbol{r}_i$ for the i-th clause and the final search result $\mathcal{R}$.

Theorem 1. SESAME+ *is an $\mathcal{L}$-secure Boolean Searchable Symmetric Encryption scheme with non-adaptive security*[7] *that supports arbitrary Boolean queries, if the inner-product functional encryption is secure.*

6 Experimental Evaluation

In this section, we report the implementation and performance of SESAME and SESAME+. We evaluate the performance on real-world data set and compare the storage overhead and search efficiency of SESAME+ with those of $\mathsf{TWINSSE}_{\mathsf{OXT}}$[8] [6] in conjunctive normal form. Furthermore, we evaluate the storage overhead of CNFFilter [29] and present an efficiency comparison with $\mathsf{TWINSSE}_{\mathsf{OXT}}$ (in CNF and DNF form) and SESAME+.

Data Set and Platform. We utilize the Enron email data set [1], comprising a total of 515,705 documents (emails). To ensure a more enriched and meaningful set of keywords in each document, we chose 17,006 documents that are greater than 10KB in size. The experiments are conducted using Python3 on a system running macOS Monterey 12.4 with an Intel Core i7 2.9 GHz CPU.

Implementation Details. We extract 500 keywords from the Enron dataset with a total of 2,553,585 document-keyword pairs. For cryptographic primitives, we implement PRF and encryption using HMAC and AES algorithms, respectively, as provided by the Crypto library [2]. In our implementation, we set the prime order p of functional encryption to 256 bits. In the implementation of scheme $\mathsf{TWINSSE}_{\mathsf{OXT}}$, we use the Pairing-Based Cryptography Library [3] and set both *qbits* and *rbits* to 256. Additionally, we set the bucket size to 10, which is consistent with the configuration used in [6]. For CNFFilter, we take the first 8 bytes for the output of PRFs, which is the same as the setting in [29].

6.1 Evaluation of Our Constructions

We present the performance evaluation of both our basic construction, SESAME, and its enhanced version, SESAME+, in terms of search efficiency and accuracy. This evaluation includes various configurations of Bloom filters, where we vary the filter length and the number of hash functions. The results are summarized in Table 2. In the experimental setting, we consider Boolean queries of the form $D_1 \vee D_2$, where D_i represents a conjunction of three labels.

Within our proposed schemes, alongside the documents that satisfy the query, the query results also encompass certain erroneous documents, which are evaluated using accuracy as a metric. The occurrence of errors can be attributed to

[7] Adaptive security denotes that the adversary can issue queries depending on previous queries, whereas non-adaptive security means that the adversary must prepare all the queries at the beginning of the BSSE security game.

[8] In this paper, unless explicitly specified, $\mathsf{TWINSSE}_{\mathsf{OXT}}$ is used to represent a scheme specifically designed for processing Boolean queries in CNF form.

Table 2: Performance Comparison: Search Efficiency and Accuracy

	1200				1800				2400			
	SESAME		SESAME+		SESAME		SESAME+		SESAME		SESAME+	
	time	acc	time	acc	time	acc	time	acc	time	acc	time	acc
2	78.678	0.942	3.256	0.942	115.477	0.947	3.290	0.947	156.806	0.989	3.326	0.989
3	79.770	0.739	4.157	0.739	118.784	0.957	4.269	0.957	160.209	0.827	4.328	0.827
4	77.025	0.587	5.000	0.587	118.532	0.878	5.105	0.878	163.714	0.844	5.273	0.844
5	73.810	0.503	5.679	0.503	119.807	0.827	6.053	0.827	162.508	0.807	6.174	0.807

[1] Search time is measured in seconds and "acc" stands for "accuracy".
[2] The leftmost column corresponds to the number of hash functions, while the top row denotes the length of the Bloom filter.

two factors. Firstly, the utilization of the Bloom filter as an indexing mechanism inherently introduces errors, which can be adjusted through parameter modifications. Secondly, to protect access and search patterns, we have incorporated a dummy keyword, which simultaneously increases the false positive rate.

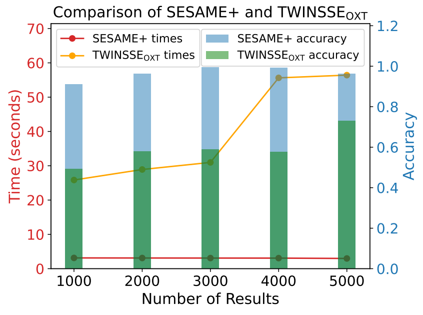
Based on the empirical findings presented in Table 2, we observe that when the length of the Bloom filter remains constant, the accuracy of the query results decreases with an increasing number of hash functions. On the other hand, increasing the length of the Bloom filter improves the performance of our constructions. Therefore, our proposed constructions allow for parameter adjustments within the Bloom filter to achieve the desired level of accuracy.

Through a comparative analysis of SESAME and SESAME+, notable distinctions emerge. SESAME+ demonstrates a search time that is at least ten times faster than that of SESAME and is unaffected by the length of the Bloom filter. In contrast, the search time of SESAME escalates with the expansion of the Bloom filter's length. The discrepancy arises from SESAME+ selectively computing relevant vector elements, ignoring nonsensical ones, compared to SESAME that calculates the entire vector regardless of element relevance. This enhancement in our construction leads to a substantial improvement in search efficiency.

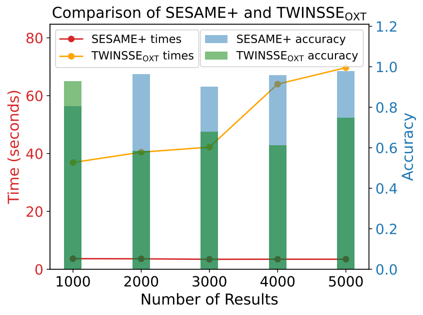
6.2 Performance Comparison

We evaluate and compare the search and accuracy performance of SESAME+ and $\mathsf{TWINSSE}_{\mathsf{OXT}}$ by varying queries. Each query is composed of two clauses, with each clause containing 2 or 3 keywords, as depicted in Fig. 1(a) and Fig. 1(b), respectively. The resulting size is varied by carefully selecting the keywords. For our implementation, we employ a Bloom filter with a length of 2400 bits and a hash family with two hash functions.

Our proposed construction, SESAME+, exhibits superior performance compared to $\mathsf{TWINSSE}_{\mathsf{OXT}}$ in terms of both search efficiency and result accuracy. In terms of search efficiency, SESAME+ achieves a more than tenfold improvement, which remains consistent regardless of the number of results or changes in query formulas. This is attributed to the linear search nature of our construction, where the number of elements involved in the computation is typically



(a) Each clause with two keywords



(b) Each clause with three keywords

Fig. 1: Search Efficiency and Accuracy Performance

small. In contrast, $\mathsf{TWINSSE_{OXT}}$ utilizes meta-keywords that often contain numerous elements, requiring individual verification and resulting in search times that fluctuate with the number of results or changes in query formula. Our construction also outperforms in result accuracy, as our scheme allows for enhanced accuracy by adjusting the parameters of the Bloom filter. On the other hand, $\mathsf{TWINSSE_{OXT}}$ introduces errors through meta-keywords, which are query-dependent and consequently limit improvements in accuracy across all queries. Fig. 2 illustrates the comparison of our scheme with $\mathsf{TWINSSE_{OXT}}$ and CNFFilter in terms of search time. CNFFilter achieves faster search efficiency at the expense of storage overhead and information leakage.

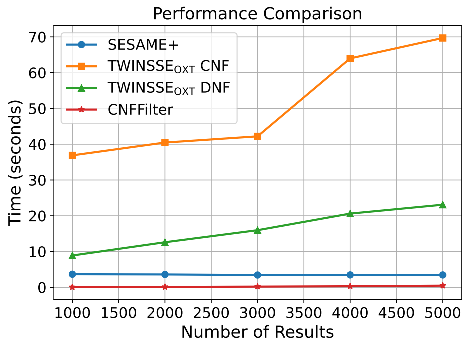


Fig. 2: Efficiency Comparison

In addition, we conduct a comparison of storage overhead and token size, as illustrated in Table 3. The storage overhead is determined by serializing the encrypted database using the *pickle* library, while the token size is computed using the `getsizeof()` function from the *sys* library. It is important to note that the token size solely captures the information transmitted from the client to the server and does not account for any information returned by the server.

From the comparison, we observe that SESAME+ demonstrates significantly lower storage overhead compared to other constructions. This is attributed to the linear relationship between the storage overhead of SESAME+ and the number of documents, which is independent of the number of keywords present in each document. In contrast, the storage overhead of the other two constructions is influenced by the number of document-key pairs, and the generation of meta-keywords results in an expansion of storage space.

Table 3: Performance Comparison: Storage Overhead and Token Size

	Storage Size	Token Size (KB)				
	(GB)	1000	2000	3000	4000	5000
SESAME+	1.55	416	416	416	416	416
$\mathsf{TWINSSE_{OXT}}$	6.69	147816	295264	295264	295264	295264
CNFFilter	23.0	208	208	208	208	208

Furthermore, we observe SESAME+ shows a smaller token size due to its linear relationship with the number of keywords in the Boolean formula. In contrast, the token size of CNFFilter is quadratically related to the number of keywords in the Boolean formula, as it necessitates the generation of double tag seeds. The search protocol of $\mathsf{TWINSSE_{OXT}}$ involves two rounds of interaction, resulting in a token size that is influenced not only by the number of keywords in the Boolean formula but also by the number of results in the first clause.

7 Conclusion

This paper further advanced the design of Boolean Searchable Symmetric Encryption (BSSE) schemes with a focus on reducing leakage and improving storage efficiency. Our proposed scheme, SESAME+, addresses the issue of volume leakage and provides enhanced protection for search and access pattern leakage that previous works have overlooked. Regarding storage overhead, SESAME+ demonstrates superiority over existing schemes, offering a more efficient solution. Additionally, the token size in SESAME+ exhibits a linear relationship with the number of keywords in the Boolean formula. These results highlight the effectiveness of our approach in achieving improved security and efficiency in BSSE schemes. However, our current constructions do not consider dynamic aspects, nor forward privacy or backward privacy. The efficiency of these constructions is primarily influenced by the functional encryption for inner product primitive and linear search time. We leave the design of a BSSE scheme with the same merits and properties meanwhile enhancing its functionality and efficiency, as our future work.

Acknowledgement. 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.

References

1. Enron Email Dataset, https://www.cs.cmu.edu/~enron/. Last modified 8 May 2015
2. The PyCryptodome, https://pycryptodome.readthedocs.io/en/latest/index.html.
3. The Pairing-Based Cryptography Library, https://crypto.stanford.edu/pbc/.
4. Abdalla, M., Bourse, F., Caro, A.D., Pointcheval, D: Simple Functional Encryption Schemes for Inner Products. In: Katz, J. (eds.) PKC 2015, LNCS, vol. 9020, pp. 733–751. Springer (2015). https://doi.org/10.1007/978-3-662-46447-2_33
5. Agrawal, S., Libert, B., Stehlé, D.: Fully Secure Functional Encryption for Inner Products, from Standard Assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, LNCS, vol. 9816, pp. 333–3362. Springer (2016). https://doi.org/10.1007/978-3-662-53015-3_12
6. Bag, A., Talapatra, D., Rastogi, A., Patranabis, S., Mukhopadhyay, D.: TWo-IN-one-SSE: Fast, Scalable and Storage-Efficient Searchable Symmetric Encryption for Conjunctive and Disjunctive Boolean Queries. Proc. Priv. Enhancing Technol. **2023**(1), 115–139 (2023)
7. Bishop, A., Jain, A., Kowalczyk, L.: Function-Hiding Inner Product Encryption. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, LNCS, vol. 9452, pp. 470–491. Springer (2015). https://doi.org/10.1007/978-3-662-48797-6_20
8. Boneh, D., Sahai, A., Waters, B.: Functional Encryption: Definitions and Challenges. In: Ishai, Y. (eds.) TCC 2011, LNCS, vol. 6597, pp. 253–273. Springer (2011). https://doi.org/10.1007/978-3-642-19571-6_16
9. Bost, R.: $\sum o\varphi o\varsigma$: Forward Secure Searchable Encryption. In: 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 1143–1154. ACM (2016)
10. Bost, R., Minaud, B., et al.: Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. In: 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, pp. 1465–1482. ACM (2017)
11. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. IEEE Trans. Parallel Distributed Syst. **25**(1), 222–233 (2014)
12. Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.-C., Steiner, M.: Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, LNCS, vol. 8042, pp. 353–373. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_20
13. Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.-C., Steiner, M.: Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In: 21st Annual Network and Distributed System Security Symposium, NDSS 2014. The Internet Society (2014)
14. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: 2006 ACM Conference on Computer and Communications Security, CCS 2006, pp. 79–88. ACM (2006)
15. Demertzis, I., Papadopoulos, D., Papamanthou, C.: Searchable Encryption with Optimal Locality: Achieving Sublogarithmic Read Efficiency. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, LNCS, vol. 10991, pp. 371–406. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_13
16. Fu, Z., Huang, F., Ren, K., Weng, J., Wang, C.: Privacy-Preserving Smart Semantic Search Based on Conceptual Graphs Over Encrypted Outsourced Data. IEEE Trans. Inf. Forensics Secur. **12**(8), 1874–1884 (2017)

17. Grubbs, P., Lacharité, M.-S., Minaud, B., Paterson, K.G.: Pump up the Volume: Practical Database Reconstruction from Volume Leakage on Range Queries. In: 2018 ACM Conference on Computer and Communications Security, CCS 2018, pp. 315–331. ACM (2018)
18. Gui, Z., Johnson, O., Warinschi, B.: Encrypted Databases: New Volume Attacks against Range Queries. In: 2019 ACM Conference on Computer and Communications Security, CCS 2019, pp. 361–378. ACM (2019)
19. Islam, M.S., Kuzu, M., et al.: Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In: 19th Annual Network and Distributed System Security Symposium, NDSS 2012, pp. 12. The Internet Society (2012)
20. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 965–976. ACM (2012)
21. Kamara, S., Moataz, T.: Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017, LNCS, vol. 10212, pp. 94–124. Springer (2017). https://doi.org/10.1007/978-3-319-56617-7_4
22. Kellaris, G., Kollios, G., Nissim, K., O'Neill, A.: Generic Attacks on Secure Outsourced Databases. In: 2016 ACM Conference on Computer and Communications Security, CCS 2016, pp. 1329–1340. ACM (2016)
23. Kim, S., Lewi, K., et al.: Function-Hiding Inner Product Encryption Is Practical. In: Catalano, D., Prisco, R.D. (eds.) SCN 2018, LNCS, vol. 11035, pp. 544–562. Springer (2018). https://doi.org/10.1007/978-3-319-98113-0_29
24. Kornaropoulos, E.M., Papamanthou, C., Tamassia, R.: The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In: 2020 IEEE Symposium on Security and Privacy, S&P 2020, pp. 1223–1240. IEEE (2020)
25. Lai, S., Patranabis, S., et al.: Result Pattern Hiding Searchable Encryption for Conjunctive Queries. In: 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, pp. 745–762. ACM (2018)
26. Liu, C., Zhu, L., Wang, M., Tan, Y.a.: Search pattern leakage in searchable encryption: Attacks and new construction. Inf. Sci. **265**, 176–188 (2014)
27. Ning, J., Xu, J., Liang, K., Zhang, F., Chang, E.-C.: Passive Attacks Against Searchable Encryption. IEEE Trans. Inf. Forensics Secur. **14**(3), 789–802 (2019)
28. Oya, S., Kerschbaum, F.: Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In: 30th USENIX Security Symposium, USENIX Security 2021, pp. 127–142. USENIX Association (2021)
29. Patel, S., Persiano, G., et al.: Efficient Boolean Search over Encrypted Data with Reduced Leakage. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, LNCS, vol. 13092, pp. 577–607. Springer (2021). https://doi.org/10.1007/978-3-030-92078-4_20
30. Pouliot, D., Wright, C.V.: The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption. In: 2016 ACM Conference on Computer and Communications Security, CCS 2016, pp. 1341–1352. ACM (2016)
31. Shang, Z., Oya, S., Peter, A., Kerschbaum, F.: Obfuscated Access and Search Patterns in Searchable Encryption. In: 28th Annual Network and Distributed System Security Symposium, NDSS 2021. The Internet Society (2021)
32. Song, D.X., Wagner, D.A., Perring A.: Practical Techniques for Searches on Encrypted Data. In: 2000 IEEE Symposium on Security and Privacy, S&P 2000, pp. 44–55. IEEE Computer Society (2000)
33. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: 2014 IEEE Conference on Computer Communications, INFOCOM 2014, pp. 2112–2120. IEEE (2014)

Appendix A Proof of Theorem 1

We provide a formal security proof of our construction SESAME+. We consider a database DB and a sequence of DNF queries $\mathcal{Q} = \{Q_1, \cdots, Q_n\}$, where $Q_i = q_{i,1} \vee \cdots \vee q_{i,m}$ consists of m conjunctions.

The leakage function $\mathcal{L}_{\mathsf{Setup}}$ captures information that is leaked from the Setup algorithm. In our construction, we use Bloom filters to represent documents and encrypt them using functional encryption. As the adversary is restricted to access only the encrypted vectors, the acquired information is confined to the total number of encrypted vectors and their respective lengths, represented as d and l, respectively. Hence, the Setup leakage function is defined as $\mathcal{L}_{\mathsf{Setup}} = (d, l)$.

The leakage function $\mathcal{L}_{\mathsf{Token}}$ is a summary of the information that an adversary can acquire in the context of the Token algorithm. It is noteworthy that both the vector $\boldsymbol{\alpha}$, which records the number of non-zero elements, and the vector $\boldsymbol{\beta}$, which records the positions of non-zero elements, are sent to the server as auxiliary query information, thereby making them susceptible to the adversary. Additionally, $\boldsymbol{U}$ can be derived from $\boldsymbol{\beta}$, which means that it is not part of $\mathcal{L}_{\mathsf{Token}}$. Furthermore, $\boldsymbol{\beta}$ discloses the number of clauses in the query Q as m. Consequently, the Token leakage function is represented as $\mathcal{L}_{\mathsf{Token}} = (m, \boldsymbol{\alpha}, \boldsymbol{\beta})$.

Regarding the information that is leaked in the Search algorithm, it is important to note that the output from the Token is received by the server, and this output has already been included in the $\mathcal{L}_{\mathsf{Token}}$. During query execution, the server prunes the matrix $\boldsymbol{A}$ based on β_i to derive $\boldsymbol{A}'$ for each clause in Q, where $\boldsymbol{A}$ represents the ciphertext vectors encrypted by functional encryption generated in the Setup, and its security is guaranteed by functional encryption. Subsequently, the server decrypts $\boldsymbol{A}'$ to obtain the inner product result $\boldsymbol{r}_i$, which can be acquired by the adversary. Additionally, the server discloses the query's result set $\mathcal{R}$, which constitutes information accessible to the adversary. Therefore, the Search leakage function is defined as $\mathcal{L}_{\mathsf{Search}} = (\{\boldsymbol{r}_1, \cdots, \boldsymbol{r}_m\}, \mathcal{R})$.

Proof. To demonstrate that $\mathbf{Real}_{\mathcal{A}}^{\mathsf{SESAME+}}(\lambda)$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathsf{SESAME+}}(\lambda)$ are computationally indistinguishable, we characterize a probabilistic polynomial-time simulator $\mathcal{S}$ capable of simulating the three protocols in our SESAME+ scheme. The simulator $\mathcal{S}$ must be able to regenerate the encrypted database and tokens from the leakage information $\mathcal{L}$, with the regenerated tokens satisfying the the dependencies among the leakage functions $\mathcal{L}_{\mathsf{Setup}}$, $\mathcal{L}_{\mathsf{Token}}$, and $\mathcal{L}_{\mathsf{Search}}$, in order to prevent the adversary $\mathcal{A}$ from distinguishing between the real world and ideal world scenarios. The adversary $\mathcal{A}$ has access to the simulated encrypted database and can retrieve data using the simulated tokens.

Provided the leakage information $\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Token}}, \mathcal{L}_{\mathsf{Search}})$, the simulations can be formulated as follows:

To simulate the Setup protocol, $\mathcal{S}$ selects a cyclic group $\mathbb{G}$ of prime order $p > 2^{\lambda}$. Then, $\mathcal{S}$ randomly samples $s_i, t_i \leftarrow \mathbb{Z}_p$ for each $i \in \{1, \cdots, l\}$, where l is determined by $\mathcal{L}_{\mathsf{Setup}}$, randomly samples $k \leftarrow \{0,1\}^{\lambda}$ and computes $h_i = g^{s_i} \cdot h^{t_i}$, where g and h are two randomly generated generators in $\mathbb{G}$. As a result, $\mathcal{S}$

simulates the master secret key and master public key as $\mathsf{msk} := (\mathsf{sk}_{\mathsf{IPFE}} = \{(s_i, t_i)\}_{i=1}^{l}, k)$ and $\mathsf{mpk} := (\mathbb{G}, g, h, \{h_i\}_{i=1}^{l})$, respectively.

For simulating the EDB, $\mathcal{S}$ generates d Bloom filters $\boldsymbol{v}_i$ of length l. These vectors are constructed to maintain dependencies with the leakage functions $\mathcal{L}_{\mathsf{Token}}$ and $\mathcal{L}_{\mathsf{Search}}$, ensuring that the adversary's verification using simulated tokens remains valid. The adversary can only learn the length l of the vectors and the number of vectors d, as they only have access to the encrypted vectors. Finally, the simulator $\mathcal{S}$ employs functional encryption for inner product with the mpk to encrypt the vectors and simulate the encrypted database EDB.

In the context of the Setup protocol, given the leakage information $\mathcal{L}$, the simulator $\mathcal{S}$ generates simulated outputs, including the encrypted database EDB, the master public key mpk, and the master secret key msk. The difference between the simulated EDB and the real-world scenario lies in the selection of $\boldsymbol{v}_i$. Instead of obtaining $\boldsymbol{v}_i$ based on the document mapping, $\mathcal{S}$ selects $\boldsymbol{v}_i$ using the leakage functions $\mathcal{L}_{\mathsf{Token}}$ and $\mathcal{L}_{\mathsf{Search}}$, followed by its encryption. The advantage of distinguishing them is negligible if functional encryption is fully secure. The simulations of the mpk and msk are equivalent with those of the real world.

In the simulation of the Token protocol, $\mathcal{S}$ simulates tokens for Boolean queries based on the leakage function $\mathcal{L}_{\mathsf{Token}}$ and ensures that these tokens can operate on the simulated encrypted database EDB. The leakage information provided by $\mathcal{L}_{\mathsf{Token}}$ reveals the positions of non-zero elements in the vector for each Boolean query clause, as well as the number of clauses for each Boolean query. Consequently, $\mathcal{S}$ can generate tokens that are identical to those in the real experiment. For simulating the decryption key, $\mathcal{S}$ leverages the leaked positions information to simulate the decryption key using the Keygen algorithm of functional encryption. The advantage of $\mathcal{A}$ in distinguishing between the real world and the ideal world becomes negligible if the functional encryption is secure.

When simulating the Search protocol, $\mathcal{S}$ retrieves documents from the encrypted database EDB based on a given Boolean query. Upon receiving the simulated token tok, $\mathcal{S}$ prunes the simulated EDB according to the corresponding $\boldsymbol{\beta}$, then performs the decryption process on the pruned EDB to obtain the identifiers of documents that satisfy the query. Since both EDB and tok are simulated based on the leakage function $\mathcal{L}$, the search process performed on the simulated token leaks the same information as $\mathcal{L}_{\mathsf{Search}}$. Consequently, $\mathcal{A}$ cannot distinguish between the real world and the ideal world with more than negligible probability.

In the above proof, we describe a probabilistic polynomial-time simulator $\mathcal{S}$ that simulates the real experiment by using a given leakage information from $\mathcal{L}$. Assuming that functional encryption for inner product is secure, then our scheme $\mathsf{SESAME+}$ achieves $\mathcal{L}$-secure, that is

$$|\Pr[\mathbf{Real}_{\mathcal{A}}^{\mathsf{SESAME+}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathsf{SESAME+}}(\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

Remark. Due to subtle issues from the underlying inner product functional encryption, we prove $\mathsf{SESAME+}$ with *non-adaptive* security, *i.e.*, the adversary issues all queries before running the game. Designing an adaptively secure BSSE scheme with similar properties as $\mathsf{SESAME+}$ seems to require fundamentally different primitives and proof techniques, for which we leave as a future work.