Multi-Client Secure and Efficient DPF-Based Keyword Search for Cloud Storage

Cheng Huang, *Member, IEEE*, Dongxiao Liu, *Member, IEEE*, Anjia Yang, *Member, IEEE*, Rongxing Lu, *Fellow, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—In this paper, we propose a multi-client secure and efficient keyword search scheme for cloud storage, which is built upon distributed point function (DPF). Specifically, outsourced keyword indexes are encoded by using garbled bloom filter and cuckoo filter, instead of bloom filter adopted by most of the state-of-the-art DPF-based schemes. In this way, clients can apply cuckoo hashing into DPF and utilize a segmentation method to interact with cloud servers for keyword search, and servers can obliviously aggregate DPF evaluation results to perform the search. Accordingly, the computational complexity at server side can be significantly reduced. Furthermore, the proposed scheme preserves constant downlink overheads, which is more communication-efficient for multi-keyword conjunctive search. To achieve privacy preservation and access control for multiple clients, we propose a double encryption method to encrypt outsourced indexes and correspondingly put forward an authorization algorithm from set-constrained pseudorandom functions by which fine-grained search-authorized keys can be generated, and collusion attacks among clients are addressed by integrating Wegman-Carter message authentication codes and cover-free systems. Since our scheme is designed under both semi-honest and malicious models (i.e., malicious servers may return incorrect query results), we use a simulation-based proof to formally demonstrate its security properties. Finally, we develop a proof-of-concept prototype and perform extensive experiments to show our scheme's practicality and efficiency in terms of computation, communication, and storage overheads.**

***Index Terms*—Cloud storage, distributed point function, keyword search, multi-client, privacy.**

I. Introduction

With the advancement of cloud computing ecosystems, cloud-based data storage services such as Dropbox have become popular and affordable solutions for not only individuals but also enterprises. With the services, users can save local storage resources by storing and managing their documents on remote cloud servers. However, frequent data breaches have become one of the major obstacles that hinder users from using these services [1], [2]. To provide security protection, end-to-end data encryption methods can be integrated into cloud-based data management, i.e., data are encrypted by users' cryptographic keys before outsourcing to cloud servers, and cloud servers store only ciphertexts. By doing so, even if attackers compromise cloud servers, they can only obtain encrypted data. Nevertheless, the straightforward cryptographic storage system will raise another issue: without the capability of decrypting the stored ciphertexts, cloud servers cannot facilitate keyword search, one of the critical functionalities that plaintext storage providers can provide.

To solve the dilemma, many research works [3], [4], [5], [6] have been proposed to achieve secure keyword search for cloud-based data storage, which can be roughly categorized into: (1) efficient schemes without considering the leakages of search pattern and access pattern [3], [4], and (2) leakage-free schemes with expensive costs [5], [6]. The works in the first category enable users to search over ciphertexts with high performance in terms of computation and communication costs, since they are commonly constructed from lightweight cryptosystems such as symmetric encryption. Yet it is not free to achieve high efficiency. When a user searches documents with keywords, attackers can learn whether the user searches with the same keyword (if the user searches multiple times) and which specific documents match the query keywords. Though attackers cannot directly see the content of keywords and documents, they can exploit the pattern leakages to recover part of keywords and document contents, which is proven to be feasible to some extent [7], [8]. The works in the second category can deal with the pattern leakages by utilizing the techniques such as homomorphic encryption [5] and oblivious RAM (ORAM) [6]. They can provide high security guarantees but are prohibitively expensive in real-world deployments. Therefore, how to achieve efficient leakage-free keyword search remains a pressing challenge. Fortunately, a new technique proposed recently, named distributed point function (DPF) [9], paves a new way to meet the efficiency and security requirements of keyword search simultaneously.

Basically, DPF can be regarded as a multi-server private information retrieval (MPIR) technique that allows a user to retrieve an item from multiple servers in possession of identical databases without revealing which item is retrieved, as long as

Manuscript received 25 November 2022; revised 10 February 2023; accepted 2 March 2023. Date of publication 7 March 2023; date of current version 12 January 2024. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grants 2020B0101090004 and 2020B0101360001, in part by the National Natural Science Foundation of China under Grant 62072215, and in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. *(Corresponding author: Cheng Huang.)*

Cheng Huang, Dongxiao Liu, and Xuemin Shen are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: cheng.huang@uwaterloo.ca; dongxiao.liu@uwaterloo.ca; sshen@uwaterloo.ca).

Anjia Yang is with the College of Cyber Security, Jinan University, Guangzhou, Guangdong Province 510632, China, and also with Pazhou Lab, Guangzhou, Guangdong Province 510330, China (e-mail: anjiayang@gmail.com).

Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Saint John, NB E3B 5A3, Canada (e-mail: rlu1@unb.ca).

Digital Object Identifier 10.1109/TDSC.2023.3253786

not all servers collude. Compared to traditional MPIR [10], DPF is more promising, as it significantly reduces communication overheads while maintaining high computational efficiency. It can be naturally extended to design secure keyword search schemes with bitmap-based indexes, where each row corresponds to a bitmap of words for a document [9], [11]. The bitmap-based indexes are encrypted before being outsourced, and users can search a keyword by privately retrieving the column corresponding to the keyword. In spite of the simplicity, the bitmap-based indexes may become extremely large to accommodate words of all stored documents. To bypass the limitation, bloom-filter-encoded indexes are introduced into the state-of-the-art secure DPF-based keyword search schemes [12]. Particularly, all keywords of a document are compressed into a bloom filter (BF), and the bloom filter is inserted as one row of keyword indexes. Given the maximum number of keywords associated a document, a BF's size is fixed and hence BF-encoded indexes have smaller storage costs. Similarly, BF-encoded indexes can be encrypted and searched: when searching a keyword, users can first map the keyword into a bloom filter to locate k positions using k hash functions, and then privately retrieve k columns (associated with the mapped k positions) by means of DPF. All retrieved columns can be combined to obtain a k-column query result, and users can scan each row of the result. If a row is all-ones, it means the document corresponding to that row contains the keyword.

Despite its considerable performance, the current DPF-based schemes are not always suitable in some cases due to the characteristics of BF. Generally, a BF's size (m) and the number of hash functions needed (k) are mainly determined by the number of inserted elements n and the false positive rate $\epsilon \in (0, 1)$. Note that, for BF-encoded indexes, when there exist more keywords associated with one document, n will become larger. In the meantime, if negligible false positive is required for keyword search, ϵ will become smaller. These two constraints will lead to a situation in which $k \times m$ becomes larger, and here $k \times m$ is the number of DPF evaluation performed at server side to search one keyword. More heavy computational burden at server side will cause longer search latency and significantly degrade user experience. According to our experiments, with a database of 4,000 documents and 1,000 keyword per document, searching one keyword with false positive 10^{-5} needs more than 17 seconds. Therefore, the first challenge is how to reduce the search latency under the circumstance. Moreover, although the designers of the state-of-the-art DPF-based schemes suggested distributing data owners' master keys to clients to make their schemes useful for a multi-client data sharing scenario (i.e., one writer and multiple readers), it is not desirable since the likelihood of key exposure increases and no access control based on clients' permissions is enforced. Since cloud servers may be compromised, we cannot simply apply server-side authentication-based access control to achieve fine-grained access control. To this end, a fine-grained access control on clients' search permissions should be achieved and managed by data owners, which is the second challenge. Furthermore, the multi-client setting brings another security challenge: malicious clients may collude with each other and part of cloud servers to gain more information about outsourced databases, beyond what their permissions have granted. The security concern was clearly defined and addressed in [13] under the semi-honest model, without considering the pattern leakages. Differently, our objective is to not only limit the cross-client leakages under the semi-honest model, but also to address collusion attacks under the malicious model, where part of cloud servers may deceive honest clients by returning incorrect query results based on the information obtained through collusion.

Aiming at three challenges mentioned above, we propose a multi-client secure and efficient DPF-based keyword search scheme for cloud storage in this paper. To reduce the search latency, our intuitive idea is to enable cloud servers to obliviously "aggregate" all columns associated with query keywords such that servers only perform m-times DPF evaluation instead of $k \times m$-times (k corresponds to the number of hash functions used in a BF, and m is the size of a BF). To achieve the goal, new encoding methods based on garbled bloom filter and cuckoo filter are proposed for generating keyword indexes, and we borrow the idea from [14] to realize multi-point DPF evaluation using cuckoo hashing. In this way, the complexity of DPF evaluation at server side can be reduced to $O(3\ m)$. On top of that, we extend DPF-based keyword search with a segmentation method that splits a search query into M segments ($M \in [1, m]$), and further reduce the complexity at server side to $O(3\ M)$. Certainly, the improvements are not free: system initialization time and updating delay are longer, and the communication overheads of searching one keyword is slightly increased (e.g., extra 14%). However, the extra costs can trade for significant search latency reducing by more than $100\times$, which is still worthwhile. Similar to [12], our scheme also achieves verifiability to detect malicious servers that return incorrect query results. We achieve this property via applying Wegman-Carter message authentication codes (WCMAC) that support homomorphic xor and integrity checking, since original MAC methods used in [12] cannot be adapted to oblivious aggregation operations. To protect the confidentiality of outsourced indexes as well as be compatible with multi-client environments, we design a double encryption method and correspondingly put forward an authorization algorithm based on set-constrained pseudorandom functions (SCPRF) [15]. With the method, data owners can encrypt keyword indexes meanwhile authorizing and constraining clients' search permissions by distributing fine-grained search keys. In addition, cover-free families (CFF) [16], [17] are integrated into the proposed authorization algorithm to resist collusion attacks among clients, which enables data owners to create and distribute unique verification keys to clients. By doing so, even if a small part of clients collude with malicious adversaries that compromise part of cloud servers, adversaries cannot successfully forge query results that can pass honest clients' verification. In summary, the contributions are three-folds:

- The proposed multi-client secure keyword search scheme consists of two detailed constructions, which can satisfy diverse security and efficiency requirements in various cloud-based data storage applications.
- The proposed scheme can protect search and access pattern, forward and backward privacy, achieve soundness and access control, and resist collusion attacks at the same

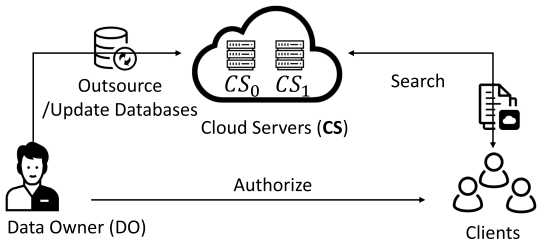


Fig. 1. System model.

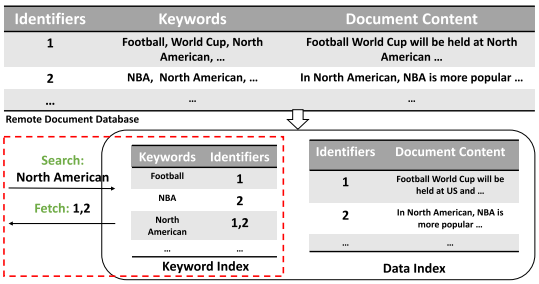


Fig. 2. Outsourced data indexes.

time. We formally prove these security properties in a simulation-based paradigm;

- We implement a Java-based prototype to evaluate our scheme's performance with real-world and synthetic datasets, and the experiment results are in accordance with theoretic analysis. The source code of the prototype is available at [18].

The remainder of this paper is organized as follows. In Section II, we introduce the system model, define the adversarial and security models, and identify the design goals. Then, we present preliminary building blocks in Section III and give two detailed constructions of our secure keyword search scheme in Section IV. Subsequently, security analysis and performance evaluation are presented in Sections V and VI, respectively. Finally, Section VII reviews some related works and Section VIII draws the conclusion.

II. Models and Design Goals

A. System Model

As shown in Fig. 1, a cloud-based data search system consists of three types of entities: a data owner (DO), clients, and cloud servers ($\mathbf{CS}$). In the system, DO can outsource their documents to cloud servers to build remote databases. When the databases are established, clients authorized by DO can query the documents associated with particular keywords, and cloud servers will return a matching subset of documents, instead of returning the entire databases.

Generally, a remote document database can be divided into two indexes [8], as shown in Fig. 2, and correspondingly the data search procedures can be separated into two parts. In the first part (Part-I), clients search the keywords over a keyword index and retrieve the identifiers of the documents that contain the keywords (highlighted with a red box). In the second part (Part-II), clients fetch the documents' content stored in the data index according to the identifiers. We mainly focus on Part-I, i.e., keyword search, in this paper, which is commonly deemed as a "structure-only" keyword search model [8].

Notations. We use ind to denote the identifier of a document. A keyword index, $\mathbb{DB}$, of N documents $(\mathsf{Doc}_1, \ldots, \mathsf{Doc}_N)$, is represented by a collection of (document identifier, keyword set) pairs, i.e., $\mathbb{DB} = (\mathsf{ind}_i, \mathbf{w}_i)_{i=1}^N$, where $\mathbf{w}_i = (w_{i,1}, w_{i,2}, \ldots)$ are all keywords of Doc_i. For notational simplicity, we usually set $\mathsf{ind}_i = i$. Let $|\mathbf{w}_i|$ be the number of keywords of Doc_i. We denote the i-th (document identifier, keyword set) pair in $\mathbb{DB}$ as $\mathbb{DB}[i]$ and denote the maximum number of keywords contained by one document as n, i.e., $max(|\mathbf{w}_1|, |\mathbf{w}_2|, \ldots, |\mathbf{w}_N|) \leq n$. The universe of keywords is defined as $\mathbf{W} = \bigcup_{i=1}^N \mathbf{w}_i$, and $|\mathbf{W}|$ means the total number of unique keywords. Let $\mathbb{DB}(w)$ be the identifiers of all documents that contain w. Moreover, we use $\mathbb{M}[i, :]$ to denote the i-th row of a matrix $\mathbb{M}$, and use $\mathbb{M}[:, j]$ to denote its j-th column. For a vector, V, its i-th element is denoted by $V[i]$, and let $V[i:j]$ $(i < j)$ be the subvector, $(V[i], V[i+1], \ldots, V[j-1], V[j])$.

B. Adversarial Model & Security Model

Adversarial Model. In an outsourced model, clients should not be allowed to perform unauthorized search, i.e., their trust comes from authorization. Cloud servers cannot be fully trusted since they may be compromised by semi-honest or malicious adversaries. The semi-honest $\mathbf{CS}$ will honestly follow a secure keyword search scheme, but may be curious and try to extract DO's data and clients' queries by launching passive attacks. Furthermore, the malicious $\mathbf{CS}$ may deviate from a secure search scheme to cheat the clients about the query results. In addition, we consider a collusion attack where part of clients may collude with adversarial cloud servers to help extract information beyond authorization and hide malicious cheating behavior. Other malicious attacks on system availability (e.g., intentionally deleting clients' data and rejecting any clients' queries), are out of scope.

Trust Model. In our system, DO is always trusted, and we make a reasonable assumption that at least one cloud service provider is not compromised by adversaries and behaves honestly. In reality, two different cloud service providers can be selected by DO to achieve such an environment. *As a result, a two-server model is considered in this paper, which are denoted by CS_0 and CS_1.*

Syntax of Secure Keyword Search. A secure keyword search scheme, Π, mainly consists of two algorithms, $Setup$ and $ClientKeyGen$, and two protocols, $Search$ and $Update$:

- $Setup(\mathbb{DB}, \mathsf{params}) \rightarrow (\mathbb{EDB}, \mathsf{stat}, \mathsf{MK})$: By inputting the plaintext keyword index, $\mathbb{DB}$, and public parameters, $\boldsymbol{params}$, the algorithm is run by DO to initialize the system by outputting master symmetric keys, MK, $\mathbb{DB}$'s public state, stat, as well as an encrypted keyword index, $\mathbb{EDB}$, that is outsourced to $\mathbf{CS}$;
- $ClientKeyGen(\mathsf{MK}, \mathbf{ind}^*, \mathbf{w}^*, \mathsf{stat}) \rightarrow \mathsf{MK}^*$: By inputting MK, stat, a set of keywords $\mathbf{w}^* \subseteq \mathbf{W}$, and a set of document identifiers $\mathbf{ind}^* \subseteq [1, N]$, the algorithm outputs

search-authorized keys, MK^*, for a client. With the keys, the client is permitted to search on $\mathbf{w}^*$ associated with $\mathbf{ind}^*$;

- $Search(\mathsf{MK}^*, \mathsf{stat}, q; \mathbb{EDB}) \rightarrow \mathbb{DB}^*(q)$: The protocol is executed between a client (with the inputs of MK^*, stat, and a query keyword $q = w$) and $\mathbf{CS}$ (with the input of $\mathbb{EDB}$). At the end of the protocol, the client outputs the queried results, $\mathbb{DB}^*(q) \subseteq \mathbb{DB}(q)$, and $\mathbf{CS}$ output $\perp$;
- $Update(\mathsf{MK}, \mathsf{stat}, \mathsf{op}, (\mathsf{ind}, \mathbf{w}); \mathbb{EDB}) \rightarrow (\mathsf{stat}; \mathbb{EDB})$: The protocol is for DO to update (e.g., add, delete, or modify) an entry $(\mathsf{ind}, \mathbf{w})$ in $\mathbb{DB}$. It is executed between DO and $\mathbf{CS}$, and DO outputs an updated public state, stat, and $\mathbf{CS}$ output a new encrypted keyword index, $\mathbb{EDB}$, at the end of the protocol;

Security w.r.t Adversarial CS. The security (a.k.a. the data privacy property) is defined by a family of stateful leakage functions $\mathcal{L} = (\mathcal{L}^{\text{stp}}, \mathcal{L}^{\text{srch}}, \mathcal{L}^{\text{upd}})$ that describe what information is leaked to a polynomial-probabilistic-time (PPT) adversary $\mathcal{A}$ during the executions of $Setup$, $Search$, and $Update$. The security is formulated using a simulation-based paradigm that involves a real experiment $\mathsf{REAL}^{\Pi}_{\mathcal{A}}(1^{\lambda})$ and an ideal experiment $\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^{\lambda})$, where λ is a security parameter. If $\mathcal{A}$ cannot distinguish the two experiments, there is no other information revealed except the information obtained from $\mathcal{L}$. The experiments are defined as follows:

- $\mathsf{REAL}^{\Pi}_{\mathcal{A}}(1^{\lambda})$ - $\mathcal{A}$ chooses $\mathbb{DB}$ and sends $\mathbb{DB}$ to the experiment who runs $Setup(\mathbb{DB})$ and returns $\mathbb{EDB}$. Then, $\mathcal{A}$ adaptively performs search queries q (or update queries $(\mathsf{op}, (\mathsf{ind}, \mathbf{w}))$). The experiment outputs the transcripts generated by running $Search(q)$ (or $Update(\mathsf{op}, (\mathsf{ind}, \mathbf{w}))$) to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a bit $\{0, 1\}$;
- $\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^{\lambda})$ - On input $\mathbb{DB}$ chosen by $\mathcal{A}$, the experiment generates $\mathbb{EDB}$ by using a simulator $\mathcal{S}(\mathcal{L}^{\text{stp}}(\mathbb{DB}))$ and returns $\mathbb{EDB}$ to $\mathcal{A}$. Then, $\mathcal{A}$ adaptively performs search queries q (or update queries $(\mathsf{op}, (\mathsf{ind}, \mathbf{w}))$), and the experiment relies on $\mathcal{S}(\mathcal{L}^{\text{srch}}(q))$ (or $\mathcal{S}(\mathcal{L}^{\text{upd}}(\mathsf{op}, (\mathsf{ind}, \mathbf{w})))$) to generate the transcripts and returns them to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a bit $\{0, 1\}$.

Definition 1 ($\mathcal{L}$-Adaptive Security). A keyword search scheme, Π, is $\mathcal{L}$-adaptively-secure if, for a PPT adversary, $\mathcal{A}$, there exists a PPT algorithm $\mathcal{S}$ such that

$$|\Pr[\mathsf{REAL}^{\Pi}_{\mathcal{A}}(1^{\lambda}) = 1] - \Pr[\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^{\lambda}) = 1]| \leq negl(1^{\lambda}).$$

Leakage Functions. The leakage functions are defined as follows: $\mathcal{L}^{\text{stp}}(\mathbb{DB}) = (\mathsf{ts}^{\text{stp}}, N, n)$, $\mathcal{L}^{\text{srch}}(q) = \mathsf{ts}^{\text{srch}}$, and $\mathcal{L}^{\text{upd}}(\mathsf{op}, (\mathsf{ind}, \mathbf{w})) = (\mathsf{ts}^{\text{upd}}, \mathsf{op}, \mathsf{ind})$ where $(\mathsf{ts}^{\text{stp}}, \mathsf{ts}^{\text{srch}}, \mathsf{ts}^{\text{upd}})$ are the timestamps when clients initiate the executions of $Setup$, $Search$, and $Update$.

Search Pattern & Access Pattern. Search pattern reveals queries with the same underlying keyword, and access pattern reveals the matching results between documents and queried keywords. We recall the formal definitions of search pattern and access pattern [19]:

Definition 2 (Search Pattern). Given a search query vector $\mathbf{q} = (q_1, q_2, \ldots, q_t)$, the search pattern is a symmetric binary matrix of $t \times t$ such that

$$\mathcal{SP}_{\mathbf{q}}[i, j] = \begin{cases} 1 & \text{if } q_i = q_j; \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Definition 3 (Access Pattern). Given a search query vector $\mathbf{q} = (q_1, q_2, \ldots, q_t)$ and $\mathbb{DB}$, the access pattern is a binary matrix of $t \times N$ such that

$$\mathcal{AP}_{\mathbf{q}}[i, j] = \begin{cases} 1 & \text{if } q_i \in \mathbb{DB}[j]; \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

According to the defined leakage functions, an $\mathcal{L}$-adaptively-secure keyword search scheme protects the *search pattern* and the *access pattern*.

Forward & Backward Privacy. Forward privacy and backward privacy are two common privacy properties utilized to capture the privacy leakages caused by update [20]. Informally, forward privacy guarantees that an update operation does not expose any information about the updated keywords, i.e., adversaries cannot learn that the updated document matches a previously-queried keyword. Backward privacy guarantees that, even if clients query the same keyword twice within any period, the information about the documents associated with the keyword that are previously added and later deleted is not leaked to adversaries. We borrow the classical definitions of forward privacy and type-I backward privacy with slight modification [21]. We make such modification since the original forward privacy and backward privacy are defined in the situation where search pattern and access pattern are disclosed to adversaries. Specifically, forward privacy* and identifier-revealing backward privacy are defined as follows:

Definition 4 (Forward Privacy*). An $\mathcal{L}$-adaptively-secure keyword search scheme is forward-private if $\mathcal{L}^{\text{upd}}$ can be written as $\mathcal{L}^{\text{upd}}(\mathsf{op}, (\mathsf{ind}, \mathbf{w})) = (\mathsf{ts}^{\text{upd}}, \mathsf{op}, \mathsf{ind})$.

The definition of forward privacy* is similar to the definition of the former forward privacy. The difference is that forward privacy* does not permit the leakage of the number of keywords modified in the document with the identifier, ind.

Definition 5 (Identifier-Revealing Backward Privacy). An $\mathcal{L}$-adaptively-secure keyword search scheme is backward-private with identifier revealing if $\mathcal{L}^{\text{srch}}$ and $\mathcal{L}^{\text{upd}}$ can be written as $\mathcal{L}^{\text{upd}}(\mathsf{op}, (\mathsf{ind}, \mathbf{w})) = (\mathsf{ts}^{\text{upd}}, \mathsf{op}, \mathsf{ind})$, and $\mathcal{L}^{\text{srch}}(w \in \mathbf{w}) = \mathsf{ts}^{\text{srch}}$.

Compared with the former type-I backward privacy, our modified definition does not leak the documents currently matching w and the total number of updates for w, but it will reveal the updated document's identifier, ind. However, leaking ind does not violate the requirement of backward privacy. That is because, adversaries cannot infer the updated keywords related to ind no matter whether the document with the identifier, ind, is inserted/deleted, due to the hidden search and access pattern.

Soundness. The soundness (a.k.a. verifiability) property ensures that clients can catch the malicious adversaries who try to return incorrect search query responses, and it is captured by the following game [22]:

$\mathsf{SOUND}^{\Pi}_{\mathcal{A}}(1^{\lambda})$ - $\mathcal{A}$ chooses $\mathbb{DB}$ and sends $\mathbb{DB}$ to a challenger who runs $Setup(\mathbb{DB})$ and returns $\mathbb{EDB}$ to $\mathcal{A}$. Then, $\mathcal{A}$ and the challenger execute $Search(q)$ where the search queries, q, are adaptively chosen by $\mathcal{A}$, and the challenger returns

$(\mathbb{DB}'(q), \mathsf{val}')$ to $\mathcal{A}$, where val' indicates whether the challenger accepts $\mathbb{DB}'(q)$. If $\mathsf{val}' = '\text{ACCEPT}'$ and $\mathbb{DB}'(q) \neq \mathbb{DB}(q)$, $\mathcal{A}$ outputs 1; otherwise, outputs 0. The output of the game is what $\mathcal{A}$ outputs.

Definition 6 (Soundness). A keyword search scheme, Π, is sound if for a PPT adversary, $\mathcal{A}$, we have

$$\Pr[\mathsf{SOUND}^{\Pi}_{\mathcal{A}}(1^{\lambda}) = 1] \leq negl(1^{\lambda}).$$

Security w.r.t Adversarial Clients. The security (a.k.a. the access control property) means that DO can specify fine-grained search policies of keywords and documents for clients according to their identity attributes, and a PPT adversary $\mathcal{A}$ that controls clients cannot search unauthorized keywords and documents. A formal security game is defined to capture the access control property:

$\mathsf{ACCESS}^{\Pi}_{\mathcal{A}}(1^{\lambda})$ - A challenger runs $Setup(\mathbb{DB})$ to generate MK and initializes two empty sets (Q_1, Q_2), and returns N and $\mathbf{W}$ to $\mathcal{A}$. Then, $\mathcal{A}$ adaptively makes authorization queries of keywords and documents, i.e. $\mathbf{ind}^* \subset [1, N]$ and $\mathbf{w}^* \subset \mathbf{W}$, and the challenger returns $\mathsf{MK}^* = ClientKeyGen(\mathsf{MK}, \mathbf{ind}^*, \mathbf{w}^*)$ and adds $\mathbf{ind}^*$ and $\mathbf{w}^*$ into Q_1 and Q_2, respectively. Later, $\mathcal{A}$ outputs a search-authorized key MK' for the keyword $\mathsf{w}' \in \mathbf{W}$ and the document $\mathsf{ind}' \in [1, N]$. The game outputs 1 if $\mathsf{MK}' = ClientKeyGen(\mathsf{MK}, \mathsf{ind}', \mathsf{w}')$ where $\mathsf{ind}' \notin Q_1$ or $w' \notin Q_2$; otherwise, outputs 0.

Definition 7 (Access Control). A keyword search scheme, Π, achieves access control if for a PPT adversary, $\mathcal{A}$, we have

$$\Pr[\mathsf{ACCESS}^{\Pi}_{\mathcal{A}}(1^{\lambda}) = 1] \leq negl(1^{\lambda}).$$

C. Design Goals

Under the aforementioned system model and security model, our goal is to design a keyword search scheme that supports dynamic update, realize fine-grained access control for multiple clients, guarantees $\mathcal{L}$-adaptive security and soundness, and provides practical performance in terms of *computational efficiency*, *communication overheads*, and *storage costs: computational efficiency* involves the computational costs of $Setup$, $ClientKeyGen$, $Search$, and $Update$ at client side and server side; *communication overheads* include the uplink overheads (clients send search queries and update queries) and the downlink overheads (cloud servers respond query results); and *storage costs* contain the local storage costs on clients and the outsourced storage costs on cloud servers.

III. Building Blocks

A. Bloom Filter & Garbled Bloom Filter

Bloom Filter (BF) is a space-efficient data structure that compresses a set of n variable-length elements into an m-bit binary, B, and supports efficient membership query without false negative and with controllable false positive. We can use the following three algorithms to build a BF for a vector of elements $\mathbf{x} = (x_1, \ldots, x_n)$ and test whether an element x' exists in the BF:

- $BF.Gen(\epsilon, n) \rightarrow (k, m, \mathbf{h})$: By inputting a false positive rate (FPR), $\epsilon \in (0, 1)$, and the maximum size of elements, n, the algorithm first outputs the optimal number of hash functions, k, and the optimal size of the filter, m, such that FPR is satisfied. Then, the algorithm outputs a family of universal hash functions $\mathbf{h} = (h_1(\cdot), \ldots, h_k(\cdot))$ where $h_j(\cdot) \rightarrow [1, m]$ for $j = 1$ to k;
- $BF.Build(\mathbf{x}, m, \mathbf{h}) \rightarrow B$: The algorithm first initializes an all-zero binary B where $B[i] = 0$ for $i = 1$ to m. Then the algorithm traverses $x \in \mathbf{x}$ and sets $B[h_j(x)] = 1$ for $j = 1$ to k, and finally outputs B;
- $BF.Query(x', B) \rightarrow 0/1$: The algorithm tests whether $\wedge_{j=1}^{k} B[h_j(x')] \stackrel{?}{=} 1$. If it holds, the algorithm outputs 1; otherwise, outputs 0 (x' does not exist in B).

Garbled Bloom Filter (GBF) is a variant of Bloom Filter that can compress a set of n variable-length elements into an m-dimension array of ψ-bit binaries [23] instead of an m-bit binary. GBF also contains three algorithms:

- $GBF.Gen(\epsilon, n) \rightarrow (k, m, \mathbf{h}, fp)$: Similar to $BF.Gen(\cdot)$, the difference is that, the algorithm additionally outputs a fingerprint algorithm $fp(\cdot) \rightarrow \{0, 1\}^{\psi}$, and ϵ only indicates the success probability of building a GBF while ψ decides the false positive rate;
- $GBF.Build(\mathbf{x}, m, \mathbf{h}, fp) \rightarrow A$: An empty array A of the length m, where $A[i] = \{0\}^{\psi}$ for $i = 1$ to m, is initialized. For each $x_i \in \mathbf{x}$, the algorithm sets $A[h_j(x_i)]$ for $j = 1$ to k and ensures that $\oplus_{j=1}^{k} A[h_j(x_i)] = fp(x_i)$. At the end, the algorithm outputs A;
- $GBF.Query(x', A) \rightarrow 0/1$: The algorithm tests whether $\oplus_{j=1}^{k} A[h_j(x_i)] \stackrel{?}{=} fp(x_i)$. If it holds, the algorithm outputs 1; otherwise, outputs 0.

Theorem 1 (Collision & False Positive of GBF). A GBF with the parameters, $(m, k, \mathbf{h}, \psi)$, that compresses a set of n elements, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, into an array, A, satisfies that: (i) $\forall x' \in \mathbf{x}, x \in \mathbf{x}$: $\Pr[(\oplus_{j=1}^{k} A[h_j(x')]) = fp(x)] \leq \epsilon$; and (ii) $\forall x' \in \mathbf{x}$: $\Pr[(\oplus_{j=1}^{k} A[h_j(x')]) = fp(x')] \leq 2^{-\psi}$: $\epsilon = p^k \times (1 + O(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}))$, where $p = 1 - (1 - \frac{1}{m})^{kn}$.

B. Cuckoo Hashing & Cuckoo Filter

Cuckoo Hashing (CH) is a special design of a hash table for resolving collision issues, where n variable-length elements can be inserted into different positions of a one-dimensional hash table T with the size m, using k hash functions. We can use the following two algorithms to build a CH-based table for storing a vector of elements $\mathbf{x} = (x_1, \ldots, x_n)$:

- $CH.Gen(n, k, \tau) \rightarrow (m, \mathbf{h})$: By inputting the maximum size of elements, n, and a collision parameter $\tau \in \mathbb{N}$, the algorithm outputs the size of the table m and a family of hash functions, $\mathbf{h} = (h_1(\cdot), \ldots, h_k(\cdot))$ where $h_j(\cdot) \rightarrow [1, m]$;
- $CH.Build(\mathbf{x}, \mathbf{h}, \Phi) \rightarrow T/'\text{FAIL}'$: With the input of a termination parameter, Φ, the algorithm first generates an empty table T where $T[i] = \mathsf{null}$ for $i = 1$ to m and then inserts $x_i \in \mathbf{x}$ into T for $i = 1$ to n using the following steps: (i) sets $\mathsf{incr} = 0$; (ii) randomly chooses $j \in [1, k]$; (iii) if $T[h_j(x_i)] = \mathsf{null}$, sets $T[h_j(x_i)] = x_i$ and $i = i + 1$, and goes back to (i); otherwise, swaps $T[h_j(x_i)]$ and x_i; (iv) sets $\mathsf{incr} = \mathsf{incr} + 1$ and goes back to (ii). If $\mathsf{incr} > \Phi$, the

algorithm stops and outputs 'FAIL' to indicate T is full; otherwise, outputs T;

Theorem 2 (Cuckoo-hashing Failure Probability). For a CH-based table, T, with the parameters, $(k, m, \mathbf{h})$, when $k = 3$, $m = \iota \cdot n$ ($\iota > 1$), and $\mathbf{h}$ is a family of collision-resistant hash functions, the probability that n elements fail to be inserted into T is $2^{-\tau}$, where $\tau = e_1 \cdot \iota - e_2 - \log_2(n)$, $e_1 = 123.5 \cdot \mathsf{cdf}_1$, $e_2 = 130 \cdot \mathsf{cdf}_2$, and cdf_1 and cdf_2 are two CDF values evaluated at n for the normal distributions of different means and standard deviations (6.3,2.3) and $(6.45, 2.18)$, respectively.

Cuckoo Filter (CF) is a CH-based space-efficient data structure that compresses n variable-length elements into m buckets, BK, for efficient membership query. We use $\mathsf{BK}[i]$ to represent the i-th bucket. Each bucket contains α entries, and j-th entry of $\mathsf{BK}[i]$ is denoted by $\mathsf{BK}[i][j]$. The following three algorithms can be utilized to establish a CF for a vector of elements $\mathbf{x} = (x_1, \ldots, x_n)$:

- $CF.Gen(n, m, \epsilon, \mu) \rightarrow (\alpha, h, fp)$: By inputting the maximum size of elements, n, an FPR $\epsilon \in (0, 1)$, and a load factor, $\mu \in (0, 1)$, the algorithm outputs $\alpha = \lceil \frac{\lceil n/\mu \rceil}{m} \rceil$, a hash function $h(\cdot) \rightarrow [1, m]$, and a fingerprint algorithm $fp \rightarrow \{0,1\}^{\theta}$;
- $CF.Build(\alpha, \mathbf{x}, m, h, fp, \Phi) \rightarrow \mathsf{BK}$: Empty buckets BK where $\mathsf{BK}[i][j] = \{0\}^{\theta}$ for $i = 1$ to m and $j = 1$ to α, are initialized first. The algorithm then performs the following steps to insert $x_i \in \mathbf{x}$ into BK for $i = 1$ to n: (i) sets $\mathsf{incr} = 0$; (ii) calculates $\rho = fp(x_i)$, $\mathsf{idx}_1 = h(x_i)$, and $\mathsf{idx}_2 = h(\rho) \oplus \mathsf{idx}_1$; (iii) if $\mathsf{BK}[\mathsf{idx}]$ is not full (idx can be either idx_1 or idx_2), i.e., $|\mathsf{BK}[\mathsf{idx}]| < \alpha$, inserts ρ into $\mathsf{BK}[\mathsf{idx}]$, sets $i = i + 1$, and goes back to (i); otherwise, randomly chooses $j \in [1, \alpha]$ and sets $\mathsf{idx} = \mathsf{idx}_1$ (or idx_2), swaps $\mathsf{BK}[\mathsf{idx}][j]$ and ρ; (iv) computes $\mathsf{idx} = \mathsf{idx} \oplus h(\rho)$ and $\mathsf{incr} = \mathsf{incr} + 1$, and goes back to (iii). If $\mathsf{incr} > \Phi$, the algorithm stops and outputs failures to indicate BK is full; otherwise, outputs BK;
- $CF.Query(x, \mathsf{BK}) \rightarrow 0/1$: The algorithm calculates $\mathsf{idx}_1 = h(x')$ and $\mathsf{idx}_2 = h(fp(x')) \oplus \mathsf{idx}_1$, and tests whether $fp(x') \in \mathsf{BK}[\mathsf{idx}_1]$ or $fp(x') \in \mathsf{BK}[\mathsf{idx}_2]$. If one of the conditions is hold, the algorithm outputs 1; otherwise, outputs 0 (x' does not exist in BK).

C. Two-Party Distributed Point Function

Distributed Point Function (DPF) is one of the most efficient techniques to achieve private information retrieval (PIR) [9]. A point function f means that, the function always returns 0 unless the function is evaluated on a particular point x, and $f(x) = y$. With a two-party DPF (P_0 and P_1), a point function f can be split into two function shares $(\mathsf{key}_0, \mathsf{key}_1)$ that can be evaluated on arbitrary inputs at two independent parties to get two separated evaluation results. The results can be combined later to recover the outputs of evaluating f. Typically, DPF consists of the following two algorithms:

- $DPF.Gen(1^{\lambda}, x, y) \rightarrow (\mathsf{key}_0, \mathsf{key}_1)$: The algorithm inputs a security parameter, λ, x and y, and returns two function shares $(\mathsf{key}_0, \mathsf{key}_1)$ for the point function $f(x) = y$;
- $DPF.Eval(\mathsf{key}_b, x') \rightarrow y'_b$: The algorithm takes key_b ($b \in \{0, 1\}$) and an input x', and returns the evaluation y'_b.

The evaluation results (y'_0, y'_1) on x' of two function shares can be combined to recover $f(x')$, i.e., $y'_0 \oplus y'_1 = y' = f(x')$. If $x' = x$, we have $y = y'$; otherwise, we have $y' = 0$.

Theorem 3 (Simulation Security of Two-Party DPF). For one corrupted party (P_0 or P_1 is controlled by a PPT-adversary $\mathcal{A}$), there exists a PPT algorithm, SIM, such that the following two experiments' outputs are computationally indistinguishable with the leakage of the point function f's input size and output size, i.e., $\mathcal{L}^{DPF}(f) = (|x|, |y|)$:

- $\mathsf{REAL}^{DPF}(1^{\lambda})$: Output $DPF.Gen(1^{\lambda}, x, y) \rightarrow (\mathsf{key}_0, \mathsf{key}_1)$;
- $\mathsf{IDEAL}^{DPF}(1^{\lambda})$: Output $SIM(1^{\lambda}, \mathcal{L}^{DPF}(f))$.

DPF-Based PIR. DPF can be used for constructing two-party PIR. Considering that two servers store identical copies of an outsourced database $\mathbb{DB} = (\mathsf{val}_1, \ldots, \mathsf{val}_m)$, and a client wants to retrieve val_i but hide i, the client can first run $DPF.Gen(x = i, y = 1) \rightarrow (\mathsf{key}_0, \mathsf{key}_1)$ and send $(\mathsf{key}_0, \mathsf{key}_1)$ to two servers, respectively. Each server returns $\mathsf{res}_b = \oplus_{j=1}^{m}(DPF.Eval(\mathsf{key}_b, j) \cdot \mathsf{val}_j)$ to the client ($b \in \{0, 1\}$). The client finally combines $\mathsf{val}_i = \mathsf{res}_0 \oplus \mathsf{res}_1$.

D. Wegman-Carter Message Authentication Codes

Message Authentication Codes (MAC). MAC is a symmetric cryptographic primitive for data source authentication and integrity checking. A MAC tag for the data, D, using a symmetric key, $K \in \{0,1\}^{\lambda}$, is denoted by $\mathsf{tag} = MAC(K, D)$, and multiple MAC tags $(\mathsf{tag}_1, \ldots \mathsf{tag}_n)$ generated for multiple data $(D_1, \ldots, D_n)$ using keys $(K_1, \ldots, K_n)$, can be aggregated to one MAC tag $\sigma = \oplus_{i=1}^{n}\mathsf{tag}_i$ if MAC is secure and unforgeable [24].

Wegman-Carter (WC) MAC. WCMAC is one special type of the polynomial MAC, which can be used for privately data integrity checking [25]. With a long-term symmetric key $K \in \mathbb{F}_{2^{\lambda}}$ and a one-time nonce $\Gamma \in \{0,1\}^{\lambda}$, a WCMAC tag for the data, D (is divided into $\mathcal{N}$ λ-bit binaries, and $D[i]$ is the i-th binary), can be constructed $WCMAC(D, K, \Gamma) = \Gamma \oplus \mathsf{Poly}_K(D)$, where $\mathsf{Poly}_K(D) = \oplus_{i=1}^{\mathcal{N}}(D[i] \cdot K^{\mathcal{N}+1-i})$ is a polynomial-based hash function and is a $\frac{\mathcal{N}}{2^{\lambda}}$-almost XOR universal (AXU) hash function. The security is defined as follows:

Theorem 4 (Security of WCMAC [25]). For any (q_m, q_v)-adversary, $\mathcal{A}$, who obtains at most q_m honestly generated MAC tags and makes at most q_v verification queries, as long as $q_m \leq 2^{\lambda/2}$, the probability that $\mathcal{A}$ succeeds in forging a MAC tag is $\frac{C\mathcal{N}q_v}{2^{\lambda}}$ where $(C \leq 2)$ is a small constant.

E. Pseudorandom Function & Set-Constrained PRF

A pseudorandom function (PRF), is denoted by $PRF : \mathcal{K} \times \{0,1\}^{f_1(\lambda)} \rightarrow \{0,1\}^{f_2(\lambda)}$, where $\mathcal{K} \in \{0,1\}^{\lambda}$ is a symmetric key, f_1 and f_2 are polynomials, and $\{0,1\}^{f_1(\lambda)}$ and $\{0,1\}^{f_2(\lambda)}$ are input and output domains. Set-constrained PRF (SCPRF) is a special class of constrained PRFs, which consists of the following two algorithms:

- $SCPRF.Cons(\mathcal{K}, S) \rightarrow \hat{\mathcal{K}}$: By inputting $\mathcal{K}$ and a set of elements S where each element is $\{0,1\}^{f_1(\lambda)}$, the algorithm outputs a constrained key $\hat{\mathcal{K}}$;
- $SCPRF.Eval(\hat{\mathcal{K}}, s) \rightarrow \varphi$: By inputting $\hat{\mathcal{K}}$ and an element $s \in \{0,1\}^{f_1(\lambda)}$, the algorithm outputs $\varphi = PRF(\mathcal{K}, s) \in \{0,1\}^{f_2(\lambda)}$ if $s \in S$; otherwise, outputs $\varphi = \perp$.

The security of SCPRF is captured by $\mathsf{GAME}_{\mathcal{A}}^{SCPRF}(1^\lambda)$: A challenger chooses $\mathcal{K} \in \{0,1\}^\lambda$, a bit $b \in \{0,1\}$, and initializes three empty sets (Q_1, Q_2, Q_3). Then, a PPT adversary, $\mathcal{A}$, adaptively invokes three types of queries:

- With an evaluation query on $s \in \{0,1\}^{f_1(\lambda)}$, the challenger returns $PRF(\mathcal{K}, s)$ and adds s into Q_1;
- With a key extraction query on S, the challenger returns $SCPRF.Cons(\mathcal{K}, S)$ and adds S into Q_2;
- With a challenge query on $s^* \in \{0,1\}^{f_1(\lambda)}$, the challenger returns $\varphi = PRF(\mathcal{K}, s^*)$ if $b = 1$ or $\varphi \in_{\mathcal{R}} \{0,1\}^{f_2(\lambda)}$ if $b = 0$, and adds s^* into Q_3

Finally, $\mathcal{A}$ outputs b', and the game outputs 1 if $b' = b$, $Q_1 \cap Q_3 = \emptyset$, and $s^* \notin \bigcup_{S' \in Q_2} S$ for all $s^* \in Q_3$; otherwise, outputs 0.

Definition 8 (Security of SCPRF [26]). SCPRF is secure if for a PPT adversary, $\mathcal{A}$, we have

$$\mathsf{Adv}_{\mathcal{A}}^{SCPRF}(1^\lambda) = \Pr[\mathsf{GAME}_{\mathcal{A}}^{SCPRF}(1^\lambda) = 1] \leq negl(1^\lambda).$$

F. Cover-Free Families (Systems)

Cover-free Families (CFF) are a set of sets with a special property: the union of # of these sets does not contain any of the other sets. Formally, a ζ-CFF(χ, ν) is defined as follows:

Definition 9 (Cover-free Families). A ζ-cover-free family $(\mathbf{X}, \mathbf{Y})$ is a set of χ elements and a set $\mathbf{Y}$ of ν subsets of $\mathbf{X}$, with the following property: for any ζ sets $(\boldsymbol{\mathcal{Y}}_{i_1}, \boldsymbol{\mathcal{Y}}_{i_2}, \ldots, \boldsymbol{\mathcal{Y}}_{i_\zeta}) \in \mathbf{Y}$, and for all other $\boldsymbol{\mathcal{Y}} \in \mathbf{Y}$, it holds that $\boldsymbol{\mathcal{Y}} \not\subseteq \bigcup_{j=1}^{\zeta} \boldsymbol{\mathcal{Y}}_{i_j}$.

If we define $\mathbf{X} = (X_1, X_2, \ldots, X_\chi)$ and $\mathbf{Y} = (\boldsymbol{\mathcal{Y}}_1, \boldsymbol{\mathcal{Y}}_2, \ldots, \boldsymbol{\mathcal{Y}}_\nu)$, a ζ-CFF(χ, ν) can be denoted by an incidence matrix $\mathbb{I}$ with χ rows and ν columns as follows:

$$\mathbb{I}[i, j] = \begin{cases} 1 & \text{if } X_i \in \boldsymbol{\mathcal{Y}}_j; \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Many approaches have been proposed for constructing a ζ-CFF(χ, ν), e.g., using polynomials over a finite field. Please refer to [16] for more details.

IV. Secure DPF-Based Keyword Search

A. Baseline

We start by describing a baseline DPF-based scheme that achieves secure keyword search against malicious adversaries [9], [12]. As shown in Fig. 3, the plaintext keyword index is first encoded using a BF that is set up by running $BF.Gen(\epsilon, n) \rightarrow (k, m, \mathbf{h})$. Each document's keywords (row in the index), $\mathbf{w}_i$ for $i = 1$ to N (are highlighted in green color), are encoded by running $BF.Build(\mathbf{w}_i, m, \mathbf{h}) \rightarrow B_i$. The BF-encoded keyword index is more space-efficient than a conventional bitmap-encoded keyword index where each keyword is mapped to one column, considering that $|\mathbf{W}|$ is quite

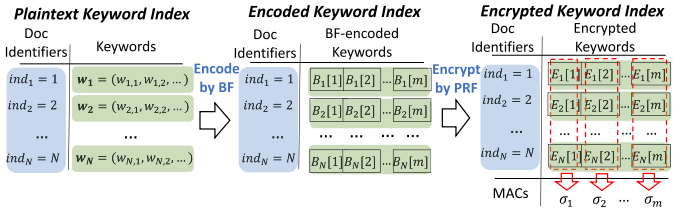


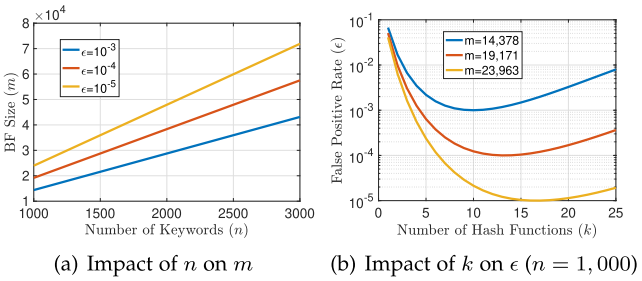
Fig. 3. High-level flows of constructing an encrypted keyword index that can be securely searched using DPF.

large. By introducing the BF, Doc_i's keywords can be accommodated into B_i whose length is fixed and computed based on n, regardless of $|\mathbf{W}|$. To protect the keywords' confidentiality, B_i is further encrypted using PRF and a long-term symmetric key $\mathcal{K} \in \{0,1\}^\lambda$. The encrypted keywords for Doc_i are $E_i = B_i \oplus PRF(\mathcal{K}, i||\mathsf{ver}_i)$, where $\mathsf{ver}_i \in \mathbb{N}$ is a unique document version.

Moreover, a cryptographic checksum, $\boldsymbol{\sigma}$, is created to help detect malicious **CS**'s cheating behavior. By using MAC, all (document identifier, keyword set) pairs are bound to the checksum using another long-term symmetric key, K. It consists of m aggregated MAC tags, $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ corresponding to m columns in the index (are highlighted in red dotted rectangle in Fig. 3), and the j-th MAC tag is $\sigma_j = \oplus_{i=1}^{N} MAC(K, E_i[j]||i||j||\mathsf{ver}_i)$. Since MAC tags are aggregated, the size of the checksum will not linearly increase with the number of documents. Finally, DO sets $\mathbb{C} = (E_1, E_2, \ldots, E_N)$ and outsources the encrypted keyword index, $\mathbb{EDB} = (\mathbb{C}, \boldsymbol{\sigma})$, to **CS**, keeps $\mathsf{MK} = (\mathcal{K}, K)$, and publishes $\mathsf{stat} = (k, m, \mathbf{h}, \mathsf{ver}_1, \mathsf{ver}_2, \ldots, \mathsf{ver}_N)$. Note that, $\mathbb{C}$ can be viewed as an $m \times N$ matrix and $\mathbb{EDB}$ can be regarded as an $m \times (N+1)$ matrix. To authorize search permissions to other clients, DO can share MK with them.

Following the membership query algorithm $BF.Query$, if $\wedge_{j=1}^{k} B_i[h_j(w')] = 1$, it means that w' exists in B_i with overwhelming probability, i.e., Doc_i contains w'. In other words, if clients can find a secure way to retrieve k columns of $\mathbb{EDB}$ that are decided by $\mathbf{h}$, securely searching a keyword w' in $\mathbb{EDB}$ is achievable. Considering that clients have $\mathbb{C}[:, h_j(w')]$ for $j = 1$ to k, they can recover $B_i[h_j(w')] = E_i[h_j(w')] \oplus PRF(\mathcal{K}, i||\mathsf{ver}_i)[h_j(w')]$, and can test $\wedge_{j=1}^{k} B_i[h_j(w')] \stackrel{?}{=} 1$ to find out whether w' exists in Doc_i for $i = 1$ to N.

To securely fetch k columns of the encrypted keyword index, clients can run DPF-based PIR k times. In particular, clients can run $DPF.Gen(1^\lambda, h_j(w'), 1) \rightarrow (\mathsf{key}_0, \mathsf{key}_1)$ and send key_b to CS_b $(b \in \{0,1\})$ that scans m columns and returns $\mathsf{res}_b = \oplus_{j=1}^{m} (DPF.Eval(\mathsf{key}_b, j) \cdot \mathbb{EDB}[:, j])$, where $\mathbb{EDB}[:, j] = (E_1[j], E_2[j], \ldots, E_N[j], \sigma_j)$. After receiving all query responses, $\mathbb{EDB}[:, h_j(w')] = \mathsf{res}_0 \oplus \mathsf{res}_1$ can be recovered and its integrity can be verified by checking whether $\mathbb{EDB}[:, h_j(w')][N+1] = \oplus_{i=1}^{N} MAC(K, \mathbb{EDB}[:, h_j(w')][i]||i||h_j(w')||\mathsf{ver}_i)$. For an update, clients can add a new $(\mathsf{ind}, \mathbf{w})$ into $\mathbb{DB}$ by requesting **CS** to insert a new row into $\mathbb{C}$, and the row is the ciphertext of BF-encoded $\mathbf{w}$. Using the similar idea, clients can delete and replace existing (document identifier, keyword set) pairs to update the outsourced



(a) Impact of n on m (b) Impact of k on ϵ ($n = 1,000$)

Fig. 4. BF parameters' impacts on search efficiency and accuracy.

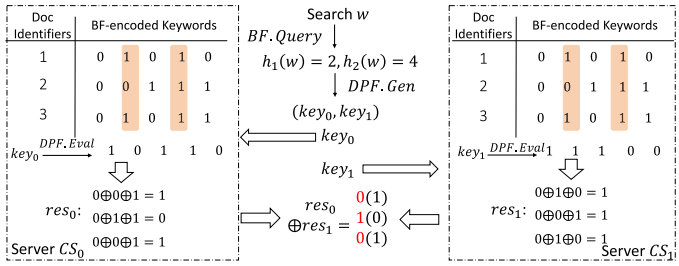


Fig. 5. A toy example of false positives and false negatives caused by using a BF-encoded keyword index (plaintext) with 3 documents: clients want to retrieve the aggregation of 2-th column and the 4-th column (are highlighted in orange color) for searching a keyword, w.

keyword index. At the same time, aggregated MAC tags, $\boldsymbol{\sigma}$, can be updated correspondingly.

Remarks. The baseline design, though is secure and efficient, is not always suitable. As the outsourced keyword index is encoded with a BF, at least k columns need to be retrieved by clients to complete one search, and CS_b ($b \in \{0,1\}$) needs to scan m columns to generate res_b when receiving one DPF-based PIR query. This inevitably leads to heavy computational burden on **CS**: m columns of the outsourced keyword index have to be repeatedly scanned k times for processing one single query. Under certain circumstances, one document may contain a great number of keywords, i.e., n is large. Based on the following equations, $k \times m$ will become explosively large if clients want to maintain false positives at a negligible level:

$$m = \left\lceil -\frac{n \ln \epsilon}{(\ln 2)^2} \right\rceil, \; k = \left\lceil \frac{m}{n} \ln 2 \right\rceil. \tag{4}$$

Fig. 4(a) illustrates how n affects m ($n = 1,000$ to 3,000) and Fig. 4(b) shows that how the number of hash functions affects the false positive rate. When $\epsilon = 10^{-5}$ and $n = 3,000$, we have $k = 17$ and $m = 71,888$. This setting implies that **CS** must scan $k * m = 1,222,096$ columns. Even though DPF only requires lightweight symmetric-key cryptography, the search process is still cumbersome, and the costs are linear.

In addition, the baseline requires DO to share its master symmetric keys, MK, to comply with a multi-client setting. It does not allow DO to set fine-grained search permissions to authorized clients, and thus may not be appropriate for sharing applications where clients own different permissions. In particular, the baseline is not collision-resistant. If any authorized client colludes with malicious **CS**, the whole outsourced keyword index is disclosed, and it can cause even worse results under the malicious model: malicious **CS** can forge any incorrect query results using K without being detected. Aiming at the aforementioned issues, we propose a new multi-client secure and efficient DPF-based keyword search scheme.

B. Design Overview: Differences and Improvements

Three new techniques are exploited to improve the search efficiency compared to the baseline construction. We first borrow the idea of distributed multiple-point function (DMPF) that extends DPF's single-point evaluation to multiple-point evaluation [14]. The intuitive motivation is that, if k points of a function can be evaluated simultaneously via employing *cuckoo hashing*, we can correspondingly design a new secure keyword scheme that just requires *one-time* DPF-based PIR rather than k-*times*. In other words, **CS** only scan m columns once but can obliviously "aggregate" the k columns of the outsourced keyword index associated with a query keyword during the process. With the aggregated responses from **CS**, clients can decrypt to extract query results.

However, the original BF-encoded index does not support oblivious column-based aggregation. Fig. 5 shows an example: if **CS** naively aggregate m columns of the BF-encoded keyword index, i.e., $\mathsf{res}_b = \oplus_{i=1}^{m}(DPF.Eval(\mathsf{key}_b, i) \cdot \mathbb{EDB}[:, i])$, clients will obtain *incorrect* query results. To resolve the challenge, we introduce a different keyword index encoded by *garbled bloom filter*, where each row is an m-length array of ψ-bit binaries. With the GBF-encoded keyword index, **CS** can safely conduct oblivious aggregation to ensure the correctness of the query results.

Furthermore, we adopt a *segmentation* method to speed up the search process at server side. In the baseline, clients generate function shares $(\mathsf{key}_0, \mathsf{key}_1)$ by running $DPF.Gen(1^\lambda, x, y = 1)$ where x is a column number, and each server CS_b ($b \in \{0,1\}$) has to compute $DPF.Eval(\mathsf{key}_b, i)$ for $i = 1$ to m to complete one search. Differently, our scheme enables clients to split m columns into M segments before performing search, and each segment incorporates $\Delta = \lceil \frac{m}{M} \rceil$ columns. Clients can traverse all segments and learn if a segment includes the columns they would like to retrieve, and for that segment, they can execute $DPF.Gen(1^\lambda, x, y)$ where $x \in [1, M]$ is the segment number and $y \in \{0,1\}^\Delta$ is a Δ-bit binary. If multiple segments involve the queried columns, clients can utilize cuckoo hashing and DPF to obliviously aggregate the evaluation results of multiple segments likewise. Fig. 6 gives how the segmentation method works with the parameters, $m = 16$, $M = 4$, and $\Delta = 4$. In this situation, **CS** only perform the DPF evaluation on [1,4] instead of [1,16] ($4\times$ speed up).

The proposed scheme also needs to handle malicious adversaries' cheating behavior, but the general MAC aggregation techniques used in the baseline do not apply to our scheme due to the idea of oblivious aggregation. The reason can be explained by the following example: with two general MAC tags (σ_1, σ_2) of (D_1, D_2) that are constructed using the same key K, clients can easily test the data integrity of (D_1, D_2), respectively. Nevertheless, with general MAC, clients cannot check the data integrity of the aggregated data, $D_1 \oplus D_2$ due to the fact that

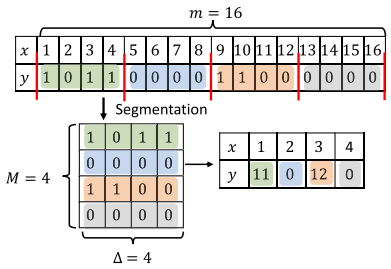


Fig. 6. An example of segmentation: clients split 16 columns (x is the column number) into 4 segments (are highlighted in four colors) for speeding up search.

$MAC(K, D_1 \oplus D_2) \neq \sigma_1 \oplus \sigma_2$ with high probability. Fortunately, *Wegman-Carter MAC* fulfills our requirements due to a unique feature: *supporting homomorphic xor* (addition in $\mathbb{F}_{2^\lambda}$). That is, we have $WCMAC(K, D_1 \oplus D_2, \Gamma_1 \oplus \Gamma_2) = \sigma_1 \oplus \sigma_2$, where $\sigma_i = WCMAC(K, D_i, \Gamma_i)$ for $i = 1, 2$. Therefore, clients can leverage WCMAC to verify the query responses, and we achieve the soundness property in the proposed scheme.

Last but not least, our proposed scheme allows DO to assign fine-grained search permissions of keywords and documents to multiple clients and can resist collusion attacks launched by malicious adversaries control **CS** and a small part of malicious clients. To this end, we first design a double encryption method that enables DO to encrypt (document identifier, keyword set) pairs twice with two access keys derived from master symmetric keys, before outsourcing. By utilizing *set-constrained PRF*, DO can customize clients' search-authorized keys by constraining the access keys and share the customized keys to clients to satisfy different access control on keywords and documents. Since PRF-based one-time pad is applied, access keys for encrypting (document identifier, keyword set) pairs cannot be reused. Hence, the biggest challenge of designing such a double encryption method is how to deal with dynamic update. The update of (document identifier, keyword set) pairs means new access keys should be used, but the usage of new keys should not affect the validity of previously-shared clients' search-authorized keys derived from past access keys. We address the challenge by resorting to a prefix-predicate SCPRF such that clients can self-derive updated keys from their original keys for searching updated databases [15], [27]. Furthermore, inspired by Agrawal et al.'s work [17], we design a MAC-based verification key distribution method based on *cover-free families*. In our method, several WCMAC tags generated by independent MAC keys are bound to each column of the outsourced keyword index. Each client is given a unique subset of all MAC keys that satisfy CFF and can use them to validate the integrity of query results. Due to the properties of CFF, malicious **CS** cannot forge query results even if colluding with a small part of clients.

C. Detailed Scheme Construction

Setup. In Algorithm 1, DO first creates $\chi + 1$ master symmetric keys MK and derives two access keys $\mathcal{K}^{'\text{col}'}$ and $\mathcal{K}^{'\text{row}'}$ (lines 1-7). For each document Doc_i $(i \in [1, N])$, keywords are encoded with GBF and are doubly encrypted (lines 9-16). Correspondingly, an $n \times N$ ciphertext matrix, $\mathbb{C}$, is established, and its integrity is protected through WCMAC (lines 18-31). Particularly, the j-th column of $\mathbb{C}$ is bound with χ aggregated WCMAC tags, $\boldsymbol{\sigma}_j$ for $j = 1$ to m, i.e., $\mathbb{EDB}[:, j] = (E_1[j], E_2[j], \ldots, E_N[j], \sigma_{1,j}, \ldots, \sigma_{\chi,j})$. Finally, DO outsources $\mathbb{EDB}$ to **CS**, keeps MK, and publishes stat.

Algorithm 1: *Setup* Algorithm.

Input: $\mathbb{DB} = (\mathsf{ind}_i, \mathbf{w}_i)_{i=1}^N$, $\mathsf{params} = (n, \lambda, \epsilon, \tau, \psi, \tilde{\Phi}, \zeta, \chi, \nu)$
Output: $\mathbb{EDB}$, MK, stat
1: $\mathcal{K} \in_{\mathcal{R}} \{0,1\}^\lambda$ ▷ choose a master key
2: **for** $t = 1$ to χ **do**
3: $K_t \in_{\mathcal{R}} \mathbb{F}_{2^\lambda}$ ▷ select MAC keys
4: $\hat{\mathcal{K}}_t = PRF(\mathcal{K}, K_t)$ ▷ derive keys for creating randomness
5: **end for**
6: $\mathcal{K}^{'\text{col}'} = PRF(\mathcal{K}, '\text{KEYWORD}')$ ▷ derive keys for columns
7: $\mathcal{K}^{'\text{row}'} = PRF(\mathcal{K}, '\text{DOCUMENTS}')$ ▷ derive keys for rows
8: $(k, m, \mathbf{h}, fp) = GBF.Gen(\epsilon, n)$ ▷ generate GBF encoding parameters
9: **for** $i = 1$ to N **do**
10: $\mathsf{ver}_i = 0$ ▷ initialize the version number of Doc_i to 0
11: $A_i = GBF.Build(\mathbf{w}_i, m, \mathbf{h}, fp)$ ▷ encode keywords with GBF
12: **for** $j = 1$ to m ▷ encrypt rows and columns **do**
13: $E_i[j] = A_i[j] \oplus PRF(\mathcal{K}^{'\text{row}'}, i||\mathsf{ver}_i)[(j-1) * \psi : j * \psi]$
14: $E_i[j] = E_i[j] \oplus PRF(\mathcal{K}^{'\text{col}'}, j||\mathsf{ver}_i)[(i-1) * \psi : i * \psi]$
15: **end for**
16: **end for**
17: $\mathbb{C} = (E_1, E_2, \ldots, E_N)$ ▷ reorganize ciphertexts to an $m \times N$ matrix
18: **for** $t = 1$ to χ **do**
19: **for** $i = 1$ to N **do**
20: **for** $j = 1$ to m **do**
21: $\Gamma_{t,i,j} = PRF(\hat{\mathcal{K}}_t, i||j||\mathsf{ver}_i)$ ▷ generate random nonce
22: $\mathsf{tag}_{t,i,j} = WCMAC(E_i[j], K_t, \Gamma_{t,i,j})$
23: **end for**
24: **end for**
25: **end for**
26: **for** $j = 1$ to m **do**
27: **for** $t = 1$ to χ **do**
28: $\sigma_{t,j} = \oplus_{i=1}^N \mathsf{tag}_{t,i,j}$ ▷ aggregate WCMAC tags
29: **end for**
30: $\boldsymbol{\sigma}_j = (\sigma_{1,j}, \sigma_{2,j}, \ldots, \sigma_{\chi,j})$
31: **end for**
32: construct $\mathbb{I}$ satisfying $\zeta-\text{CFF}(\chi, \nu)$
33: $\boldsymbol{\Sigma} = (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_m)$, $\mathsf{MK} = (\mathcal{K}, K_1, K_2, \ldots, K_\chi)$, $\mathbb{EDB} = (\mathbb{C}, \boldsymbol{\Sigma})$
34: $\mathsf{stat} = (k, m, \mathbf{h}, fp, \mathsf{th}, (1, \mathsf{ver}_i), (2, \mathsf{ver}_i), \ldots, (N, \mathsf{ver}_i), \mathbb{I})$
35: **return** $(\mathbb{EDB}, \mathsf{MK}, \mathsf{stat})$

Algorithm 2: $ClientKeyGen$ Algorithm.

Input: $\mathsf{MK}, \mathbf{ind}^*, \mathbf{w}^*, \mathsf{stat}$
Output: MK^*
1: set num as the current number of authorized clients
2: $\mathsf{num} = \mathsf{num} + 1$ ▷ add a new client
3: $\mathsf{MK}^* = \emptyset$ ▷ initialize the client's search-authorized keys
4: $S^{'\text{col}'} = \emptyset$, $S^{'\text{row}'} = \emptyset$ ▷ initialize authorized keywords and documents
5: **for all** $w \in \mathbf{w}^*$ **do**
6: **for** $j = 1$ to k **do**
7: add $h_j(w)$ into $S^{'\text{col}'}$ ▷ add authorized keywords
8: **end for**
9: **end for**
10: **for all** $\mathsf{ind} \in \mathbf{ind}^*$ **do**
11: add ind into $S^{'\text{row}'}$ ▷ add authorized documents
12: **end for**
13: derive $\mathcal{K}^{'\text{col}'}$ and $\mathcal{K}^{'\text{row}'}$ from $\mathcal{K}$ (lines 6-7 in Algorithm 1)
14: $\hat{\mathcal{K}}^{'\text{col}'} = SCPRF.Cons(\mathcal{K}^{'\text{col}'}, S^{'\text{col}'})$
15: $\hat{\mathcal{K}}^{'\text{row}'} = SCPRF.Cons(\mathcal{K}^{'\text{row}'}, S^{'\text{row}'})$
16: add $(\hat{\mathcal{K}}^{'\text{col}'}, \hat{\mathcal{K}}^{'\text{row}'})$ into MK^* ▷ construct the search keys
17: choose the num-th column from $\mathbb{I}$, i.e., $\mathbb{I}[:, \mathsf{num}]$
18: $\boldsymbol{\mathcal{P}} = \emptyset$
19: **for** $i = 1$ to χ **do**
20: **if** $\mathbb{I}[i, \mathsf{num}] == 1$ **then**
21: add i into $\boldsymbol{\mathcal{P}}$
22: derive $\hat{\mathcal{K}}_i$ from $\mathcal{K}$ (line 4 in Algorithm 1)
23: add $(K_i, \hat{\mathcal{K}}_i)$ into MK^* ▷ construct the verification keys
24: **end if**
25: **end for**
26: add $\boldsymbol{\mathcal{P}}$ into MK^*
27: **return** MK^*

ClientKeyGen. Algorithm 2 gives the procedures of generating a client's search-authorized keys, MK^*. The search-authorized keys contain two types of keys: search keys and verification keys. By constraining the access keys through SCPRF, DO can create specific search keys and adjust clients' decryption capabilities to control their search permissions (lines 4-16). To allow clients to self-derive search keys for updated documents, SCPRF is instantiated using a GGM tree, as the GGM-based PRF construction has a special prefix-predicate property: it enables the calculation of $PRF(\mathcal{K}, x||y)$ based on $PRF(\mathcal{K}, x)$ [15]. Other instantiations of SCPRF can also be applied, as long as they are designed for prefix predicates [27]. In our scheme, x is set to be the column number $i \in [1, m]$ or the row number $j \in [1, N]$ in our scheme, and y is the version number. Given the original search-authorized keys $PRF(\mathcal{K}^{'\text{col}'}, i)$ and $PRF(\mathcal{K}^{'\text{row}'}, j)$, clients can self-derive the updated keys $PRF(\mathcal{K}^{'\text{col}'}, i||\{0,1\}^l)$ and $PRF(\mathcal{K}^{'\text{row}'}, j||\{0,1\}^l)$, where $\{0,1\}^l$ is the bit representation of the version number and l is the bit length. Also, to resist collusion attacks, DO chooses a unique combination of $(K_1, K_2, \ldots, K_\chi)$ satisfying $\zeta-\text{CFF}(\chi, \nu)$ and distributes it to a client to ensure that each client owns a distinctive combination.

Algorithm 3: $CH.Gen^*$ Algorithm.

Input: $M, \Delta, k, 3, \tau, \lambda, \mathbf{pos}, \Phi$
Output: $\hat{m}, \hat{\mathbf{h}}, \mathbf{loc}, \gamma$
1: calculate ι satisfying Theorem 2
2: $\hat{m} = \iota \cdot k$
3: $\delta = \lceil \frac{3M}{\hat{m}} \rceil$
4: $\gamma \in_{\mathcal{R}} \{0,1\}^\lambda$
5: **for** $j = 1$ to 3 ▷ define cuckoo hashing functions **do**
6: $\hat{h}_j(x) = \lfloor PRP(\gamma, x + M \cdot (j-1))/\delta \rfloor$
7: $loc_j(x) = PRP(\gamma, x + M \cdot (j-1)) \bmod \delta$
8: **end for**
9: $\hat{\mathbf{h}} = (\hat{h}_1(x), \hat{h}_2(x), \hat{h}_3(x))$, $\mathbf{loc} = (loc_1(x), loc_2(x), loc_3(x))$
10: **if** $CH.Build(\mathbf{pos}, \mathbf{h}, \Phi) \neq$ 'FAIL' **then**
11: **return** $(\hat{m}, \hat{\mathbf{h}}, \mathbf{loc}, \gamma)$
12: **else**
13: go to step 3
14: **end if**

Search. When clients search one keyword, i.e., $q = w$, they execute Protocol 1. Concretely, clients split m to M segments and encode q into M segments with GBF algorithms (lines 1-17). All non-zero segments' locations, $\mathbf{pos}$ are encoded into a CH-based table, T (lines 18-19). With such a table, clients can construct $\hat{m}$ function shares for two cloud servers that just need to perform 3 M-times DPF evaluation rather than km-times DPF evaluation, and thus the computational costs are significantly reduced (lines 20-39). Different from traditional algorithms for building a CH-based table, we present a modified algorithm (Algorithm 3), $CH.Gen^*$, based on a pseudorandom permutation (PRP) denoted by $PRP : \{0,1\}^\lambda \times [M] \to [\hat{m}]$ [14]. The PRP is employed for generating two types of hash functions: $\hat{\mathbf{h}}$ and $\mathbf{loc}$ that replace the original hash functions used by CH. Finally, search queries, $\mathbf{QR}_b$ where $b \in \{0,1\}$ are sent to CS_b. (lines 40-41).

After receiving clients' search queries, CS_b can unzip segments and obtain M evaluation results, col_i, for $i = 1$ to M (lines 4-6). Each DPF evaluation result is a Δ-bit binary. Using the total $M \cdot \Delta = m$ bits, CS_b can generate an aggregated query response, res_b, and return it back to clients. Clients combine two responses, res_0 and res_1, to recover ciphertext of the matched documents identifiers (line 1), check its integrity (lines 2-14), and finally decrypt it to extract the matched documents identifiers, $\mathbf{IDS}$ (lines 15-23). The decryption results are the membership query results of GBF-encoded keywords with the input of q:

$$\begin{aligned}\mathsf{res}[i] \oplus r_{1,i} \oplus r_{i,2} &= \left(\oplus_{j=1}^{k} E_i[h_j(q)]\right) \oplus r_1 \oplus r_2 \\ &= \oplus_{j=1}^{k} A_i[h_j(q)]. \end{aligned} \tag{5}$$

For $i = 1$ to N, if $\oplus_{j=1}^{k} A_i[h_j(q)] = fp(q)$, it means Doc_i contains the keyword q, and accordingly i is added to $\mathbf{IDS}$.

The search protocol also enables clients to search multiple keywords in one query, $q = (w_1, w_2, \ldots)$, and retrieve the identifiers of the documents that contain all queried keywords

Protocol 1: *Search* Protocol

Clients - **Input:** $(\mathsf{MK}, \mathsf{stat}, q = w)$, **Output:** $(\mathbf{QR}_0, \mathbf{QR}_1)$

$M \in [1, m]$, $\Delta = \lceil \frac{m}{M} \rceil$ ▷ set parameters of segmentation
$\mathbf{\Xi} = \emptyset$, $\hat{i} = 1$
for $i = 1$ to M **do**
  $\mathsf{seg}_i = \{0\}^{\Delta}$ ▷ initialize i-th segment's value to 0
end for
for $j = 1$ to k **do**
  $i = \lfloor \frac{h_j(q)}{\Delta} \rfloor + 1$ ▷ locate segments
  $\mathsf{seg}_i[h_j(q) \bmod \Delta] = 1$ ▷ set queried columns
  if $i \notin \mathbf{\Xi}$ **then**
    add i into $\mathbf{\Xi}$ ▷ count an involved segment
  end if
end for
for all $i \in \mathbf{\Xi}$
  $\mathsf{pos}_{\hat{i}} = i$
  $\mathsf{ele}_{\hat{i}} = \mathsf{seg}_i$, $\hat{i} = \hat{i} + 1$ ▷ set queried values
end for
$\hat{n} = \hat{i} - 1$, $\mathbf{pos} = (\mathsf{pos}_1, \ldots, \mathsf{pos}_{\hat{n}})$
$(\hat{m}, \hat{\mathbf{h}}, \mathbf{loc}, \gamma) = CH.Gen^*(M, \Delta, k, 3, \tau, \lambda, \mathbf{pos}, \Phi)$
$T = CH.Build(\mathbf{pos}, \hat{\mathbf{h}}, \Phi)$ ▷ build a CH-based table
for $i = 1$ to $\hat{m}$ **do** ▷ construct function shares for the table
  if $T[i] = \mathsf{null}$ **then** ▷ set the table's unused entries
    $(\mathsf{key}_{i,0}, \mathsf{key}_{i,1}) = DPF.Gen(1^{\lambda}, 0, 0)$
  else
    for $j = 1$ to 3 **do** ▷ extract matched hash functions' indexes
      if $\hat{h}_j(T[i]) = i$ **then**
        $j' = j$
        break
      end if
    end for
    $\mathsf{index} = loc_{j'}(T[i])$ ▷ calculate DPF's inputs
    for $j = 1$ to $\hat{n}$ **do** ▷ calculate DPF's outputs
      if $T[i] = \mathsf{pos}_j$ **then**
        $\ell = j$
        break
      end if
    end for
    $(\mathsf{key}_{i,0}, \mathsf{key}_{i,1}) = DPF.Gen(1^{\lambda}, \mathsf{index}, \mathsf{ele}_{\ell})$
  end if
end for
$\mathbf{QR}_0 = (M, \hat{m}, \gamma, \mathsf{key}_{1,0}, \mathsf{key}_{2,0}, \ldots, \mathsf{key}_{\hat{m},0})$
$\mathbf{QR}_1 = (M, \hat{m}, \gamma, \mathsf{key}_{1,1}, \mathsf{key}_{2,1}, \ldots, \mathsf{key}_{\hat{m},1})$
return $(\mathbf{QR}_0, \mathbf{QR}_1)$

Cloud Server $CS_b(b = 0, 1)$ - **Input:** $(\mathbb{EDB}, \mathbf{QR}_b)$, **Output:** res_b

$\Delta = \lceil \frac{m}{M} \rceil$, $\mathsf{res}_b = \{0\}^{N*\psi+\lambda*\chi}$ ▷ initialize query responses
for $i = 1$ to M **do**
  $\mathsf{col}_i = \{0\}^{\Delta}$
  for $j = 1$ to 3 **do** ▷ unzip segments with DPF evaluation
    $\mathsf{col}_i = \mathsf{col}_i \oplus DPF.Eval(\mathsf{key}_{\hat{h}_j(i),b}, loc_j(i))$
  end for
  for $j = 1$ to Δ **do** ▷ compute an aggregated query response
    $\mathsf{res}_b = \mathsf{res}_b \oplus (\mathsf{col}_i[j] \cdot \mathbb{EDB}[:, j + M \cdot (i-1)])$
  end for
end for
return res_b

Clients - **Input:** $(\mathsf{res}_0, \mathsf{res}_1, \mathsf{stat}, \mathsf{MK}, q = w)$, **Output: IDS** or 'FAIL'

IDS $= \emptyset$, $\mathsf{res} = \mathsf{res}_0 \oplus \mathsf{res}_1$
for all $t \in \mathcal{P}$ **do**
  for $i = 1$ to N **do**
    $\Gamma_{t,i} = \{0\}^{\lambda}$
    for $j = 1$ to k **do**
      $\Gamma_{t,i} = \Gamma_{t,i} \oplus PRF(\hat{\mathcal{K}}_t, i || h_j(q) || \mathsf{ver}_i)$
    end for
  end for
end for
for all $t \in \mathcal{P}$ **do**
  if $\mathsf{res}[N+t] \neq \oplus_{i=1}^{N} WCMAC(\mathsf{res}[i], K_t, \Gamma_{t,i})$ **then**
    return 'FAIL' ▷ query responses' integrity checking fails
  end if
end for
for $i = 1$ to N **do**
  $r_{1,i} = \oplus_{j=1}^{k} SCPRF.Eval(\hat{\mathcal{K}}'^{\mathrm{row}'}, i || \mathsf{ver}_i)[h_j(q) * \psi : (h_j(q)+1) * \psi]$
  $r_{2,i} = \oplus_{j=1}^{k} SCPRF.Eval(\hat{\mathcal{K}}'^{\mathrm{col}'}, h_j(q) || \mathsf{ver}_i)[(i-1) * \psi : i * \psi]$
  $\mathsf{res}[i] = r_{1,i} \oplus r_{i,2} \oplus \mathsf{res}[i]$
  if $\mathsf{res}[i] = fp(q)$ **then**
    add i into **IDS** ▷ record matched document identifiers
  end if
end for
return IDS

in a communication-efficient way. To achieve the conjunctive query, clients can encode multiple keywords into M segments with GBF algorithms. Encoding multiple keywords is slightly different from encoding a single keyword: clients should remove some repeated queried columns (the number of repetitions is multiplies of 2) when setting queries, since these columns will cancel out each other during the aggregation at server side. With the encoded queries, clients generate a search query for fetching the aggregated evaluation results of all columns associated with queried keywords. When receiving the query responses, clients can check whether $\mathsf{res}[i] \oplus r_{i,1} \oplus r_{i,2} \stackrel{?}{=} fp(w_1) \oplus fp(w_2) \oplus \ldots$ for $i = 1$ to N to obtain the matched documents' identifiers.

Update. DO updates $\mathbb{EDB}$ by instructing **CS** to add, replace, or delete (document identifier, keyword set) pairs and update the cryptographic checksum, $\mathbf{\Sigma}$, correspondingly. Since the update procedures are similar to the baseline scheme, we omit them.

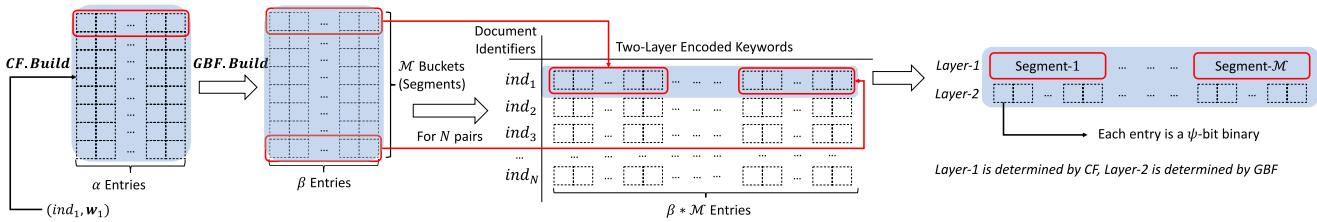


Fig. 7. A high-level illustration of constructing a variant keyword index using two-layer encoding.

D. A Variant Construction With Cuckoo Filter

We also propose a variant construction using *cuckoo filter* that has a better search efficiency in terms of single-keyword search queries ($q = w$) but has to tolerate more downlink overheads and storage costs.

In our first construction, a query keyword is encoded into M segments and the number of involved segments may be much greater than 1. In the variant construction, we build an alternative keyword index such that the number of involved segments can be fixed to 1 to reduce the complexity of DPF evaluation and thus achieve higher search efficiency. Such an index is built on *a two-layer encoding method* sourced from CF and GBF, where the first-layer encoding is a bucket (segment) mapping, and the second-layer encoding is a GBF compression. Different from the first construction, when clients build the keyword index, they need to determine the number of segments and split documents' keywords into multiple segments in advance. In this way, they can guarantee that one query keyword, if it is contained by any document, must locate in one and only one segment after being encoded. Although the design fixes the computational complexity of DPF evaluation on each column to 2 for single-keyword search, it is less communication-efficient for multi-keyword search compared to the first construction, especially when the number of query keywords are large. The reason is that the downlink overheads may linearly increase with the number of query keywords.

As shown in Fig. 7, supposing that clients set the CF parameters as $(\alpha, h, \overline{fp}) = CF.Gen(n, \mathcal{M}, \epsilon_1, \mu)$, they can execute $CF.Build(\alpha, \mathbf{w}_1, \mathcal{M}, h, \overline{fp}, \Phi)$ to encode $\mathbf{w}_1$ into $\mathcal{M}$ buckets (first-layer encoding, $\mathcal{M} \leq n$), $\mathsf{BK} = (\mathsf{BK}[1], \mathsf{BK}[2], \ldots, \mathsf{BK}[\mathcal{M}])$. Similar to a traditional CF, each bucket (highlighted in red lines) has at most α entries, but each entry stores an inserted item in place of the item's fingerprint. Given the GBF parameters, $(k', \beta, \mathbf{h}', fp') = GBF.Gen(\epsilon_2, \alpha)$, clients then perform the second-layer encoding on the i-th bucket, $A'[i] = GBF.Build(\mathsf{BK}[i], \beta, \mathbf{h}', fp')$, from $i = 1$ to $\mathcal{M}$, and each bucket is expanded to β entries. Finally, clients unfold $\mathcal{M}$ segments (red arrow line) to obtain two-layer encoded keywords. All N (document identifier, keyword set) pairs can be encoded using the method to build the keyword index (plaintext). The plaintext keyword index will be encrypted into $\mathbb{EDB}$ using the double encryption, before being outsourcing to **CS**.

In the following, we briefly summarize key procedures of searching a keyword that are different from the first construction. When searching a keyword, $q = w$, clients first locate two possible segments (buckets) $(\mathsf{idx}_1, \mathsf{idx}_2)$: $\rho = \overline{fp}(q)$, $\mathsf{idx}_1 = h(q)$, $\mathsf{idx}_2 = h(\rho) \oplus \mathsf{idx}_1$. Subsequently, clients find the positions of the query keyword in a GBF, $\mathbf{gp}' = (\mathsf{gp}'_1, \ldots, \mathsf{gp}'_{k'})$ where $\mathsf{gp}'_j = h'_j(q)$ for $j = 1$ to k', and generate a β-bit binary, $\mathsf{val} = \{0,1\}^\beta$, where i-th bit ($i \in [1, \beta]$) satisfies

$$\mathsf{val}[i] = \begin{cases} 1 & \text{if } i \in \mathbf{gp}'; \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Clients then generate function shares

$$\begin{aligned} (\mathsf{key}'_{0,1}, \mathsf{key}'_{1,1}) &= DPF.Gen(1^\lambda, x = \mathsf{idx}_1, y = \mathsf{val}), \\ (\mathsf{key}'_{0,2}, \mathsf{key}'_{1,2}) &= DPF.Gen(1^\lambda, x = \mathsf{idx}_2, y = \mathsf{val}), \end{aligned} \tag{7}$$

and send $(\mathsf{key}'_{b,1}, \mathsf{key}'_{b,2})$ to CS_b ($b \in \{0,1\}$) as a single-keyword search query. When receiving the query, CS_b evaluates all $\mathcal{M}$ segments using $\mathsf{key}'_{b,1}$ and $\mathsf{key}'_{b,2}$, aggregates the evaluation results:

$$\begin{aligned} \mathsf{res}_{b,1} = \oplus_{i=1}^{\mathcal{M}} \big(\oplus_{j=1}^{\beta} (DPF.Eval(\mathsf{key}'_{b,1}, i)[j] \\ \cdot \mathbb{EDB}[:, j + \beta \cdot (i-1)])\big), \\ \mathsf{res}_{b,2} = \oplus_{i=1}^{\mathcal{M}} \big(\oplus_{j=1}^{\beta} (DPF.Eval(\mathsf{key}'_{b,2}, i)[j] \\ \cdot \mathbb{EDB}[:, j + \beta \cdot (i-1)])\big), \end{aligned} \tag{8}$$

and returns $(\mathsf{res}_{b,1}, \mathsf{res}_{b,2})$ back. Clients can combine $\mathsf{res}_1 = \mathsf{res}_{0,1} \oplus \mathsf{res}_{1,1}$ and $\mathsf{res}_2 = \mathsf{res}_{0,2} \oplus \mathsf{res}_{1,2}$ and check the equality between the decrypted value of $\mathsf{res}_1[i]$ (resp. $\mathsf{res}_2[i]$) and $\mathsf{fp}'(q)$ for $i = 1$ to N to extract all matched document identifiers. The integrity of the query responses can still be guaranteed using WCMAC, and the logics of updating the variant keyword index is the same as that of the first construction, and hence we elide them.

V. Security Analysis

With the simulation-based paradigm, we formally prove that the proposed scheme is $\mathcal{L}-$adaptively secure under the security of some preliminary primitives. Since proving the security of the first construction and the variant construction is similar, we omit the proof of the latter for conciseness. In short, at the heart of the security analysis is to construct a simulator, $\mathcal{S}$, that is able to use the pre-defined leakage functions to simulate an ideal experiment for a PPT adversary, $\mathcal{A}$, that is computationally indistinguishable from a real experiment.

Theorem 5. When PRF is a secure pseudorandom function, PRP is a secure pseudorandom permutation, and DPF is

Setup: Simulation of *Honest Clients*

Input: $\mathcal{L}^{\mathsf{stp}}(\mathbb{DB}) = (\mathsf{ts}^{\mathsf{stp}}, N, n)$
Output: $\mathbb{EDB}$
for $i = 1$ to N **do**
  $E_i \in_{\mathcal{R}} \{0,1\}^{m*\psi}$
end for
$(k, m, \mathbf{h}, \mathtt{fp}) = \mathtt{GBF.Gen}(\epsilon, n)$
for $t = 1$ to χ **do**
  for $i = 1$ to N **do**
    for $j = 1$ to m **do**
      $\mathsf{tag}_{t,i,j} \in_{\mathcal{R}} \{0,1\}^{\lambda}$
    end for
  end for
end for
for $i = 1$ to N **do**
  $\mathsf{DEL}[i] = \emptyset$
  for $t = 1$ to χ **do**
    for $j = 1$ to m **do**
      add $\mathsf{tag}_{t,i,j}$ into $\mathsf{DEL}[i]$.
    end for
  end for
end for
for $j = 1$ to m **do**
  for $t = 1$ to χ **do**
    $\sigma_{t,j} \oplus_{i=1}^{N} \mathsf{tag}_{t,i,j}$
  end for
  $\boldsymbol{\sigma}_j = (\sigma_{1,j}, \sigma_{2,j}, ..., \sigma_{\chi,j})$
end for
$\boldsymbol{\Sigma} = (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, ..., \boldsymbol{\sigma}_m)$
$\mathbb{EDB} = ((1, E_1), ..., (N, E_N), \boldsymbol{\Sigma})$
return $\mathbb{EDB}$

Search: Simulation of *Honest Clients*

Input: $\mathcal{L}^{\mathsf{srch}}(q) = \mathsf{ts}^{\mathsf{srch}}$
Output: $(\mathbf{QR}_0, \mathbf{QR}_1)$
$M \in [1, m]$, $\Delta = \lceil \frac{m}{M} \rceil$
calculate ι satisfying Theorem 2
$\hat{m} = \iota \cdot k$, $\gamma \in_{\mathcal{R}} \{0,1\}^{\lambda}$
for $i = 1$ to $\hat{m}$ **do**
  $x \in_{\mathcal{R}} [0, \delta]$, $y \in_{\mathcal{R}} [0, M]$, $f(x) = y$
  $(\mathsf{key}_{i,0}, \mathsf{key}_{i,1}) = \mathtt{SIM}(1^{\lambda}, \mathcal{L}^{\mathtt{DPF}}(f))$
end for
end for
$\mathbf{QR}_0 = (M, \hat{m}, \gamma, \mathsf{key}_{1,0}, ..., \mathsf{key}_{\hat{m},0})$
$\mathbf{QR}_1 = (M, \hat{m}, \gamma, \mathsf{key}_{1,1}, ..., \mathsf{key}_{\hat{m},1})$
return $(\mathbf{QR}_0, \mathbf{QR}_1)$

Update: Simulation of *Honest Clients*

Input: $\mathcal{L}^{\mathsf{upd}}(\mathsf{op}, (\mathsf{ind}, \mathbf{w})) = (\mathsf{ts}^{\mathsf{upd}}, \mathsf{op}, \mathsf{ind})$
Output: $(E_{\mathsf{ind}}, \boldsymbol{\Sigma}')$
if op = 'Add'/'Edit' **then**
  $E_{\mathsf{ind}} \in_{\mathcal{R}} \{0,1\}^{m*\psi}$
  for $j = 1$ to m **do**
    for $t = 1$ to χ **do**
      $\mathsf{tag}'_{t,\mathsf{ind},j} \in_{\mathcal{R}} \{0,1\}^{\lambda}$
    end for
    $\boldsymbol{\sigma}'_j = (\mathsf{tag}'_{1,\mathsf{ind},j}, ..., \mathsf{tag}'_{\chi,\mathsf{ind},j})$
  end for
  $\boldsymbol{\Sigma}' = (\boldsymbol{\sigma}'_1, \boldsymbol{\sigma}'_2, ..., \boldsymbol{\sigma}'_m)$
else if op = 'Del' **then**
  $E_{\mathsf{ind}} = \mathsf{null}$, $\boldsymbol{\Sigma}' = \mathsf{DEL}[\mathsf{ind}]$
end if
return $(E_{\mathsf{ind}}, \boldsymbol{\Sigma}')$

Fig. 8. The construction of the simulator, $\mathcal{S}$.

a simulation-secure distributed point function, the proposed keyword search scheme, Π, achieves $\mathcal{L}$-adaptive security.

Proof. The proof proceeds with a sequence of games. We start from the first game that is same as the real experiment, and ends with a game that is the ideal experiment. We show that the advantages that a PPT adversary, $\mathcal{A}$, can distinguish every two consecutive games is negligible.

Game 0: G_0 is exactly same as the real experiment, $\mathsf{REAL}^{\Pi}_{\mathcal{A}}(1^{\lambda})$, i.e.,

$$\Pr[\mathsf{REAL}^{\Pi}_{\mathcal{A}}(1^{\lambda}) = 1] = \Pr[G_0 = 1].$$

Game 1: G_1 is identical to G_0 except that $\mathcal{S}$ picks random binaries from $\{0,1\}^{f_2(\lambda)}$ to replace the outputs of calling one-time pads with keys $\mathcal{K}$, $\mathcal{K}^{'\mathrm{col}'}$, $\mathcal{K}^{'\mathrm{row}'}$. The probability that $\mathcal{A}$ can distinguish G_0 and G_1, is negligible since the probability that $\mathcal{A}$ can distinguish a truly random function and PRF is negligible, namely, we have

$$|\Pr[G_0 = 1] - \Pr[G_1 = 1]| \leq 3\mathsf{Adv}^{PRF}_{\mathcal{A}}(1^{\lambda}).$$

Game 2: G_2 is identical to G_1 except that $\mathcal{S}$ chooses random binaries from $\{0,1\}^{f_2(\lambda)}$ to replace the outputs of calling $WCMAC$, and we have

$$|\Pr[G_1 = 1] - \Pr[G_2 = 1]| \leq \chi \cdot \mathsf{Adv}^{PRF}_{\mathcal{A}}(1^{\lambda}).$$

The reason that $\mathcal{S}$ can make the substitution is that we use one-time pads generated by PRF keys $\mathcal{K}_t$ for $t = 1$ to χ to construct one-time randomness nonce used in WCMAC.

Game 3: G_3 is identical to G_2 except that $\mathcal{S}$ randomly selects a seed of PRP, $\gamma \in_{\mathcal{R}} \{0,1\}^{\lambda}$, when creating T. Note that, the game aborts if $\mathbf{pos}$ fail to be inserted into T by using $(\hat{h}_1, \hat{h}_2, \hat{h}_3)$ that invokes PRP with γ. According to Theorem 2, assuming that $\mathcal{A}$ makes polynomial p queries, the possibility of the game aborts is $\frac{p}{2^{\tau}}$. In other words, we have

$$|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \frac{p}{2^{\tau}}.$$

Game 4: The difference between G_3 and G_4 is that $\mathcal{S}$ substitutes the DPF algorithm, $DPF.Gen(1^{\lambda})$, with the DPF simulator $SIM(1^{\lambda})$ defined in Theorem 3, given the input size $[0, \delta]$ and the output size $[0, M]$ of point functions. So we have

$$|\Pr[G_3 = 1] - \Pr[G_4 = 1]| \leq \mathsf{Adv}^{DPF}_{\mathcal{A}}(1^{\lambda}).$$

Game 5: G_5 is identical to G_4 except that $\mathcal{S}$ makes slight changes to the Setup algorithm, the Search protocol, and the Update protocol. We show how $\mathcal{S}$ modifies them in Fig. 8 to simulate the role of clients. The modifications do not provide any advantage for $\mathcal{A}$ to distinguish the game from G_5, since only redundant steps are removed. That is,

$$\Pr[G_4 = 1] = \Pr[G_5 = 1].$$

Actually, $\mathcal{S}$ simulates the ideal experiment in this game. It is obvious that the inputs of the simulation are the pre-defined leakages, and the outputs of the simulation are the exact outputs of the idea experiment, i.e.,

$$\Pr[G_5 = 1] = \Pr[\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^{\lambda}) = 1].$$

Finally, we have

$$|\Pr[\mathsf{REAL}^{\Pi}_{\mathcal{A}}(1^{\lambda}) = 1] - \Pr[\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(1^{\lambda}) = 1]|$$
$$\leq (\chi + 3) \cdot \mathsf{Adv}^{PRF}_{\mathcal{A}}(1^{\lambda}) + \mathsf{Adv}^{DPF}_{\mathcal{A}}(1^{\lambda}) + \frac{p}{2^{\tau}},$$

which completes the proof.

The soundness of the proposed keyword search scheme can be reduced to the security of WCMAC.

Theorem 6. When $WCMAC$ is secure and no more than ζ clients collude with $\mathcal{A}$, the proposed keyword search scheme, Π, is sound.

Proof. We show that if there exists a PPT adversary, $\mathcal{A}$, can break the soundness property of the proposed keyword search scheme, it can correspondingly construct a PPT algorithm to forge WCMAC tags. Supposing that $\mathcal{A}$ compromises any group of clients and the group size is equal to or smaller than ζ. Without loss of generality, search-authorized keys of

these compromised clients are $(MK_1^*, MK_2^*, \ldots, MK_\phi^*)$ where $\phi \leq \zeta$. The leaked search-authorized keys include part of MAC keys $\mathbf{K}' \subset ((K_1, \hat{\mathcal{K}}_1), (K_2, \hat{\mathcal{K}}_2), \ldots, (K_\chi, \hat{\mathcal{K}}_\chi))$, and according to Definition 9, we have the following property: other uncompromised clients own at least one pair $(K_i, \hat{\mathcal{K}}_i) \notin \mathbf{K}'$ $(i \in [1, \chi])$. It is straightforward that if $\mathcal{A}$ can win $\mathsf{SOUND}_{\mathcal{A}}^{\Pi}(1^\lambda)$, it must be able to extract a valid WCMAC tag generated by using $(K_i, \hat{\mathcal{K}}_i)$. Assuming that $\mathcal{A}$ makes p verification queries about WCMAC tags generated by using $(K_i, \hat{\mathcal{K}}_i)$, according to Theorem 4, as long as $m \leq 2^{\lambda/2}$, we have

$$\Pr[\mathsf{SOUND}_{\mathcal{A}}^{\Pi}(1^\lambda) = 1] \leq \chi \cdot \frac{C\mathcal{N}p}{2^\lambda},$$

which completes the proof.

The security of access control over multiple clients can be reduced to the security of SCPRF.

Theorem 7. When $SCPRF$ is secure, the proposed keyword search scheme, Π, achieves access control.

Proof. We demonstrate that, if a PPT adversary $\mathcal{A}$ can break the access control property and win $\mathsf{ACCESS}_{\mathcal{A}}^{\Pi}(1^\lambda)$ with overwhelming probability, there exists another PPT adversary, $\mathcal{B}$, can use $\mathcal{A}$ as a subroutine to win $\mathsf{GAME}_{\mathcal{B}}^{SCPRF}(1^\lambda)$ with non-negligible probability. There are three cases where $\mathcal{A}$ wins: (1) $\mathsf{ind}' \in Q_1$ and $w' \notin Q_2$; (2) $\mathsf{ind}' \notin Q_1$ and $w' \in Q_2$; and (3) $\mathsf{ind}' \notin Q_1$ and $w' \notin Q_2$. We only analyze Case-1 here, and other cases can be analyzed similarly.

Given the security parameter 1^λ and the SCPRF's key extraction oracle $\mathcal{O}_{\mathcal{K}'^{\text{col}'}}^{SCPRF}$, $\mathcal{B}$ creates $\mathbb{DB}$, chooses $\mathcal{K}'^{\text{row}'}$, generates stat, initializes an empty set Q, and returns N, $\mathbf{W}$, and stat to $\mathcal{A}$. For an authorization query on $(\mathbf{ind}^*, \mathbf{w}^*)$ invoked by $\mathcal{A}$, $\mathcal{B}$ sets $S_w'^{\text{col}'} = (h_1(w), h_2(w), \ldots, h_k(w))$ for all $w \in \mathbf{w}^*$, calculates $\hat{\mathcal{K}}'^{\text{row}'} = SCPRF.Cons(\mathcal{K}'^{\text{row}'}, \mathbf{ind}^*)$, gets $\hat{\mathcal{K}}'^{\text{col}'}$ by querying $\mathcal{O}_{\mathcal{K}'^{\text{col}'}}^{SCPRF}(\cup_{w \in \mathbf{w}^*} S_w'^{\text{col}'})$, adds $\cup_{w \in \mathbf{w}^*} S_w'^{\text{col}'}$ to Q, and returns $(\hat{\mathcal{K}}'^{\text{row}'}, \hat{\mathcal{K}}'^{\text{col}'})$ to $\mathcal{A}$. Later, $\mathcal{A}$ outputs $(\tilde{\mathcal{K}}'^{\text{row}'}, \tilde{\mathcal{K}}'^{\text{col}'})$ for searching $w' \in \mathbf{W}$ and $\mathsf{ind}' \in [1, N]$, and $\mathcal{B}$ can find i satisfying $h_i(w') \cap Q = \emptyset$ $(i \in [1, k])$. $\mathcal{B}$ forwards $h_i(w')$ as a challenge to its own challenger who inputs a random bit $b \in \{0, 1\}$ and returns a response φ satisfying

$$\varphi = \begin{cases} PRF(\mathcal{K}'^{\text{col}'}, h_i(w')) & \text{if } b = 1; \\ \varphi^* \in_{\mathcal{R}} \{0,1\}^{f_2(\lambda)} & \text{if } b = 0. \end{cases}$$

Then, $\mathcal{B}$ checks whether $SCPRF.Eval(\tilde{\mathcal{K}}'^{\text{col}'}, h_i(w')) = \varphi$. If the answer is yes, $\mathcal{B}$ outputs $b' = 1$; otherwise, outputs $b' = 0$. If $\mathcal{A}$ wins, $\tilde{\mathcal{K}}'^{\text{col}'}$ is a valid search key for w', we have $SCPRF.Eval(\tilde{\mathcal{K}}'^{\text{col}'}, h_i(w')) = PRF(\mathcal{K}'^{\text{col}'}, h_i(w'))$, and thus $\mathcal{B}$ can win $\mathsf{GAME}_{\mathcal{B}}^{SCPRF}(1^\lambda)$ with non-negligible probability, which completes the proof.

VI. Performance Evaluation

We implement two constructions of the proposed keyword search scheme using Java, and compare them to the baseline under various experiment settings to show that the proposed scheme can achieve practical performance. Our experiments are conducted on a MAC mini with M1 and 16 GB memory. The developed Java-based prototype is single-thread, but can be extended to a multi-thread version similar to [12]. A multithreading deployment can improve search efficiency by roughly a factor of a processor's real thread number.

Experiment Parameters & Settings. We choose the 128-bit security level for the baseline scheme and the proposed scheme, and accordingly we set $\lambda = \tau = 128$. Two datasets are chosen for extensive experiments: one is a small part of a real Wikipedia dataset that includes 4,000 documents[1], and the other is a synthetic dataset. We use Python and KeyBERT library[2] to preprocess each Wikipedia document and extract keywords, and the maximum number of keywords extracted from a Wikipedia document is set to be 1,000. For the synthetic dataset, $N \in [1, 50000]$ and $n \in [1, 3000]$ are chosen to simulate different various application scenarios.

In the experiments, the false positive of searching one keyword, ϵ, is set to be $(10^{-3}, 10^{-4}, 10^{-5})$. Given n and ϵ, the optimal m and k are calculated based on Eq. 4. Other parameters related to GBF, CH, and CF are set as follows: $\Phi = 1,000$, $\psi = k$ satisfying $2^{-\psi} \approx \epsilon$, $\mu = 0.9$, and $\epsilon_1 = \epsilon_2 = 2^{-\psi}$. According to Theorem 1, the parameter settings can make the comparison fair: the baseline scheme and the proposed scheme have the same false positive rate, ϵ, which is almost negligible.

Under the semi-honest model and the malicious model with/without considering collusion attacks, the performance varies since applying MAC/WCMAC and CFF causes extra costs. To make the performance evaluation results comprehensive, we consider 8 experiment settings for the four phases of secure keyword search: $Setup$, $ClientKeyGen$, $Search$, and $Update$. The settings are denoted by: BL-SEMI [12], BL-MAL [12], OC-SEMI, OC-MAL, OC-MAL-CA, VC-SEMI, VC-MAL, and VC-MAL-CA. Here, BL, OC, and VC denote the baseline, the first construction of our proposed scheme, and the variant construction. We use SEMI and MAL to indicate the experiments under the semi-honest model and the malicious model (extra applying MAC/WCMAC), and the experiments considering collusion attacks are represented by CA (extra applying CFF). For example, OC-MAL-CA means the experiments based on our first construction under the malicious model with considering collusion attacks. In the implementation, we use a deterministic polynomial-based method [16] to construct the cover-free system, and set $\chi = 1$ and $\chi = 25$ for simulating non-collusion/collusion attack cases. The detailed mathematical analysis of the cover-free system's parameters can be found in [28]. When χ is 1, we have $\nu = 1$ and $\zeta = 1$, and one malicious client colludes with a cloud server can break the soundness. When $\chi = 25$, we have $\nu = 125$, and $\zeta = 2$, and our proposed scheme allows DO to authorize 125 clients with unique WCMAC keys and can resist no more than 2 clients colluding with one malicious cloud server.

A. Experiment Results & Analysis

$Setup$ & $Update$ Costs. Since the operations performed by DO are almost the same during $Setup$ and $Update$ (except that

[1] https://dumps.wikimedia.org

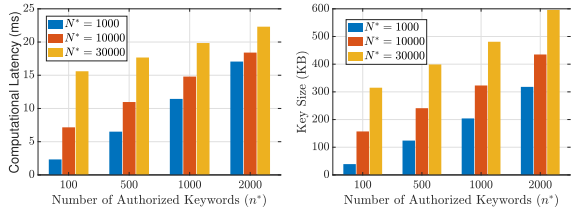
[2] https://github.com/MaartenGr/KeyBERT

TABLE II
COMPUTATIONAL LATENCY OF $Setup$ AND $Update$ (MS) AT DO SIDE: THE FIRST 8 COLUMNS ILLUSTRATE HOW PARAMETER CHANGES AFFECT THE COMPUTATION LATENCY, AND THE LAST COLUMN SHOWS THE COMPUTATION LATENCY OF INITIALIZING A REAL-WORLD DATASET

Maximum Keywords per Document (n)	100	1,000	100	1,000	100	1,000	100	1,000	1,000
Number of Documents (N)	1	1	100	100	1	1	100	100	4,000
False Positive (ϵ)	10^{-3}				10^{-4}				10^{-5}
BL-SEMI	0.31	0.55	1.94	18.81	0.33	1.46	3.79	38.79	668.88
BL-MAL	1.36	6.26	33.41	314.09	1.68	7.25	40.31	381.61	11936.02
OC-SEMI	1.83	7.98	56.27	496.02	2.3	9.35	63.51	617.38	46113.43
OC-MAL	6.45	18.45	109.42	841.78	6.92	23.11	149.03	1219.47	74065.95
OC-MAL-CA	28.82	173.37	1512.51	10341.27	30.49	218.92	1750.05	14467.78	706460.94
VC-SEMI ($\mathcal{M} = n/10$)	1.55	7.21	53.38	480.12	2.3	8.64	60.63	593.78	39539.83
VC-MAL ($\mathcal{M} = n/10$)	6.04	17.32	103.63	800.24	6.3	21.94	135.44	1140.08	69551.31
VC-MAL-CA ($\mathcal{M} = n/10$)	26.88	161.73	1442.61	9221.25	28.14	204.63	1663.01	13040.43	689075.12

TABLE I
STORAGE AND COMMUNICATION (UPDATING) COMPLEXITIES

Settings	Storage	Update
BL-SEMI	mN	m
BL-MAL	$m(N+\lambda)$	$m(\lambda+1)$
OC-SEMI	$mN\psi$	$m\psi$
OC-MAL	$m(N\psi+\lambda)$	$m(\lambda+\psi)$
OC-MAL-CA	$m(N\psi+\chi\lambda)$	$m(\chi\lambda+\psi)$
VC-SEMI	$\eta\mathcal{M}N\psi$	$\eta\mathcal{M}\psi$
VC-MAL	$\eta\mathcal{M}(N\psi+\lambda)$	$\eta\mathcal{M}(\psi+\lambda)$
VC-MAL-CA	$\eta\mathcal{M}(N\psi+\chi\lambda)$	$\eta\mathcal{M}(\psi+\chi\lambda)$



(a) Computational Latency (b) Search Key Size

Fig. 9. The costs of creating a search key for authorizing the search permissions of n^* keywords and N^* documents, given $N = 50,000$, $n = 3,000$, and $\epsilon = 10^{-5}$.

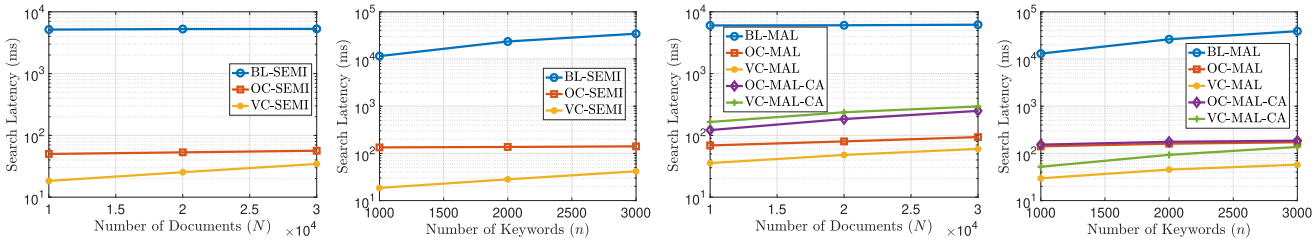
CS need to insert/delete/replace updated rows in the outsourced keyword index meanwhile updating stored MACs/WCMACs at the phase of $Update$), we analyze their performance together. The computational costs of $Setup$ at DO side and the storage costs of the outsourced keyword indexes at server side are determined by N, n, ϵ, and χ (only affects OC and VC). Table II shows how these parameters affect the computational latency. When n, N, and χ are large and ϵ is small, it takes longer time for DO to initialize the system, and the generated outsourced keyword index is large. Compared to the baseline, DO relies on complex encoding methods including GBF and CF to encode keywords, exploits a double encryption method to encrypt the encoded keywords, and uses WCMAC and CFF to achieve verifiability in our scheme. Therefore, more initialization and updating time are required. Considering that $Setup$ only runs once at the beginning, longer latency is acceptable. For updating, DO spends less than 300 ms to generate a request for updating one (document identifier, keyword set) pair, and **CS** take less 5 ms to process an update request. To support oblivious column-based aggregation, our scheme applies the GBF encoding method or the two-layer encoding method that requires around ψ-times larger storage space than the baseline. The storage and communication (updating) complexities of 8 experiment settings are shown in Table I, and we have $\eta = \lceil -\frac{\lceil \frac{n}{\mu\mathcal{M}} \rceil \ln \epsilon}{(\ln 2)^2} \rceil$ that is the segmentation length in VC. When $n = 1,000$ and $\epsilon = 10^{-5}$, the communication costs of updating one (document identifier, keyword set) pair are 386.4 KB for BL-MAL. For OC-MAL and VC-MAL, the costs are 431.4 KB and 522 KB. Obviously, our scheme sacrifices some storage and communication overheads at the phases of $Setup$ and $Update$.

$ClientKeyGen$ **Costs.** In our experiment, authorized keywords and documents are randomly chosen from all keywords and documents. The performance of creating a search key is shown in Fig. 9(a), and the corresponding size of a search key is shown in Fig. 9(b). Since the GGM-based PRF construction is not compact, i.e., the set-constrained property is achieved by puncturing internal nodes in a GGM tree, the costs increase linearly with more authorized keywords and documents. Despite the drawback, it takes less than 25 ms to generate a search key and the size of a search key is smaller than 600 KB. The computational latency of generating a verification key is negligible, as a verification key is a unique combination of all WCMAC keys generated during $Setup$. The size of a verification key is determined by how CFF is constructed, e.g., the average size of a verification key is 768 bits when $\chi = 25$ and $\zeta = 2$.

$Search$ **Costs.** The experiment results of single-keyword search over the real-world dataset are shown in Table III. In the experiment, we categorize the computations of single-keyword search into (1) clients generate search queries (Query), (2) **CS** perform the search (Search), and (3) clients recover query results (Recover). An end-to-end search latency can be obtained by combining them. The experiment confirms that our scheme is more efficient: VC trades more downlink bandwidth for the best computational efficiency at server side; OC is less computation-efficient than VC at server side but has lower downlink bandwidth; and BL has the best communication efficiency at the expense of higher search latency. For instance, compared to BL-SEMI($\epsilon = 10^{-4}$), OC-SEMI($\epsilon = 10^{-4}$) sacrifices only 14% uplink overheads (total 3% communication overheads)

TABLE III
THE PERFORMANCE EVALUATION OF SINGLE-KEYWORD SEARCH WITH THE REAL-WORLD DATASET ($N = 4,000$, $n = 1,000$, $M = \mathcal{M} = n/10$)

Performance	Computation Latency (ms)						Communication Overheads (KB)			
Operations	Query		Search		Recover		Uplink		Downlink	
False Positive (ϵ)	10^{-4}	10^{-5}	10^{-4}	10^{-5}	10^{-4}	10^{-5}	10^{-4}	10^{-5}	10^{-4}	10^{-5}
BL-SEMI	1.7	1.9	11849.1	17397.84	1	1.2	3.864	4.416	7	8
OC-SEMI	19.9	25.6	114.5	122.3	0.55	0.61	4.426	5.344	7	8
VC-SEMI	0.8	0.9	22.5	28.7	0.94	0.98	0.34975	0.364	14	17
BL-MAL	1.7	1.9	13504.9	20259.8	11.2	13.8	3.864	4.416	7.224	8.256
OC-MAL	20	25.6	128.3	134.6	1.6	2.07	4.426	5.344	7.016	8.016
VC-MAL	0.8	0.9	35.2	44.2	3.2	3.3	0.34975	0.364	14.032	17.032
OC-MAL-CA	20	25.6	130.1	137.4	22.9	26.1	4.426	5.344	7.4	8.4
VC-MAC-CA	0.8	0.9	41.2	59.2	37.8	46.2	0.34975	0.364	14.8	17.8



(a) Semi-honest Model (b) Semi-honest Model (c) Malicious Model (d) Malicious Model

Fig. 10. The search latency of the baseline and our scheme ($\epsilon = 10^{-4}$): (a) $n = 500$, (b) $N = 1,000$, (c) $n = 500$, and (d) $N = 1,000$.

TABLE IV
COMMUNICATION OVERHEADS OF SINGLE-KEYWORD SEARCH

Settings	Uplink	Downlink
BL-SEMI	$k(\lambda + \lceil \log_2 m + 1 \rceil (\lambda + 2))$	Nk
BL-MAL		$k(N + \lambda)$
OC-SEMI	$\hat{m}(\lambda + \lceil \log_2 \frac{3M}{\hat{m}} + 1 \rceil (\lambda + 2)$	$N\psi$
OC-MAL	$+\Delta) + \lambda$	$N\psi + \lambda$
OC-MAL-CA		$N\psi + \chi\lambda$
VC-SEMI	$2(\lambda + \lceil \log_2 \mathcal{M} + 1 \rceil (\lambda + 2)$	$2N\psi$
VC-MAL	$+\eta) + \lambda$	$2(N\psi + \lambda)$
VC-MAL-CA		$2(N\psi + \chi\lambda)$

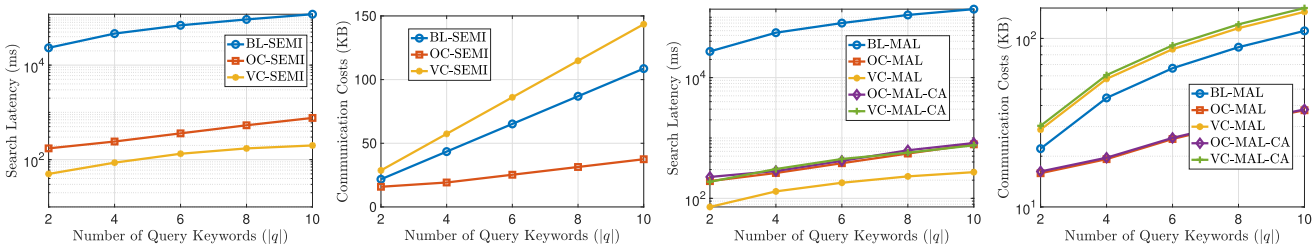
while speeding up the search by more than $100\times$. The communication overhead analysis of the baseline and our scheme during $Search$ is shown in Table IV.

The experiment results also demonstrate that, when false positive is smaller, the effect of search efficiency improvement is more significant. The reason is that, compared to BL, OC and VC reduce the complexities of DPF evaluation at server side from $O(km)$ to $O(3\ M)$ and $O(2\ M)$, and smaller ϵ leads to larger k. Similarly, the effect also applies when M is smaller. The performance of single-keyword search with different numbers of keywords and documents is shown in Fig. 10. It is clear that our scheme outperforms the baseline in terms of the search latency, and we can see that the keyword number has more significant impacts than the number of documents in terms of the search latency. This is because **CS** only perform more lightweight xor operations with the increase number of documents, while the increase number of keywords will cause more heavy DPF evaluation.

The experiment results of multi-keyword conjunctive search are shown in Fig. 11. Compared to BL and VC, OC is more suitable for the applications that require multi-keyword conjunctive search, since it not only provides relatively low search latency but also preserve low communication overheads. The low communication overheads benefit from the design of oblivious aggregation: no matter how many keywords are queried, their associated columns in the outsourced keyword index can be aggregated to fix the downlink overheads (related to the number of documents). The increased communication overheads fully come from the uplink overheads affected by $\hat{m}$ which increases with the number of query keywords. Due to the constant downlink overheads, OC notably outperforms BL when there exist more documents. For example, DO can use OC-MAL to conjunctively search 10 keywords within 800 ms, and the total communication overheads are 37.482 KB. With BL-MAL, DO has to spend 136.24 s to achieve the same goal with the communication costs around 110.88 KB. VC offers the best search efficiency in terms of computation, but its communication overheads, especially downlink overheads, are doubled and increase almost linearly. It may be suitable for some applications that has high communication (downlink) bandwidth but requires low search latency.

VII. RELATED WORK

Secure keyword search for cloud storage has become one of the hot research topics, as clients' concerns about data security and privacy are increasing [3], [4], [8], [26], [29], [30], [31], [32], [33]. One core cryptographic primitive for achieving secure search is searchable encryption (SE), which enables clients to encrypt their data before uploading to cloud servers and also



(a) Semi-honest Model (b) Semi-honest Model (c) Malicious Model (d) Malicious Model

Fig. 11. The performance evaluation of multi-keyword conjunctive search over the real-world dataset ($N = 4,000, n = 1,000, \epsilon = 10^{-4}$).

allow them to search their data without exposing query keywords. It can achieve the basic data confidentiality for clients while satisfying the basic requirements of search. There has been a fruitful line of research on further studying SE from different perspectives. For instance, many symmetric searchable encryption (SSE) have been proposed that combine with sophisticated data structures to provide highly computational and communication efficiency in terms of search [8]. Furthermore, numerous SE variants have been proposed to support various functionalities, including dynamic keyword update, conjunctive query, disjunctive query, boolean query, ranked queries, and range queries [26], [30], [31], [32], [34], [35]. In the meantime, multi-client SE schemes have been proposed to enhance data sharing [3], [4], [33]. Due to the flexibility of key management in public-key settings, most of the previous multi-client schemes are designed using public-key cryptosystems and can support a multi-writer/multi-reader model with fine-grained access control [4], [36], [37]. Different from them, symmetric-key-enabled multi-client keyword search schemes have been proposed and drawn much attention recently [3], [13], [33], [38]. Compared with public-key-enabled schemes, these works are more computation-efficient and suitable for a one-writer/multi-reader model. For instance, based on oblivious cross tag techniques (OXT), Sun et al. firstly proposed a multi-client non-interactive SSE scheme where a data owner can be offline when authorized clients perform search [26], and then Kermanshahi et al. improved their scheme by reducing search costs [33]. Subsequently, Wang et al. [13] and Chamani et al. [38] considered client-server collusion attacks where compromised clients may collude with servers to learn information beyond their permissions. To address the issues, they design new multi-client collusion-resistant SSE schemes by combining oblivious RAM and oblivious Map with classical SSE schemes such as MITRA [21].

Most of the above-mentioned secure search schemes, though are practical for certain applications, suffer from malicious attacks that exploit the leakage of search pattern and access pattern. One of the famous attacks on access pattern is the file-injection attack [39]: if an attacker could inject carefully tailored files into clients' outsourced databases, they can recover clients' queries by observing clients' access pattern. Besides, there are other attacks proposed to explore the leakage of search pattern, and these works show that, even if an adversary just knows statistics of clients' databases, it can still successfully identify clients' query keywords with overwhelming probability [7], [40]. To prevent such attacks, three techniques are usually integrated into secure search schemes: differential privacy (DP) [19], oblivious RAM (ORAM) [41], and private information retrieval (PIR) [5]. These techniques both have advantages and limitations, and we specially focus on PIR in this paper since our work is in line with research on secure keyword search based on *multi-server PIR* [9], [12], [14]. Multi-server PIR (MPIR) is first proposed by Chor et al. [10]. The essential logic behind MPIR is secret sharing: replicated databases are outsourced to multiple cloud servers, and clients' search queries are randomly split into xor-shared binaries based data indexes to hide which data are fetched. The security of MPIR is established on decentralized trust, i.e., at least one of cloud servers cannot be compromised by adversaries. For multi-client scenarios, most of the existing works pay more attention to single-server PIR, while few of them focuses on MPIR [42], [43].

By combining with symmetric cryptosystems, MPIR can be natively generalized to achieve secure keyword search for cloud storage, i.e, the data index outsourced to cloud servers is an encrypted bitmap-based keyword index [44]. Although MPIR-based keyword search schemes are computationally efficient at client side and server side, high communication overheads limit its potential success. Fortunately, Gilboa et al. [11] proposed a novel technique named distributed point function (DPF) that is enhanced by Boyle et al. [9]. DPF-based PIR is attractive and practical compared to traditional MPIR: not only does it keep high computational efficiency, but also it significantly reduces the communication overheads. Similarly, DPF-based PIR can be naturally utilized as a building block for secure keyword search schemes. Gilboa et al. [11] and Boyle et al. [9] briefly discussed how to make such constructions using bitmap-based keyword indexes. Based on their constructions, Dauterman et al. introduced bloom filter data structures into secure DPF-based keyword search. By doing so, their proposed scheme can greatly reduce computational costs, communication overheads, and storage costs, since outsourced keyword indexes are compressed into equal-length BF-encoded keyword indexes [12]. To detect malicious adversaries' cheating behavior, they applied aggregate MAC techniques to DPF-based keyword search to provide verifiability for query responses. The bloom filter-based keyword index (a.k.a. one-hot index) has been utilized as a core component to design a private search scheme for time-series databases to achieve complex range queries [45]. To support

expressive search like boolean queries, Falk et al. further combined multi-party computation and DPF-based search, which is more flexible compared to single-keyword search [46].

VIII. Conclusion

We have proposed a multi-client secure and efficient keyword search scheme for cloud storage services. In our scheme, with encoding methods derived from garbled bloom filter and cuckoo filer, novel keyword indexes have been constructed to offer high search efficiency in terms of computation and communication overheads. To enable high security guarantees in multi-client environments, we have designed a double encryption method, put forward an authorization algorithm based on SCPRF, and applied WCMAC with cover-free systems. In addition, we have implemented a proof-of-concept prototype to demonstrate our scheme's practicality. For the future work, we will further explore DPF-based expressive keyword search such as ranked search, and investigate the opportunity to achieve multi-client efficient and leakage-free keyword search based on single-server PIR.

References

[1] X. Shen et al., "Data management for future wireless networks: Architecture, privacy preservation, and regulation," *IEEE Netw.*, vol. 35, no. 1, pp. 8–15, Jan./Feb. 2021.

[2] H. Cui, X. Yuan, and C. Wang, "Harnessing encrypted data in cloud for secure and efficient mobile image sharing," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1315–1329, May 2017.

[3] S.-F. Sun et al., "Non-interactive multi-client searchable encryption: Realization and implementation," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 452–467, Jan./Feb. 2022.

[4] Y. Miao, R. H. Deng, X. Liu, K.-K. R. Choo, H. Wu, and H. Li, "Multi-authority attribute-based keyword search over encrypted cloud data," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1667–1680, Jul./Aug. 2021.

[5] S. G. Choi, D. Dachman-Soled, S. D. Gordon, L. Liu, and A. Yerukhimovich, "Compressed oblivious encoding for homomorphically encrypted search," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2021, pp. 2277–2291.

[6] E. Stefanov et al., "Path ORAM: An extremely simple oblivious RAM protocol," *J. ACM*, vol. 65, no. 4, pp. 1–26, 2018.

[7] S. Oya and F. Kerschbaum, "Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption," in *Proc. USENIX Secur. Symp.*, 2021, pp. 127–142.

[8] Z. Gui, K. G. Paterson, and S. Patranabis, "Rethinking searchable symmetric encryption," *Proc. IEEE Secur. Privacy*, May 22/25, 2023.

[9] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1292–1303.

[10] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[11] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *Proc. 33rd Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2014, pp. 640–658.

[12] E. Dauterman, E. Feng, E. Luo, R. A. Popa, and I. Stoica, "DORY: An encrypted search system with distributed trust," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2020, pp. 1101–1119.

[13] Y. Wang and D. Papadopoulos, "Multi-user collusion-resistant searchable encryption with optimal search time," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2021, pp. 252–264.

[14] L. de Castro and A. Polychroniadou, "Lightweight, maliciously secure verifiable function secret sharing," in *Proc. 41st Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2022, pp. 150–179.

[15] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[16] R. Kumar, S. Rajagopalan, and A. Sahai, "Coding constructions for blacklisting problems without computational assumptions," in *Proc. 19th Annu. Int. Cryptol. Conf.*, 1999, pp. 609–623.

[17] S. Agrawal and D. Boneh, "Homomorphic MACs: Mac-based integrity for network coding," in *Proc. 7th Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2009, pp. 292–305.

[18] [Online]. Available: https://github.com/EnderCheng/KeywordSearch

[19] Z. Shang, S. Oya, A. Peter, and F. Kerschbaum, "Obfuscated access and search patterns in searchable encryption," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[20] X. Wang, J. Ma, X. Liu, Y. Miao, Y. Liu, and R. H. Deng, "Forward/backward and content private dsse for spatial keyword queries," *IEEE Trans. Dependable Secure Comput*, to be published, doi: 10.1109/TDSC.2022.3205670.

[21] J. G. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 1038–1055.

[22] R. Bost, "Σο φ ος: Forward secure searchable encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.

[23] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets Big Data: An efficient and scalable protocol," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2013, pp. 789–800.

[24] J. Katz and A. Y. Lindell, "Aggregate message authentication codes," in *Proc. Cryptographers Track RSA Conf.*, 2008, pp. 155–169.

[25] D. J. Bernstein, "Stronger security bounds for Wegman-Carter-Shoup authenticators," in *Proc. 24th Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2005, pp. 164–180.

[26] S.-F. Sun et al., "Practical non-interactive searchable encryption with forward and backward privacy," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[27] A. Davidson, S. Katsumata, R. Nishimaki, S. Yamada, and T. Yamakawa, "Adaptively secure constrained pseudorandom functions in the standard model," in *Proc. 40th Annu. Int. Cryptol. Conf.*, 2020, pp. 559–589.

[28] G. Hartung, B. Kaidel, A. Koch, J. Koch, and A. Rupp, "Fault-tolerant aggregate signatures," in *Proc. 19th IACR Int. Conf. Pract. Theory Public-Key Cryptogr.*, 2016, pp. 331–356.

[29] M. Shen, B. Ma, L. Zhu, X. Du, and K. Xu, "Secure phrase search for intelligent processing of encrypted data in cloud-based IoT," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1998–2008, Apr. 2019.

[30] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Proc. 36th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2017, pp. 94–124.

[31] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption with forward and stronger backward privacy," in *Proc. 24th Eur. Symp. Res. Comput. Secur.*, 2019, pp. 283–303.

[32] C. Zuo, S. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and L. Xu, "Forward and backward private DSSE for range queries," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 328–338, Jan./Feb. 2022.

[33] S. K. Kermanshahi et al., "Multi-client cloud-based symmetric searchable encryption," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2419–2437, Sep./Oct. 2021.

[34] W. Lin, H. Cui, B. Li, and C. Wang, "Privacy-preserving similarity search with efficient updates in distributed key-value stores," *IEEE Trans Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1072–1084, May 2021.

[35] Y. Miao, W. Zheng, X. Jia, X. Liu, K.-K. R. Choo, and R. Deng, "Ranked keyword search over encrypted cloud data through machine learning method," *IEEE Trans. Serv. Comput.*, vol. 16, no. 1, pp. 525–536, Jan./Feb. 2023.

[36] Y. Zhang, C. Xu, J. Ni, H. Li, and X. Shen, "Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1335–1348, Fourth Quarter 2021.

[37] L. Xu, X. Yuan, R. Steinfeld, C. Wang, and C. Xu, "Multi-writer searchable encryption: An lwe-based realization and implementation," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2019, pp. 122–133.

[38] J. Gharehchamani, Y. Wang, D. Papadopoulos, M. Zhang, and R. Jalili, "Multi-user dynamic searchable symmetric encryption with corrupted participants," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 114–130, Jan./Feb. 2023.

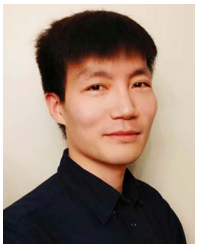
[39] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. USENIX Secur. Symp.*, 2016, pp. 707–720.

[40] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1223–1240.

[41] S. Garg, P. Mohassel, and C. Papamanthou, "TWORAM: Efficient oblivious ram in two rounds with applications to searchable encryption," in *Proc. 36th Annu. Int. Cryptol. Conf.*, 2016, pp. 563–592.
[42] W. Lueks and I. Goldberg, "Sublinear scaling for multi-client private information retrieval," in *Proc. 19th Int. Conf. Financial Cryptogr. Data Secur.*, 2015, pp. 168–186.
[43] W. Cheng, D. Sang, L. Zeng, Y. Wang, and A. Brinkmann, "Tianji: Securing a practical asynchronous multi-user ORAM," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2023.3241184.
[44] X. Zhang, C. Huang, Y. Su, J. Qin, and X. Shen, "Divertible searchable symmetric encryption for secure cloud storage," in *Proc. IEEE Glob. Commun. Conf.*, 2022, pp. 3785–3790.
[45] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica, "Waldo: A private time-series database from function secret sharing," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 2450–2468.
[46] B. H. Falk, S. Lu, and R. Ostrovsky, "DURASIFT: A robust, decentralized, encrypted database supporting private searches with complex policy controls," in *Proc. 18th ACM Workshop Privacy Electron. Soc.*, 2019, pp. 26–36.

Cheng Huang (Member, IEEE) received the PhD degree in electrical and computer engineering, from the University of Waterloo, ON, Canada in 2020. He is currently a postdoctoral research fellow with the Department of Electrical and Computer Engineering, University of Waterloo. His research interests are in the areas of security and privacy in mobile networks, databases, and blockchain.

Dongxiao Liu (Member, IEEE) received the PhD degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada in 2020. He is a postdoctoral research fellow with the Department of Electrical and Computer Engineering, University of Waterloo. His research interests include security and privacy in intelligent transportation systems, blockchain, and mobile networks.



Anjia Yang (Member, IEEE) received the PhD degree in department of Computer Science from the City University of Hong Kong in 2015. He is currently an associate professor with Jinan University, Guangzhou. His research interests include security and privacy in Internet of Things, vehicular networks, blockchain and cloud computing, etc. He has published more than 40 international papers including journals and conferences.

Rongxing Lu (Fellow, IEEE) received the PhD degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2012. He worked as a postdoctoral fellow with the University of Waterloo from May 2012 to April 2013. He is currently a mastercard IoT research chair, a University research scholar, and an associate professor with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an assistant professor with the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore, from April 2013 to August 2016. He has published extensively in his areas of expertise. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security, and privacy. He was a recipient of nine best (student) paper awards from some reputable journals and conferences. He was awarded the most prestigious "Governor General's Gold Medal," when he received his PhD degree. He has won the Eighth IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award in 2013. He also serves as the chair of IEEE Communications and Information Security Technical Committee (ComSoc CIS- TC), and the Founding co-chair of IEEE TEMS Blockchain and Distributed Ledgers Technologies Technical Committee (BDLT-TC). He is the Winner of 2016–2017 Excellence in Teaching Award, FCS, UNB.

Xuemin Shen (Fellow, IEEE) received the PhD degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is a University professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research focuses on network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular ad hoc and sensor networks. He is a registered professional engineer of Ontario, Canada, an Engineering Institute of Canada fellow, a Canadian Academy of Engineering fellow, a Royal Society of Canada fellow, a Chinese Academy of Engineering foreign member, and a distinguished lecturer of the IEEE Vehicular Technology Society and Communications Society. He received the Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, the R.A. Fessenden Award in 2019 from IEEE, Canada, Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society, and Technical Recognition Award from Wireless Communications Technical Committee (2019) and AHSN Technical Committee (2013). He has also received the Excellent Graduate Supervision Award in 2006 from the University of Waterloo and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He served as the Technical Program Committee chair/co-chair for IEEE Globecom' 16, IEEE Infocom'14, IEEE VTC'10 Fall, IEEE Globecom'07, and the chair for the IEEE Communications Society Technical Committee on Wireless Communications. He is the president of the IEEE Communications Society. He was the vice president for Technical & Educational Activities, vice president for Publications, member-at-large on the Board of Governors, chair of the Distinguished Lecturer Selection Committee, member of IEEE Fellow Selection Committee of the ComSoc. He served as the editor-in-chief for *IEEE IoT JOURNAL*, *IEEE Network*, and *IET Communications*.