Published in IEEE Transactions on Computers, 2023, 72 (11). DOI: 10.1109/TC.2023.3285103

Privacy-Preserving Bloom Filter-Based Keyword Search Over Large Encrypted Cloud Data

Yanrong Liang, Jianfeng Ma, *Member, IEEE*, Yinbin Miao, *Member, IEEE*, Da Kuang, Xiangdong Meng, and Robert H. Deng, *Fellow, IEEE*

***Abstract*—To achieve the search over encrypted data in cloud server, Searchable Encryption (SE) has attracted extensive attention from both academic and industrial fields. The existing Bloom filter-based SE schemes can achieve similarity search, but will generally incur high false positive rates, and even leak the privacy of values in Bloom filters (BF). To solve the above problems, we first propose a basic Privacy-preserving Bloom filter-based Keyword Search scheme using the Circular Shift and Coalesce-Bloom Filter (CSC-BF) and Symmetric-key Hidden Vector Encryption (SHVE) technology (namely PBKS), which can achieve effective search while protecting the values in BFs. Then, we design a new index structure T-CSCBF utilizing the Twin Bloom Filter (TBF) technology. Based on this, we propose an improved scheme PBKS+, which assigns a unique inclusion identifier to each position in each BF with privacy protection. Formal security analysis proves that our schemes are secure against Indistinguishability under Selective Chosen-Plaintext Attack (IND-SCPA), and extensive experiments using real-world datasets demonstrate that our schemes are feasible in practice.**

***Index Terms*—Searchable symmetric encryption, bloom filter-based keyword search, circular shift and coalesce-bloom filter, symmetric-key hidden vector encryption.**

I. Introduction

With the development of cloud computing, more and more enterprises and individuals choose to outsourced their data to the cloud server to reduce the local storage and computing costs. To further avoid the privacy leakage, all data should be encrypted before being outsourced to the cloud server. However, this will prevent the cloud server from efficiently searching over encrypted data. Searchable Encryption (SE) has attracted wide attention, which allows data users to search ciphertext according to the query keyword. Among the index structure in SE, the index structure of Bloom Filter (BF)-based is efficient, which can achieve similarity search. Therefore, we focus on the BF-based keyword search schemes [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. However, the BF-based keyword search schemes still have some challenges.

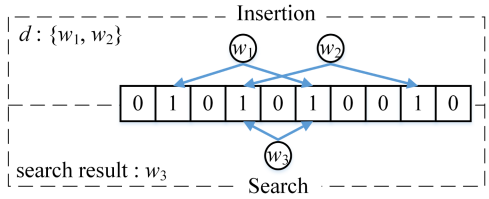


Fig. 1. The insecure and inaccurate of BF.

Challenge 1. How to construct a BF-based index structure that reduces the false positive rate and achieves efficient search: On the one hand, in the existing BF-based keyword search schemes, the different keywords may be mapped to the same position in BF, which brings the problem of high false positive rates. As shown in Fig. 1, BF only contains w_1 and w_2. However, in the search phase, w_3 will be mistakenly considered to be included in the BF. On the other hand, in existing BF-based keyword search schemes, each data needs to build a BF. All BFs need to be traversed in the search phase. Therefore, as the number of data increases, the search time will increase. In addition, if all data is mapped to one BF, as the number of keywords increases by K times, we need to increase the length of BF by the same multiple K to keep the false positive rate unchanged [14], which also increase the storage cost. At the same time, this operation also does not achieve the efficient search because it is impossible to determine which data the keyword is included in.

Challenge 2. How to provide privacy protection for the BF-based index structure while supporting efficient search: BF can conveniently implement efficient plaintext search. However, as shown in Fig. 1, uploading the unencrypted BF to the cloud server will directly leak the values of BF. Therefore, most of the existing BF-based SE schemes [1], [2], [8] use the secure k-Nearest Neighbors (kNN) [15] to encrypt BF, but the secure kNN has been proved to not resistant to Know Plaintext Attacks (KPA) [16]. The schemes [5], [9], [10] use the Twin Bloom Filter (TBF) [10] to confuse the values in BF. However, as the

Manuscript received 6 November 2022; revised 23 February 2023; accepted 28 May 2023. Date of publication 12 June 2023; date of current version 10 October 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3101100, in part by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant 62121001, in part by the National Natural Science Foundation of China under Grant 62072361, in part by the Key Research and Development Program of Shaanxi Province under Grant 2022GY-019, in part by the Henan Key Laboratory of Network Cryptography Technology under Grant LNCT2020-A06, and in part by the Fundamental Research Funds for the Central Universities under Grant ZYTS23166. Recommended for acceptance by D. B. Whalley. *(Corresponding author: Yinbin Miao.)*

Yanrong Liang, Jianfeng Ma, Yinbin Miao, and Da Kuang are with the School of Cyber Engineering, Xidian University, Xi'an 710071, China (e-mail: liangyr163@163.com; jfma@mail.xidian.edu.cn; ybmiao@xidian.edu.cn; 1043096175@qq.com).

Xiangdong Meng is with the Henan Key Laboratory of Network Cryptography Technology, Zhengzhou 450001, China (e-mail: xiangdong129@163.com).

Robert H. Deng is with the School of Information Systems, Singapore Management University, Singapore 188065 (e-mail: robertdeng@smu.edu.sg).

Digital Object Identifier 10.1109/TC.2023.3285103

number of searches increases, the cloud server will capture the real values in BF according to statistical location information, and TBF will cause double storage cost. The above schemes can achieve efficient search, but cannot provide enough privacy protection. The BF-based SE schemes [11], [12] use the traditional public-key encryption to achieve secure search, but it will incur large search cost due to the bilinear pairing. For example, the schemes [17], [18] use Homomorphic Encryption (HE) to protect privacy, which will incur high storage and computation costs.

As mentioned above, the existing BF-based SE schemes still suffer from high false positive rates and costs regarding storage and computation. Therefore, we first propose a basic Privacy-preserving Bloom filter-based Keyword Search scheme using the Circular Shift and Coalesce-Bloom Filter (CSC-BF) and Symmetric-key Hidden Vector Encryption (SHVE) technology (namely PBKS), which can achieve secure and efficient search without high false positive rates. In the search phase, although the values in BF are encrypted by SHVE, the values of the positions that successfully match must be 1 in BF, otherwise 0. Thus, as the number of searches increases, the values in BF will still be leaked to the cloud server. Therefore, we design a new index structure Twin-Circular Shift and Coalesce Bloom Filter (T-CSCBF) to enhance PBKS scheme and propose a PBKS+ scheme, which assigns a unique inclusion identifier to each position in each BF and ensures that the values in BF will not be leaked after search. The main contributions of our work are as follows:

- *W*e propose a basic privacy-preserving Bloom filter-based keyword search scheme PBKS using CSC-BF and SHVE. Specifically, we store all keywords by CSC-BF. Then, we encrypt CSC-BF by using SHVE to effectively realize keyword search.
- *T*o further protect the values in CSC-BF from being leaked after search, we design a new index structure T-CSCBF using TBF, which assigns the unique inclusion identifier to each position in BF to break the association of inclusion identifiers at different positions.
- *W*e give the security analysis and performance analysis. The security analysis proves that our PBKS schemes are secure under Selective Chosen-Plaintext Attacks (SCPA). The experiments on real datasets show that our PBKS schemes are efficient in the search phase.

The rest of this paper is organized as follows. Section II presents the related work. Section III reviews the preliminaries involved in our schemes. Section IV gives the specific system model & threat model, problem definition, scheme definition and security model of the work. Section V describes two specific schemes in detail. Sections VI and VII analyze the security and performance of our proposed schemes, respectively. Section VIII concludes this paper.

II. Related Work

Searchable Encryption: SE can implement the search in encrypted data without leaking privacy. Therefore, SE can be used to search encrypted data outsourced to the cloud server. Song et al. [19] proposes a searchable symmetric encryption scheme for the first time, which opens the research in the SE field. At present, according to different demands, researchers have proposed various SE schemes. The schemes [20], [21], [22] combine SE with the Attribute-Based Encryption (ABE). ABE can be used to achieve fine-grained access control over encrypted data without privacy leakages. The schemes [1], [2], [5] subdivide the keywords, and use BF and Locality Sensitive Hashing (LSH) to achieve fuzzy keyword search. When a keyword is misspelled, fuzzy keyword search can achieve search as accurately as possible. To prevent the malicious cloud servers from returning incorrect search results driven by evasive calculations or other interest incentives, the schemes [1], [5], [23] realize the integrity verification of the search results. In addition, the schemes [8], [24], [25] transform two-dimensional spatial data into one-dimensional using gray code, hilbert curve or curve fitting technique, and realize double search of space and text at the same time. The schemes [26], [27], [28] extract the feature vector of the image to construct the index, and realize the search of the encrypted image data.

Bloom Filter-Based Searchable Encryption: Given n sets $S_0, \ldots, S_{n-1}$, *Multi-Set Multi-Membership Querying* (MS-MMQ) can answer which sets contain query element q. Currently, the more popular MS-MMQ solutions are BIGSI [14], RAMBO [29] and Circular Shift and Coalesce-Bloom Filter (CSC-BF) [30]. This three solutions are all implemented with the help of BF. CSC-BF is improved on the BIGSI and RAMBO. In one query of CSC-BF, the number of hash operations is independent of the number of sets, which can achieve efficient storage and computation. The set in MS-MMQ can be regarded as the file, and then the elements in a set can be regarded as the keywords contained in a file. By constructing a BF for each file as the index, the MS-MMQ can implement keyword search. The SE scheme that uses BF to achieve search is called Bloom Filter-based Searchable Encryption (BF-based SE) scheme [1], [2], [5], [11], [12], [13], which can achieve more efficient search than other SE schemes because of the high efficiency of BF. The schemes [1], [5] use n-gram and LSH to generate BF, and use kNN to encrypt the BF to achieve fuzzy keyword search. ESVSSE [13] uses a pseudo-random function to encrypt keyword and generate trapdoor to initiate search. These schemes suffer from the same problem as BIGSI. That is, when a large number of ciphertext are outsourced in the cloud server, the search efficiency will decrease because all ciphertexts have to be traversed. In order to improve the search efficiency, Chen et al. [2] used the inverted index and realizes fuzzy keyword search, in which the number of keywords is much smaller than that of ciphertexts. In addition, the above schemes have shortcomings in protecting the values of BF, which will leak the privacy of BF to the cloud server. The schemes [11], [12] use the public-key encryption to protect the privacy of BF. However, in the search phase, the bilinear pairing operation will incur high time cost. Due to the large computation and storage costs caused by HE, the schemes [17], [18] also face the problem of low search efficiency.

A comparative summary of PBKSs and the existing schemes is presented in Table I.

TABLE I
COMPARISON BETWEEN PREVIOUS SCHEMES AND TWO PBKS SCHEMES

Schemes	Match type	Search type	MS-MMQ solution	Search complexity	Hidden the value of BF	Secure type
[1]	Fuzzy	Multi	BIGSI	$O(n)$	×	KCA
[2]	Fuzzy	Multi	BIGSI	$O(w)$	×	KBA
[5]	Fuzzy	Single	BIGSI	$O(\log n)$	×	IND-CKA
[11]	Exact	Multi	BIGSI	$O(\log n)$	×	IND-SCPA
[12]	Exact	Multi	BIGSI	$O(\log n)$	×	IND-SCPA
PBKS	Exact	Single	CSC-BF	$O(b)$	×	IND-SCPA
PBKS+	Exact	Single	CSC-BF	$O(b)$	✓	IND-SCPA

– n: number of data; w: number of keywords; b: number of partitions in CSC-BF.
– KCA: semantic security under the known ciphertext attack; KBA: semantic security under the known background attack; IND-CKA: semantic security against chosen keyword attack; IND-SCPA: semantic security against selective chosen-plaintext attack.

III. PRELIMINARIES

In this section, we review the relevant background knowledge used in this paper, including Twin Bloom Filter (TBF) [10], Circular Shift and Coalesce-Bloom Filter (CSC-BF) [30] and Symmetric-key Hidden Vector Encryption (SHVE) [31].

A. Twin Bloom Filter

Twin Bloom Filter (TBF) [10] is an extension of the Bloom Filter, which protects the privacy of the BF by obfuscating the values in BF. TBF consists of two m-bit BFs side-by-side, and any two cells in the same positions of two BFs are called a twin. Each twin is divided into one chosen cell and one unchosen cell containing 0 or 1, respectively. TBF contains four algorithms: TBF.Setup, TBF.Initi, TBF.Insert and TBF.Check.

- *TBF.Setup*$(1^\eta, k) \rightarrow \{H_{key}, \mathcal{H}\}$*:* Generate the keyed-hash function family $H_{key} = \{h_1, \ldots, h_{k+1}\}$ and a hash function $\mathcal{H}(x) = \mathbb{H}(x)\%2$ with a range of [0, 1], where $\mathbb{H}$ is a random oracle.
- *TBF.Initi*$(m, H_{key}, \mathcal{H}, \gamma) \rightarrow$ *TBF:* Calculate the $\mathcal{H}(h_{k+1}(i) \oplus \gamma)$ to determine which cell in each twin is chosen, where γ is a random number and $i \in \{0, m-1\}$.
- *TBF.Insert*(TBF, W, H_{key}, $\mathcal{H}$, γ) $\rightarrow$ *TBF:* For each keyword $w \in W$, let TBF[$(h_i(w)$][$\mathcal{H}(h_{k+1}(h_i(w)) \oplus \gamma)$] = 1 and TBF[$(h_i(w)$][1 - $\mathcal{H}(h_{k+1}(h_i(w)) \oplus \gamma)$] = 0.
- *TBF.Check*(TBF, w', γ) $\rightarrow$ *0/1:* Test whether TBF[$(h_i(w')$][$\mathcal{H}(h_{k+1}(h_i(w')) \oplus \gamma)$] = 1 for query keyword w'. If the values of k chosen cells are 1, w' is included in TBF; otherwise, not included.

A TBF is correct if for all parameters η, k, m, γ, and any keyword $w \in W$, after running TBF.Setup(1^η, k) $\rightarrow$ {H_{key}, $\mathcal{H}$}, TBF.Initi(m, H_{key}, $\mathcal{H}$, γ) $\rightarrow$ TBF, TBF.Insert(TBF, W, H_{key}, $\mathcal{H}$, γ) $\rightarrow$ TBF, if $w' \in W$, then TBF.Check(TBF, w', γ) = 1, otherwise

$$Pr[\mathsf{TBF.Check}(\mathrm{TBF}, w', \gamma) = 0] = 1 - \mathsf{negl}_{\mathrm{TBF}}(m, k, |W|),$$

where the false positive rate of TBF is equal to that of traditional BF, i.e., $\mathsf{negl}_{\mathrm{TBF}}(m, k, |W|) = \mathsf{negl}_{\mathrm{BF}}(m, k, |W|) \approx (1 - (1 - \frac{1}{m})^{k|W|})^k \approx (1 - e^{-k|W|/m})^k$. Fig. 2 shows an example when $m = 8$ and $k = 2$. It can be seen that TBF not only obfuscates the insertion position but also achieves efficient search.

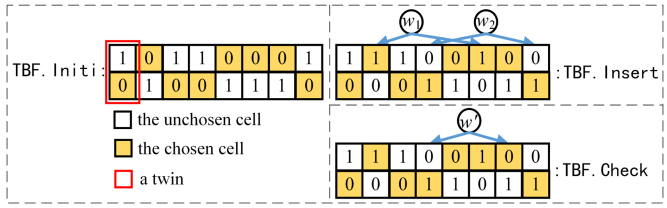


Fig. 2. Example of TBF.

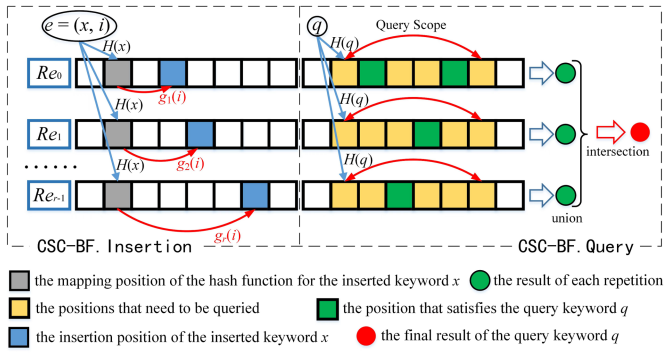


Fig. 3. The Insertion and Query of CSC-BF.

B. Circular Shift and Coalesce-Bloom Filter

Based on the Bloom filter, Circular Shift and Coalesce-Bloom Filter (CSC-BF) [30] can solve the problem of *Multi-Set Multi-Membership Querying* (MS-MMQ), i.e., determining which sets contain the query element for n sets $S_0, \ldots, S_{n-1}$. The current methods for solving the MS-MMQ problem still have some disadvantages, for example, BIGSI [14] suffers from the low accuracy and RAMBO [29] has the low search efficiency. However, CSC-BF can balance the accuracy and efficiency. CSC-BF consists of r m-bit BFs and k independent hash functions $h_0, \ldots, h_{k-1}$. All bits of BFs are initialized to 0. CSC-BF contains two algorithms: CSC − BF.Insertion and CSC − BF.Query.

- *CSC-BF.Insertion:* For each element (x, i) in set $\mathbb{S} = \{(x, i) : x \in S_i, i \in [0, n-1]\}$ in repetition Re_j, $j \in [0, r-1]$, CSC-BF uses partition function g_j to calculate partition $g_j(i) \in [0, b-1]$ of S_i. And then, CSC-BF computes k hash values $h_0(x), \ldots, h_{k-1}(x)$. Finally, CSC-BF updates the values at k positions $(h_0(x)\%m + g_j(i))\%m$,..., $(h_{k-1}(x)\%m + g_j(i))\%m$ in BF to 1. Fig. 3 shows an example of the Insertion phase.
- *CSC-BF.Query:* For a query element q in repetition Re_j, $j \in [0, r-1]$, CSC-BF computes k hash values $h_0(q)\%m$,..., $h_{k-1}(q)\%m$. For each $h_t(q)\%m$, $t \in [0, k-1]$, CSC-BF checks the next b positions $(h_t(q)\%m + 0)\%m$,..., $(h_t(q)\%m + b-1)\%m$ to determine which partitions contain q. For positions that imply the existence of q, CSC-BF takes the union of the sets contained in these partitions to get the result M_j of repetition Re_j. Finally,

CSC-BF computes the intersection of r repetition results $M_0, \ldots, M_{r-1}$ to obtain the final result M_q. Fig. 3 shows an example of Query phase.

Fig. 3 shows the CSC-BF specific framework in the Insertion and Query phases, where the CSC-BF is consisted of r 8-bit BFs.

C. Symmetric-Key Hidden Vector Encryption

Symmetric-key Hidden Vector Encryption (SHVE) [31] is a predicate encryption that supports conjunctive, equality, comparison, and subset queries of encrypted data. Σ represents a predefined set of attributes, "$*$" represents a wildcard and $\Sigma_* = \Sigma \cup \{*\}$. A SHVE consists of four algorithms: SHVE.Setup, SHVE.KeyGen, SHVE.Enc and SHVE.Query.

- *SHVE.Setup*(λ) $\rightarrow$ *msk:* On input a security parameter λ, this algorithm outputs a master secret key *msk* and the message space *M:*
- *SHVE.KeyGen*(*msk*, $\vec{v} \in \Sigma_*^m$) $\rightarrow$ *s:* On input a predicate vector $\vec{v}$ and the master secret key *msk*, this algorithm outputs a decryption key *s*.
- *SHVE.Enc*(*msk*, μ = "True", $\vec{x} \in \Sigma^m$) $\rightarrow$ $\boldsymbol{c}$*:* On input a message μ, an index vector $\vec{x}$ and the master secret key *msk*, this algorithm outputs the ciphertext $\boldsymbol{c}$ associated with ($\vec{x}$, μ).
- *SHVE.Query*(*s*, $\boldsymbol{c}$) = μ*:* On input a ciphertext $\boldsymbol{c}$ corresponding to the index vector $\vec{x}$ and a decryption key *s* corresponding to the predicate vector $\vec{v}$, this algorithm outputs the message μ, if $P_{\vec{v}}^{\mathrm{SHVE}}(\vec{x})$=1.

In addition, a Symmetric-key Hidden Vector Encryption [31] is correct if for all security parameters λ, all index vectors $\vec{x} \in \Sigma^m$ and predicate vectors $\vec{v} \in \Sigma_*^m$, after running SHVE.Setup(λ) $\rightarrow$ *msk*, SHVE.KeyGen(*msk*, $\vec{v} \in \Sigma_*^m$) $\rightarrow$ *s*, SHVE.Enc(*msk*, μ = "True", $\vec{x} \in \Sigma^m$) $\rightarrow$ $\boldsymbol{c}$, if $P_{\vec{v}}^{\mathrm{SHVE}}(\vec{x})$=1, then SHVE.Query(*s*, $\boldsymbol{c}$) = μ, otherwise

$$Pr[\mathsf{SHVE.Query}(s, \mathbf{c}) = 0] = 1 - \mathsf{negl}_{\mathrm{SHVE}}(\lambda).$$

Obviously, for each $\vec{v} = (v_1, \ldots, v_m) \in \Sigma_*^m$ and a $\vec{x} = (x_1, \ldots, x_m) \in \Sigma^m$, $P_{\vec{v}}^{\mathrm{SHVE}}(\vec{x}) = 1$ only if $v_i = x_i$ or $v_i = *$, otherwise 0.

IV. Problem Formulation

In this section, we will give the system model & threat model, problem definition, scheme definition and security model of PBKS.

A. System Model & Threat Model

As shown in Fig. 4, PBKS mainly consists of three entities: Data Owner (DO), Data User (DU) and Cloud Service Provider (CSP). The role of each entity is as follows:

- *Data owner:* DO is responsible for generating keys for all authorized DUs and sending them by a secure channel. At the same time, DO generates the index for dataset, and finally sends them to CSP. Note that this work only focuses on the generation of encrypted indexes, and the generation of encrypted data is beyond the scope of our discussion.

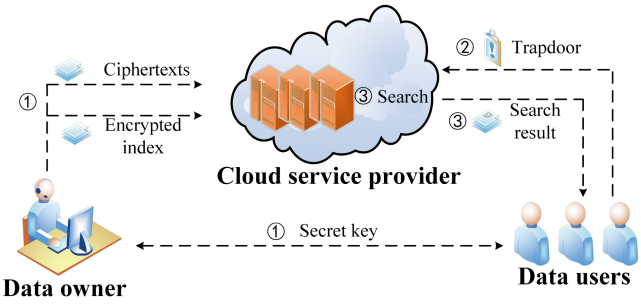


Fig. 4. The system model of PBKS.

- *Data users:* When an authorized DU needs to search, he or she generates a trapdoor and sends it to CSP.
- *Cloud service provider:* CSP with powerful storage and computing capabilities is responsible for storing massive encrypted data outsourced by DO and responding to the search requests from authorized DUs.

As shown in Fig. 4, PBKS consists of the following three steps. First, DO shares the secret key with DU and builds the index for the local data so that they can be searched after being outsourced to CSP. Then, DO sends the encrypted data and encrypted indexes to CSP (Step ①). When an authorized DU needs to perform a search, he or she needs to generate a trapdoor for the query keyword, and then sends it to CSP to initiate one search (Step ②). After CSP receives one trapdoor sent by a DU, CSP uses this trapdoor to search the encrypted indexes to obtain the encrypted data containing the query keyword, and finally returns the search result to the DU (Step ③).

Threat Model: We assume that CSP is honest but curious. CSP conducts the preset search protocol, but will try to infer the specific content of the indexes and trapdoors in the search phase. DO and DU are considered as fully trusted entities and do not collude with CSP. It is worth noting that our proposed schemes focus on the accurate and secure search of encrypted data, but do not consider the protection of access pattern and search pattern. In addition, the privacy protection of the data content can be achieved by the traditional symmetric or public-key encryption algorithm, which is beyond the scope of our discussion.

B. Problem Definition

Let $\mathcal{D} = \{d_0, \ldots, d_{n-1}\}$ be a dataset owned by DO, where the keyword set contained in each data d_i is represented as $W_i = \{w_0, w_1, \ldots\}$, and DO constructs BFs for $\mathcal{D}$. Let $Q = \{w_q\}$ denote a search request. In the BF-based keyword search scheme, the more data are stored, the problems of high storage cost, low search efficiency and high false positive rates are inevitable. Shortening BF length can reduce storage cost, but will increase false positive rate. Conversely, increasing BF length can reduce false positive rate, but will increase storage cost. At the same time, since the search needs to traverse all data, the search efficiency will reduce as the number of stored data increases. In addition, although the values in BF do not directly reflect the content of keyword, it will reveal the positions with a value of 1 in BF to the cloud server. Therefore, we need to implement the

privacy-preserving Bloom filter-based efficient keyword search. Now we give the definition of our scheme as follows:

Definition 1. (Privacy-Preserving Bloom Filter-based Efficient Keyword Search). Given a dataset $\mathcal{D} = \{d_0, \ldots, d_{n-1}\}$ and a search request $Q = \{w_q\}$, a privacy-preserving Bloom filter-based keyword search is to retrieve a subset $\mathcal{R} = \{d_1, d_2, \ldots, d_h\}$ from *Enc*($\mathcal{D}$) and *Enc*(Q), such that $\forall d_i \in \mathcal{R}, W_i \bigcap Q \neq \varnothing, 1 \leq i \leq h$. Note that this scheme uses a Bloom filter structure to implement the search. Therefore, this scheme suffers from the low false positive rate.

C. Scheme Definition

Based on the above system model, we describe the scheme definition of PBKS as follows, which consists of five algorithms:

- *Setup*(1^λ) $\rightarrow$ *msk:* Given a security parameter λ, DO outputs the master key *msk*.
- *KeyGen*(1^λ, *msk*) $\rightarrow$ {*PP*, *SK*}*:* Given the security parameter λ and the master key *msk*, DO outputs the public parameters *PP* and the secret key *SK*.
- *IndexGen*($\mathcal{D}$, *PP*, *SK*) $\rightarrow BF_{Enc}$*:* Given the dataset $\mathcal{D}$, DO first calls $\mathsf{CSC-BF.Insertion}$ to generate the Bloom filter set $\mathbf{BF} = \{\mathrm{BF}_0, \ldots, \mathrm{BF}_{r-1}\}$ of $\mathcal{D}$, where r is the number of Bloom filters. Then, DO calls SHVE.KeyGen to encrypt **BF** and gets $\mathbf{BF_{Enc}} = \{\mathrm{BF}^0_{\mathrm{Enc}}, \ldots, \mathrm{BF}^{r-1}_{\mathrm{Enc}}\}$. Finally, DO sends $\mathbf{BF_{Enc}}$ to CSP.
- *TrapGen*(w', *SK*) $\rightarrow$ *T:* Given a query keyword w', DU calls SHVE.Enc to output the trapdoor $\mathbf{T} = \{T_0, \ldots, T_{k-1}\}$ and initiates one search, where k is the number of hash functions.
- *Search*($\mathbf{BF_{Enc}}$, *T*) $\rightarrow \mathcal{R}$*:* CSP uses the trapdoor **T** to search the encrypted index $\mathbf{BF_{Enc}}$ and obtains the search result $\mathcal{R}$. CSP returns $\mathcal{R}$ to the DU.

Correctness: Let **Search**($\mathcal{D}$, w') $\rightarrow \mathcal{R}'$ be the plaintext search result of the query keyword w' on the dataset $\mathcal{D}$. The PBKS is correct for all index $\mathbf{BF_{Enc}}$ output by *IndexGen*, all trapdoor **T** output by *TrapGen*, if **Search**($\mathcal{D}$, w') $\rightarrow \mathcal{R}'$ and **Search**($\mathbf{BF_{Enc}}$, **T**) $\rightarrow \mathcal{R}$, then $\mathcal{R}' = \mathcal{R}$.

D. Security Model

In this subsection, we give the security definition of PBKS. Since PBKS implements encryption using SHVE, we widely and consistently subsume the traditional security definitions for SHVE in the indistinguishability setting. The adversary A should not learn anything about the dataset and search content beyond some explicit leaks in leakage function $\mathcal{L}$.

Definition 2: (IND-SCPA Secure for PBKS) Let $\prod$ = (**Setup**, **KeyGen**, **IndexGen**, **TrapGen**, **Search**) be a PBKS scheme over security parameter λ. The security game between a challenger C and an adversary A is defined as follows:

- *Init:* The adversary A submits two distinct search requests Q_0 and Q_1.
- *Setup:* The challenger C runs **KeyGen**(1^λ, *msk*) to get {*PP*, *SK*}, note that *SK* should be kept private.
- *Phase 1:* The adversary A adaptively initiates a number of requests:

  $Index$. On the j-th index request, the adversary A outputs a dataset $\mathcal{D}_j$. The challenger C gets $\mathbf{BF_{Enc}} \leftarrow$ **IndexGen**($\mathcal{D}_j$, *PP*, *SK*) and sends it to A, note that $\mathcal{L}(\mathcal{D}_j, Q_0) = \mathcal{L}(\mathcal{D}_j, Q_1)$.

  $Trapdoor$*:* On the j-th trapdoor request, the adversary A outputs a search request Q_j. The challenger C gets $\boldsymbol{T}_{Q_j} \leftarrow$ **TrapGen**(Q_j, *SK*) and responds the trapdoor $\boldsymbol{T}_{Q_j}$ to A.
- *Challenge:* With Q_0 and Q_1 selected in *Init*, the challenger C randomly chooses a bit $b \in \{0, 1\}$, and gets $\boldsymbol{T}_{Q_b} \rightarrow$ **TrapGen**(Q_b, *SK*). Finally, C sends $\boldsymbol{T}_{Q_b}$ to adversary A.
- *Guess:* The adversary A takes a guess b' of b.

We say that $\prod$ is secure against Selective Chosen-Plaintext Attack (SCPA) if for any Probabilistic Polynomial Time (PPT) adversary A in the above game, it has at most a negligible advantage

$$Adv^{\mathrm{IND-SCPA}}_{\mathrm{PBKS},A}(1^\lambda) = |Pr[b == b'] - 1/2| \leq \mathsf{negl}(\lambda).$$

In addition, our PBKS+ scheme can really hide the 0 or 1 values in BF by changing the inclusion identifier, which is not considered in other schemes.

TABLE II
NOTATION DESCRIPTIONS IN PBKS

Notations	Descriptions
m	Length of BF
F_0	Pseudo-random function used in SHVE
C_i	The binary vector representing the inclusion identifier of the i-th BF
$\boldsymbol{T} = \{T_0, \ldots, T_{k-1}\}$	Trapdoor
$\mathcal{D} = \{d_0, \ldots, d_{n-1}\}$	Outsourced dataset
$H = \{h_0, \ldots, h_{k-1}\}$	Hash function family used to insert elements in BF
$G = \{g_0, \ldots, g_{r-1}\}$	Partition hash functions used to classify data
$\mathbf{BF_{Enc}} = \{\mathrm{BF}^0_{Enc}, \ldots, \mathrm{BF}^{r-1}_{Enc}\}$	Index (r encrypted BFs)
(*Sym.Enc, Sym.Dec*)	IND-CPA secure symmetric encryption algorithm used in SHVE

V. PROPOSED PBKS FRAMEWORKS

In this section, we first present a efficient Bloom filter-based keyword search scheme based on CSC-BF and SHVE, namely PBKS. Then, we design a new index structure called T-CSCBF and propose an improved scheme based on PBKS to achieve higher security. Some notations used in this paper are shown in Table II.

A. PBKS: Basic Scheme

Main Idea: As mentioned in Section I, Bloom filter-based keyword search schemes suffer from some challenges, such as high false positive rates and inefficiency. To achieve keyword search on encrypted dataset, we take advantage of a new Bloom filter structure: CSC-BF, which can achieve efficient and low false positive rate search. We map all keywords W to Bloom filter: **BF** $\leftarrow \mathsf{CSC-BF.Insertion}(W)$. However, unencrypted CSC-BF will directly reveal the values in BF. There are many candidate methods to protect BF security, such as secure kNN, public-key encryption and HE. However, the above methods are not efficient or secure enough. Thus, we adopt a more efficient and secure algorithm: SHVE. We encrypt all position in BF: $\mathbf{BF_{Enc}} \leftarrow$

Algorithm 1: Details of *IndexGen* in PBKS.

Input: the outsourced dataset $\mathcal{D} = \{d_0, ..., d_{n-1}\}$, the keywords set $W = \{w_1, w_2, ...\}$, *PP*, *SK*.
Output: the encrypted index $\mathbf{BF_{Enc}}$.

Partition:
for $i = 0; i < r; i++$ **do**
  $\mathbb{S}_i = \varnothing$;
  for $j = 0; j < n; j++$ **do**
    for $w \in d_j$ **do**
      $\mathbb{S}_i$.append$((w, g_i(j)))$;
$\{\mathbb{S}_0, ..., \mathbb{S}_{r-1}\}$;
Insertion:
for $i = 0; i < r; i++$ **do**
  for $(w, j) \in \mathbb{S}_i$ **do**
    for $t = 0; t < k; t++$ **do**
      $loc_t = (h_t(w)\%m + j)\%m$;
      $\mathrm{BF}_i[loc_t] = 1$;
BF = $\{\mathrm{BF}_0, ..., \mathrm{BF}_{r-1}\}$;
Encryption:
for $i = 0; i < r; i++$ **do**
  for $j = 0; j < m; j++$ **do**
    $d_{j0} = F_0(msk, \mathrm{BF}_i[j]||j) \oplus \alpha_{ij}$;
    $d_{j1} = Sym.Enc(\alpha_{ij}, 0^{\lambda + log\lambda})$;
  $\mathrm{BF}^i_{\mathrm{Enc}} = \{\{\vec{d_j} = \{d_{j0}, d_{j1}\}\}_{j \in [0, m-1]}\}$;
$\mathbf{BF_{Enc}} = \{\mathrm{BF}^0_{\mathrm{Enc}}, ..., \mathrm{BF}^{r-1}_{\mathrm{Enc}}\}$.

SHVE.KeyGen(**BF**). Then, given a query keyword w', we locate the query positions in $\mathbf{BF_{Enc}}$: **loc** $\leftarrow$ BF.Query(w'), and encrypt the query positions to get the trapdoor: $\boldsymbol{T} \leftarrow$ SHVE.Enc(**loc**). Finally, the search phase is to match $\mathbf{BF_{Enc}}$ and $\boldsymbol{T}$ in the query positions, which is performed by SHVE.Query. Therefore, we construct an efficient PBKS scheme.

Scheme Details: According to the above idea, we describe the details of PBKS as follows.

Setup(1^λ) $\rightarrow$ *{msk}:* Given the security parameter λ, DO runs SHVE.Setup to get the master secret key *msk*.

KeyGen(1^λ, *msk*) $\rightarrow$ *PP*, *{SK}:* Given the security parameter λ, DO generates r partition hash functions $G = \{g_0, \ldots, g_{r-1}\}$, $g_i : \{0,1\}^* \rightarrow \{0,1\}^b$ to classify data and a hash function family $H = \{h_0, \ldots, h_{k-1}\}$, $h_i : \{0,1\}^* \rightarrow \{0,1\}^m$ to map keywords. In addition, DO randomly generates a pseudorandom function F_0: $\{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^*$. Then, DO chooses an IND-CPA secure symmetric encryption algorithm (*Sym.Enc, Sym.Dec*). Finally, DO outputs the public parameters *PP* and the secret key *SK* of DO as follows. Then, DO shares the hash function family *H* with DU through a secret channel as the secret key of DU.

$$PP = \{(Sym.Enc, Sym.Dec), F_0\}, SK = \{H, G\}.$$

IndexGen($\mathcal{D}$, *PP*, *SK*) $\rightarrow$ $\{BF_{Enc}\}$*:* For the outsourced dataset $\mathcal{D} = \{d_0, \ldots, d_{n-1}\}$ and the keyword set $W = \{w_1, w_2, \ldots\}$, DO generates the index of $\mathcal{D}$ by Algorithm 1, which can be divided into following steps:

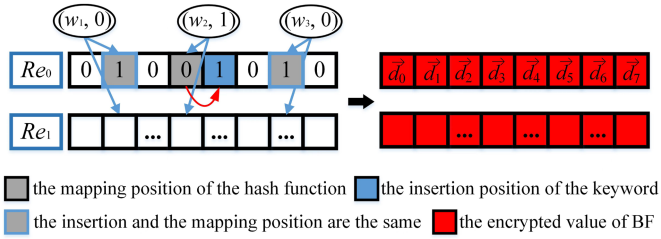


Fig. 5. Example of *IndexGen* in PBKS.

- *Partition:* Let's take one repetition as an example. DO initializes a m-bit Bloom filter BF with all zeros, and then classifies n data in $\mathcal{D}$ using a partition hash function $g(x)$: the j-th data d_j is divided into the $g(j)$-th partition $P_{g(j)}$. Accordingly, the keywords contained in d_j are also classified as keywords contained in $P_{g(j)}$. Finally, DO gets the set $\mathbb{S} = \{(w, i), i \in [0, b-1]\}$, indicating $w \in P_i$. DO repeats the partition r times in this way. It is worth noting that the partition hash function used for each repetition is different, i.e., $\{g_t(x), t \in [0, r-1]\}$. Thus, r all-zero m-bit Bloom filters $\{\mathrm{BF}_0, ..., \mathrm{BF}_{r-1}\}$ and their corresponding partitions $\{\mathbb{S}_0, \ldots, \mathbb{S}_{r-1}\}$ are obtained.
- *Insertion:* Let's take one repetition (one Bloom filter BF) as an example. For each pair (w, j) contained in $\mathbb{S}$, the hash function family $H = \{h_0, \ldots, h_{k-1}\}$ and offset j are used to get the k positions $loc_t, t \in [0, k-1]$ of w, and the values of the corresponding positions in BF[loc_t] are set to 1. The above operation is repeated until all elements in $\mathbb{S}$ are inserted into BF. DO uses this method to get the Bloom filter set $\mathbf{BF} = \{\mathrm{BF}_0, ..., \mathrm{BF}_{r-1}\}$ that completes all insertions.
- *Encryption:* Let's take one repetition (one Bloom filter BF) as an example. DO uses SHVE.KeyGen to encrypt the Bloom filter BF obtained by CSC-BF. For each value BF[j] in BF, DO calculates the encrypted values d_{j0} and $d_{j1}, j \in [0, m-1]$ to generate $\mathrm{BF_{Enc}}$. Because the position j and the secret key α_j are different, $\vec{d_j}$ and $\vec{d_{j'}}$ are different when BF[j] = BF[j']. Similarly, DO repeats and gets the encrypted Bloom filter set $\mathbf{BF_{Enc}} = \{\mathrm{BF}^0_{\mathrm{Enc}}, ..., \mathrm{BF}^{r-1}_{\mathrm{Enc}}\}$. Finally, $\mathbf{BF_{Enc}}$ is sent to the cloud server as index.

Example: Suppose $m = 8$, $k = 1$, $r = 2$ and $b = 2$. There are two 8-bit all-zero Bloom filters $\{\mathrm{BF}_0, \mathrm{BF}_1\}$, one hash function $\{h\}$, and three data $\mathcal{D} = \{d_0, d_1, d_2\}$, where $d_0 = \{w_3\}, d_1 = \{w_1\}, d_2 = \{w_2\}$. From $r = 2$ and $b = 2$, we can know that there are two repetitions $\{Re_0, Re_1\}$ and two partitions $\{P_0, P_1\}$, respectively. Let's take the repetition Re_0 as an example to introduce the *IndexGen* phase in Fig. 5. We first classify $\mathcal{D}$ by $g_0(x)$: $g_0(0) = g_0(1) = 0, g_0(2) = 1$, that is, d_0 and d_1 are classified as P_0 and d_2 is classified as P_1, i.e., $P_0 = \{w_1, w_3\}$, $P_1 = \{w_2\}$ and $\mathbb{S}_0 = \{(w_1, g_0(1)), (w_2, g_0(2)), (w_3, g_0(0))\} = \{(w_1, 0), (w_2, 1), (w_3, 0)\}$. Next, for each item $(w_i, j) \in \mathbb{S}_0$, we calculate $loc_t = (h(w_i)\%8 + j)\%8$ by h to obtain 1, 4, 6, respectively. Then, we change the values of corresponding positions in BF_0 to 1 and finally get the BF_0 of repetition

Algorithm 2: Details of *Search* in PBKS.

Input: the trapdoor $\boldsymbol{T} = \{T_0, T_1, ..., T_{k-1}\}$, the Bloom filters $\mathbf{BF_{Enc}} = \{\mathrm{BF}^0_{Enc}, ..., \mathrm{BF}^{r-1}_{Enc}\}$.
Output: the search result $\mathcal{R}$.

```
for i = 0; i < r; i++ do
    R_i = ∅;
    a = 0;
    for j = 0; j < b; j++ do
        for t = 0; t < k; t++ do
            BF^i_Enc[h_i(w') + j] = {d⃗_(h_i(w')+j) = {d_(h_i(w')+j)0, d_(h_i(w')+j)1}};
            K'_j = c_j ⊕ d_(h_i(w')+j)0;
            μ' = Sym.Dec(K'_j, d_(h_i(w')+j)1);
            if μ' == 0^(λ+logλ) then
                a++;
        if a == k then
            R_i.append(j);
B = {R_0, ..., R_(r-1)};
R = R_0 ∩ ... ∩ R_(r-1).
```

Re_0. With the same method, the Bloom filter BF_1 of repetition Re_1 is obtained and the Insertion phase is completed. Finally, SHVE.KeyGen is used to encrypt the $\{\mathrm{BF}_0, \mathrm{BF}_1\}$ to get the index.

TrapGen(w', *SK*) $\rightarrow$ *{T}:* For each query keyword w', DU first calculates the corresponding position $h_t(w')$ of w' in the Bloom filter by H, where $t \in [0, k-1]$. For each $h_t(w')$, DU uses SHVE.Enc to generate T_t as follows:

$$T_t = \{h_t(w'), c_0 = F_0(msk, 1||h_t(w')), \ldots,$$
$$c_{b-1} = F_0(msk, 1||h_t(w') + b - 1)\}.$$

In this way, DU gets the trapdoor $\mathbf{T} = \{T_0, T_1, \ldots, T_{k-1}\}$, where k is the number of hash functions in H.

Search(BF_{Enc}, T) $\rightarrow$ $\{\mathcal{R}\}$*:* After receiving the trapdoor $\mathbf{T} = \{T_0, T_1, \ldots, T_{k-1}\}$, CSP checks which data contains the query keyword in $\mathbf{T}$ through Algorithm 2. Take one $\mathrm{BF}^i_{\mathrm{Enc}}$ as an example, where $i \in [0, r-1]$. CSP searches each $T_t = \{h_t(w'), c_0, \ldots, c_{b-1}\}$, $t \in [0, k-1]$. First of all, CSP uses $h_t(w')$ to locate the corresponding positions $\mathrm{BF}^i_{\mathrm{Enc}}[h_t(w') + j]$, $j \in [0, b-1]$, and then uses SHVE.Query to match. CSP computes K'_j and μ'. If $\mu' = 0^{\lambda+log\lambda}$, it indicates the c_j and $\mathrm{BF}^i_{\mathrm{Enc}}[h_t(w') + j]$ are matched successfully for hash function h_t. Correspondingly, if c_j and $\mathrm{BF}^i_{\mathrm{Enc}}[h_t(w') + j]$, $t \in [0, k-1]$ all can match successfully for all, it means that the query keyword in c_j is included in the j-th partition. In this way, the candidate result R_i of one Bloom filter $\mathrm{BF}^i_{\mathrm{Enc}}$ is obtained, where $i \in [0, r-1]$. Similarly, r candidate results $\boldsymbol{R} = \{R_0, \ldots, R_{r-1}\}$ can be obtained. Finally, the search result $\mathcal{R}$ can be obtained by performing set intersection operation on these r candidate results.

Remark: As shown in Section V-A, PBKS can achieve efficient BF-based keyword search on encrypted data, and also achieve the privacy protection by encrypting the values of BF.

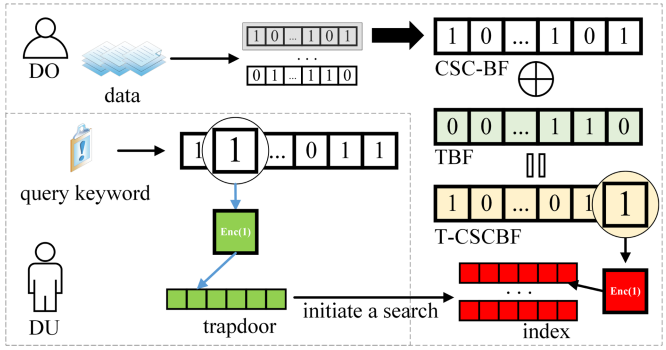


Fig. 6. The main idea of PBKS+.

However, PBKS still have some shortcomings in terms of security. In the *Search* phase, since BF is encrypted by SHVE, the cloud server cannot know the values of BF. However, when one query position is successfully matched, CSP will assume that the value of BF at this position is 1 due to the fixed rules of Bloom filter, which still leaks the privacy of BF.

B. PBKS+: Extended Scheme

Main Idea: To solve the privacy leakage problem in the basic PBKS scheme, we can change the inclusion identifiers at different positions in BF. TBF can obfuscate the mapped position of the keyword, but will incur the double storage cost. To change the inclusion identifier and avoid double storage cost, we generate the binary vectors inspired by TBF and perform XOR operations with BF to construct a new index structure T-CSCBF. Then, given a query keyword w', we compute the inclusion identifiers for all query positions and then perform match. Note that, like PBKS, index and trapdoor are encrypted by SHVE.KeyGen and SHVE.Enc, respectively. On the basis of the PBKS, only the *KeyGen*, *IndexGen* and *TrapGen* algorithms need to be changed, and the contents of *Setup* and *Search* remain the same. As shown in Fig. 6, we construct an PBKS+ scheme, which can achieve privacy-preserving and efficient Bloom filter-based keyword search.

Scheme Details: According to the above idea, we only describe in detail the *KeyGen*, *IndexGen* and *TrapGen* algorithms of PBKS+ as follows.

KeyGen(1^λ, *msk*) $\rightarrow$ *PP*, *{SK}:* In the *KeyGen* of the PBKS scheme, we have obtained $PP = \{(Sym.Enc, Sym.Dec), F_0\}$ and $SK = \{H, G\}$. On this basis, DO uses a pseudo-random function $\mathbb{H}$: $\{0,1\}^* \rightarrow \{0,1\}^*$ to obtain a hash function $\mathcal{H}(x)$, where $\mathcal{H}(x) = \mathbb{H}(x)\%2$. Additionally, DO generates r random numbers $\gamma = \{\gamma_0, \ldots, \gamma_{r-1}\}$. Finally, DO gets the new public parameters *PP* and the new secret key *SK* of DO as follows. Then, DO shares the hash function family H with DU through a secret channel as the secret key of DU.

$$PP = \{(Sym.Enc, Sym.Dec), F_0, \mathcal{H}\}, SK = \{H, G, \boldsymbol{\gamma}\}.$$

IndexGen($\mathcal{D}$, *PP*, *SK*) $\rightarrow$ $\{BF_{Enc}\}$*:* For the outsourced dataset $\mathcal{D} = \{d_0, \ldots, d_{n-1}\}$ and keyword set $W = \{w_1, w_2, \ldots\}$, DO generates r Bloom filters $\mathbf{BF} = \{\mathrm{BF}_0, \ldots, \mathrm{BF}_{r-1}\}$ according to

Algorithm 3: Details of **IndexGen**.*Change* in PBKS+.

Input: the Bloom filters BF = {BF_0, ..., BF_{r-1}}.
Output: the new privacy preserving Bloom filter set $\mathbf{BF}^{new}$.

Change:
BF = {BF_0, ..., BF_{r-1}};
for $i = 0; i < r; i++$ **do**
  for $j = 0; j < m; j++$ **do**
    $C_i[j] = \mathcal{H}(j \oplus \gamma_i)$;
    $C_i'[j] = (C_i[j] \oplus 1)\%2$;
    $BF_i^{new}[j] = C_i'[j] \oplus BF_i[j]$;
$\mathbf{BF}^{new}$ = {BF_0^{new}, ..., BF_{r-1}^{new}};

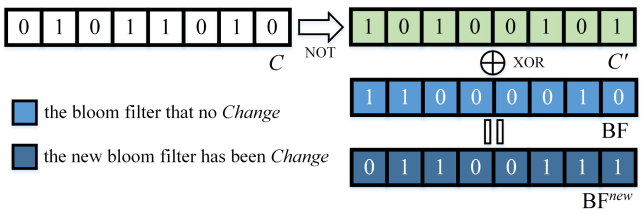


Fig. 7. Example of *Change* in PBKS+.

Partition and *Insertion* of *IndexGen* algorithm in PBKS scheme. On this basis, we add a step *Change* to change the inclusion identifier that is represented as 0 or 1 in each position of BF, where the special details are given in Algorithm 3.

- *Change:* Let's take one repetition (one Bloom filter BF) as an example. The Bloom filter BF obtained by CSC-BF will be changed by TBF.Iniți. For the j-th position in BF, DO calculates $C[j] = \mathcal{H}(j \oplus \gamma)$ to obtain an m-bit binary vector C, where $C[j]$ directly denotes the inclusion identifier in BF[j]. For example, $C[j] = 0$ means that 0 is the inclusion identifier in BF[j]. Then, we invert the value at each position in C to get C', i.e., $C'[j]$ directly denotes the exclusion identifier in BF[j]. Afterwards, DO calculates $BF^{new} = BF \oplus C'$. In the same way, we can get the new Bloom filter set $\mathbf{BF}^{new} = \{BF_0^{new}, ..., BF_{r-1}^{new}\}$. Note that γ used in C of different BF^{new} is different to eliminate the correlation between different BF^{new}. Finally, DO encrypts $\mathbf{BF}^{new}$ to get $\mathbf{BF_{Enc}} = \{BF_{Enc}^0, ..., BF_{Enc}^{r-1}\}$ and sends it to CSP.

Example: Suppose the 8-bit Bloom filter BF generated by CSC-BF is 11000010. Correspondingly, the vector C representing the inclusion relationship calculated by $C[i] = H(i \oplus \gamma)$, $i \in [0, 7]$ is 01011010, then $C' = 10100101$. Therefore, the new Bloom filter BF^{new} that has been *Change* can be obtained as 01100111. Fig. 7 shows an example of the *Change* in *IndexGen*.

TrapGen(w', *SK*) $\rightarrow$ {*T*}*:* Similar to *TrapGen* in PBKS, for a query keyword w', DU first calculates the corresponding position $h_t(w')$ in BF through *H*, where $t \in [0, k\text{-}1]$. Then, DU needs to calculate the inclusion identifier in each query position for i-th BF_i, where $i \in [0, r\text{-}1]$. For each $h_t(w')$, $t \in [0, k\text{-}1]$, DU calculates

$$C_i[0] = \mathcal{H}((h_t(w') + 0) \oplus \gamma_i), \ldots,$$

$$C_i[b-1] = \mathcal{H}((h_t(w') + b - 1) \oplus \gamma_i),$$

and

$$T_t^i = \{h_t(w'), c_0^i = F_0(msk, C_i[0] || h_t(w')), \ldots,$$

$$c_{b-1}^i = F_0(msk, C_i[b-1] || h_t(w') + b - 1)\}.$$

In this way, DU gets trapdoor $\mathbf{T} = \{T_t^i\}_{i \in [0,r-1], t \in [0,k-1]}$, where k is the number of hash functions and r is the number of repetitions in the scheme.

Finally, similar to *Search* in PBKS, CSP uses the received $\mathbf{T}$ to search $\mathbf{BF_{Enc}}$. Note that, unlike PBKS, different BF_i^{new} matches different T_t^i, where $i \in [0, r\text{-}1]$.

Remark: PBKS+ can realize privacy-preserving BF-based keyword search. However, since CSC-BF maps all keywords of the dataset to the r Bloom filters and the identifier information of each data is introduced in keyword insertion phase, this makes PBKS+ unable to use the method that multiply the index and trapdoor, in traditional Bloom filter-based search scheme, to achieve multi-keyword search. At the same time, since the hash values of different query keyword are different in PBKS+, it is impossible to realize multiple keyword searches at the same time. Therefore, PBKS+ can only achieve single keyword search, which affects search efficiency and user experience.

VI. Correctness and Security Analysis

In this section, we first prove the correctness of PBKS+. Then, the security of PBKS is analyzed.

A. Correctness Analysis

In this subsection, we verify the correctness of **IndexGen**.*Change* in PBKS+.

Theorem 1: For a m-bit Bloom filter BF and a m-bit binary vector C, $BF^{new} = BF \oplus C'$, where $C[i]$ denotes the inclusion identifier of BF[i]. BF^{new} is a new Bloom filter obtained by reassigning each bit according to the new inclusion identifier redefined by C.

Proof: We can know that 1 means inclusion and 0 means exclusion in traditional BF. That is, there are only two possible values 0 or 1 in BF[i], $i \in [0, m\text{-}1]$. In addition, it can be known from the definition of C that $C[i]$ means inclusion, and $C'[i]$ means exclusion in BF[i]. That is, when $C[i] = 1$, 1 means inclusion, 0 means exclusion in BF[i]. Conversely, when $C'[i] = 1$, 1 means exclusion, 0 means included in BF[i]. Thus, there are also two possible values 0 or 1 in $C'[i]$, $i \in [0, m\text{-}1]$. Therefore, there are only four possible cases for BF[i] and $C'[i]$. Next, we will prove by case that when BF[i] = 1, i.e., BF[i] contains some keywords, and then $BF^{new}[i]$ also contains some keywords, where $BF^{new}[i] = BF[i] \oplus C'[i]$, and the opposite is also true.

- When BF[i] = 1 and $C'[i] = 1$, $BF^{new}[i] = BF[i] \oplus C'[i] = 0$. BF[$i$] = 1 means that BF[$i$] contains some keywords. In addition, it can be seen from $C'[i] = 1$ that $BF^{new}[i] = 0$ also means that $BF^{new}[i]$ contains some keywords.
- When BF[i] = 1 and $C'[i] = 0$, $BF^{new}[i] = BF[i] \oplus C'[i] = 1$. BF[$i$] = 1 means that BF[$i$] contains some keywords. In addition, it can be seen from $C'[i] = 0$ that $BF^{new}[i] = 1$ also means that $BF^{new}[i]$ contains some keywords.

- When $\mathrm{BF}[i]=0$ and $C'[i]=1$, $\mathrm{BF}^{new}[i]=\mathrm{BF}[i]\oplus C'[i]=1$. $\mathrm{BF}[i]=0$ means that $\mathrm{BF}[i]$ does not contain any keyword. In addition, it can be seen from $C'[i]=1$ that $\mathrm{BF}^{new}[i]=1$ also means that $\mathrm{BF}^{new}[i]$ does not contain any keyword.
- When $\mathrm{BF}[i]=0$ and $C'[i]=0$, $\mathrm{BF}^{new}[i]=\mathrm{BF}[i]\oplus C'[i]=0$. $\mathrm{BF}[i]=0$ means that $\mathrm{BF}[i]$ does not contain any keyword. In addition, it can be seen from $C'[i]=0$ that $\mathrm{BF}^{new}[i]=0$ also means that $\mathrm{BF}^{new}[i]$ does not contain any keyword.

As can be seen from the above, Theorem 1 was established in all four cases. Therefore, the correctness of PBKS+ is proved. □

In term of the false positive rate, the false positive rates of three MS-MMQ solutions introduced in the Section I: BIGSI, RAMBO and CSC-BF are:

$$\delta_{\mathrm{BIGSI}}=\sum_{i\in\{0,\ldots,n-1\}\setminus M_q} negl_{\mathrm{BF}}(m,k,|S_i|),$$

$$\delta_{\mathrm{RAMBO}}\le n\left(1-(1-negl_{\mathrm{BF}}(m,k,|P|))\left(1-\frac{1}{b}\right)^{v}\right)^{r},$$

$$\delta_{\mathrm{CSC-BF}}\le\left(1-\left(1-negl_{\mathrm{BF}}\left(m,k,\sum_{i=0}^{n-1}|S_i|\right)\right)\left(1-\frac{1}{b}\right)^{v}\right)^{r},$$

where $negl_{\mathrm{BF}}(m,k,|S|)\approx(1-(1-\frac{1}{m})^{k|S|})^k\approx(1-e^{-k|S|/m})^k$, n is the number of data, m is the length of each bloom filter, k is the number of hash functions, b is the number of partitions, r is the number of repetitions, P is a partition in RAMBO, S_i is the keyword set of the i-th data, M_q is the data containing the query element q, and $v=|M_q|$.

It can be seen that when the query q is fixed, the false positive rate of BIGSI δ_{BIGSI} is fixed. However, the false positive rate of RAMBO δ_{RAMBO} and CSC-BF $\delta_{\mathrm{CSC-BF}}$ can be adjusted according to the values of b and r. Because $negl_{\mathrm{BF}}(m,k,\sum_{i=0}^{n-1}|S_i|)\le 1$, i.e., δ_{RAMBO} and $\delta_{\mathrm{CSC-BF}}$ will decrease as b and r increase. In addition, according to the false positive rate equation of BF, we can know $negl_{\mathrm{BF}}(m,k,\sum_{i=0}^{n-1}|S_i|)\ge negl_{\mathrm{BF}}(m,k,|S_i|)$. However, we can increase b to keep it in equilibrium. Therefore, we can achieve the low false positive rate of $\delta_{\mathrm{CSC-BF}}$ by increasing b and r.

B. Security Analysis

In this subsection, according to the security definition in Section IV-D, we first prove that PBKS is semantic security against IND-SCPA. And then, we prove that the 0 or 1 value of the BF is not revealed in the *Search* phase.

Theorem 2: Our PBKS scheme is IND-SCPA secure if SHVE is IND-SCPA secure.

Proof: The index and trapdoor of our PBKS are encrypted by SHVE, thus, the security of PBKS depends on the security of SHVE. Specifically, the proof is as follows:

- *Init:* The adversary A submits two distinct search requests Q_1 and Q_2.
- *Setup:* The challenger C runs **KeyGen**(1^λ, *msk*) to get the public parameters *PP* and the secret key *SK*, while keeping *SK* private.
- *Phase 1:* The adversary A adaptively submits a number of requests:
  $Index$*:* The adversary A outputs the dataset $\mathcal{D}_j$ for the j-th index request. The challenger C runs **IndexGen**($\mathcal{D}_j$, *PP*, *SK*) and gets $\mathbf{BF}_{\mathbf{Enc}}$*:* Note that $\mathcal{L}(\mathcal{D}_j, Q_0)=\mathcal{L}(\mathcal{D}_j, Q_1)$, where $\mathcal{L}$ denotes the leak function of PBKS.
  $Trapdoor$*:* The adversary A outputs the search request Q_j for the j-th trapdoor request. The challenger C runs **TrapGen**(Q_j, *SK*) and gets a trapdoor $\mathbf{T}_{Q_j}$. Finally, C responses with $\mathbf{T}_{Q_j}$ to A:
- *Challenge:* According to the Q_0, Q_1 selected in *Init*, the challenger C randomly chooses a bit $b\in\{0,1\}$, and runs **TrapGen**(Q_b, *SK*) to get $\mathbf{T}_{Q_b}$. Finally, $\mathbf{T}_{Q_b}$ is sent to A.
- *Guess*: The adversary A takes a guess b' of b.

Since SHVE is used to encrypt in our PBKS, the indistinguishability of index and trapdoor of our PBKS is directly derived from the indistinguishability of SHVE. The security game of our PBKS is essentially simulated by q instances of SHVE, where q indicates the number of SHVE instances in the game. Therefore, as long as the adversary A' can distinguish between two SHVE encrypted values, A can distinguish indexes and trapdoors in PBKS, which can be expressed as:

$$Adv^{\mathrm{IND-SCPA}}_{\mathrm{PBKS},A}(1^\lambda)\le Adv^{\mathrm{IND-SCPA}}_{\mathrm{SHVE},A'}(1^\lambda)\le q\cdot\mathsf{negl}(\lambda).$$

As can be seen from the above, the IND-SCPA secure of PBKS is proved. Note that since both PBKS and PBKS+ are encrypted by SHVE, their security proofs are the same.

Theorem 3: In the PBKS+ scheme, the 0 or 1 value at each position of each BF is semantically secure, if $\mathcal{H}(x)$ is a pseudo-random hash function.

Proof: In the traditional BF, 1 means that the position contains some keywords, and 0 means not contains. Because the data user will only search the position of the 1 in the BF, that is, only when this position is 1 in the BF, the data user's search condition will be satisfied. Even if BF is encrypted, once one position is successfully matched, it will directly reveal that this position must be 1 in the BF. PBKS+ can prevent such information leakage. In **IndexGen**.*Change* algorithm of PBKS+, a pseudorandom hash function $\mathcal{H}(x)$ is used to operate on all positions in the BF. At the same time, in order to eliminate the correlation between different BFs, different random values γ are used for different BFs in $\mathcal{H}(x\oplus\gamma)$. The inclusion identifier at different position in different BFs is changed by $\mathcal{H}(x\oplus\gamma)$. In this way, even if the index and trapdoor are successfully matched at one position, CSP also cannot know the real value at this position in BF. Thus, the privacy of the real value in BF is protected. □

VII. Performance Analysis

In this section, we analyze the performance of PBKS by comparing PBKS, PBKS+ and VFSA [5] in theoretical and experimental aspects, respectively.

TABLE III
THEORETICAL COMPUTATION COST ANALYSIS: A COMPARATIVE SUMMARY

Schemes	**IndexGen**	**TrapGen**	**Search**
VFSA [5]	T_{GKP}+$2nm\mathrm{T}_{Hash}$+$3\sum_{i=1}^{n}\|W_i\|k\mathrm{T}_{Hash}$	$2k\mathrm{T}_{Hash}$	$\|node\|k\mathrm{T}_{Hash}$
PBKS	$rn\mathrm{T}_{Hash}+\sum_{i=1}^{n}\|W_i\|\mathrm{T}_{Hash}$+$\mathrm{T}_{\mathsf{SHVE.KeyGen}}$	$k(\mathrm{T}_{Hash}$+$\mathrm{T}_{\mathsf{SHVE.Enc}})$	$kr\mathrm{T}_{\mathsf{SHVE.Query}}+rU_{\bigcup}+I_{\bigcap}$
PBKS+	$(n+m)r\mathrm{T}_{Hash}+\sum_{i=1}^{n}\|W_i\|\mathrm{T}_{Hash}$+$\mathrm{T}_{\mathsf{SHVE.KeyGen}}$	$kr((b+1)\mathrm{T}_{Hash}$+$\mathrm{T}_{\mathsf{SHVE.Enc}})$	$kr\mathrm{T}_{\mathsf{SHVE.Query}}+rU_{\bigcup}+I_{\bigcap}$

Notes. T_{GKP}: Time complexity of running the fraph-based keyword partition algorithm; $|W_i|$: Number of keywords extracted from the document d_i; $|node|$: Number of tree nodes need to be checked; $\mathrm{T}_{\mathsf{SHVE.KeyGen}}$: Time complexity of running the SHVE.KeyGen algorithm; $\mathrm{T}_{\mathsf{SHVE.Enc}}$: Time complexity of running the SHVE.Enc algorithm; $\mathrm{T}_{\mathsf{SHVE.Query}}$: Time complexity of running the SHVE.Query algorithm; $U_{\bigcup}$: Time complexity of running once union operation; $I_{\bigcap}$: Time complexity of running once intersection operation.

A. Theoretical Analysis

In order to analyze the computational cost of PBKS, PBKS+ and comparison scheme VFSA, we first introduce some time-consuming operations in these schemes, such as: graph-based keyword partition algorithm in VFSA, SHVE.KeyGen algorithm, SHVE.Enc algorithm, SHVE.Query algorithm, hash operation in PBKS. Let T_{GKP}, $\mathrm{T}_{\mathsf{SHVE.KeyGen}}$, $\mathrm{T}_{\mathsf{SHVE.Enc}}$, $\mathrm{T}_{\mathsf{SHVE.Query}}$ and T_{Hash} denote the corresponding time complexity, respectively.

We present the computation cost of PBKS, PBKS+ and VFSA [5] schemes in Table III. In *IndexGen*, PBKS first uses CSC-BF to construct the BFs, and then uses SHVE.KeyGen to encrypt the value at each BF position, which costs $rn\mathrm{T}_{Hash}+\sum_{i=1}^{n}|f_i|\,\mathrm{T}_{Hash}$ and $\mathrm{T}_{\mathsf{SHVE.KeyGen}}$, respectively. The cost of *Partition* is $rn\,\mathrm{T}_{Hash}$, and the cost of *Insertion* is $\sum_{i=1}^{n}|f_i|\mathrm{T}_{Hash}$. In PBKS+, since it is necessary to change the inclusion identifier of each position in BF, the cost of *Change* is $mr\,\mathrm{T}_{Hash}$. In *TrapGen*, PBKS first calculates the hash values of the query positions, and then encrypts the query positions. Since k hash functions are involved in the scheme, the above operation needs to be repeated k times. Therefore, the costs of the above two operations are $k\mathrm{T}_{Hash}$ and $k\,\mathrm{T}_{\mathsf{SHVE.Enc}}$, respectively. In PBKS+, since the random number γ used by each BF is different in the *Change* operation, the trapdoor of different BF is different. Therefore, the costs of the two operations mentioned above are $kr(b+1)\mathrm{T}_{Hash}$ and $kr\,\mathrm{T}_{\mathsf{SHVE.Enc}}$, respectively. In *Search*, the costs of PBKS and PBKS+ are the same. The cloud server first matches the query positions, and then performs the union operation on the results obtained by the same BF. Finally, the cloud server performs a intersection operation on r different results to obtain the final result, which costs $kr\mathrm{T}_{\mathsf{SHVE.Query}}$, $rU_{\bigcup}$ and $I_{\bigcap}$, respectively.

B. Experimental Tests

We conduct experiments on an Ubuntu 20.04 Server with 2.90 GHz × 64 Intel Xeon(R) Gold 6266R CPU by using C++. We randomly select 1500 data from the NSF Research Awards Abstracts 1990-2003 dataset as the test dataset. The number of hash functions used in the experiments is 3.

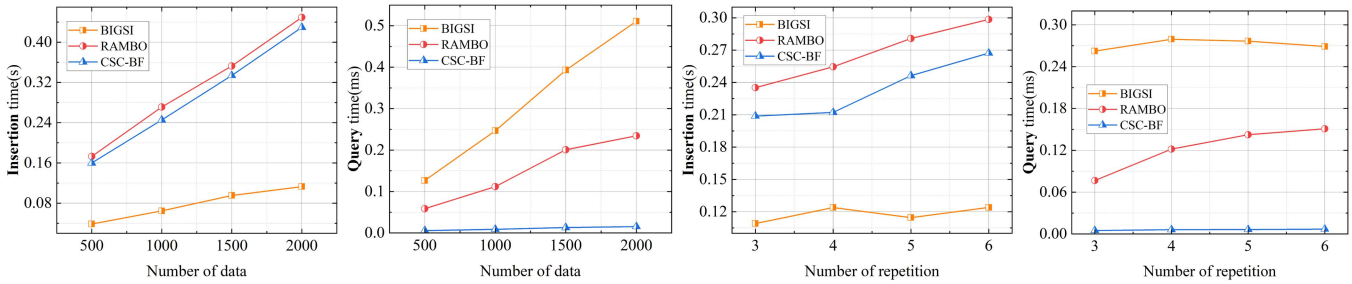
1) Experimental of Bloom Filter: We first explain why we choose CSC-BF to build the Bloom filter. At present, there are three main methods to solve the MS-MMQ problem, namely BIGSI, RAMBO and CSC-BF. BIGSI needs to construct one Bloom filter for each data. And $n \cdot k$ hash operations need to be performed in the *Query* phase. RAMBO needs to construct $r \cdot b$ BFs. And $r \cdot b \cdot k$ hash operations need to be performed in the *Query* phase. Different from the previous two methods, CSC-BF needs to construct r BFs. Only $r \cdot k$ hash operations need to be performed in the *Query* phase. Below, we analyze the effects of the number of data and the number of repetitions on the *Insertion* and *Query* times of the three MS-MMQ methods.

Fig. 8(a) and (b) show the time cost in *Insertion* and *Query* phases of the three methods as the number of data increases form 500 to 2000. And let $r = 5$, $b = 100$, where r is the number of repetitions in RAMBO and CSC-BF, b is the number of partitions in RAMBO and CSC-BF. As can be seen, on the one hand, CSC-BF spends the least time in *Query* phase because the number of hash operations that needs to be calculated is less than the other two methods, and BIGSI spends the most time. On the other hand, since each keyword needs to be inserted r times repeatedly to improve accuracy in CSC-BF, the *Insertion* phase takes more time than BIGSI but less than RAMBO, and BIGSI spends the least time. Fig. 8(c) and (d) show the time of three methods in *Insertion* and *Query* phases as the number of repetitions increases form 3 to 6, where $n = 1000$, $b = 100$. Same as Fig. 8(a) and (b), due to the different number of hash operations, CSC-BF spends the least time in *Query* phase and spends more time in *Insertion* phase.

As can be seen from Fig. 8, the *Insertion* time and *Query* time of BIGSI do not change with r, but increase with n. In addition, the *Insertion* time and *Query* time of CSC-BF and RAMBO increase with r. Therefore, in order to improve the search efficiency, we choose CSC-BF to build Bloom filter.

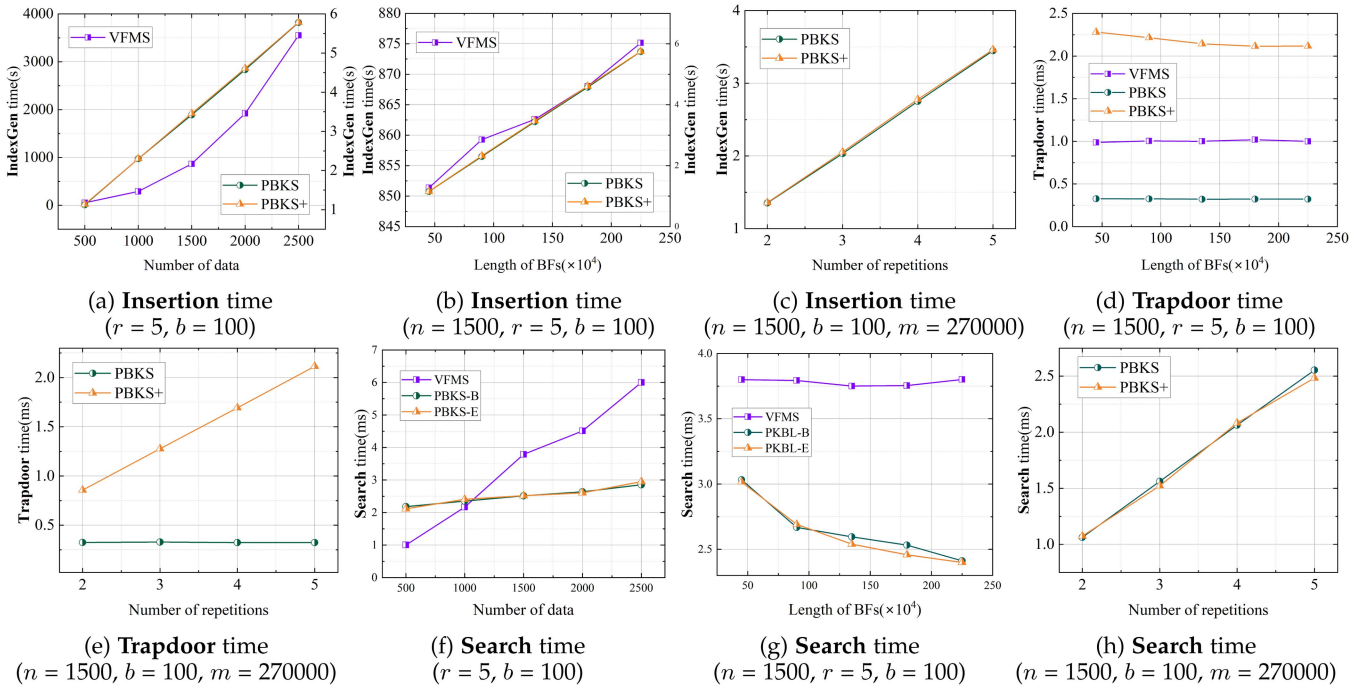
2) Experimental of Related Scheme: To illustrate the feasibility of PBKS and PBKS+, we compare them with VFMS [5]. Different from PBKS and PBKS+, VFMS uses the traditional BIGSI method to construct BF to implement search and uses Twin Bloom Filter to protect BF privacy.

Fig. 9(a) and 9(b) show the variety of *IndexGen* time as the number of data and the total length of the BF, respectively. Note that the curve of VFMS varies refer to the left Y-axis, and the curves of PBKS and PBKS+ vary refer to the right Y-axis in these two figures. In Fig. 9(a), we can see that the *IndexGen* time of three schemes increases with the number of data. At the same time, since PBKS+ needs to calculate the inclusion identifiers for different BFs, it takes more time in the *IndexGen* phase than PBKS. In Fig. 9(b), we can see that the *IndexGen* time for PBKS, PBKS+ and VFMS increases with the BF length. Because the longer the BF, the longer time for encrypting index. At the same time, PBKS+ still requires more *IndexGen* time than PBKS.



(a) **Insertion** time ($r = 5$, $b = 100$) (b) **Query** time ($r = 5$, $b = 100$) (c) **Insertion** time ($n = 1000$, $b = 100$) (d) **Query** time ($n = 1000$, $b = 100$)

Fig. 8. Time cost analysis of three Bloom filter methods.



(a) **Insertion** time ($r = 5$, $b = 100$) (b) **Insertion** time ($n = 1500$, $r = 5$, $b = 100$) (c) **Insertion** time ($n = 1500$, $b = 100$, $m = 270000$) (d) **Trapdoor** time ($n = 1500$, $r = 5$, $b = 100$)

(e) **Trapdoor** time ($n = 1500$, $b = 100$, $m = 270000$) (f) **Search** time ($r = 5$, $b = 100$) (g) **Search** time ($n = 1500$, $r = 5$, $b = 100$) (h) **Search** time ($n = 1500$, $b = 100$, $m = 270000$)

Fig. 9. Practical performance analysis.

Note that for the fair comparison, in Fig. 9(b), (d) and (g), we unify the total lengths of BFs in the three schemes.

In Fig. 9(d), we present the trapdoor generation time for different numbers of BF lengths in the *Trapdoor* phase. It can be seen that the *Trapdoor* times of the three schemes are independent of m. In addition, since different BFs need to use different random values γ, the *Trapdoor* time of PBKS+ is significantly longer than that of PBKS. Meanwhile, since we use the SHVE algorithm to improve security, the *Trapdoor* time of PBKS+ is higher than that of VFMS.

In Fig. 9(f) and (g), we show the time cost in *Search* phase as the number of data and the total length of the BF increase, respectively. In Fig. 9(f), it can be seen that the *Search* time of VFMS increases with the number of data. Although the operations performed in the *Search* phase of PBKS and PBKS+ schemes are independent of the number of data, the time cost of PBKS and PBKS+ in *Search* phase also increases with the number of data, because the time cost for the union and intersection operations increases as the number of data increases. In addition, as the number of data increases, PBKS and PBKS+ gradually dominate. It can be seen in Fig. 9(g) that the *Search* time of VFMS is independent of the BF length. However, as the time costs of union and intersection operations decrease with the increase of BF length, the time costs of PBKS and PBKS+ will also decrease gradually. In addition, the time costs of PBKS and PBKS+ are equal and significantly less than VFMS's.

In addition, in Fig. 9(c), (e) and (h), we show the time costs in different phases of PBKS and PBKS+ with the increase of the number of repetitions r. It can be seen that, as r increases, the number of keyword insertions increases multiply, so the *IndexGen* time increases accordingly. In addition, in *IndexGen* phase, PBKS+ has the *Change* stage that PBKS does not have, PBKS+ takes longer time than PBKS. In *Trapdoor* phase, PBKS uses the same trapdoor for different BFs, so the *Trapdoor* time

TABLE IV
TIME COSTS OF PBKS AND PBKS+ IN EACH PHASE VARYING WITH THE NUMBER OF REPETITIONS ($n = 1500$, $b = 100$, THE TOTAL LENGTH OF BFS IS 1350000)

r	PBKS			PBKS+		
	Insertion (s)	**Trapdoor** (ms)	**Search** (ms)	**Insertion** (s)	**Trapdoor** (ms)	**Search** (ms)
2	3.418	0.336	1.07	3.420	0.858	1.05
3	3.410	0.328	1.55	3.432	1.280	1.58
4	3.435	0.331	2.19	3.457	1.760	2.12
5	3.461	0.323	2.51	3.472	2.150	2.54

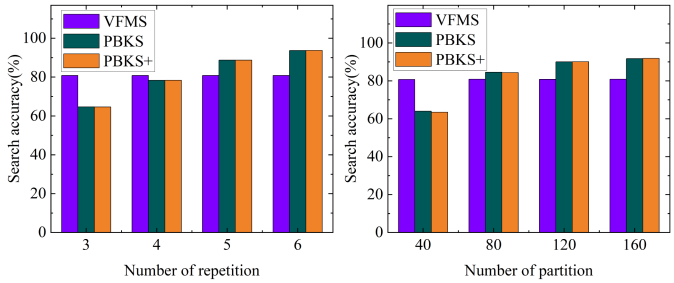
TABLE V
TIME COSTS OF PBKS AND PBKS+ IN EACH PHASE VARYING WITH THE NUMBER OF DATA ($r = 5$, $b = 1000$, $m = 400000$)

n ($\times 10^4$)	PBKS			PBKS+		
	Insertion (s)	**Trapdoor** (ms)	**Search** (s)	**Insertion** (s)	**Trapdoor** (s)	**Search** (s)
2	5.965	4.09	0.1662	6.018	0.027	0.1502
4	6.520	4.08	0.2521	6.515	0.0269	0.2501
6	7.272	4.04	0.3638	7.305	0.0267	0.3402
8	8.086	4.05	0.4145	8.041	0.0269	0.4249
10	9.076	4.05	0.5121	9.062	0.0268	0.5215

is constant. However, PBKS+ needs to generate r different trapdoors for r different BFs, so the *Trapdoor* time gradually increases. In *Search* phase, PBKS and PBKS+ need to search all BFs, so the *Search* time also increases linearly, and PBKS and PBKS+ take the same amount of time.

In Fig. 9(c), (e) and (h), the total length of BF increases with the increase of repetitions r. That is, the length of single BF m is constant, and the total length of BF is $m \times r$. To demonstrate the variety of time cost at each phase when r increases but the total BF length is constant, Table IV compares the time costs of PBKS and PBKS+ in *Insertion*, *Trapdoor* and *Search* phases, where the length of a single BF is equal to the total length of BF divided by r. It can be seen that PBKS+ takes longer time than PBKS in the *Insertion* phase. However, the both schemes spend the same time cost in the *Search* phase. Moreover, the time cost of PBKS in *Trapdoor* phase is not affected by r, but the time cost of PBKS+ in *Trapdoor* phase increases with r. In addition, to demonstrate the efficiency of our PBKS and PBKS+ schemes in each phase under the context of large cloud data, Table V shows the time cost of each phase when the number of data increases from 20000 to 100000. It can be seen that when n=100000, PBKS+ spends 9.062 s, 0.0268 s and 0.5215 s in *Insertion*, *Trapdoor* and *Search* phase, respectively, which is obviously acceptable to the data users.

Fig. 10 shows the search accuracies of the three schemes as the number of repetitions and partitions increases. It can be seen that the search accuracy of PBKS and PBKS+ increases as r and b increase and the search accuracy of VFMS is constant, because the Bloom filters used in PBKS and PBKS+ are CSC-BF and the Bloom filter used in VFMS is BIGSI. Finally, the search accuracies of PBKS and PBKS+ can reach more than 90%.



(a) Search accuracy (n = 1500, b = 100) (b) Search accuracy (n = 1500, r = 5)

Fig. 10. Search accuracy analysis.

VIII. CONCLUSION

In this paper, we proposed two privacy-preserving Bloom filter-based keyword search schemes over large encrypted cloud data. First, CSC-BF is used to store the keywords of all data and implement the search simultaneously. Then, we use SHVE to encrypt CSC-BF and propose the basic PBKS scheme. In addition, inspired by TBF, a new index structure T-CSCBF is constructed to further protect the BF value of the basic PBKS scheme. We rigorously analyze the security of our schemes under IND-SCPA. Finally, performance analysis shows that our schemes can achieve efficient keyword search.

REFERENCES

[1] X. Li et al., "VRFMS: Verifiable ranked fuzzy multi-keyword search over encrypted data," *IEEE Trans. Serv. Comput.*, vol. 16, no. 1, pp. 698–710, Jan./Feb. 2023, doi: 10.1109/TSC.2021.3140092.
[2] J. Chen et al., "EliMFS: Achieving efficient, leakage-resilient, and multi-keyword fuzzy search on encrypted cloud data," *IEEE Trans. Serv. Comput.*, vol. 13, no. 6, pp. 1072–1085, Nov./Dec. 2020.
[3] S. Gao, Y. Chen, J. Zhu, Z. Sui, R. Zhang, and X. Ma, "BPMS: Blockchain-based privacy-preserving multi-keyword search in multi-owner setting," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2022.3196712.
[4] Q. Jiang, E.-C. Chang, Y. Qi, S. Qi, P. Wu, and J. Wang, "Rphx: Result pattern hiding conjunctive query over private compressed index using Intel SGX," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 1053–1068, 2022.
[5] Q. Tong, Y. Miao, J. Weng, X. Liu, K.-K. R. Choo, and R. Deng, "Verifiable fuzzy multi-keyword search over encrypted data with adaptive security," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 5386–5399, May 2023.
[6] C. Guo, S. Su, K.-K. R. Choo, P. Tian, and X. Tang, "A provably secure and efficient range query scheme for outsourced encrypted uncertain data from cloud-based Internet of Things systems," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 1848–1860, Feb. 2022.
[7] J. Du, J. Zhou, Y. Lin, W. Zhang, and J. Wei, "Secure and verifiable keyword search in multiple clouds," *IEEE Syst. J.*, vol. 16, no. 2, pp. 2660–2671, Jun. 2022.
[8] X. Wang, J. Ma, F. Li, X. Liu, Y. Miao, and R. H. Deng, "Enabling efficient spatial keyword queries on encrypted data with strong security guarantees," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 4909–4923, 2021.
[9] R. Li and A. X. Liu, "Adaptively secure and fast processing of conjunctive queries over encrypted data," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 4, pp. 1588–1602, Apr. 2022.
[10] R. Li and A. X. Liu, "Adaptively secure conjunctive query processing over encrypted data for cloud computing," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 697–708.
[11] X. Zhu, E. Ayday, and R. Vitenberg, "A privacy-preserving framework for outsourcing location-based services to the cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 1, pp. 384–399, Jan./Feb. 2021.

[12] Q. Tong, X. Li, Y. Miao, X. Liu, J. Weng, and R. Deng, "Privacy-preserving boolean range query with temporal access control in mobile computing," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 5159–5172, May 2023, doi: 10.1109/TKDE.2022.3152168.
[13] Z. Shi, X. Fu, X. Li, and K. Zhu, "ESVSSE: Enabling efficient, secure, verifiable searchable symmetric encryption," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 7, pp. 3241–3254, Jul. 2022.
[14] P. Bradley, H. C. D Bakker, E. P. C. Rocha, G. M. Vean, and Z. Iqbal, "Ultrafast search of all deposited bacterial and viral genomic data," *Nature Biotechnol.*, vol. 37, pp. 152–159, Feb. 2019.
[15] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 139–152.
[16] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*. Berlin, Germany: Springer, 2007, pp. 1–287.
[17] M. Umer, T. Azim, and Z. Pervez, "Reducing communication cost of encrypted data search with compressed bloom filters," in *Proc. IEEE 16th Int. Symp. Netw. Comput. Appl.*, 2017, pp. 1–4.
[18] H. Yao, N. Xing, J. Zhou, and Z. Xia, "Secure index for resource-constraint mobile devices in cloud computing," *IEEE Access*, vol. 4, pp. 9119–9128, 2016.
[19] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, 2000, pp. 44–55.
[20] Q. Huang, J. Du, G. Yan, Y. Yang, and Q. Wei, "Privacy-preserving spatio-temporal keyword search for outsourced location-based services," *IEEE Trans. Serv. Comput.*, vol. 15, no. 6, pp. 3443–3456, Nov./Dec. 2022, doi: 10.1109/TSC.2021.3088131.
[21] P. Chaudhari and M. L. Das, "Privacy preserving searchable encryption with fine-grained access control," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 753–762, Second Quarter 2021.
[22] M. Zeng, H. Qian, J. Chen, and K. Zhang, "Forward secure public key encryption with keyword search for outsourced cloud storage," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 426–438, First Quarter 2022.
[23] A. Wu, A. Yang, W. Luo, and J. Wen, "Enabling traceable and verifiable multi-user forward secure searchable encryption in hybrid cloud," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1886–1898, Second Quarter 2023, doi: 10.1109/TCC.2022.3170362.
[24] X. Wang et al., "Search me in the dark: Privacy-preserving Boolean range query over encrypted spatial data," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2253–2262.
[25] F. Song, Z. Qin, L. Xue, J. Zhang, X. Lin, and X. Shen, "Privacy-preserving keyword similarity search over encrypted spatial data in cloud computing," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 6184–6198, Apr. 2022.
[26] Y. Li et al., "DVREI: Dynamic verifiable retrieval over encrypted images," *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1755–1769, Aug. 2022.
[27] Z. Xia, L. Wang, J. Tang, N. N. Xiong, and J. Weng, "A privacy-preserving image retrieval scheme using secure local binary pattern in cloud computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 1, pp. 318–330, First Quarter 2021.
[28] Z. Xia, L. Jiang, D. Liu, L. Lu, and B. Jeon, "BOEW: A content-based image retrieval scheme using bag-of-encrypted-words in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 202–214, Jan./Feb. 2022.
[29] G. Gupta, B. Coleman, T. Medini, V. Mohan, and A. Shrivastava, "RAMBO: Repeated and merged bloom filter for multiple set membership testing (MSMT) in sub-linear time," 2019, *arXiv: 1910.02611*.
[30] R. Li et al., "Building fast and compact sketches for approximately multi-set multi-membership querying," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2021, pp. 1077–1089.
[31] S. Lai et al., "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 745–762.

Yanrong Liang received the MS degree from the Department of Applied Mathematics, Shaanxi Normal University, Xi'an, China, in 2021. She is currently working toward the PhD degree with the Department of Cyber Engineering, Xidian University, Xi'an, China. Her research interests include cloud computing security and applied cryptography.

Jianfeng Ma (Member, IEEE) received the ME and PhD degrees in computer software and communications engineering from Xidian University, in 1988 and 1995, respectively. He is currently a professor and a PhD supervisor with the School of Cyber Engineering, Xidian University, China. He is also the director with the Shaanxi Key Laboratory of Network and System Security. His research interests include information and network security, coding theory, and cryptography.

Yinbin Miao (Member, IEEE) received the BE degree from the Department of Telecommunication Engineering, Jilin University, Changchun, China, in 2011, and the PhD degree from the Department of Telecommunication Engineering, Xidian University, Xi'an, China, in 2016. He is also a postdoctor with Nanyang Technological University from September 2018 to September 2019, and a postdoctor with the City University of Hong Kong from December 2019 to December 2021. He is currently an associate professor with the Department of Cyber Engineering, Xidian University, Xi'an, China. His research interests include information security and applied cryptography.

Da Kuang received the BS degree from the Department of Information Security, Hainan University, Hainan, China, in 2022. He is currently working toward the MS degree with the Department of Cyber Engineering, Xidian University, Xi'an, China. His research interests include information security and federated learning.

Xiangdong Meng, photograph and biography not available at the time of publication.

Robert H. Deng (Fellow, IEEE) is AXA chair professor of cybersecurity and professor of information systems with the School of Information Systems, Singapore Management University since 2004. His research interests include data security and privacy, multimedia security, network and system security. He has served on the editorial boards of many international journals, including the TFIS, *IEEE Transactions on Dependable and Secure Computing*. He has received the Distinguished Paper Award (NDSS 2012), Best Paper Award (CMS 2012), Best Journal Paper Award (IEEE Communications Society 2017).