Efficient Private Set Intersection and Searchable Encryption using Homomorphic Bloom Filters

Anil Kumar Pradhan[1][0000−0002−1976−2529], Killari Nandini[2][0009−0007−1869−0440], Harsh Kasyap[2][0000−0002−8313−6354], and Sayantan Mukherjee[3][0000−0002−1236−8086]

[1] Vaultree Ltd, Ireland
akpradhan.math@gmail.com
[2] Indian Institute of Technology (BHU), Varanasi, India
{killari.nandini.cse23, hkasyap.cse}@iitbhu.ac.in
[3] Indian Institute of Technology Jammu, India
csayantan.mukherjee@gmail.com

Abstract. Existing encrypted search and private set intersection (PSI) protocols struggle to reconcile post-quantum security with practical efficiency, often leaking search and access patterns or requiring prohibitively deep fully homomorphic encryption (FHE) circuits. We address these limitations by introducing a new *Homomorphic Bloom Filters* (HBF) framework, a quantum-resilient framework that embeds length-m Bloom filters directly into the plaintext space of an RLWE-based FHE scheme, enabling shallow homomorphic evaluation and matching without structural leakage. Building on HBF, we construct a searchable encryption (SE) scheme and a private set intersection (PSI) protocol, both based on a depth-1 homomorphic *missing-bit* circuit. The SE scheme requires no rotations or bootstrapping at server side and incurs no additional computational cost as the number of query keywords increases. The PSI protocol reduces each packed Bloom-filter comparison to a single ciphertext–plaintext multiplication, with cost depending on the Bloom-filter length rather than direct element-wise comparisons with the responder's set. This framework confines leakage to benign dataset dimensions, tunable false-positive rates, and other public metadata, thereby eliminating explicit pattern leakage.

Keywords: Fully Homomorphic Encryption · Bloom Filter · Searchable Encryption · Private Set Intersection.

1 Introduction

Heavily regulated domains such as healthcare, genomics and finance increasingly rely on cloud infrastructure for sensitive data storage and management, exposing critical assets to risks of unauthorized access and re-identification [2, 20]. Supporting encrypted search and distributed membership testing with post-quantum guarantees therefore demands strong cryptographic constructions. Searchable

encryption (SE) and private set intersection (PSI) are foundational primitives, and lattice-based fully homomorphic encryption (FHE) remains among the most promising approaches for quantum-resilient, privacy-preserving data operations.

Existing searchable encryption schemes often trade sub-linear query efficiency for search and access-pattern leakage [1,9,15,19,25,29]. Such structural leakage is incompatible with regulatory and ethical obligations. FHE over (R)LWE lattices eliminates explicit leakage while providing post-quantum security [1, 19, 23], but remains costly due to deep arithmetic circuits, ciphertext expansion, and bootstrapping, limiting scalability in large cloud deployments [1, 19].

Bloom filters offer compact, probabilistic set membership with strong space efficiency [4, 24]. They have been widely used to encode document attributes for sub-linear and multi-keyword queries, including homomorphic or attribute-based secure querying [15, 16, 21, 30]. However, inadequately masked filter bit-patterns expose structural leakage and correlation attacks [3, 24]. Prior works have combined Bloom filters with homomorphic primitives for 0-RTT handshakes [12], location services [14], crowd monitoring [26], and biometric template protection [3]. Yet these remain application-specific, often rely on Paillier, RSA, or bilinear assumptions, and do not provide a general RLWE-based FHE framework where Bloom filters serve as the *central* plaintext space for encrypted search and private set operations.

Existing work and observations. Recent post-quantum PSI and SE protocols include several lattice-based and homomorphic constructions: Chen et al. [7] use BFV-style SIMD packing with low-degree polynomials for unbalanced PSI; Cai et al. [6] evaluate encrypted Bloom filters under somewhat homomorphic encryption; Hu and Dai [17] combine Bloom filter encoding with high-degree oblivious polynomial evaluation for threshold PSI and cardinality estimation; and other lattice-based approaches achieve linear complexity without FHE [10]. However, Bloom filters remain auxiliary encodings rather than a central cryptographic primitive, and no existing construction provides a unified framework for encrypted search and set intersection under a shallow RLWE-based FHE circuit.

Despite these advances, current RLWE/FHE-based PSI schemes still evaluate membership through rotations, bootstrapping, or high-degree polynomial circuits rather than simple algebraic operations on natively compressed representations. For example, Hu and Dai's membership indicators $\delta(c_i, S)$ require high-degree polynomial evaluation with non-constant multiplicative depth and numerous ciphertext multiplications [17]; similar bottlenecks persist across other FHE and lattice-based designs [6, 7, 10, 11]. Moreover, these architectures do not share primitives with searchable encryption indices or support fuzzy membership semantics, and relegate Bloom filters to transient implementation details rather than the core plaintext space.

Our work. Motivated by these observations, we adopt a new design principle: place *all confidentiality* in an RLWE-based FHE layer, treating Bloom filters strictly as compact, probabilistic encodings. Document keyword sets (including fuzzy n-gram or LSH expansions) and queries are mapped to Bloom filters $B_D, q \in \{0,1\}^m$ sharing identical parameters; further packed into BFV plaintexts

and encrypted, the server stores and evaluates only ciphertexts, never observing bit patterns, query tokens, or match scores in the clear.

Search and membership testing leverage the shallow, depth-1 circuit $\mathsf{CT}_{\mathrm{miss},D} = \mathsf{CT}_Q - (\mathsf{CT}_D \odot \mathsf{CT}_Q)$, which corresponds slotwise to $r_i = q_i(1 - B_D[i])$ and reveals required missing bits upon client decryption. This single primitive unifies both searchable encryption and private set membership: in SE, it yields exact multi-keyword conjunction and fuzzy search via pattern thresholding; in PSI/PMT, it permits private server-side membership testing. Thus, the framework combines Bloom-filter compactness and fuzzy flexibility with the post-quantum security of RLWE-based FHE at a constant multiplicative depth of 1.

Key contributions in this work are as follows.

- We introduce a new *homomorphic Bloom filters* (HBF), which treat Bloom filters themselves as the native plaintext domain of an RLWE-based FHE scheme.
- We design an *FHE-native Bloom-filter matching technique* based on a depth-1 homomorphic circuit, implementing encrypted bitwise matching and missing-bit detection with a single multiplication, and applying uniformly to searchable encryption and private set membership/intersection.
- We demonstrate that the scheme remains *practically scalable*: Bloom filters keep the plaintext representation compact, SIMD packing enables batching, and the resulting costs make encrypted search and PSI/PMT realistic for large, privacy-sensitive datasets.

2 Preliminaries

In this section we fix notation, recall basic facts about Bloom filters, introduce the Ring-Learning-With-Errors (RLWE) assumption and the BFV fully homomorphic encryption scheme, and review the standard syntax and leakage-based security notions for searchable encryption and private set intersection. These ingredients will be combined in our constructions.

2.1 Notation

We write $[n] := \{1, \ldots, n\}$ and denote by $\{0,1\}^*$ the set of all finite binary strings. For a vector $x = (x_1, \ldots, x_m) \in \mathbb{Z}^m$, we write $\|x\|_0 := |\{i : x_i \neq 0\}|$ for its Hamming weight and $\langle x, y \rangle := \sum_{i=1}^{m} x_i y_i$ for the standard inner product. The security parameter is denoted by $\lambda \in \mathbb{N}$; a function $\mu(\lambda)$ is *negligible* if for every polynomial p there exists N such that $|\mu(\lambda)| < 1/p(\lambda)$ for all $\lambda > N$.

For a finite set S, we write $x \xleftarrow{\$} S$ for sampling x uniformly at random from S. For a probabilistic algorithm $\mathcal{A}$, we write $y \leftarrow \mathcal{A}(x)$ for running $\mathcal{A}$ on input x with fresh randomness and obtaining output y. Two distribution ensembles $\{X_\lambda\}_\lambda$ and $\{Y_\lambda\}_\lambda$ are *computationally indistinguishable*, written $X_\lambda \approx_c Y_\lambda$, if for every probabilistic polynomial-time (PPT) distinguisher $\mathcal{D}$ the distinguishing advantage

$$|\Pr[\mathcal{D}(X_\lambda) = 1] - \Pr[\mathcal{D}(Y_\lambda) = 1]|$$

is negligible in λ.

2.2 Bloom Filters

Bloom filters are classical probabilistic data structures for approximate membership queries [4, 5]: a finite set $S \subseteq U$ is represented by a bit array $B_S \in \{0,1\}^m$ and k (pseudo)independent hash functions $h_1, \ldots, h_k : U \to \{0, \ldots, m-1\}$, where each $x \in S$ sets the bits $B_S[h_i(x)] \leftarrow 1$ and a query y is answered "yes" iff all $B_S[h_i(y)] = 1$. This yields no false negatives, while the standard analysis shows that after inserting $n = |S|$ distinct elements the false-positive probability for $y \notin S$ satisfies

$$p(m,n,k) = \left(1 - \left(1 - \tfrac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k,$$

which is minimised (up to lower-order terms) by choosing $k \approx (m/n)\ln 2$ and $m/n \approx -\ln p/(\ln 2)^2$ for a target $p \in (0,1)$ [4, 5]. Bloom filters are asymptotically space-efficient among approximate membership structures, and numerous variants (counting, compressed, scalable, partitioned) and related filters are surveyed in [5, 28]; in this work we rely solely on the standard static variant and its classical parameter relations.

2.3 RLWE and the Fully Homomorphic Encryption Scheme

Our constructions rely on the Ring-Learning-With-Errors (RLWE) assumption over the cyclotomic ring $R_q \cong \mathbb{Z}_q[X]/(X^N + 1)$, where N is a power of two and q is a ciphertext modulus [22]. Informally, decisional RLWE states that, for suitable parameters and an error distribution concentrated on "small" polynomials, polynomially many noisy ring-linear samples $(a, as + e) \in R_q^2$ with secret $s \in R_q$ are computationally indistinguishable from uniformly random pairs. Lyubashevsky, Peikert, and Regev show that, for appropriate cyclotomic rings and parameters, RLWE is at least as hard as certain worst-case problems on ideal lattices for quantum adversaries [22], providing a standard post-quantum hardness foundation.

On top of RLWE we employ the Brakerski–Fan–Vercauteren (BFV) leveled fully homomorphic encryption scheme [13], instantiated over (R_q, R_t) with plaintext modulus t and ciphertext modulus q. BFV provides public-key encryption, homomorphic addition and homomorphic multiplication for plaintexts in R_t, supports SIMD "batch" encodings of up to N slots, and is IND-CPA secure under the RLWE assumption for appropriate parameter choices [1, 13, 19]. In this work, we use BFV as a depth-1 leveled FHE scheme: all of our homomorphic computations consist of a single ciphertext–ciphertext or ciphertext–plaintext multiplication plus a small number of additions, with no bootstrapping, and we rely on BFV's batch encoding to store an entire Bloom filter as a single plaintext vector.

2.4 Functionalities and Adversarial Model

We briefly describe the abstract functionalities and the adversarial model adopted in the rest of the paper.

Searchable encryption functionality. Let $\mathcal{D}$ be a document collection, each document represented as a subset of a keyword (or feature) universe $\mathcal{W}$. Following the static searchable-symmetric-encryption (SSE) framework of Curtmola et al. [9] and subsequent surveys [1, 19], an SE scheme consists of PPT algorithms

$$\mathsf{SE} = (\mathsf{Setup}, \mathsf{EncIndex}, \mathsf{QueryGen}, \mathsf{Search}, \mathsf{Dec})$$

where Setup outputs secret state and public parameters, $\mathsf{EncIndex}$ takes $(\mathcal{D}, \cdot)$ and produces an encrypted index (and encrypted documents) stored on the server, $\mathsf{QueryGen}$ maps a user query to a search token or encrypted query, Search runs on the server and returns an encrypted answer, and Dec runs at the client and recovers the set of matching documents (up to Bloom-filter false positives in our setting). Security is defined via a leakage function $\mathcal{L} = (\mathcal{L}_{\mathsf{setup}}, \mathcal{L}_{\mathsf{search}})$ and the standard real/ideal indistinguishability notion: for any PPT adversary controlling the server, there exists a PPT simulator that can generate a view indistinguishable from the real execution given only $\mathcal{L}$ and no access to the plaintexts [1, 9, 19].

PMT and PSI functionalities. For two-party private set operations, let $X, Y \subseteq \mathcal{U}$ be finite sets held by parties P_1 and P_2, respectively. The private membership test (PMT) functionality $\mathcal{F}_{\mathrm{PMT}}$ (one-sided PSI) takes as input a set Y from P_2 and an element d (or small set X) from P_1 and returns to P_1 the membership bit $b = \mathbf{1}[d \in Y]$ (or the intersection $X \cap Y$), revealing nothing to P_2. The two-sided PSI functionality $\mathcal{F}_{\mathrm{PSI}}$ takes (X, Y) as input and outputs $X \cap Y$ to both parties. Existing RLWE/FHE-based PSI and PSI-cardinality constructions [7, 10, 17] typically realise these functionalities in the semi-honest model; our goal is analogous, but with Bloom filters as the primary plaintext representation and depth-1 homomorphic circuits.

Adversarial model. In the SE setting, we consider an honest-but-curious (semi-honest) cloud server: it faithfully executes ($\mathsf{EncIndex}, \mathsf{Search}$) on the encrypted index and queries, but may attempt to infer information from all values it sees. Concretely, the server's view consists of the public parameters, the encrypted index, all encrypted queries, and all encrypted result ciphertexts; it never observes decrypted documents or match scores. Security is defined via the SE real/ideal experiment with respect to an explicit leakage function $\mathcal{L}$ that may include benign size information (e.g. number of documents, Bloom-filter length, number of queries) but is designed not to reveal plaintext search or access patterns.

In the PMT and PSI setting, we consider semi-honest parties: each party follows the protocol but records its messages, randomness and local state, and may attempt to infer additional information about the other party's set. We require that for each corrupted party there exists a simulator that, given only that party's input and output to $\mathcal{F}_{\mathrm{PMT}}$ or $\mathcal{F}_{\mathrm{PSI}}$ (and trivial leakage such as set sizes and public parameters), can produce an indistinguishable view of the protocol execution. In all cases we assume that the underlying BFV/BGV scheme

is IND-CPA secure under RLWE (Section 2.3), so that ciphertexts of Bloom filters and derived values are semantically secure.

3 Homomorphic Bloom Filters

This section formalises a new homomorphic Bloom-filter (HBF) primitive that integrates classical Bloom filters with RLWE-based FHE schemes such as BFV [13]. The primitive provides encrypted representations of Bloom filters and supports homomorphic evaluation of bitwise conjunction and a "missing-bit" transformation using circuits of multiplicative depth 1.

HBF Parameter Sets and Embedding. Let U be a fixed universe of elements and let $\Pi_{\mathrm{BF}} = (m, k, \{h_i\}_{i=1}^{k})$ be Bloom-filter parameters, where $m, k \in \mathbb{N}$ and $h_i : U \to \{0, \ldots, m-1\}$ are (pseudo)independent hash functions as in Section 2.2. For any finite set $S \subseteq U$, denote by $B_S \in \{0,1\}^m$ the Bloom filter of S constructed with Π_{BF}.

Let (N, q, t) be BFV parameters and let $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ and $R_t = \mathbb{Z}_t[X]/(X^N + 1)$ be the ciphertext and plaintext rings, respectively, as in Section 2. As the BFV plaintext space supports a SIMD/batching encoding with at least m usable slots, assume $m \leq N$ so that Bloom-filter bits can be embedded as plaintext slots. Under this embedding, plaintext multiplication acts slotwise on the encoded Bloom-filter bits. Let $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a BFV scheme over (R_q, R_t) that supports addition and multiplication of ciphertexts and ciphertext–plaintext multiplication [13]. The construction also applies to BGV-style plaintext batching, but we describe and evaluate the BFV instantiation.

Definition 1 (Bloom-filter embedding). *A Bloom-filter embedding with parameters (m, N, t) is a pair of maps*

$$\mathsf{pack} : \{0,1\}^m \to R_t, \qquad \mathsf{unpack} : R_t \to \{0,1\}^m,$$

such that for all $B, B' \in \{0,1\}^m$ and all $j \in \{0, \ldots, m-1\}$,

1. $\mathsf{unpack}(\mathsf{pack}(B)) = B$,
2. $\big(\mathsf{pack}(B) \cdot \mathsf{pack}(B')\big)_j \equiv B[j] \cdot B'[j] \pmod{t}$,

where $(\cdot)_j$ denotes the j-th plaintext slot of the product and $B[j] \cdot B'[j] \in \{0,1\}$.

A concrete instantiation is obtained by treating the m bits of B as a vector and applying batch encoding to obtain a polynomial in R_t; in this case, the coefficient-wise product realizes the bitwise conjunction due to the identification $\{0,1\} \subset \mathbb{Z}_t$ and the multiplicative structure of R_t [13].

3.1 Homomorphic Bloom-Filter Primitive

Definition 2 (Homomorphic Bloom Filter (HBF)). *Fix Bloom-filter parameters $\Pi_{\mathrm{BF}} = (m, k, \{h_i\})$, BFV parameters (N, q, t), and an embedding* $(\mathsf{pack}, \mathsf{unpack})$

as in Definition 1. A homomorphic Bloom-filter scheme

$$\mathsf{HBF} = (\mathsf{HBF.Setup}, \mathsf{HBF.EncBF}, \mathsf{HBF.ANDctpt}, \mathsf{HBF.Missing})$$

is defined as follows in Table 1.

Table 1. HBF Scheme Underlying Algorithms.

Algorithm	**Operation Steps**
$\mathsf{HBF.Setup}(1^\lambda)$	1. Define $\mathsf{params} := (\Pi_{\mathrm{BF}}, N, q, t, \mathsf{pack}, \mathsf{unpack})$; 2. Obtain FHE key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$; 3. Output $\{\mathsf{params}, \mathsf{pk}, \mathsf{sk}\}$
$\mathsf{HBF.EncBF}(\mathsf{pk}, B)$	1. Compute $P := \mathsf{pack}(B) \in R_t$; 2. Output: Ciphertext $C_B := \mathsf{Enc}_{\mathsf{pk}}(P) \in \mathcal{C}$.
$\mathsf{HBF.ANDctpt}(C_B, B')$	1. Compute $P' := \mathsf{pack}(B')$; 2. Evaluate : $C_{B\wedge B'} := C_B \odot_{\mathrm{pt}} P'$; 3. Output $C_{B\wedge B'}$ (which encrypts $\mathsf{pack}(B \wedge B')$).
$\mathsf{HBF.Missing}(C_D, C_Q)$	1. Accept ciphertexts C_D, C_Q encrypting document and query filters B_D, B_Q; 2. Compute $C_{\mathrm{miss}} := C_Q - (C_D \cdot C_Q)$; 3. Output C_{miss} (which encrypts the vector with coordinates $q_i(1 - B_D[i])$).

The primitive HBF.ANDctpt corresponds to homomorphic Bloom-filter intersection, while HBF.Missing corresponds to an encrypted "missing-bit" indicator vector suitable for searchable encryption. Both rely on the fact that multiplication in the plaintext ring implements bitwise conjunction on packed Bloom-filter bits when these bits are represented in $\{0, 1\} \subset \mathbb{Z}_t$ [13].

3.2 Correctness and Complexity of HBF Operations

Correctness of the HBF operations follows from BFV correctness and the properties of the embedding.

Lemma 1 (Correctness of HBF Operations). *Let* HBF *be as in Definition 2. Let* $B_1, B_2 \in \{0,1\}^m$*, let* $C_{B_i} \leftarrow \mathsf{HBF.EncBF}(\mathsf{pk}, B_i)$ *for* $i \in \{1, 2\}$*, and let* $C_{B_1\wedge B_2} \leftarrow \mathsf{HBF.ANDctpt}(C_{B_1}, B_2)$ *and* $C_{\mathrm{miss}} \leftarrow \mathsf{HBF.Missing}(C_{B_1}, C_{B_2})$*. Then, except with negligible probability, decryption satisfies:*

1. $\mathsf{unpack}\big(\mathsf{Dec}_{\mathsf{sk}}(C_{B_1\wedge B_2})\big) = B_1 \wedge B_2$*, where* $(B_1 \wedge B_2)[j] = B_1[j] \cdot B_2[j]$ *for all* $j \in \{0, \ldots, m-1\}$.

2. $r := \mathsf{unpack}\big(\mathsf{Dec}_{\mathsf{sk}}(C_{\mathrm{miss}})\big) \in \mathbb{Z}_t^m$ *satisfies* $r[j] \equiv B_2[j] \cdot (1 - B_1[j]) \pmod t$ *for all* $j \in \{0, \ldots, m-1\}$.

Proof. The claim follows from BFV correctness [13] and Definition 1. Ciphertext–plaintext multiplication decrypts to $\mathsf{pack}(B_1)\mathsf{pack}(B_2)$, whose j-th slot is $B_1[j]B_2[j]$. Similarly,

$$\mathsf{Dec}_{\mathsf{sk}}(C_{\mathrm{miss}}) = \mathsf{pack}(B_2) - \mathsf{pack}(B_1)\mathsf{pack}(B_2),$$

whose j-th slot is $B_2[j](1 - B_1[j])$. Applying unpack gives both statements.

Lemma 1 implies that $r[j] = 0$ whenever $B_Q[j] = 0$ or $B_D[j] = 1$, and $r[j] = 1$ precisely at positions where $B_Q[j] = 1$ and $B_D[j] = 0$. In particular, if all positions set by the query Bloom filter are also set in the document Bloom filter (i.e., the Bloom-filter membership test passes), then r is the all-zero vector.

Theorem 1 (Complexity of HBF Operations). *Let* HBF *be as in Definition 2 and let* $m \leq N$. *Then:*

1. HBF.ANDctpt *uses one ciphertext–plaintext multiplication.*
2. HBF.Missing *uses one ciphertext–ciphertext multiplication and one ciphertext subtraction.*

In particular, all HBF operations are supported by a leveled BFV instantiation with multiplicative depth 1 *and do not require bootstrapping.*

Proof. The cost bounds follow from the definition of BFV encryption and homomorphic operations.

Security of HBF-based constructions against semi-honest adversaries reduces to the IND-CPA security of BFV under the RLWE assumption [13, 22].

4 Construction of the PMT and PSI Protocols

We now instantiate the private membership test (PMT) and two-sided private set intersection (PSI) protocols using the homomorphic Bloom-filter primitive HBF from Section 3.

In the two-sided PSI setting, P_1 holds $X_1 \subseteq U$ and P_2 holds $X_2 \subseteq U$. In the PMT setting, P_1 holds a query element $d \in U$ or a small query set, while P_2 holds X_2.

4.1 Two-Sided PSI from HBF

We now describe a symmetric two-sided PSI protocol Π_{PSI} realizing $\mathcal{F}_{\mathrm{PSI}}(X_1, X_2) = X_1 \cap X_2$. Each party P_i executes the following steps, with $j = 3 - i$ denoting the counterparty.

Setup.

1. Executes: $(\mathsf{params}, \mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{HBF.Setup}(1^\lambda)$

2. Publishes the public key pk_i and retains the secret key sk_i privately.

Encryption.

1. Compute Bloom filter $B_i \in \{0,1\}^m$ of set X_i.
2. Compute: $C_i \leftarrow \mathsf{HBF.EncBF}(\mathsf{pk}_i, B_i)$.
3. Send the ciphertext C_i to party P_j.

Homomorphic Intersection. Upon receiving C_j from the counterparty, P_i computes:

1. $C_{j\wedge i} := \mathsf{HBF.ANDctpt}(C_j, B_i) = C_j \odot_{\mathrm{pt}} \mathsf{pack}(B_i)$
2. Sends the evaluation ciphertext $C_{j\wedge i}$ to party P_j.

Local Decryption and Extraction. Upon receiving $C_{i\wedge j}$ from the counterparty, P_i computes:

1. Decrypt the ciphertext: $P^{(i)}_{i\cap j} := \mathsf{Dec}_{\mathsf{sk}_i}(C_{i\wedge j})$.
2. Unpack the plaintext element: $B^{(i)}_{i\cap j} := \mathsf{unpack}(P^{(i)}_{i\cap j})$.
3. Return $\Big\{x \in X_i \mid \forall \ell \in [k],\, B^{(i)}_{i\cap j}[h_\ell(x)] = 1\Big\}$.

The set-level correctness follows directly from the algebraic correctness of the HBF combined with standard Bloom filter false-positive bounds (Lemma 1). Both parties recover the true intersection elements with no false negatives, while their outputs may include Bloom-filter false positives controlled by the chosen Bloom-filter parameters.

4.2 One-Sided Private Membership Test

As a simpler variant, we describe a one-sided PMT protocol Π_{PMT} realising $\mathcal{F}_{\mathrm{PMT}}$, in which only P_1 learns the membership result and only P_1 holds an FHE key.

Protocol Π_{PMT}.

Setup. P_1 runs $\mathsf{HBF.Setup}(1^\lambda)$ to obtain $(\mathsf{params}, \mathsf{sk})$, including pk. Structural parameters are published; sk is kept secret. P_2 learns Π_{BF} and (N, q, t) but holds no secret key.

Server preprocessing. P_2 computes the Bloom filter $B_Y \in \{0,1\}^m$ of its set Y and retains it in plaintext.

Query generation. For a query element $d \in U$, P_1 computes the Bloom filter $B_d \in \{0,1\}^m$ corresponding to the singleton set $\{d\}$, packs it as $P_d := \mathsf{pack}(B_d)$, and computes $C_d := \mathsf{Enc}_{\mathsf{pk}}(P_d)$, which it sends to P_2.

Answer computation. P_2, holding C_d and B_Y, computes

$$C_{d\wedge Y} := \mathsf{HBF.ANDctpt}(C_d, B_Y) = C_d \odot_{\mathrm{pt}} \mathsf{pack}(B_Y),$$

and returns $C_{d\wedge Y}$ to P_1.

Decision. P_1 decrypts and unpacks

$$P_{d\wedge Y} := \mathsf{Dec}_{\mathsf{sk}}(C_{d\wedge Y}), \qquad B_{d\wedge Y} := \mathsf{unpack}(P_{d\wedge Y}),$$

and outputs $b = 1$ ("present") iff $B_{d\wedge Y}[h_i(d)] = 1$ for all $i \in [k]$. Otherwise it outputs $b = 0$.

By Lemma 1, $B_{d\wedge Y} = B_d \wedge B_Y$ except with negligible probability; if $d \in Y$ then $b = 1$ with probability 1, while if $d \notin Y$ then $b = 1$ with probability at most p. Thus Π_{PMT} realises $\mathcal{F}_{\mathrm{PMT}}$ with correctness governed solely by the Bloom-filter false-positive rate; a formal security proof in the semi-honest model, under the IND-CPA security of BFV and the RLWE assumption, is given in Appendix 6.

5 Construction of the Searchable Encryption Scheme

We construct a Searchable Encryption (SE) scheme directly upon the homomorphic Bloom-filter primitive (HBF) from Section 3. Let U be a fixed universe of elements and let $\Pi_{\mathrm{BF}} = (m, k, \{h_i\}_{i=1}^{k})$ be Bloom-filter parameters, where $m, k \in \mathbb{N}$ and $h_i : U \to \{0, \ldots, m-1\}$ are (pseudo)independent hash functions as in Section 2.2. For any finite set $S \subseteq U$, denote by $B_S \in \{0,1\}^m$ the Bloom filter of S constructed with Π_{BF}.

5.1 System Parameters and Feature Encoding

Let $\mathcal{D} = \{D_1, \ldots, D_{n_{\mathsf{doc}}}\}$ denote a document collection. A public, deterministic extractor $\mathsf{FeatExtract} : \mathcal{D} \to 2^U$ maps each document D_j to a finite set $S_{D_j} \subseteq U$. This extractor dictates retrieval semantics (e.g., exact keywords, n-grams, or LSH-based fuzzy encodings) without altering the underlying cryptographic structure. Similarly, user queries $Q \in \mathcal{Q}$ are mapped to feature sets $S_Q := \mathsf{FeatQuery}(Q) \subseteq U$ via a corresponding deterministic encoder. Both document and query feature sets are then encoded into standard Bloom filters $B_{D_j}, B_Q \in \{0,1\}^m$, respectively, preparing them for homomorphic evaluation.

The client and server agree on fixed universe of elements U, bloom-filter parameters Π_{BF}, bloom-filter embedding $(\mathsf{pack}, \mathsf{unpack})$, BFV parameters (N, q, t) and the homomorphic Bloom-filter primitive.

$$\mathsf{HBF} = (\mathsf{HBF.Setup}, \mathsf{HBF.EncBF}, \mathsf{HBF.AND}_{\mathsf{ct-pt}}, \mathsf{HBF.Missing})$$

For each document D_j, the Bloom filter $B_{D_j} \in \{0,1\}^m$ is defined by

$$B_{D_j}[\ell] = 1 \iff \exists f \in S_{D_j}, \exists i \in [k] \text{ such that } h_i(f) = \ell.$$

Queries are treated analogously via a (possibly different) public, deterministic encoder $\mathsf{FeatQuery} : \mathcal{Q} \to 2^{\mathcal{F}}$ mapping user queries Q to feature sets $S_Q := \mathsf{FeatQuery}(Q)$.

5.2 Scheme Definition

The searchable encryption scheme is given by

$$\mathsf{HBF\text{-}SE} = (\mathsf{Setup}, \mathsf{IndexGen}, \mathsf{QueryGen}, \mathsf{Search}, \mathsf{Dec}),$$

with algorithms defined in Table 2.

Table 2. HBF-SE algorithms.

Algorithm	**Operation Steps**
$\mathsf{Setup}(1^\lambda)$	1. Run $(\mathsf{params}, \mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{HBF.Setup}(1^\lambda)$; 2. Publish $(\mathsf{params}, \mathsf{pk})$.
$\mathsf{IndexGen}(\mathsf{pk}, \mathcal{D})$	For each document $D_j \in \mathcal{D}$: a. Compute S_{D_j} and build B_{D_j}; b. Encrypt $C_{D_j} \leftarrow \mathsf{HBF.EncBF}(\mathsf{pk}, B_{D_j})$; c. Upload $I = \{(j, C_{D_j})\}_{j \in [n_{\mathrm{doc}}]}$.
$\mathsf{QueryGen}(\mathsf{pk}, Q)$	1. Compute S_Q and build B_Q; 2. Encrypt $C_Q \leftarrow \mathsf{HBF.EncBF}(\mathsf{pk}, B_Q)$; 3. Send C_Q and retain B_Q locally.
$\mathsf{Search}(I, C_Q)$	For each $(j, C_{D_j}) \in I$: a. Compute $C_{\mathrm{miss},j} \leftarrow \mathsf{HBF.Missing}(C_{D_j}, C_Q)$; b. Return all $(j, C_{\mathrm{miss},j})$.
$\mathsf{Dec}(\mathsf{sk}, C_{\mathrm{miss},j}, B_Q)$	1. Decrypt each $C_{\mathrm{miss},j}$; 2. Add j to M iff all positions in $\mathrm{supp}(B_Q)$ decrypt to zero.

Correctness. The correctness follows directly from the properties of HBF (Lemma 1) with the Bloom-filter analysis. HBF-SE realises exact Bloom-filter membership testing of the query feature set S_Q in each document's Bloom filter, with no false negatives and false positives governed solely by the standard Bloom-filter parameter p. When S_Q is derived from a fuzzy encoder (e.g. n-grams or LSH), the same analysis applies; retrieval accuracy is then determined jointly by the encoder and the Bloom-filter parameters.

6 Security Analysis

We analyse security in the standard simulation-based searchable-encryption paradigm, instantiated with explicit leakage functions. Since the real/ideal framework and the associated hybrid methodology are standard, we state only the scheme-specific leakage and give proof sketches reducing security to the IND-CPA security of the underlying BFV/BGV scheme.

6.1 Security Goals and Leakage

Our objective is to protect the confidentiality of document-side feature sets S_{D_j}, query-side feature sets S_Q, and server-side match outcomes against an honest-but-curious server. In particular, the server should not learn plaintext Bloom-filter contents, query keywords, or decrypted match scores. The only intended

leakage is benign metadata such as public parameters, collection size, and the number of issued queries.

Accordingly, we define

$$\mathcal{L}_{\mathsf{setup}}(\mathcal{D}) := (n, \text{params})$$

to reveal only the number of indexed documents and public system parameters, and

$$\mathcal{L}_{\mathsf{search}}(\mathcal{D}, Q^{(1)}, \ldots, Q^{(q)}) := (q, n)$$

to reveal only the number of queries and the number of indexed documents. In particular, our leakage does *not* include explicit search patterns, access patterns, or result patterns, provided that the server processes all indexed records uniformly and returns ciphertexts of fixed structure.

6.2 Simulation-Based Security Notion

We use the standard real/ideal simulation paradigm for searchable encryption parameterised by the leakage functions $\mathcal{L}_{\mathsf{setup}}$ and $\mathcal{L}_{\mathsf{search}}$. Informally, in the real world the adversarial server observes the genuine encrypted index, encrypted queries, and encrypted responses produced by the protocol; in the ideal world a simulator is given only the prescribed leakage and must generate a computationally indistinguishable view. We say that the scheme is $\mathcal{L}$-secure if no PPT adversary can distinguish these two worlds with non-negligible advantage.

6.3 Security of the Encrypted Search Scheme

The server view in our encrypted-search construction consists of: (i) ciphertexts of document Bloom filters $\{\mathsf{CT}_{D_j}\}_{j\in[n]}$, (ii) query ciphertexts $\mathsf{CT}_Q^{(1)}, \ldots, \mathsf{CT}_Q^{(q)}$, and (iii) encrypted result vectors or encrypted aggregate scores derived from homomorphic evaluation. All of these objects are BFV/BGV ciphertexts under the same public key.

Theorem 2. *Assume that the underlying BFV/BGV encryption scheme is IND-CPA secure under the RLWE assumption. Then the proposed encrypted-search construction is $\mathcal{L}$-secure against any PPT honest-but-curious server with leakage*

$$\mathcal{L} = (\mathcal{L}_{\mathsf{setup}}, \mathcal{L}_{\mathsf{search}}).$$

Proof. The simulator receives only $\mathcal{L}_{\mathsf{setup}}$ and $\mathcal{L}_{\mathsf{search}}$ and generates a simulated server view by replacing all genuine ciphertexts with fresh encryptions of dummy plaintexts of the correct shape, e.g. encryptions of 0^m for Bloom-filter vectors and encryptions of 0 for aggregated scores. By Lemma 3 (IND-CPA security of BFV/BGV), encryptions of the true Bloom-filter encodings are computationally indistinguishable from encryptions of these dummy vectors. The same argument applies to encrypted query Bloom filters and to all ciphertexts produced by homomorphic evaluation, since these are again BFV/BGV ciphertexts of fixed format. Consequently, the server's real view is computationally indistinguishable from the simulated view generated from leakage only, which proves $\mathcal{L}$-security.

The theorem captures the main security advantage of the construction over classical Bloom-filter-based SSE: the server never sees Bloom-filter bit patterns, deterministic search tokens, or plaintext result sets, and therefore cannot directly infer keyword statistics, token reuse, or access patterns from exposed index structures.

6.4 Security of the PMT/PSI Application

In the PMT/PSI application, the server holds a plaintext Bloom filter for its own set, while the client sends encrypted Bloom-filter queries under BFV/BGV. The server's cryptographic view therefore consists only of the client's encrypted query Bloom filter and the ciphertexts it homomorphically derives from it.

Theorem 3. *Assume that BFV/BGV is IND-CPA secure under the RLWE assumption. Then the PMT/PSI protocol derived from our construction preserves client-query privacy against any PPT honest-but-curious server: the server learns nothing about the client's queried element or set beyond trivial metadata such as the occurrence of a query.*

Proof. The proof is identical in structure to that of Theorem 2. A simulator replaces the client's encrypted Bloom-filter query by an encryption of a dummy vector such as 0^m, and replaces any derived ciphertext response by an encryption of a dummy scalar or vector of matching format. By Lemma 3, these substitutions are computationally indistinguishable from the genuine ciphertexts. Hence the server's view can be simulated from trivial leakage only, proving client-query privacy.

If the scalar-response variant is used, the client learns only a single membership bit (up to the Bloom-filter false-positive probability), while the server still learns nothing about the query content. Thus the PMT/PSI application inherits the same post-quantum cryptographic protection as the main encrypted-search construction.

7 Performance Evaluation

7.1 Experimental Setup

All experiments were conducted on a Dell laptop with a 13th Generation Intel® Core™ i5-1340P processor running Windows 11. The proposed HBF-SE scheme was implemented in C++ using the Microsoft SEAL library with the BFV homomorphic encryption scheme. Two polynomial modulus degrees were evaluated: $n \in \{2048, 4096\}$, with coefficient modulus $\{24, 24\}$ bits and plaintext modulus 2^{20}. Server-side keyword sets of sizes $|\mathcal{S}| \in \{10^4, 10^5, 10^6\}$ were constructed, and query workloads of $q \in \{1, 10, 100\}$ keywords were tested for scalability. Bloom filters were configured with a false-positive probability of $\varepsilon = 0.01$. The results are reported with an average of 10 runs.

7.2 Evaluation Metrics

We used following metrices to evaluate. The **Bloom Filter Size** ($|BF|$, KB) captures the storage footprint of the server-side index Bloom filter, which is determined by the keyword set size $|\mathcal{S}|$ and the target false-positive probability ε. The **Query Size** (KB) denotes the total ciphertext volume sent from the client to the server per query, while the **Response Size** (KB) measures the size of the homomorphically evaluated result ciphertext returned by the server. Together, these two sum to the **Total Communication Cost** (KB), which reflects the end-to-end bandwidth requirement of a single query round. Finally, **Server Time** (ms) records the end-to-end wall-clock query processing time, and **Throughput/QPS** (queries/sec) quantifies the number of keyword queries the server can process per second under each configuration.

7.3 Empirical Results

Table 3 reports the performance of the HBF-based PSI protocol across all set sizes and both polynomial modulus degrees. These results empirically validate the complexity bound established that each membership test requires exactly one ciphertext-plaintext multiplication and one ciphertext addition at multiplicative depth 1, with no bootstrapping. This is directly reflected in the server times, which are substantially lower.

Table 3. HBF-based PSI Performance

FHE Batch Size(n)	$\lvert\mathcal{S}\rvert$ Keyword	$\lvert BF\rvert$ **(KB)**	**Query (KB)**	**Resp. (KB)**	**Comm. (KB)**	**Time (ms)**
2048	10^4	93.68	3034.44	3034.44	6068.88	**11**
	10^5	936.812	30247.5	30247.5	60495.1	**151**
	10^6	9368.12	302443	302443	604886	**2160**
4096	10^4	93.68	3026.36	3026.36	6052.72	**24**
	10^5	936.812	30199.2	30199.2	60398.4	**322**
	10^6	9368.12	301670	301670	603340	**2792**

Table 4 compares the HBF-based PSI protocol against the recurrent unbalanced PSI scheme [8], which uses leveled FHE with Cuckoo hashing over the BFV library. It reports three configurations corresponding to the querier set sizes $|X| \in \{4, 16, 64\}$, with a fixed polynomial degree $n = 4096$.

Our protocol achieves competitive and in several configurations for superior server-side performance. At $|\mathcal{S}| = 10^4$, ours completes in 21 ms, outperforming [8] across all three querier set sizes. This advantage arises directly from the depth-1 circuit structure of HBF. At larger set sizes, [8] maintains its fixed communication and timing advantage due to its recurrent optimisation, while ours scales linearly in $|\mathcal{S}|$ as expected.

Table 4. PSI Comparison ($n = 4096$, $\varepsilon = 0.01$)

Scheme	$\lvert\mathcal{S}\rvert$	Query (KB)	Resp. (KB)	Comm. (KB)	Time (ms)
[8]	10^4	543/2161/8653	543/2161/8653	1086/4322/17306	**30/100/370**
Ours	10^4	3026.36	3026.36	6052.72	**21**
[8]	10^5	543/2161/8653	543/2161/8653	1086/4322/17306	**30/100/370**
Ours	10^5	30199.20	30199.20	60398.41	**186**
[8]	10^6	543/2161/8653	543/2161/8653	1086/4322/17306	**30/100/370**
Ours	10^6	301670.08	301670.08	603340.16	**2300**

Table 5 reports performance for single-keyword queries ($q = 1$) across all set sizes and both polynomial modulus degrees. The response size remains fixed at 32.28 KB for $n = 2048$ and 64.39 KB for $n = 4096$ across all values of $|\mathcal{S}|$, because the server always returns a single constant-size result ciphertext independent of the index size. Query size grows proportionally with $|\mathcal{S}|$ since the Bloom filter length m is determined by the set size and the target false-positive probability $\varepsilon = 0.01$. Server time observes a linear scaling.

Table 5. HBF-SE Performance: Single-Keyword Queries ($q = 1$)

n	$\lvert\mathcal{S}\rvert$	$\lvert BF\rvert$ (KB)	Query (KB)	Resp. (KB)	Comm. (KB)	Time (ms)	QPS
2048	10^4	93.68	3034.44	32.28	3066.72	**225**	4.44
	10^5	936.81	30247.5	32.28	30279.8	**2252**	0.44
	10^6	9368.12	302443	32.28	302475	**23113**	0.043
4096	10^4	93.68	3026.36	64.39	3090.75	**906**	1.10
	10^5	936.81	30199.2	64.39	30263.6	**8934**	0.11
	10^6	9368.12	301670	64.39	301734	**95968**	0.01

Table 6. HBF-SE Performance: Multi-Keyword Queries ($q = 10$)

n	$\lvert\mathcal{S}\rvert$	$\lvert BF\rvert$ (KB)	Query (KB)	Resp. (KB)	Comm. (KB)	Time (ms)	QPS
2048	10^4	93.68	3034.44	32.28	3066.72	**222**	45.04
2048	10^5	936.81	30247.5	32.28	30279.8	**2237**	4.47
2048	10^6	9368.12	302443	32.28	302475	**23236**	0.19
4096	10^4	93.68	3026.36	64.39	3090.75	**941**	10.62
4096	10^5	936.81	30199.2	64.39	30263.6	**10334**	0.96
4096	10^6	9368.12	301670	64.39	301734	**94056**	0.10

Table 6 reports performance for ten-keyword queries. It shows that query cost is *independent* of the number of query keywords, since all keywords are inserted into a single client-side Bloom filter and encrypted as one ciphertext, so neither the query size nor the server computation increases with the keyword count. Thus, we do not report results on 100 or more queries.

7.4 Comparison with Related Work

Classical SSE schemes [9, 15, 25, 29] achieve excellent concrete performance using only symmetric-key primitives and simple index lookups, often sub-linear in n_{doc}, but inherently reveal search and access patterns through deterministic tokens and plaintext indices [1, 9, 19]. Bloom-filter-based SSE [15, 16, 18, 21, 27] compresses indices and supports rich queries but exposes Bloom-filter bit patterns, enabling structural leakage. In contrast, our SE construction replaces all such plaintext structures by encrypted Bloom filters and depth-1 "missing-bit" circuits. The server cost grows linearly in n_{doc} with a small constant and no rotations or bootstrapping , positioning the scheme as a shallow-circuit homomorphic alternative to HE-based SE that must evaluate deeper comparison or ranking circuits [1, 19].

RLWE-FHE-based PMT/PSI protocols in [6, 7, 17] typically encode sets into encrypted integers and test membership by homomorphically evaluating polynomials or related encodings. As a result, they require many ciphertext–ciphertext multiplications and non-constant multiplicative depth; for example, Hu and Dai's threshold PSI evaluates a membership polynomial with dozens of multiplications and depth about $\log\log B + 2$ for domain size B [17]. Post-quantum PSI and PSI-cardinality schemes without FHE [10, 11] avoid this homomorphic cost, but operate on larger direct encodings and do not exploit compact aggregated representations. Our HBF-based PMT/PSI instead uses Bloom filters as the plaintext domain and reduces each membership test to one ciphertext–plaintext multiplication plus one addition at depth 1, with no rotations or bootstrapping.

8 Conclusion

This paper introduced a new homomorphic Bloom filters (HBF), embedding Bloom filters directly into the plaintext space of RLWE-based FHE. The resulting depth-1 primitive supports searchable encryption and private membership/intersection while hiding Bloom-filter patterns, query tokens, and match scores under semi-honest security. Experiments show that encrypted Bloom filters reduce homomorphic complexity, with efficiency governed mainly by Bloom-filter size and false-positive parameters.

References

1. I. Amorim and I. Costa. Leveraging searchable encryption through homomorphic encryption: A comprehensive analysis. *Mathematics*, 11(13):2948, 2023.

2. N. N. Basil, S. Ambe, C. Ekhator, and E. Fonkem. Health records database and inherent security concerns: A review of the literature. *Cureus*, 14(10):e30168, 2022.
3. A. Bassit, F. Hahn, R. Veldhuis, and A. Peter. Hybrid biometric template protection: Resolving the agony of choice between bloom filters and homomorphic encryption. *IET Biometrics*, 11(4), 2022.
4. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
5. A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.
6. Y. Cai, C. Tang, and Q. Xu. Two-party privacy-preserving set intersection with FHE. *Entropy*, 22(12):1339, 2020.
7. H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1243–1255, 2017.
8. E. Chielle and M. Maniatakos. Recurrent private set intersection for unbalanced databases with cuckoo hashing and leveled fhe. In *Network and Distributed System Security (NDSS) Symposium*, 2025.
9. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, pages 79–88, 2006.
10. S. K. Debnath, N. Kundu, and T. Choudhury. An efficient post-quantum private set intersection protocol. *International Journal of Information and Computer Security*, 20(3/4):278–299, 2022.
11. S. K. Debnath, P. Stănică, T. Choudhury, and N. Kundu. Post-quantum protocol for computing set intersection cardinality with linear complexity. *IET Information Security*, 14(6):729–739, 2020.
12. D. Derler, K. Gellert, T. Jager, D. Slamanig, and C. Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. *Journal of Cryptology*, 34, 2021.
13. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
14. Y. Feng, Z. Lu, and Q. Cao. Secure sharing of private locations through homomorphic bloom filters. In *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity)*, pages 107–113, 2018.
15. E.-J. Goh. Secure indexes. Technical Report 2003/216, IACR Cryptology ePrint Archive, 2003.
16. V. J. Gondil and H. A. Hingoliwala. ABE and bloom filter based searchable attribute-based encryption scheme over cloud data. *International Journal of Scientific Research in Science and Technology*, 8(3):208–215, 2021.
17. J. Hu and W. Dai. Efficient threshold private set intersection via BFV fully homomorphic encryptions. *IACR Cryptology ePrint Archive*, 2023:1407, 2023.
18. X. Jin et al. Research on dynamic searchable encryption method based on bloom filter. *Applied Sciences*, 14(8):3379, 2024.
19. B. Kishiyama and I. Alsmadi. A review on searchable encryption functionality and the evaluation of homomorphic encryption. *arXiv preprint arXiv:2312.14434*, 2023.
20. T.-T. Kuo, X. Jiang, H. Tang, X. Wang, A. Harmanci, M. Kim, K. Post, D. Bu, T. Bath, J. Kim, W. Liu, H. Chen, and L. Ohno-Machado. The evolving privacy and security concerns for genomic data analysis and sharing as observed from the

iDASH competition. *Journal of the American Medical Informatics Association*, 29(12):2182–2190, 2022.
21. Y. Liang, J. Ma, Y. Miao, D. Kuang, X. Liu, and R. H. Deng. Privacy-preserving bloom filter-based keyword search over large encrypted cloud data. *IEEE Transactions on Computers*, 72(11):3086–3098, 2023.
22. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.
23. Ş. L. Niţă and M. I. Mihăilescu. *Advances to Homomorphic and Searchable Encryption*. Springer, 2023.
24. M. A. Owaid and O. A. Dawood. A survey in privacy-preserving by bloom filters. *AIP Conference Proceedings*, 2979(1):070001, 2023.
25. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 44–55, 2000.
26. V.-D. Stanciu, M. van Steen, C. Dobre, and A. Peter. Privacy-preserving crowd-monitoring using bloom filters and homomorphic encryption. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '21)*, pages 37–42, 2021.
27. T. Suga, H. Nishide, and K. Sakurai. Secure keyword search using bloom filter with specified character positions. In *Provable Security (ProvSec 2012)*, volume 7496 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2012.
28. H. Wang, H. Dai, S. Chen, Z. Chen, and G. Chen. Filter data structures: A survey. *Computer Science*, 50(4):1–38, 2023.
29. C. Xu, R. Wang, L. Zhu, C. Zhang, R. Lu, and K. Sharif. Efficient strong privacy-preserving conjunctive keyword search over encrypted cloud data. *arXiv preprint arXiv:2203.13662*, 2022.
30. Q. Zhang, S. Fu, N. Jia, J. Tang, and M. Xu. Secure multi-keyword fuzzy search supporting logic query over encrypted cloud data. In *Security and Privacy in New Computing Environments (SPNCE 2019)*, volume 284 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 210–225. Springer, 2019.