VeriFuzzy: A Dynamic Verifiable Fuzzy Search Service Framework for Encrypted Cloud Data

Jie Zhang, *Student Member, IEEE*, Xiaohong Li, *Member, IEEE*, Man Zheng, Ruitao Feng, Shanshan Xu, Zhe Hou and Guangdong Bai, *Member, IEEE*

***Abstract*—Enabling search over encrypted cloud data is essential for privacy-preserving data outsourcing. While searchable encryption has evolved to support individual requirements like fuzzy matching (tolerance to typos and variants in query keywords), dynamic updates, and result verification, designing a service that supports Dynamic Verifiable Fuzzy Search (DVFS) over encrypted cloud data remains a fundamental challenge due to inherent conflicts between underlying technologies. Existing approaches struggle with simultaneously achieving efficiency, functionality, and security, often forcing impractical trade-offs.**

This paper presents VeriFuzzy, a novel DVFS service framework that cohesively integrates three innovations: an *Enhanced Virtual Binary Tree (EVBTree)* that decouples fuzzy semantics from index logic to support $O(\log n)$ search/updates; a *blockchain-reconstructed verification* mechanism that ensures result integrity with logarithmic complexity; and a *dual-repository state management* scheme that achieves IND-CKA2 security by neutralizing branch leakage. Extensive evaluation on 3,500+ documents shows VeriFuzzy achieves 41% faster search, $5\times$ more efficient verification, and constant-time index updates compared to state-of-the-art alternatives. Our code and dataset are now open source, hoping to inspire future DVFS research.

***Index Terms*—Searchable Encryption, Fuzzy Multi-keyword Search, Dynamic Updates, Result Verifiability, Cloud Security, Blockchain, IND-CKA2, Search-As-a-Service.**

I. Introduction

The global digital transformation is driving the widespread outsourcing of data to cloud platforms [1]. While cloud data services reduce storage costs and improve scalability [2], their inherent characteristics such as openness, centralization, and lack of regulation have raised serious concerns about cloud data security, especially in sensitive areas such as healthcare, finance, and government affairs [3], [4], [5]. Traditional encryption, though effective in preventing data breaches [6], significantly impairs the ability to perform efficient keyword-based search, which is essential for ensuring timely data access and maintaining quality of service.

While Searchable Encryption (SE) enables searching over encrypted data without revealing its contents [7], real-world applications demand three critical capabilities simultaneously: (i) *fuzzy multi-keyword search* to tolerate typos and variants [8], (ii) *dynamic updates* to support real-time data operations [9], and (iii) *result verifiability* to ensure integrity in untrusted environments [10].

Three existing mature technologies in SE development can address the above core requirements respectively: (i) *Fuzzy search* via Locality-Sensitive Hashing (LSH) [11] provides efficient approximate matching by mapping similar keywords to identical buckets with tunable precision; (ii) *Dynamic updates* demand tree-based structures like Virtual Binary Tree (VBTree) [12] to maintain logarithmic complexity for real-time operations; (iii) *Result verification* leverages blockchain's immutable ledger and Merkle trees [13] for trustless integrity validation. Each technique represents the state-of-the-art in its respective domain, making them natural candidates for integration.

However, despite significant progress in individual components of searchable encryption, the Dynamic Verifiable Fuzzy Search (DVFS) schemes that support fuzzy search, dynamic updates, and result verifiability remain challenging to implement, with the core issue stemming from *fundamental incompatibilities* between the above techniques.

Challenges of Prior Art in DVFS. Current solutions face the following core challenges:

1) **LSH leads to Semantic-Index Coupling.** While fuzzy techniques like LSH [11] or MinHash [14] can effectively map semantically similar keywords to identical buckets, fundamental incompatibility arises when directly applying LSH to dynamic indexing schemes requiring tree structures (e.g., VBTree [12]) for $O(\log n)$ operations, causing ***semantic-index coupling***, where fuzzy keywords are mapped to fixed positions in tree-based indices, causing any update to document-keyword relationships to trigger cascading index reorganizations with $O(n^2)$ complexity [15]. This architectural conflict forces existing schemes to choose between fuzzy search capability and practical update performance, either sacrifice scalability [12], [15], [16], [17], [18], [19] or revert to static indexing models [10], [11], [20], [21].
2) **Blockchain has Verification Overhead.** Although blockchain systems based on Merkle trees [22], [21], [23],

This work is supported in part by the National Key Research and Development Program of China under Grant 2023YFB3107103, in part by the National Natural Science Foundation of China under Grant 62262073, 62332005.

Jie Zhang and Xiaohong Li are with the College of Intelligence and Computing, Tianjin University, Tianjin, China. (e-mail: {jackzhang, xiaohongli}@tju.edu.cn).

Man Zheng is with the State Grid Hebei Electric Power Co., Ltd. Xiong'an New Area Power Supply Company, China (e-mail: meganzheng527@163.com).

Ruitao Feng is with the Faculty of Science and Engineering, Southern Cross University, Australia (e-mail: ruitao.feng@scu.edu.au).

Shanshan Xu is with the School of Geographic Sciences, East China Normal University, Shanghai, China. (e-mail: s.xu.ecnu@gmail.com).

Zhe Hou is with the School of Information and Communication Technology, Griffith University, Nathan, Australia. (e-mail: z.hou@griffith.edu.au).

Guangdong Bai is with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. (e-mail: baiguangdong@gmail.com).

Jie Zhang and Man Zheng contributed equally to this work.

Ruitao Feng and Guangdong Bai are the corresponding authors.

[24] theoretically provide trustless result verification with $O(\log n)$ complexity in a decentralized environment, such schemes [13], [18], [25], [26], [27] lead to ***verification overhead*** imbalance in dynamic environments with frequent index changes. Specifically, additional $O(n)$ complexity is required for tree structure reconstruction during updates, resulting in actual verification overhead growing linearly. This creates an impossible choice: either accept inefficient verification [24] or abandon completeness guarantees.

3) **VBTree Amplifies Branch Leakage.** The logarithmic search efficiency of tree structures [12] comes at the cost of transparent traversal paths, where each query reveals the specific nodes accessed, enabling adversaries to reconstruct query semantics through statistical analysis [28]. Although the proposed VBTree addresses this issue by implementing branch hiding for exact queries through its virtualized structure, it actually exacerbates the ***branch leakage*** under fuzzy search. This is because VBTree was designed with only individual branch hiding in mind, and cannot address the branch leakage introduced by fuzzy search semantics. Attackers can perform path inference through multiple sets of keyword variants to obtain core keywords, and further deduce keyword distribution, relationships, and even specific query identities [16], [21]. Consequently, most practical schemes either accept this leakage [12], [15], [18], [19] or sacrifice tree-based structures to achieve the current strongest security level (**IND-CKA2**) [21]. This creates a security-functionality trade-off where stronger privacy guarantees necessitate sacrificing either search efficiency or query expressiveness.

Our VeriFuzzy Framework: A Cohesive Solution. To systematically address these interconnected challenges, we propose VeriFuzzy, a novel dynamic verifiable fuzzy search service framework that transforms rather than merely combines existing components through three foundational innovations:

- **Resolving Semantic-Index Coupling:** We introduce an EVBTree that architecturally decouples fuzzy semantics from index logic. Unlike naive integrations that bind LSH groups to tree nodes, EVBTree processes each keyword w by: (i) converting to uni-gram vector $\vec{v}$, (ii) applying k LSH functions to generate bucket set S_w, (iii) for the bucket string S_w we compute index entries for all nodes on the path to the document's leaf. This approach maintains $O(\log n)$ search complexity while supporting fuzzy matching through multiple bucket evaluation.
- **Bridging the Dynamic-Verification Gap:** Our verification mechanism stores only cryptographic digests and deletion logs on-chain. During queries, smart contracts reconstruct search paths on-demand using Merkle proofs, eliminating $O(n)$ tree reconstruction overhead. The dual-repository design (Local + Blockchain repositories) enables forward privacy while maintaining $O(\log n)$ verification complexity and completeness guarantees through distributed trust.
- **Neutralizing Branch Leakage:** We implement a multi-layered security approach to achieve IND-CKA2 security under fuzzy search semantics. Our dual-repository design employs cryptographic chaining of keyword versions: when updating keyword w to version v, we store a cryptographic chain element linking current and previous versions in the blockchain, preventing version correlation attacks. Complemented by historical trapdoor checking, early branch termination, and post-search filter deletion in our search protocol (Section V-E), this multi-faceted defense ensures query patterns remain unlinkable across updates while maintaining fuzzy search performance.

VeriFuzzy represents an architectural breakthrough that harmonizes functionality, efficiency, and security in encrypted search systems. Our implementation on a corpus of 3,500+ documents demonstrates: (i) 41% faster search than state-of-the-art fuzzy Searchable SE (SSE) schemes, (ii) 5× verification efficiency compared to linear-verifiable models, and (iii) full support for dynamic updates with constant-time index refresh. A preliminary version of this work has been shared on arXiv [29].

Summary of Contributions:

- We design a dynamic encrypted search framework supporting efficient fuzzy multi-keyword queries and real-time updates via a novel virtual binary tree structure.
- We develop a blockchain-based verification mechanism that ensures completeness and correctness with $O(\log n)$ complexity.
- We formally prove the security of VeriFuzzy and demonstrate its practicality through comprehensive experimental evaluation.
- While our open-source release does not expose proprietary blockchain chaincode (due to commercial deployment restrictions), our code and dataset are publicly available at: https://github.com/JackAugust/VeriFuzzy.git, which are used for implementing dynamic fuzzy search and other functionalities in VeriFuzzy.

The remainder of this paper is organized as follows: Section II reviews related work; Section III presents preliminaries; Section IV describes our system model; Section V details the VeriFuzzy scheme; Section VI provides security analysis; Section VII presents experimental results; Section VIII provides discussions and future work prospects, and Section IX concludes.

II. Related Work

A. Fuzzy Searchable Encryption

The limitation of precise keyword matching in addressing spelling errors in query keywords, which leads to incorrect outputs, has created an urgent need for designing keyword search schemes that support approximate matching, thus introducing the concept of fuzzy searchable encryption. Li et al. [8] proposed the first fuzzy keyword search scheme that utilizes edit distance and wildcards to construct a predefined fuzzy set to cover spelling errors; however, it only supports single-keyword queries. Wang et al. [30] employed Bloom filters and LSH to facilitate fuzzy matching searches. Building on this, Fu et al. [11] improved [30] scheme by introducing the uni-gram model to improve the accuracy of fuzzy ranked search. Liu et al. [31] implemented an effective wildcard-based fuzzy multi-keyword search using edit distance that supports both AND and OR search semantics. Xie et al. [20] pioneered geohash-primes indexing for spatial fuzzy queries. Liu et al. [17] achieved $O(k)$ conjunctive search via binary tree indexing. However, all of the above schemes only support static data and ignore the dynamic scenario that does not meet the actual production requirements.

B. Dynamic SSE

The demand for real-time data processing has motivated Dynamic SSE (DSSE) research. Kamara et al. [9] designed a keyword red-black tree index, realizing a sublinear DSSE scheme that supports efficient updates. Xia et al. [32] constructed a special keyword balanced binary tree as the index that supports flexible dynamic operation. Li et al. [16] introduced an indistinguishable binary tree and an indistinguishable bloom filter data structure for conjunctive query. Wu et al. [12] further proposed the VBTree structure, achieving scalable index sizes and sublinear update times. Zhong et al. [15] utilized LSH and balanced binary trees (which assemble all Bloom index vectors) to achieve fault-tolerant multi-keyword fuzzy retrieval and support dynamic file updates, Xu et al. [19] enabled multi-user verification, yet integrating these two functionalities may impact search efficiency.

C. Verifiable SSE

A significant body of research has addressed data correctness and integrity in response to malicious cloud services. Liu et al. [33] employed RSA accumulators to generate proofs for search results, although this approach incurs substantial computational overhead. Wan et al. [34] utilized homomorphic MAC techniques to authenticate each index, thereby verifying the correctness of relevance score calculations. Tong et al. [23] proposed an authenticated tree-based index structure based on merkle hash trees. Zhang et al. [13] leveraged blockchain technology to achieve certificate-free public verification of data integrity without a central authority. Regarding correctness and integrity verification, Liu et al. [18] employed RSA accumulators to authenticate index structures; however, this method is not suitable for tree-based index structures. Shao et al. [10] implemented efficient verifiable fuzzy multi-keyword search using keyed hash message authentication codes and cardinality trees. Li et al. [22] and Cui et al. [24] introduced chain-indexed Merkle trees for verifiable fuzzy search. Tong et al. [21] advanced fuzzy verifiability.

D. Integrated Solutions and Research Gaps

As synthesized in Table I, our VeriFuzzy represents the first solution that simultaneously achieves all three objectives without compromising security or efficiency. Compared with Tong *et al.*'s VFSA[21], we achieve dynamic updates and reduce verification complexity from $O(n)$ to $O(\log n)$ without compromising security. Compared to Zhong *et al.*'s EDMF[15], our scheme adds verifiability and eliminates the $O(n^2)$ update bottleneck. Cui *et al.*'s DVFKF [24] represents the closest prior art. However, its $O(n)$ search complexity and lack of completeness verification leave critical gaps. Our work finally achieves the quadripartite goal of **fuzzy search**, **dynamic updates**, **dual verification**, and **sublinear complexity** simultaneously.

TABLE I
COMPARISON OF ENCRYPTED SEARCH SCHEMES. F_1: FUZZY MULTI-KEYWORD; F_2: DYNAMIC UPDATES; F_3: CORRECTNESS VERIFICATION; F_4: COMPLETENESS VERIFICATION; F_5: SEARCH COMPLEXITY; F_6: VERIFICATION COMPLEXITY; F_7: SECURITY MODEL.

Scheme	F_1	F_2	F_3	F_4	F_5	F_6	F_7
[11]	✓	×	×	×	$O(n)$	-	<KPA
[12]	×	✓	×	×	$O(\log n)$	-	IND-CKA2
[16]	×	✓	×	×	$O(n \log n)$	-	IND-CKA2
[15]	✓	✓	×	×	$O(\log n)$	-	KPA
[20]	✓	×	×	×	$O(\log n)$	-	IND-CKA
[17]	×	✓	×	×	$O(\log n)$	-	IND-CKA2
[18]	×	✓	✓	✓	$O(\log n)$	$O(n)$	UC-CKA
[19]	×	✓	✓	✓	$O(n)$	$O(n)$	IND-CKA
[21]	✓	×	✓	✓	$O(\log n)$	$O(n)$	IND-CKA2
[24]	✓	✓	✓	✓	$O(n)$	$O(n)$	IND-CKA
Ours	✓	✓	✓	✓	$O(\log n)$	$O(\log n)$	IND-CKA2

III. PRELIMINARIES

In this section, we review the relevant background knowledge, including locality sensitive hashing [30],virtual binary tree [12] and blockchain [35].

A. Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) LSH solves approximate or exact Near Neighbor Search in high dimensional space. It hashes inputs, likely mapping similar items to same buckets. Here, keywords are LSH-processed and then added to indexes or tokens for fuzzy search. A hash function family H is (R_1,R_2,P_1,P_2)-sensitive when for any p, q and $h \in H$:

- if $d(p,q) \leq R_1$, $Pr[h(p) = h(q)] \geq P_1$;
- if $d(p,q) \geq R_2$, $Pr[h(p) = h(q)] \leq P_2$;

Here, $d(p,q)$ is the distance between p and q, R_1, R_2 are the distance thresholds ($R_1 < R_2$), and P_1, P_2 are the prob thresholds ($P_1 > P_2$). In this work, for similar keyword identification, we use p-stable LSH [36] mapping d dimensional vectors to integers. The p-stable function in our scheme is $h_{(\mathbf{a},b)}(\vec{v}) = \lfloor \frac{\mathbf{a}\cdot\vec{v}+b}{c} \rfloor$. Here, $\mathbf{a}$ is a d dimensional vector with each dim randomly and independently chosen from the p-stable dist, $b \in [0, c]$ is random, and c is a fixed constant per family. Varying $\mathbf{a}$ and b generates different p stable LSH functions.

B. VBTree

Virtual Binary Tree (VBTree) VBTree, first introduced in [12], is a full binary tree that exists logically rather than physically. Its key design is top-down indexing element organization without storing branches or nodes. Each VBTree node has a path: for non-terminal nodes, a '0' is appended to the left child's path and a '1' to the right child's path. For node v, $path(v)$ is the binary string from root to v (root's $path(v)$ is empty). Let $leaf_i$ be the i^{th} leaf and $Nodes(i)$ be the nodes from the root to $leaf_i$. To index keyword w for doc

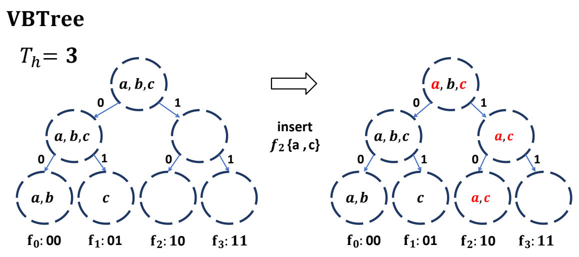


Fig. 1. An example of vbtree.

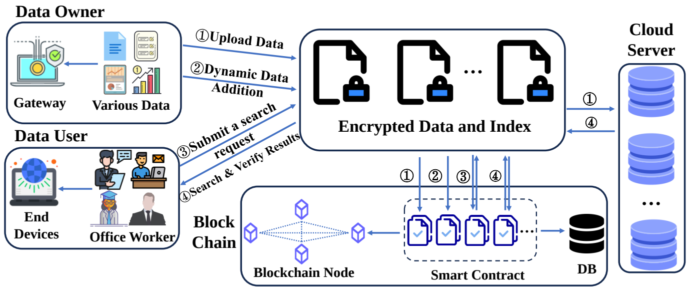


Fig. 2. System model.

i, insert $H(path(v) \| F_K(w))$ into each $Nodes(i)$ node, where H is a random oracle and F_K is a pseudo-random function.

Fig. 1 shows an example of a virtual binary tree of height $T_h = 3$. There are four leaves in the tree. $path(leaf_1)$ denotes string "01". $Nodes(1)$ denotes a set of tree nodes $\{root, node_0, node_{01}\}$. The values of keyword "c" of file identifier "1" in the hash table are a set of key-value pairs, i.e., denoted $W(\text{“c”}, 1) = \{H(path(v) \| F_K(c))\}_{v \in Nodes(1)} = \{H(\text{“”} \| F_K(w)), H(\text{“0”} \| F_K(w)), H(\text{“01”} \| F_K(w))\}$. All key-value pairs in this VBTree are $W(\text{“a”}, 0) \cup W(\text{“b”}, 0) \cup W(\text{“a”}, 1) \cup W(\text{“b”}, 1) \cup W(\text{“c”}, 1)$. To insert a file f_2 with two keywords $\{a, c\}$, the data owner converts f_2 to the path "10" and inserts the encrypted items from the root to the leaf along path "10". The inserted items are $W(\text{“a”}, 2) \cup W(\text{“c”}, 2)$.

For a VBTree storing n real files, with all leaves containing N keywords (where N is the total keyword appearances in all files and T_h is the tree height), its construction time is $O(NT_h)$ and index size is $O(\beta N) \approx O(N \log n)$ $(\beta \in [2, T_h])$. Also, for a u-dimensional keyword conjunctive query q, the top-down search algorithm has a query complexity of $O(\log_2 n)$.

C. Blockchain

Blockchain and Smart Contract: Blockchain, as a continuously evolving distributed ledger technology, constitutes the fundamental infrastructure of cryptocurrency systems [35]. Its security is underpinned by cryptographic hashing and consensus mechanisms. In a blockchain, each block acts as a permanent record of transactions, with millions of blocks interconnected to form a blockchain network. The consensus mechanism ensures the integrity and consistency of the entire blockchain network, thereby preventing data tampering [37]. Smart contracts are digital legal agreements executed as computer programs. Intended to foster trust without TTPs, their implementation was hindered by the absence of programmable digital systems. These are unmodifiable scripts on the blockchain. Once deployed in a block, they automatically execute according to predefined logic, independent of a central authority.

IV. System Model

In this section, we formalize the system components and security requirements for addressing the fundamental challenges identified in Section I. First, we define the system model involving data owners/users, cloud servers, and blockchain entities; second, we formalize the design goals encompassing fuzzy search capability, dynamic updates, verifiable results, and strong security guarantees; third, we provide precise algorithmic definitions for the six core operations of our scheme; finally, we present the security model including adaptive adversary capabilities and IND-CKA2 security notions. Collectively, these components provide the formal underpinnings for our VeriFuzzy framework, enabling the integrated solution described in subsequent sections.

A. The System Model

As shown in Fig.2, our system model consists of four types of entities: data owners, data users, cloud servers, and blockchain. The roles of the entities are as follows:

- **Data Owners (DOs)**: Departments within an enterprise act as data owners, uploading project documents, encrypting them using traditional symmetric encryption algorithms, constructing encrypted indexes, and then uploading them to cloud server and blockchain respectively.
- **Data Users (DUs)**: Employees in each department serve as data users, sending query requests or new document materials to blockchains and cloud server according to their needs.
- **Cloud Server (CS)**: CS is responsible for storing the uploaded encrypted data and returning the required search results, providing support for cross departmental data interaction.
- **Blockchain (BC)**: BC stores encrypted indexes and executes query and addition operations on searchable encrypted indexes with the help of smart contracts. During the query process, the search contract records auxiliary information, and the verification contract verifies the correctness and integrity of the results based on this information to ensure data security and consistency.

B. Design Goals

Our design goals encompass the following key aspects:

G1 **Robustness.** VeriFuzzy must support both fuzzy multi-keyword search and dynamic data management to enhance robustness in complex business scenarios. Similar to [15], [21], [24], it must achieve high accuracy in fuzzy search even when keywords contain common spelling errors (see Section VII-B), and remain effective after updates.

G2 **Verifiability.** VeriFuzzy supports the verification of the correctness and completeness of search results, ensuring that the returned data are accurate and reliable.

G3 **Efficiency.** VeriFuzzy can achieve sublinear time complexity for search and verification. Compared to traditional solutions, it effectively reduces search response times and enhances verification efficiency, optimizing system performance.

G4 **Security and Privacy.** VeriFuzzy must satisfy adaptive security and ensures forward privacy [38] for dynamic updates, ensuring that cloud servers cannot infer any sensitive information from encrypted data documents, indexes, user search tokens, or data updates.

C. Algorithm Definition

The following presents the algorithm definitions of the six algorithms that make up the research scheme of this paper:

$\textbf{Setup}(1^{\lambda}, k) \rightarrow Params$: Given a security parameter λ and an element k, the DO outputs the parameter $Params$.

$\textbf{IndexGen}(F, CT, Params) \rightarrow I$: Given a document set F, ciphertext set CT and a parameter $Params$, DO outputs an index tree I.

$\textbf{Update}(I, i, W) \rightarrow I'$: Given index construction I, a file identifier i, an additional set of keywords W, DO outputs the updated index I'.

$\textbf{TrapGen}(Q, Params) \rightarrow TK$: Given a search query Q and parameter $Params$, DU outputs a trapdoor TK for retrieval.

$\textbf{Search}(I, TK, Params) \rightarrow \{R, AP, D\}$: Given the index tree I, the trapdoor TK and the parameter $Params$, this function first prepares the historical trapdoor set H for all keywords in TK by querying the blockchain repository. Then, it invokes the internal recursive search procedure (Algorithm 1) starting from the root node (with $path = ""$) and passing H and the deletion logs DL (which are part of $Params$). The internal procedure returns the result set R, auxiliary proofs AP, and digest set D.

$\textbf{Verify}(R, AP, D, E, Params) \rightarrow 1/0$: Given the search results R, the auxiliary proof information AP, the digest information set D, the ciphertext set E and the parameter $Params$, BC outputs 1 if E are correct and complete; otherwise, it returns 0.

D. Security Definition

The Indistinguishability under Adaptive Chosen-Keyword Attack (IND-CKA2) model, first proposed by [39], combines indistinguishability and adaptive keyword attacks to ensure stronger confidentiality. IND-CKA2 comprehensively evaluates the security of searchable encryption schemes based on leakage functions, ensures that even if an adversary queries a random oracle and selects keywords, other index keywords stay hidden. This model provides a more comprehensive analysis of attackers' behavior and strategies and offers higher security than other security models.

We consider the strongest threat model in searchable encryption: an **adaptive adversary** $\mathcal{A}$ that operates as follows:

Definition 1 (Adaptive Adversary)**.** *An adaptive adversary $\mathcal{A}$ is a Probabilistic Polynomial-Time (PPT) algorithm that performs the following:*

1) ***Adaptively chooses queries**: Based on all previously observed information (indexes, trapdoors, search results), $\mathcal{A}$ can strategically select subsequent search queries Q_i and update operations to maximize information leakage.*
2) ***Dynamically interacts with the system**: $\mathcal{A}$ engages in multiple rounds of the security game, where each query Q_i may depend on the outcomes of all previous queries $\{Q_1, \ldots, Q_{i-1}\}$ and their corresponding trapdoors $\{TK_1, \ldots, TK_{i-1}\}$.*
3) ***Observes all cryptographic outputs**: $\mathcal{A}$ can observe the encrypted index I, all generated trapdoors TK_i, search results, and updated indexes I_{up} throughout the entire system lifetime.*
4) ***Attempts to distinguish real from simulated worlds**: The adversary's goal is to distinguish between interactions with the real scheme and a simulator that receives only the predefined leakage functions $L = (L_1, L_2, L_3)$.*

Our leakage function $L = (L_1, L_2, L_3)$ follows the standard formulation in most SE schemes [16], [17], [33], [40]:

- $L_1 = L_1(F) = (n, sizes, ids, T_h, M)$. Here,$L_1$ leakage encompasses M (EVBTree hashtable entry count), T_h (EVBTree height), n (file count), $sizes$ (set of file sizes encrypted by a CPA-secure scheme), and $ids = \{id_1, \cdots, id_n\}$ (set of file identifiers from 1 to n).
- $L_2(F, Q) = \{(w, SP(w), HIST(w)) : w \in Q\}$, which includes the search pattern $SP(w)$ (client's trapdoor repetition to BC) and the history $HIST(w)$[1] of queried keywords, representing returned encrypted document identities. For a keyword w, $SP(w)$ has all search trapdoors for w, and $HIST(w)$ is the collection of w containing document identifiers, which represents the *observable outcome* of search operations rather than internal system state.
- $L_3 = L_3(W, id)$: This leakage represents the information revealed during encrypted index updates. Here, W and id denote adding/deleting keyword set W to the document with identifier id.

Definition 2 (IND-CKA2(L_1, L_2, L_3)-secure)**.** *A searchable encryption scheme is IND-CKA2(L_1, L_2, L_3)-secure if for any PPT adaptive adversary $\mathcal{A}$, there exists a simulator S such that*

$$|\Pr[\mathit{Real}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathit{Sim}_{\mathcal{A},S}(\lambda) = 1]| \leq negl(\lambda)$$

where $negl(\lambda)$ denotes a negligible function in the security parameter λ. The probabilistic experiments are defined as:

- *$\mathit{Real}_{\mathcal{A}}(\lambda)$: The adversary $\mathcal{A}$ interacts with the real scheme, receiving actual cryptographic outputs.*
- *$\mathit{Sim}_{\mathcal{A},S}(\lambda)$: The simulator S, given only the leakage functions L_1, L_2, L_3, generates simulated outputs that are computationally indistinguishable from the real ones.*

This Definition 2 formalizes the adaptive security of our searchable encryption scheme under the IND-CKA2 model. The security is parameterized by three leakage functions L_1,

[1] In fact, there is a fundamental difference between the historical results $HIST(w)$ returned by this paper for describing attackers' keyword w queries and the $HIST(w)$ used in SSE to maintain w's lifecycle status.

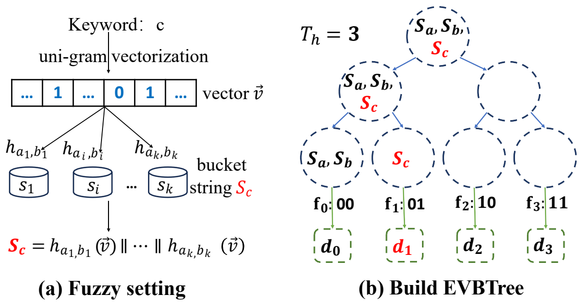


Fig. 3. An instance of the EVBTree with a height of $T_h = 3$ and four leaves.

L_2, and L_3 that quantify the information intentionally revealed to the adversary. The core idea is that for any PPT adversary A, the views in the real world (interacting with the actual scheme) and the simulated world (interacting with a simulator that only uses the allowed leakage) are computationally indistinguishable.

V. Our VeriFuzzy scheme

This section details the design of our scheme, closely adhering to the specified objectives. For robustness (**G1**), VeriFuzzy achieves this through fuzzifying index keywords and enhanced virtual binary tree construction, enabling spelling error tolerance and dynamic data updates; for verifiability (**G2**), ensured by ciphertext digest recalculation and search tree reconstruction; for efficiency (**G3**), Implemented via trapdoor computation and top-down search algorithms for optimized query processing; for security and privacy (**G4**), Security Analysis VI will be fully explained.

A. Setup

Given the security parameters λ and k, the DO outputs the parameter set $Params = \{F_K, H_1, LSH_k\}$, where F_K is a keyed-hash function: $\{0,1\}^* \times \{0,1\}^\lambda \to \{0,1\}^*$, H_1 is a hash function: $\{0,1\}^* \to \{0,1\}^*$, and LSH_k is a locality-sensitive hash family of k functions $\{h_{a_t,b_t}\}_{t=1}^k$. Specifically, F_K can be generated by using the keyed-hash message authentication code $HMAC$, i.e., $F_K = HMAC(K, \cdot)$ and K is a randomly generated secret key of length λ. H_1 can be generated using the random oracle H.

B. IndexGen

To achieve the fault tolerance requirement for user queries in **G1**, given an outsourced file set $F = \{f_1, f_2, \cdots, f_n\}$, the DO extracts keywords from each record. Applying a stemming algorithm, the DO derives keyword dictionaries $W(f_i)$ by reducing related words (e.g., “encrypted”, “encrypting”, and “encrypts” to “encrypt”). Then, as shown in Fig. 3, the DO generates an encrypted logical index tree, EVBTree.

- *Fuzzy Setting*. We employ the vectorization algorithm based on the uni-gram and LSH technique to process keywords, enhancing the fault tolerance of user queries. Specifically, for each keyword $w_j \in W(f_i)$, DO first converts w_j into a 160-dimensional uni-gram vector $\vec{v_j}$, which serves as the input for LSH. Subsequently, LSH family $LSH_k = \{h_{a_t,b_t}\}_{t=1}^k$ are utilized to compute the **bucket string** $S_{w_j} = \{h_{a_1,b_1}(\vec{v_j}) \| \cdots \| h_{a_k,b_k}(\vec{v_j})\}$, which represents the concatenation of outputs from the k LSH functions. This bucket string S_{w_j} captures the fuzzy semantics of keyword w_j, enabling tolerance to spelling variations.
- *Index EVBTree Built*. Given a file f_i with a keyword set $W(f_i) = \{w_1, w_2, \cdots\}$, the application of a fuzzy setting to this keyword set results in a bucket string value set $\{S_{w_1}, S_{w_2}, \cdots\}$. Additionally, according to the path encoding rules of the VBTree structure, the path value $path(f_i)$ of the file f_i is a string composed of 0 and 1, numerically representing the binary form of i. For each bucket string S_{w_j}, we calculate insert entries $I_{w_j,i} = \{(H_1(path(v) \| F_K(S_{w_j})), 1)\}_{v \in nodes(leaf_i)}$. Here, node v is a traversal node from the root to the leaf node $leaf_i$ of the document f_i according to the design of vbtree structure. The set $nodes(leaf_i)$ comprises the traversal nodes, and $path(v)$ denotes the string path code of node v, 1 indicates the presence of this keyword-document association. Subsequently, the digest of a file f_i is $d_i = F_K(path(leaf_i) \| c_i)$ is inserted into the index EVBTree I, where $path(leaf_i)$ denotes the path encoding information of leaf node $leaf_i$ corresponding to document identifier i, and c_i is the encrypted document f_i. Finally, all the insert entries and digests are stored in the EVBTree I, which logically adopts a binary tree structure but is actually implemented as a hashtable.

The **time complexity of the IndexGen Algorithm** is analyzed as follows:

Let n be the number of documents, $|W_i|$ be the average number of keywords per document, and k be the number of LSH functions. For each document f_i, keyword extraction is $O(|W_i|)$, then for each keyword, LSH processing requires $O(k)$ operations, and for each LSH bucket, VBTree path insertion requires $O(T_h)$ operations, where T_h is the tree height ($T_h = \lceil \log_2 n \rceil$). Thus, the total complexity is:

$$T(n) = O\left(n \cdot |W_i| \cdot k \cdot T_h\right) = O\left(n \cdot |W_i| \cdot k \cdot \log n\right)$$

Since $|W_i|$ and k are constants independent of n, the complexity simplifies to $T(n) = O(n \log n)$, confirming our claim of logarithmic scaling per document insertion during initial index construction.

C. Update

To achieve the goals of supporting efficient file addition (**G1**) while ensuring forward privacy (**G4**), we design a novel dynamic update mechanism based on dual repositories (Local Repository (LR) and Blockchain Repository (BR)) and the EVBTree structure. The update process operates in three coordinated phases.

- *Keyword State Update Mechanism:* The keyword state update mechanism ensures forward privacy through a dual-repository approach. For each keyword w, $LR[w]$ maintains $(LR[w].b, LR[w].v, LR[w].n_{\text{add}}, LR[w].n_{\text{del}})$ for its update status, current version, latest document ID and deletion.

- *First Update in Session:* When a keyword w is first updated to document m in a session ($LR[w].b = \textsf{false}$), the system computes the delta $T_{\text{add}} = I_{w,m} \setminus I_{w,LR[w].n_{\text{add}}}$, where $I_{w,m} = \{(H_1(path(v) \| F_K(S_w)), 1)\}_{v \in nodes(leaf_m)}$ represents the index entries for keyword w in document m, and $I_{w,LR[w].n_{\text{add}}}$ represents the previous index entries. Privacy-preserving padding is applied to conceal the actual delta size..
- *Subsequent Updates:* For subsequent updates ($LR[w].b = \textsf{true}$), a cryptographic chaining element $T_{\text{chain}} = (H_2(F_K(w\|v)), H_3(F_K(w\|v)) \oplus F_K(w\|(v-1)))$ is stored in the BR, and index I is updated with T_{add} as in the first case, preventing correlation between keyword versions.
- *Document Deletion:* Document deletion employs a mark-and-filter approach. The Deletion Logs (DL) is a dictionary mapping document IDs to deletion timestamps. When document m is deleted, $DL[m] \leftarrow$ True and relevant keyword states are updated. During search, IsDeleted(id, DL) filters results by checking $id \in DL$, ensuring backward privacy with $O(1)$ overhead.

Our update mechanism achieves the following properties:

- **Security**: The dual-repository designenables efficient state management while maintaining strong security guarantees. The LR keeps small-sized state information, while the BR provides tamper-proof storage for update operations.
- **Forward Privacy (G4)**: We ensure that new additions cannot be linked to previous search operations. The cryptographic chaining mechanism in BR prevents correlation between different versions of the same keyword.
  Efficiency (G1): By leveraging the EVBTree structure, updates require only $O(T_h \cdot |W_i|)$ operations, where T_h is the tree height, which is significantly better than the $O(n)$ complexity of linear schemes.
- **Verifiability**: All update operations are recorded on-chain through the BR, enabling transparent audit trails and verification of update correctness.
- **Backward Privacy**: The deletion mechanism using the DL ensures that removed documents do not appear in search results while preventing the server from learning which specific documents were deleted.

The **time complexity of the Update Algorithm** is analyzed as follows:

For each new document f_i, $O(|W_i|)$ for keyword extraction and vectorization, and $O(|W_i| \cdot k)$ for generating LSH buckets, then each keyword requires insertion along the VBTree path of length $L = O(\log n)$. The overall complexity for adding one document is:

$$T_{\text{add}}(n) = O(|W_i| \cdot k \cdot \log n) = O(\log n)$$

This logarithmic complexity demonstrates that our semantic-decoupled indexing successfully avoids the $O(n^2)$ update overhead that plagued previous integrated approaches like Zhong et al. [15].

D. TrapGen

To search the encrypted index, DU formulates a search trapdoor with encrypted conditions from user provided keywords and sends it to BC. For a query $Q = \{w'_1, w'_2, \cdots, w'_q\}$ of q keywords, each $w'_i \in Q$ is first vectorized to $\vec{v'_i}$ via a uni-gram algorithm. Then, set $LR(w_i).b$ in the local repository to $true$ to mark that the keyword has been queried, for each $\vec{v'_i}$, an LSH function calculates a bucket string $S_{w'_i}$ as per Eq. (1).

$$S_{w'_i} = LSH_k(\vec{v'_i}) = h_{a_1,b_1}(\vec{v'_i}) \| \cdots \| h_{a_k,b_k}(\vec{v'_i}) \tag{1}$$

Next, each bucket string $S_{w'_i}$ undergoes encryption using a pseudo random function F_K to generate a token tk_i,combined with the current version number $LR(w_i).v$ of the keyword w_i, as dictated by the Eq. (2):

$$tk_i = F_K(LSH_k(\vec{v_i}) || LR(w_i).v) \tag{2}$$

Upon completion of the processing for all query keywords, the search trapdoor $TK = \{tk_1, \cdots, tk_q\}$ is constructed, where q denotes the total number of query keywords in Q.

E. Search

Algorithm 1 presents the core search procedure of VeriFuzzy, which achieves **G3** by enabling efficient fuzzy multi-keyword search with $O(\log n)$ complexity through the EVBTree structure. **Note:** This algorithm is the recursive internal procedure of the Search function. The external Search interface first prepares the historical trapdoor set H for all query keywords and then calls this procedure starting from the root node (path = “”). The VeriFuzzy search protocol executes in three sequential phases as follows:

- *Phase 1- Historical Trapdoor Preparation*: For each keyword w_i in query $Q = \{w_1, \ldots, w_q\}$, the system generates the current trapdoor $tk_i = F_K(\text{LSH}_k(\vec{v_i}) \parallel \text{LR}(w_i).v)$. The BR is queried to retrieve all historical trapdoors $h(w_i) = \{x_{i1}, \ldots, x_{ip_i}\}$ where p_i represents the version count. The complete historical trapdoor set $H = \{h(w_1), \ldots, h(w_q)\}$ is constructed for the search operation.
- *Phase 2- Recursive Index Traversal*: The search begins at the root node (empty path) and recursively traverses the EVBTree structure. For each node, the algorithm verifies if *all* query keywords have at least one historical version present by checking $H_1(path\|x_{ij})$ for each $x_{ij} \in h(w_i)$. Branches are pruned early when any keyword lacks matching entries, significantly improving search efficiency. Matching nodes are recorded in auxiliary proof set AP with their path and match status (0/1). Leaf nodes reached through successful paths are added to the result set R with their corresponding digests d_i stored for verification.
- *Phase 3- Result Filtering and Verification*: The preliminary result set R is filtered against the DL to remove any documents that have been marked for deletion. The final result set R_{final} is returned along with auxiliary proofs AP and digests D for blockchain-based verification. The entire process maintains $O(\log n)$ complexity due to the tree structure and early pruning mechanism.

Algorithm 1 EVBTree Search Algorithm

```
Input: Encrypted EVBTree index $I$, full historical trapdoor
    set $H = \{h(w_1), h(w_2), \ldots, h(w_q)\}$, current node path
    $path$ (initially empty string for root) and deletion logs
    $DL$
Output: Result set $R$, Auxiliary proofs $AP$, Digest set $D$
 1: // Phase 1: Check all keyword versions at current node
 2: for each keyword $w_i \in Q$ do
 3:     $keyword\_matched \leftarrow$ false
 4:     for each historical version $x_{ij} \in h(w_i)$ do
 5:         $key \leftarrow H_1(path \| x_{ij})$
 6:         if $I$.contains($key$) then
 7:             $keyword\_matched \leftarrow$ true
 8:             break            ▷ One matching version is sufficient
 9:         end if
10:     end for
11:     if not $keyword\_matched$ then
12:         // Current node doesn't contain all keywords - prune
            this branch
13:         $AP \leftarrow AP \cup \{(path, 0)\}$      ▷ Record mismatch for
            verification
14:         return $\emptyset$                ▷ Prune this search branch
15:     end if
16: end for
17: // Phase 2: All keywords matched at this node - continue
    processing
18: $AP \leftarrow AP \cup \{(path, 1)\}$ ▷ Record match for verification
19: if isLeaf($path$) then
20:     // Reached a leaf node - add to potential results
21:     $id \leftarrow$ BinaryToDecimal($path$)
22:     $R \leftarrow R \cup \{id\}$
23:     $D \leftarrow D \cup \{d_{id}\}$      ▷ Store digest for later verification
24: else
25:     // Continue searching in both subtrees
26:     $R_{\text{left}} \leftarrow$ Search($I, H, path \| '0', DL$)
27:     $R_{\text{right}} \leftarrow$ Search($I, H, path \| '1', DL$)
28:     $R \leftarrow R \cup R_{\text{left}} \cup R_{\text{right}}$
29: end if
30: // Phase 3: Post-processing - apply deletion filtering
31: $R_{\text{final}} \leftarrow R$
32: for each document $id \in R$ do
33:     if IsDeleted($id, DL$) then
34:         $R_{\text{final}} \leftarrow R_{\text{final}} \setminus \{id\}$        ▷ DL is a set of deleted
            document identifiers
35:     end if
36: end for
37: return $R_{\text{final}}$
```

The **time complexity of the Search Algorithm 1** is analyzed considering the worst-case scenario:

Let q be the number of query keywords, p be the maximum historical versions per keyword, and T_h be the VBTree height. $O(q \cdot p)$ for retrieving all versioned trapdoors, and in worst case, we traverse $O(2^{T_h}) = O(n)$ nodes, but our early pruning strategy ensures that each node requires $O(q \cdot p)$ comparisons, and with effective pruning, only $O(T_h) = O(\log n)$ nodes are visited in practice. In deletion filtering phase needs $O(|R|)$

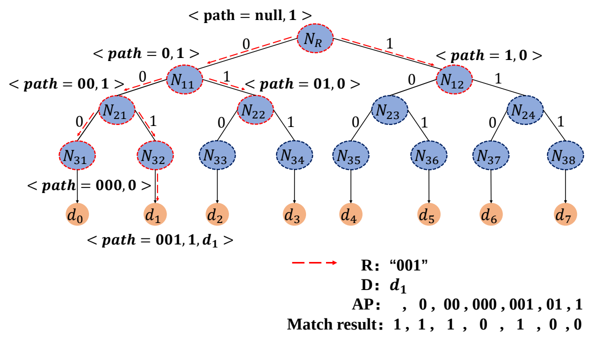


Fig. 4. Example of the auxiliary proof information for the search results c_1.

where $|R|$ is the result set size.

The dominant term is the tree traversal:

$$T_{\text{search}}(n) = O(q \cdot p \cdot \log n) = O(\log n)$$

In summary, we have made three optimizations for achieving efficient retrieval. First, we ensure that documents remain searchable after multiple updates while maintaining forward security by checking all historical versions of each keyword. Second, we significantly reduce the number of index accesses by immediately terminating the recursive search when any keyword mismatch occurs. Finally, instead of continuously maintaining deleted indexes, we perform deletion filtering after searching to improve update efficiency while ensuring accuracy. Our time complexity analysis confirms the $O(\log n)$ search complexity claimed in our introduction, representing a significant improvement over linear-scan approaches.

F. Verify

Upon receiving the result set R, the auxiliary proof set AP, the ciphertext digests set D, and the ciphertext set E, the smart contract verifies the correctness and completeness of the search results. The design not only accomplishes **G2** but also achieves the super-linear verification efficiency in **G3**.

Specially, first, verification of result correctness. Given a result set R, comprising r search results denoted as $R = \{path(leaf_0), \cdots, path(leaf_r)\}$, and the corresponding encrypted files $E = \{c_0, \cdots, c_r\}$ retrieved from the cloud server, the contract recomputes the digests set

$$\begin{aligned} D' &= \{d'_0, d'_1, \cdots, d'_r\} \\ &= \{F_K(path(leaf_0) \| c_0), \cdots, F_K(path(leaf_r) \| c_r)\} \end{aligned} \tag{3}$$

for each leaf node suffix and compares them against the search-derived set $D = \{d_0, d_1, \cdots, d_r\}$, verifying whether the equation $D' = D$ holds. The comparison outcome serves as evidence for the tampering status of the stored records. Secondly, verification of result integrity. Upon correct verification, the verify contract uses AP to build a search binary tree via node path encoding and matching. It checks if non-leaf terminal node search results are 0; if yes, it negates child node matches, skipping downward search and passing verification. Otherwise, non-zero results imply incomplete ciphertext retrieval, missing potential matches. If all verifications succeed, the smart contract returns 1 to the user, and the cloud server provides the ciphertext data. Finally, for completeness verification, if and only if any non-leaf node v with $match_v = 0$, all descendants must have $match = 0$. If a cloud server maliciously omits a document included in the result R, it must forge AP to pass the path verification, but this is cryptographically infeasible (since AP is associated with Merkle proof).

The **time complexity of our blockchain-based verification** through path reconstruction:

The smart contract reconstructs only the search path of length $T_h = \log n$, and by Merkle tree in blockchian, each node on the path requires $O(1)$ hash operations, then $O(|R|)$ for result set verification, where $|R|$ is typically small. So the overall verification complexity is:

$$T_{\text{verify}}(n) = O(\log n + |R|) = O(\log n)$$

For better understanding, let's explain it through Fig. 4. Given query $Q = \{w_1, w_2\}$ for example, assume only f_1 among $\{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$ has both keywords. Then, $path(N_{32})$ ="001" is in search result R, and ciphertext digest d_2 is in D. The auxiliary proof set $AP = \{$"","0","1","00","01","000","001"$\}$ contains traversed node path encodings, with matching results $\{1, 1, 0, 1, 0, 0, 1\}$ (1: match, 0: no match).

$$d_1' = F_K(path(N_{32}) \| c_1) = F_K('001' \| c_1) \qquad (4)$$

In verification, recalculate d_1' per Eq. (4) and compare with d_1. Since $d_1' = d_i$, the ciphertext result is correct. Then, construct a search binary tree based on search paths and node matching results (Fig. 4). As non-leaf terminal node search results are 0, the search-path integrity is validated, with no unqueried nodes missing.

VI. Security Analysis

A. IND-CKA2(L_1, L_2, L_3)-Secure Proof

Based on the security definition, we prove that our scheme is IND-CKA2(L_1, L_2, L_3)-secure in Theorem 3, achieving objective **G4**.

Theorem 3. *If F_K is a pseudo-random function, H_1 is a random oracle, the encryption scheme is CPA-secure, then our scheme is IND-CKA2(L_1, L_2, L_3)-secure against an adaptive adversary.*

Proof. We prove security through a sequence of games, where each game introduces a modification that is computationally indistinguishable from the previous one.

Game G_0 (Real World): This is the real security experiment $Real_{\mathcal{A}}(\lambda)$. The challenger:

- Runs $\mathbf{Setup}(1^\lambda, k)$ to generate $Params = \{F_K, H_1, LSH_k\}$
- Given document set F and keyword set W, computes $I \leftarrow \mathbf{IndexGen}(F, CT, Params)$ and encrypted documents CT
- For each adaptive query Q_i from $\mathcal{A}$, generates $TK_i \leftarrow \mathbf{TrapGen}(Q_i, Params)$ and returns search results
- For each update operation, generates $I_{up} \leftarrow \mathbf{Update}(I, i, W)$

The adversary receives all real cryptographic outputs: $\theta = (I, I_{up}, \{TK_1, \ldots, TK_s\}, \{R_1, \ldots, R_s\}, CT)$.

Game G_1 (Replace PRF with Random Function): We replace all instances of F_K with a truly random function R of the same domain and range. Specifically:

- In $\mathbf{IndexGen}$, index entries become $H_1(path(v) \| R(LSH_k(\vec{v})))$ instead of $H_1(path(v) \| F_K(LSH_k(\vec{v})))$
- In $\mathbf{TrapGen}$, trapdoors become $R(LSH_k(\vec{v_i}) \| LR(w_i).v)$ instead of $F_K(LSH_k(\vec{v_i}) \| LR(w_i).v)$
- All other operations remain identical to G_0

Any adversary distinguishing G_0 from G_1 can be used to break the PRF security:

$$|\Pr[G_0] - \Pr[G_1]| \le \text{Adv}_{F_K}^{\text{PRF}}(\mathcal{B}_1)$$

Game G_2 (Replace Real Ciphertexts): We replace all real document ciphertexts with encryptions of zero strings of the same length. The challenger:

- Instead of encrypting actual documents f_i, encrypts zero strings $0^{|f_i|}$ to generate CT'
- All index operations, trapdoor generation, and search algorithms use the same procedures as G_1
- Search results return the correct document identifiers but with zero-encrypted content

By the CPA-security of the encryption scheme:

$$|\Pr[G_1] - \Pr[G_2]| \le \text{Adv}_{\text{Enc}}^{\text{CPA}}(\mathcal{B}_2)$$

Game G_3 (Simulated World): We construct a simulator S that uses only the leakage functions L_1, L_2, L_3:

Setup Simulation ($S(\lambda, L_1(F))$): Given $L_1(F) = (n, \text{sizes}, \text{ids}, T_h, M)$, S:

- Generates $CT' = \{c_1', \ldots, c_n'\}$ by encrypting zero strings with lengths matching sizes
- Creates simulated index I' as a hash table with M random entries
- Sends (CT', I', ids) to $\mathcal{A}$

Search Simulation ($S(L_2(F, Q_1, \ldots, Q_i))$): For adaptive query $Q_i = \{w_{i1}, \ldots, w_{iu}\}$:

- For previously seen keywords $\{F_K(w_{i1}), \ldots, F_K(w_{ix})\}$, reuses random components $\{z_{i1}, \ldots, z_{ix}\}$
- For new keywords, generates fresh random strings $\{z_{ix+1}, \ldots, z_{iu}\}$
- Programs H_1 such that $F(z_{i1}) \cap \cdots \cap F(z_{iu}) = R_{Q_i}$, where R_{Q_i} is revealed by L_2
- Returns $TK_i' = \{z_{i1}, \ldots, z_{iu}\}$ and results $(R_{Q_i}', CT'[R_{Q_i}'])$

Update Simulation ($S(L_3(W, id))$): For update adding $W_j = \{w_1, \ldots, w_m\}$ to document id:

- Generates m random values $\{r_1, \ldots, r_m\}$ matching $H_1(path(v) \| F_K(w))$ size
- Updates I' with these random values
- Sends I_{up}' to $\mathcal{A}$

In G_2, all components are random or pseudorandom:

- Index entries: $H_1(path(v) \| \text{random})$ are uniformly random due to H_1 being a random oracle

- Trapdoors: Random strings programmed to produce correct search results through oracle programming
- Ciphertexts: Encryptions of zeros
- Search results: Correct document identifiers with zero-encrypted content

The simulator S in G_3 generates identically distributed values using the same random distributions and programming techniques. Therefore: $\Pr[G_2] = \Pr[G_3]$. Combining all transitions:

$$|\Pr[G_0] - \Pr[G_3]| \leq \text{Adv}_{F_K}^{\text{PRF}}(\mathcal{B}_1) + \text{Adv}_{\text{Enc}}^{\text{CPA}}(\mathcal{B}_2) = \text{negl}(\lambda)$$

This proves that VeriFuzzy is IND-CKA2(L_1, L_2, L_3)-secure. □

B. Branch Leakage Analysis under Fuzzy Search

VeriFuzzy neutralizes fuzzy search branch leakage through three mechanisms. In **Semantic-Index Decoupling**, EVBTree cryptographically separates LSH semantics from index logic. Each index entry $H_1(path(v) \| F_K(LSH_k(\vec{v})))$ ensures LSH bucket relationships remain hidden. In **Early Branch Termination**, conjunctive evaluation $\bigwedge_{w_i \in Q}(\bigvee_{x_{ij} \in h(w_i)} I.\text{contains}(H_1(path \| x_{ij})))$ creates binary outcomes, eliminating partial match granularity. In **Historical Obfuscation**, checking all historical trapdoor versions $H = \{h(w_1), \ldots, h(w_q)\}$ prevents version correlation and statistical inference. Therefore the adversary's information entropy is bounded by $O(\log n + \log |R|)$, preventing effective inference attacks even under fuzzy semantics.

VII. Performance Evaluation

In this section, we conduct a systematic evaluation of the core functionalities of the proposed scheme through comprehensive experiments. The experiments are primarily organized into three aspects: For Robustness, we test fuzzy search effectiveness in handling spelling errors and evaluate the time complexity of dynamic document addition. For efficiency, we measure the time overhead of the search algorithm and compare its performance with existing solutions. For verifiability, we analyze the time complexity of the verification functionality and compare it with other schemes. Based on the experimental results, we address the following key research questions:

RQ1: How effective is the proposed scheme in handling spelling errors under the fuzzy setting, and what is the time complexity of the dynamic document addition functionality?
RQ2: How does the time overhead of the search functionality in our scheme compare to existing solutions, and how much is the search efficiency improved?
RQ3: What is the computational overhead of our verification algorithm compared to other scheme?

A. Experiment Setup and Baselines

This section systematically evaluates the core functions of VeriFuzzy in terms of **robustness**, **efficiency**, and **verifiability**. Our implementation of the cryptographic algorithms and the searchable encryption scheme is publicly available at: https://github.com/JackAugust/VeriFuzzy.git. Note that the blockchain smart contract components are excluded due to commercial deployment constraints.

DataSets. We select two datasets to evaluate our scheme. One dataset is the RFC (request for comments) dataset with terminology-dense characteristics, from which 56,658 keywords were extracted[2] from 3,500 files, with the number of keywords per file about 92. The other dataset is the NSF Research Award abstracts spanning 1990 to 2003[3] with term sparsity characteristics, from which 3,167 keywords were extracted from 500 documents, with an average of approximately 45 keywords per file.

Implementation details. We conduct our experiments mainly on a PC running Windows 10 with a 1.60GHz Intel(R) Core(TM) i5-8250U CPU and 8GB memory, using the Java Pairing-Based Cryptography library. We set the number of HMAC-SHA256 functions to $k = 8$ as the pseudorandom function and use SHA-512 as the random oracle. We employ a Bloom filter of length $N = 1000$, resulting in an average false positive rate of less than $(1 - e^{-|W| \times k/N})^k = (1 - e^{-92 \times 8/1000})^8 \approx 0.5\%$. Using the LSH parameters $(\sqrt{3},\ 2,\ 0.56,\ 0.28)$. The height of the VBTree $l = 32$. Our blockchain environment is under a cloud server running Ubuntu 20.04 64-bit operating system with Hyperledger Fabric V2.2, developing the contract described in this paper based on Go 1.18.8.

Baselines. As shown in Table I, to comprehensively evaluate the performance of the fuzzy search, dynamic update, and result verification algorithms in our VeriFuzzy, we compare its performance with several existing solutions. For fuzzy search, we compared VeriFuzzy with DVFKF[24], VFSA[21], PCQS[16], and PCKS-FS[17] (as shown in Fig. 5 (a, d, e)). For dynamic update, we compared VeriFuzzy with DVFKF[24], EDMF[15] for dynamic updates (as shown in Fig. 5 (b,c)), and with DVFKF[24], VFSA[21], MVSSE[18], and MVDSE[19] for verification (as shown in Fig. 5 (f)).

B. Evaluation of Fuzzy Setting and Addition *(RQ1)*

1) Fuzzy Setting: We evaluate the effectiveness of the fuzzy setting in handling spelling errors through search accuracy. We categorize spelling errors in keywords into four common types: misspelling a letter, missing a letter, adding a letter, and swapping two letters. By varying the number of LSH functions, we evaluate the uni-gram search accuracy under these four types of spelling errors. We define this precision using the ratio $P = M'/M$, where M' represents the number of keywords that fall into the same hash bucket under both erroneous and correct conditions, and M denotes the total number of keywords. The results are presented in Table II, which illustrates the complex interrelationships among the number of LSH functions, error type, and search accuracy.

2) Accuracy assessment of fuzzy search: We evaluate the effectiveness of the fuzzy search accuracy assessment scheme. We define accuracy using the ratio $P = M'/M$, where M' represents the number of keywords that fall into the same hash

[2]By using RAKE algorithm. https://github.com/aneesha/RAKE
[3]The dataset can be found at http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html

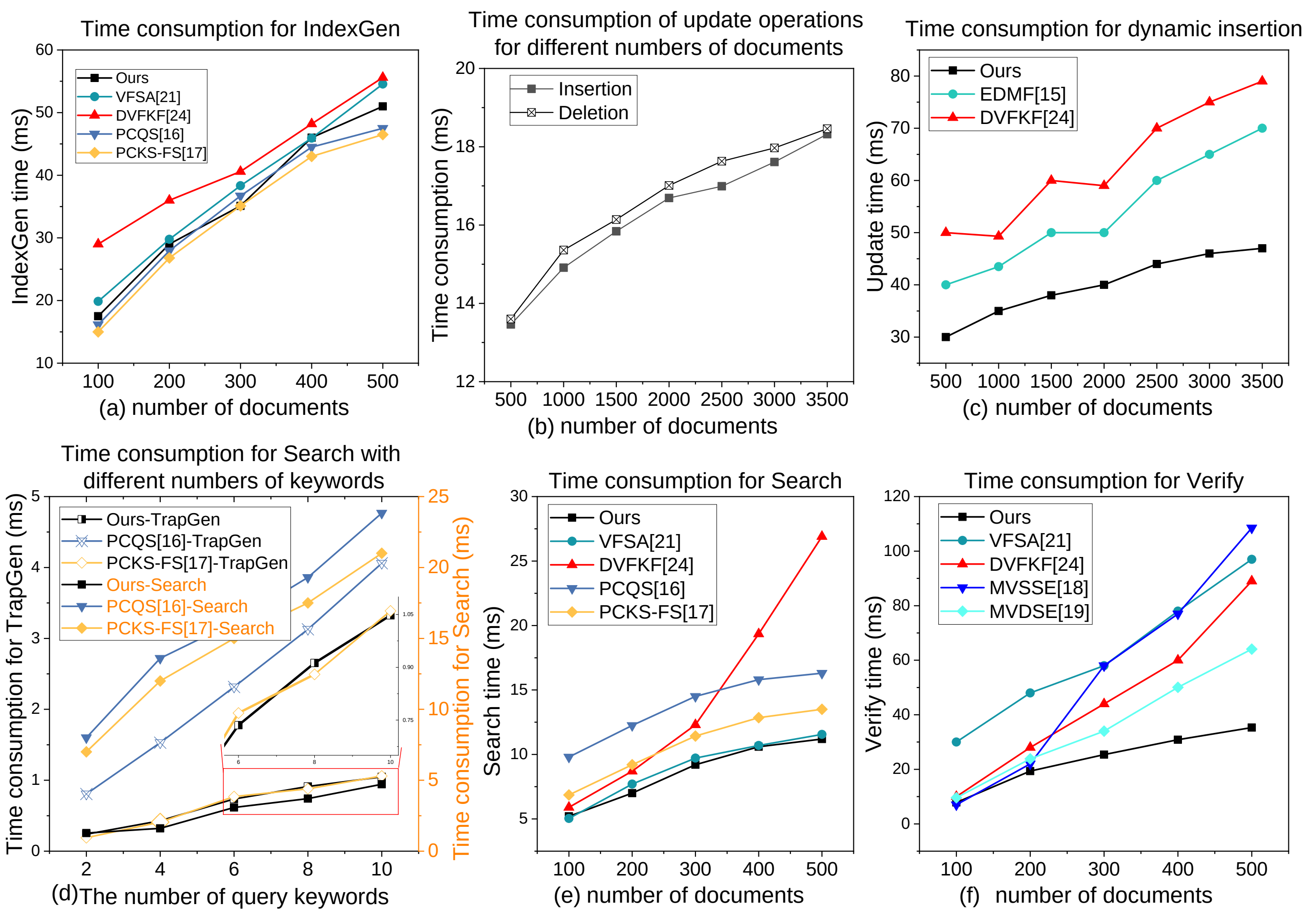


Fig. 5. Performance comparison between our VeriFuzzy and existing solutions. Time consumption for **IndexGen**(Fig.a), **Update**(Fig.b,c), **TrapGen**(Fig.d), **Search**(Fig.e), and **Verify**(Fig.f).

bucket under both incorrect and correct conditions, and M represents the total number of keywords. We first verify the retrieval accuracy under different keywords. VeriFuzzy employs uni-gram+LSH for fuzzy processing, thereby avoiding the false positives inherent in LSH (the fundamental cause of the low accuracy of EDMF[15]), and combines this with VB-Tree path intersection retrieval to improve retrieval accuracy for multiple keywords (the DVFKF[24] scheme can achieve similar accuracy using Merkle trees). We defined four error types to validate retrieval accuracy under special scenarios. As shown in the table II, VeriFuzzy and DVFKF[24] have the same accuracy rate, while EDMF[15] lacks accuracy due to its use of bi-grams for vectorization. However, VeriFuzzy achieves high accuracy in fuzzy multi-keyword retrieval.

TABLE II
ACCURACY COMPARISON OF FUZZY MULTI-KEYWORD SEARCH. T1: LETTER SUBSTITUTION (E.G.,"SECARE" $\rightarrow$"SECURE") ; T2: LETTER OMISSION/ADDITION (E.G.,"SECRE" $\rightarrow$ "SECURE") ; T3: REVERSED ORDER (E.G., "SECRUE" $\rightarrow$ "SECURE"); T4: SAME ROOT (E.G., "WALKING" $\rightarrow$ "WALK").

Schemes	Multi-keyword accuracy rate			Four types of mistakes			
	kw=1	kw=2	kw=5	T1	T2	T3	T4
VeriFuzzy	91%	87%	78%	92%	88%	100%	90%
DVFKF[24]	92%	85%	72%	92%	88%	100%	90%
EDMF[15]	85%	75%	65%	85%	80%	70%	-

3) Update: In the **IndexGen** phase, VeriFuzzy is similar to the comparison scheme, calculating the insertion values of document index keywords through k hash functions and pseudo-random functions. In the **Update** phase, since a logically full binary tree is maintained, the file identifier generated during the update is a binary path, and the complexity of any node is $O(\log n)$, with corresponding deletion consistency. In contrast, EDMF[15] computes the product of vector matrices while maintaining the tree, which is more time-consuming compared to our approach, which only performs hash computations; DVFKF[24] additionally maintains node partition indexes, resulting in higher performance overhead. The experimental results are shown in the Fig. 5 (a, b, c), particularly when the number of documents reaches 3,500, the insertion time for VeriFuzzy is only 47 ms.

C. *Search Performance* (***RQ2***)

The search process involves trapdoor generation and index traversal. During trapdoor generation, VeriFuzzy and PCKS-FS[17] perform only q hash operations and pseudo-random function evaluations, while PCQS[16] requires an additional

TABLE III
SMART CONTRACT OPERATION RESOURCE CONSUMPTION

Smart Contract	Time (ns)	CPU (%)	Storage (MB)
Trapdoor Generation	1.94	1.77	2.33
Search	7.65	3.44	12.45
Verification	3.27	1.61	2.35

k hash operations. When traversing the index, PCKS-FS[17], PCQS[16], and VFSA[21] require traversing the tree, and each node must verify the existence of the keyword. Our VeriFuzzy requires k hashes per keyword for obfuscation when querying q keywords, followed by hash index localization via historical versions and path verification via VBTree. Thus, our complexity is consistent with the aforementioned comparison schemes, but due to the pruning strategy, VeriFuzzy faces the worst-case scenario with a lower probability. In contrast, DVFKF [24] has linear complexity due to its chained retrieval. The experimental results are illustrated in Fig. 5 (d,e). Our VeriFuzzy shows advantages in search time when faced with varying numbers of query keywords. As the number of documents increases, our VeriFuzzy exhibits a sublinear growth trend in search time and achieves an improvement in search efficiency of approximately 41% when the document count $n = 500$.

D. Verification Overhead ***(RQ3)***

In terms of verification efficiency, MVSSE[18] and MVDSE[19] require the calculation of two accumulators during the verification process, with a total time complexity of $O(R(w)) + O(n)$, where $R(w)$ represents the number of documents containing the keyword w. Given that $R(w)$ is much smaller than the total number of documents n. DVFKF[24] and VFSA[21] require verification of document integrity and index integrity. During the index integrity phase, hash operations must be performed on the indexes of n documents to generate verification values. In contrast, VeriFuzzy reconstructs the search binary tree based on verification information and recalculates the ciphertext summary information to determine the consistency of the returned results, which is only $O(\log n)$. The experimental results in Fig. 5(f) show that the retrieval verification process for 500 documents takes less than 30 milliseconds, which is significantly faster than existing schemes.

E. Blockchain Performance Overhead

Since the blockchain is primarily used as an immutable storage database in this paper, the performance analysis focuses on testing three contract components: trapdoor generation, search, and verification. Based on Caliper V0.4.0^4, the relevant contracts were tested 1,000 times, and the results are shown in the table III. The results indicate that the search operation requires more time but remains within an acceptable range, while trapdoor generation and verification consume fewer resources.

The experimental results provide conclusive answers to the aforementioned research questions, summarized as follows:

- **Answer 1.** Our proposed scheme effectively handles four common types of spelling errors with a maximum search accuracy of over 80%. Furthermore, the scheme supports dynamic file addition with a time complexity of $O(\log n)$.
- **Answer 2.** The search functionality of our scheme achieves a time complexity of $O(\log n)$, and compared to existing solutions, the search efficiency is improved by approximately 41%.
- **Answer 3.** The verification algorithm in our solution demonstrates superior performance in terms of time overhead, with a time complexity of $O(\log n)$. This represents a significant improvement over the $O(n)$ complexity of baseline solutions.

VIII. DISCUSSION AND FUTURE WORK

Although VeriFuzzy demonstrates significant advantages in integrating fuzzy search, dynamic updates, and verifiable results, we acknowledge several limitations that indicate directions for future research.

A. Limitations and Practical Considerations

Multi-User Support: The current VeriFuzzy implementation focuses on single-user scenarios, which aligns with many practical enterprise deployment models where a single data owner manages the encrypted data. However, extending VeriFuzzy to multi-user settings would require additional mechanisms for key management and access control. Future work could explore attribute-based encryption or proxy re-encryption techniques to enable secure multi-user access while maintaining our core efficiency guarantees.

Language-Specific Optimizations: Our evaluation primarily uses English textual datasets, leveraging unigram-based LSH for fuzzy matching. Although the fundamental techniques are language-agnostic, specific languages (particularly character-based languages like Chinese) may require specialized tokenization and similarity metrics. The modular design of VeriFuzzy's fuzzy processing component makes it amenable to integrating language-specific preprocessing pipelines.

Blockchain Performance Considerations: Although our blockchain integration achieves $O(\log n)$ verification complexity, real-world deployment requires careful consideration of transaction costs and latency. The permissioned blockchain approach used in VeriFuzzy (Hyperledger Fabric) provides better performance than public blockchains, but organizations must still evaluate the trade-offs between verification assurance and operational costs. Future work could explore layer-2 solutions or optimized consensus mechanisms for better scalability.

B. Relation to Oblivious Computation and ML

Oblivious Computation Techniques: Systems like OblivGNN [41] employ function secret sharing to protect access patterns in graph neural network inference. While these techniques could theoretically be applied to encrypted search scenarios, their computational overhead remains prohibitive for real-time search applications. Our EVBTree architecture demonstrates that domain-specific optimizations can achieve

^{4}https://github.com/hyperledger/caliper.git comparable privacy guarantees through structural design rather than cryptographic obfuscation.

Privacy-Preserving ML Frameworks: Privacy-preserving ML systems like Leia [42] and Sonic [43] protect model parameters and input data during neural network inference, whereas VeriFuzzy addresses encrypted keyword search with fuzzy matching, dynamic updates, and verification requirements. This application divergence necessitates distinct technical approaches: ML frameworks employ secret sharing and binarized networks optimized for inference tasks, while VeriFuzzy utilizes LSH, virtual binary trees, and blockchain verification specifically designed for search operations. Consequently, VeriFuzzy achieves logarithmic complexity through domain-specific optimizations, avoiding the polynomial overhead of general-purpose privacy-preserving techniques.

C. Broader Implications

This comparison underscores that effective privacy protection requires tailoring solutions to specific application requirements, as demonstrated by VeriFuzzy's ability to provide strong security guarantees without the performance penalties of generic secure computation approaches. VeriFuzzy represents a practical advancement in encrypted search, demonstrating that careful architectural design can achieve strong security without performance compromises—a critical requirement for real-world cloud adoption. Our work showcases blockchain as a targeted solution for specific trust challenges in searchable encryption, providing a principled template for integrating cryptographic techniques with distributed systems.

IX. Conclusion

This paper presents VeriFuzzy, a dynamic verifiable fuzzy search service framework that overcomes fundamental barriers in encrypted search integration. By introducing semantic-decoupled indexing, VeriFuzzy enables efficient $O(\log n)$ fuzzy search on Virtual Binary Trees while eliminating the token explosion problem. Our blockchain-reconstructed verification achieves $O(\log n)$ completeness checks through smart contract-based path regeneration, and leakage-resilient protocols preserve privacy in fuzzy settings. Evaluations demonstrate practical performance: 47ms updates, sub-30ms verification, and 41% faster search than state-of-the-art approaches while maintaining IND-CKA2 security. VeriFuzzy establishes that functionality, efficiency, and security need not be mutually exclusive in encrypted search systems.

Artifact Availability: Our implementation is publicly available at https://github.com/JackAugust/VeriFuzzy.git to support reproducibility and future research.

References

[1] Flexera, "State of the cloud report," 2023. [Online]. Available: https://info.flexera.com/CM-REPORT-State-of-the-Cloud

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, 2010.

[3] X. Yan, C. Zheng, Y. Tang, Y. Huo, and M. Jin, "Dynamic forward secure searchable encryption scheme with phrase search for smart healthcare," *J. Syst. Archit.*, vol. 144, p. 103003, 2023.

[4] H. Zhang, S. Zeng, and J. Yang, "Backward private dynamic searchable encryption with update pattern," *Inf. Sci.*, vol. 624, pp. 1–19, 2023.

[5] R. Zhang, R. Xue, and L. Liu, "Searchable encryption for healthcare clouds: A survey," *IEEE Trans. Serv. Comput.*, pp. 978–996, 2018.

[6] Thales, "Data threat report," 2023. [Online]. Available: https://cpl.thalesgroup.com/2023/federal-data-threat-report

[7] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *2000 IEEE S&P, Berkeley, California, USA, May 14-17, 2000*. IEEE, 2000, pp. 44–55.

[8] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *2010 Proceedings IEEE INFOCOM*. IEEE, 2010, pp. 1–5.

[9] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *FC 2013, Japan, April 1-5, 2013*, vol. 7859. Springer, 2013, pp. 258–274.

[10] J. Shao, R. Lu, Y. Guan, and G. Wei, "Achieve efficient and verifiable conjunctive and fuzzy queries over encrypted data in cloud," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 124–137, 2022.

[11] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Secur.*, pp. 2706–2716, 2016.

[12] Z. Wu and K. Li, "Vbtree: forward secure conjunctive queries over encrypted data for cloud computing," *VLDB J.*, pp. 25–46, 2019.

[13] Y. Zhang, C. Xu, X. Lin, and X. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 923–937, 2021.

[14] M. Xie, X. Yang, H. Hong, G. Wei, and Z. Zhang, "A novel verifiable chinese multi-keyword fuzzy rank searchable encryption scheme in cloud environments," *Future Gener. Comput. Syst.*, pp. 287–300, 2024.

[15] H. Zhong, Z. Li, J. Cui, Y. Sun, and L. Liu, "Efficient dynamic multi-keyword fuzzy search over encrypted cloud data," *J. Netw. Comput. Appl.*, vol. 149, 2020.

[16] R. Li and A. X. Liu, "Adaptively secure conjunctive query processing over encrypted data for cloud computing," in *33rd IEEE ICDE, San Diego, CA, USA, April 19-22, 2017*, 2017, pp. 697–708.

[17] Y. Liu, X. Xiao, F. Kong, H. Zhang, and J. Yu, "Towards efficient privacy-preserving conjunctive keywords search over encrypted cloud data," *Future Gener. Comput. Syst.*, vol. 166, p. 107716, 2025.

[18] X. Liu, G. Yang, Y. Mu, and R. H. Deng, "Multi-user verifiable searchable symmetric encryption for cloud storage," *IEEE Trans. Dependable Secur. Comput.*, vol. 17, no. 6, pp. 1322–1332, 2020.

[19] Z. Xu, C. Tian, G. Zhang, W. Tian, and L. Han, "Forward-secure multi-user and verifiable dynamic searchable encryption scheme within a zero-trust environment," *Future Gener. Comput. Syst.*, p. 107701, 2025.

[20] Q. Xie, F. Zhu, and X. Feng, "Efficient and secure spatial fuzzy keyword query," *IEEE Internet Things J.*, vol. 12, no. 10, pp. 14 752–14 770, 2025.

[21] Q. Tong, Y. Miao, J. Weng, X. Liu, K. R. Choo, and R. H. Deng, "Verifiable fuzzy multi-keyword search over encrypted data with adaptive security," *IEEE Trans. Knowl. Data Eng.*, pp. 5386–5399, 2023.

[22] H. Li, T. Wang, Z. Qiao, B. Yang, Y. Gong, J. Wang, and G. Qiu, "Blockchain-based searchable encryption with efficient result verification and fair payment," *J. Inf. Secur. Appl.*, vol. 58, p. 102791, 2021.

[23] Q. Tong, Y. Miao, X. Liu, K. R. Choo, R. H. Deng, and H. Li, "VPSL: verifiable privacy-preserving data search for cloud-assisted internet of things," *IEEE Trans. Cloud Comput.*, pp. 2964–2976, 2022.

[24] N. Cui, Z. Wang, L. Pei, M. Wang, D. Wang, J. Cui, and H. Zhong, "Dynamic and verifiable fuzzy keyword search with forward security in cloud environments," *IEEE Internet Things J.*, vol. 12, no. 9, pp. 11 482–11 494, 2025.

[25] C. Xu, C. Zhang, and J. Xu, "vchain: Enabling verifiable boolean range queries over blockchain databases," in *Proceedings of the 2019 international conference on management of data*, 2019, pp. 141–158.

[26] X. Yan, S. Feng, Y. Tang, P. Yin, and D. Deng, "Blockchain-based verifiable and dynamic multi-keyword ranked searchable encryption scheme in cloud computing," *J. Inf. Secur. Appl.*, p. 103353, 2022.

[27] M. Wang, L. Rui, S. Xu, Z. Gao, H. Liu, and S. Guo, "A multi-keyword searchable encryption sensitive data trusted sharing scheme in multi-user scenario," *Comput. Networks*, vol. 237, p. 110045, 2023.

[28] R. Bost, "Forward secure searchable encryption," in *ACM SIGSAC CCS, Vienna, Austria, October 24-28, 2016*. ACM, 2016, pp. 1143–1154.

[29] J. Zhang, X. Li, M. Zheng, Z. Hou, G. Bai, and R. Feng, "VeriFuzzy: A dynamic verifiable fuzzy search service for encrypted cloud data," *CoRR*, vol. abs/2507.10927, 2025.

[30] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*. IEEE, 2014, pp. 2112–2120.

[31] Q. Liu, Y. Peng, S. Pei, J. Wu, T. Peng, and G. Wang, "Prime inner product encoding for effective wildcard-based multi-keyword fuzzy search," *IEEE Trans. Serv. Comput.*, pp. 1799–1812, 2022.

[32] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distributed Syst.*, vol. 27, no. 2, pp. 340–352, 2016.

[33] Q. Liu, Y. Tian, J. Wu, T. Peng, and G. Wang, "Enabling verifiable and dynamic ranked search over outsourced data," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 69–82, 2022.

[34] Z. Wan and R. H. Deng, "Vpsearch: Achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data," *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 6, pp. 1083–1095, 2018.

[35] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Available at SSRN 3440802*, 2008.

[36] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *20th ACM SoCG, Brooklyn, New York, USA, June 8-11, 2004*, J. Snoeyink and J. Boissonnat, Eds. ACM, 2004, pp. 253–262.

[37] J. Zhang, X. Li, R. Feng, S. Xu, Z. Hou, H. Wu, and G. Bai, "From isolation to integration: A reputation-backed auditable model for cohort data sharing," *IEEE Trans. Dependable Secur. Comput.*, pp. 1–18, 2025.

[38] J. Li, Y. Huang, Y. Wei, S. Lv, Z. Liu, C. Dong, and W. Lou, "Searchable symmetric encryption with forward search privacy," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 1, pp. 460–474, 2021.

[39] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *ACM CCS'12, Raleigh, NC, USA, October 16-18, 2012*. ACM, 2012, pp. 965–976.

[40] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *ACM CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*. ACM, 2006, pp. 79–88.

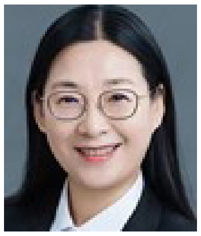
[41] Z. Xu, S. Lai, X. Liu, A. Abuadbba, X. Yuan, and X. Yi, "Oblivgnn: Oblivious inference on transductive and inductive graph neural network," in *33rd USENIX Security Symposium, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds., 2024.

[42] X. Liu, B. Wu, X. Yuan, and X. Yi, "Leia: A lightweight cryptographic neural network inference system at the edge," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 237–252, 2022.

[43] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Trans. Dependable Secur. Comput.*, pp. 620–636, 2023.

Biography Section

Jie Zhang (Student Member, IEEE) received the M.S. degree in computer application technology from the Yunnan Normal University in 2022. He is currently working toward the Ph.D. degree in computer science and technology with the Tianjin University. His research interests include efficient and secure data sharing within blockchain systems.



Xiaohong Li (Member, IEEE) received the Ph.D. degree in computer application technology from Tianjin University in 2005. She is currently a Full Tenured Professor with the Department of Cyber Security, College of Intelligence and Computing, Tianjin University. Her research interests include knowledge engineering, trusted computing, and security software engineering.

Man Zheng received B.S. and M.S. degrees in computer science from Tianjin University in 2022 and 2025. She is currently affiliated with State Grid Xiong'an Power Supply Company. Her research interests include secure and efficient searchable encryption in the cloud.

Ruitao Feng is a Lecturer at Southern Cross University, Australia. He received the Ph.D. degree from the Nanyang Technological University. His research centers on security and quality assurance in software-enabled systems, particularly AI4Sec & SE. This encompasses learning-based intrusion/anomaly detection, malicious behavior recognition for malware, and code vulnerability detection.

Shanshan Xu received the M.S. degree in computer application technology from the Yunnan Normal University in 2022. She is currently working toward the Ph.D. degree in physical geography with the School of Geographic Sciences, East China Normal University. Her research interests include Atmospheric vapor, machine learning and model building.

Zhé Hóu is a Senior Lecturer at Griffith University, Australia. He obtained his Ph.D. degree from the Australian National University in 2015. His research mainly focuses on automated reasoning, formal methods, AI, quantum computing and blockchain.

Guangdong Bai (Member, IEEE) received the B.S. and M.S. degrees in computer science from Peking University in 2008 and 2011, respectively, and the Ph.D. degree in computer science from the National University of Singapore in 2015. He is currently an Associate Professor with the Department of Computer Science, City University of Hong Kong. His research interests include cyber security, software engineering, and machine learning.