Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2025 SCPE. Volume 26, Issues 1, pp. 467–489, DOI 10.12694/scpe.v26i1.3319

ADVANCED SECURITY AND PRIVACY IN CLOUD COMPUTING: ENHANCING DATA PROTECTION WITH MULTIKEYWORD RANKED SEARCH IN ENCRYPTED ENVIRONMENTS

NARENDRA SHYAM JOSHI*, KULDEEP P. SAMBREKAR†, ABHIJIT J. PATANKAR‡, ANAND SINGH RAJAWAT§, AND MOHD MUQEEM¶

Abstract. As cloud services become more popular, encryption becomes more important for user privacy. Establishing reliable solutions for secure and fast data retrieval is crucial. This research article proposes a novel way to search encrypted cloud data. The suggested method optimises queries with multiple terms and synonyms using a greedy depth-first search (DFS) algorithm and a sophisticated rating system. The suggested architecture assumes users would search using many keywords, some of which may be synonyms for article terms. A search algorithm that uses user query synonyms was created to solve this problem. Despite the constant increase of the search universe, greedy methods help us find the most relevant information. Our depth-first search strategy improves the likelihood of finding relevant information. Our study also uses a unique ranking system that considers keyword frequency, synonym precision, and keyword proximity to determine a text's relevance to a search query. Our suggested methodology outperforms state-of-the-art methods in simulated cloud architecture experiments using encrypted datasets and industry-standard protocols. Runtime, accuracy, and recall show this superiority. The greedy Depth-First Search (DFS) algorithm optimises resources, improving efficiency. A grading method helps users quickly find the most relevant publications by naturally arranging the results. This synonym-enhanced search strategy in encrypted cloud storage systems may improve privacy and usability today.

Key words: Encrypted Data Retrieval, Synonym-Based Search Algorithms, Greedy Depth-First Searching, Cloud Security, Multi-Keyword Ranking, Searchable Encryption.

1. Introduction. In today's world, with data playing an extremely important role, the rapidly expanding cloud environments have become the primary repositories for storing large amounts of information. However, this convenience also presents significant challenges, particularly in the areas of data security and search functionality. Encrypting sensitive information before it is sent to the cloud is crucial as it ensures both the confidentiality and efficient retrieval of the data. This study introduces a new approach called "Greedy Depth-First Search and Ranking for Synonym-Enhanced Multi-Keyword Search in Encrypted Cloud Environments" to address these challenges simultaneously. The growth of cloud computing has led to a substantial increase in the amount of data transmitted and stored on remote servers. Traditional search methods are not equipped to handle the privacy requirements and complex aspects of searching encrypted material. This limitation becomes more apparent when users need to search using multiple keywords, which may include synonymous terms, making the retrieval process more complex. The proposed approach combines the computational efficiency of a modified version of the greedy depth-first search (DFS) algorithm with the flexibility of synonym-based searching. This modified DFS algorithm assigns priority to nodes based on specific criteria, allowing for a more focused search and improved retrieval efficiency. However, when applied to encrypted data, it requires the development of new indexing and search methods that can interpret and navigate the encrypted data while maintaining its confidentiality. Additionally, we enhance the ability to search for multiple keywords by in-

*Department of Computer Science and Engineering, KLS Gogte Institute of Technology, Affiliated to Visvesvaraya Technological University, Belagavi Karnataka, India (joshinarendra50@gmail.com).
†Department of Computer Science and Engineering, KLS Gogte Institute of Technology, Affiliated to Visvesvaraya Technological University, Belagavi Karnataka, India (kuldeep.git@gmail.com).
‡Department of Information Technology, D Y Patil College of Engineering, Affiliated to Savitribai Phule Pune University, Akurdi Pune, India (abhijitpatankarmail@gmail.com).
§School of Computer Sciences & Engineering, Sandip University, Nashik, Maharashtra-42213, India (anandrajawatds@gmail.com).
¶School of Computer Sciences & Engineering, Sandip University, Nashik, Maharashtra-42213, India (muqeem.79@gmail.com).

corporating synonym recognition. This enhancement recognizes the various meanings associated with human language and acknowledges that different individuals may use different terms to refer to the same concepts. By including a component that can recognize synonyms, our system significantly improves the relevance of search results, ensuring that users have access to all relevant information, not just material that exactly matches the given term. The combination of the greedy depth-first search (DFS) algorithm and synonym recognition results in a ranked search approach that effectively navigates encrypted material. This approach provides users with a sorted collection of results, prioritized based on their relevance to the specified search terms. The importance of this rating lies in its capacity to assist users in effectively identifying the most pertinent information, thus obviating the need to meticulously scrutinize each individual text that is returned. This introduction offers a comprehensive view of our system, elucidating its design concepts and the fundamental algorithms that underpin its functionality. Within this text, we delineate the architectural design of our system, exemplify its practical implementation in real-world scenarios, and substantiate its superiority over existing methodologies through exhaustive study and experimentation. The growing demand for encrypted cloud environments that provide secure, efficient, and intelligent search capabilities has spurred the development of our technique, which represents a remarkable advancement in the realm of data retrieval and security. It is our aspiration to propose a hybrid approach that bestows privacy-preserving multi-keyword search capability upon the end user, thereby retrieving relevant results with minimal delay and utmost precision. Based on this observation, the research objectives are formulated which are formally stated as:

- To study and analyze the existing searchable encryption schemes.
- To propose an efficient privacy-preserving multi-keyword ranked search scheme:
- To propose an efficient hybrid privacy-preserving multi-keyword ranked search scheme based on conjunctive and disjunctive queries over encrypted cloud data.
- To implement the proposed conjunctive, disjunctive and hybrid privacy-preserving multi-keyword ranked search schemes.
- To evaluate and compare the performance of proposed conjunctive, disjunctive and hybrid privacy-preserving multi-keyword ranked search schemes with existing scheme

2. Related work. Research has focused on developing secure and efficient search schemes over encrypted data. This includes techniques like searchable encryption, where keywords are encrypted in such a way that it's still possible to search for them without decryption.

The technique proposed in [5] aims to enhance the efficiency of doing multi-keyword searches on encrypted data stored in cloud environments. The research conducted by the authors places significant emphasis on the significance of optimising search efficiency while simultaneously upholding the anonymity of data.

The concept was further elaborated in [14] by presenting a highly efficient multi-keyword search strategy that is specifically tailored for multi-cloud situations. The technique focuses on improving the feasibility of conducting encrypted data searches across several cloud service providers.

The authors of [6] proposed a highly effective technique for ranked search of multiple keywords, utilising Latent Semantic Indexing (LSI). The researchers directed their efforts towards enhancing the search relevance by means of rating, so facilitating more precise retrieval of encrypted texts stored in the cloud.

The researchers of [1] presented a study that offers a novel perspective by integrating encryption with a graded exploration approach. The researchers used the utilisation of RC4+ encryption alongside a forest algorithm in order to enhance the security and efficacy of keyword searches in industrial applications.

The authors of [18] addressed the issue of practical multi-keyword ranked search with access control. The research conducted by the authors is particularly noteworthy due to their introduction of techniques aimed at guaranteeing that solely authorised users have the ability to access significant outcomes from encrypted cloud data.

The authors of the article [17] introduced a method for doing a semantic-based compound keyword search on encrypted data. The objective of their study was to have a comprehensive understanding of the contextual factors influencing inquiries in order to enhance the quality of search results.

The study [3] introduced a search technique that effectively addresses memory leakage issues in smart body sensor network data. This system is characterised by its dynamic nature and verifiability. The aforementioned observation underscores the wide-ranging utility of encrypted search methods in various categories of data

Table 2.1: Literature review

Paper	Methods	Limitations	Advantages	Research Gaps
[1] P. Balamurugan et al.	Searches encrypted cloud data with RC4+ and Forest.	Industrial applications only; not generalizable.	Effective for graded keyword industrial data exploration.	Could consider non-industry applications.
[3] L. Chen et al.	Fixes multi-keyword ranked search memory leaks.	Specializing in encrypted smart body sensor network data.	Solution for a niche yet crucial area.	Exploration of comparable methods in different networks.
[17] B. Lang et al.	Semantic compound keyword search.	Complex semantic techniques can be computed.	Improves search relevance through semantics.	Optimizing computing efficiency requires research
[18] J. Li et al.	Integrates cloud computing access control and prioritised search.	Access control may complicate.	Searches efficiently while protecting data.	More scalability and efficiency studies in different contexts.
[19] X. Liu et al.	Addresses distributed system privacy.	Privacy and search efficiency may be difficult to balance.	Improves dispersed data privacy.	More search efficiency study.
[31] D. Xu et al.	Designed for harsh IoV situations.	Possibly not applicable elsewhere.	Innovative solution in difficult conditions.	Explore other IoT situations.

stored in the cloud.

The authors of [19] made a significant contribution to the field with their research on privacy-preserving multi-keyword searchable encryption for distributed systems. The significance of privacy was underscored, particularly in distributed architectures where various parties may operate nodes. In the table 2.1 two represent the comparative analysis.

3. Design Goals. The proposed scheme should strive to fulfill the following design aspirations.

Creative Search: The system should possess the capacity to handle search queries encompassing multiple keywords. Alternatively, it should allow users to input a variable number of phrases, ranging from two to five, in order to imitate their authentic search patterns .

Unfruitful Exploration: To prevent searches from incurring excessive costs, users should swiftly designate them as unsuccessful. If the search terms are not found in any text within the collection, it is classified as a failed exploration. By conducting the fewest possible comparisons, a failed search can be unveiled.

Evaluated Retrieval: In order to alleviate the computational burden on end users resulting from post-processing, it is advisable to prioritize the presented search query results based on their relevance to the query.

Optimized Efficiency: When the search method is efficient, accessing the papers should be effortlessly attainable. Augmenting efficiency can be accomplished by minimizing the comparisons made between encrypted indexes and queries. Additionally, the effectiveness of search engines is determined by other parameters such as search accuracy (evaluated through metrics like recall and precision), rank efficiency, and search efficiency. In this segment, we will delve into the diverse phases constituting the general strategy for constructing the requisite search algorithms.

Figure 3.1 illustrates the employed methodologies.

The steps are the followings:

Step 1: Preliminary dataset processing. Here, we gather and prepare the open-source REUTERS-21578 dataset. To prepare a dataset, it is necessary to eliminate stop words, stem words, and generate tokens (or keywords) from each page. The TF-IDF values are computed subsequent to the discovery of keywords.

Step 2: The next stage involves the creation of a Discoverable Archive and Texts. After the initial processing

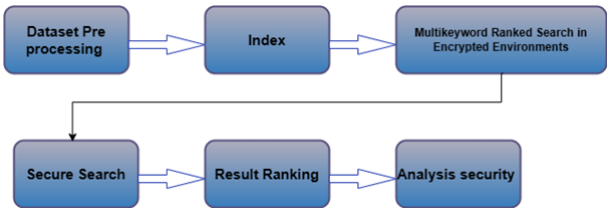

Fig. 3.1: Data processing steps

of the texts, the Department of Organization (DO) constructs both forward and inverted indexes that are presented in plain text. To transform these plain-text indexes into encrypted searchable indexes, a range of methods have been proposed in the existing body of literature. The use of encrypted keyword field-free forward and inverted indexes is preferred due to their straightforward design and efficient search capabilities. Additionally, an appropriate encryption technique, such as AES, is employed to safeguard the confidentiality of the text collection. Cs, acting as an intermediary, retrieves these encrypted texts and indexes.

Step 3: The focus shifts to the creation of a search query. The search behavior of end users is captured through multi-keyword searches. This critical step involves the generation of an encrypted search query that incorporates all the search terms entered by the user. Users have the flexibility to generate search queries using a variety of devices, including mobile phones, laptops, and desktop computers, while minimizing the utilization of processing resources.

Step 4: The process involves conducting a Secure Search Check against the outsourced indexes. In the existing literature, a range of symmetric search techniques have been proposed. Here, we propose methods to reduce the search time for text retrieval, including text clustering and keyword binning based on distinct indexes. Three distinct search strategies, namely conjunctive, disjunctive, and hybrid, are recommended, each leveraging a unique index structure.

Step 5. The next stage involves Ranking the results. By minimizing the transmission and post-processing computing burdens on end users, ranked results offer superior search efficiency compared to Boolean retrieval. At this point, the suggested conjunctive, disjunctive, and hybrid search strategies are further enhanced through the implementation of a TF-IDF score-based result ranking scheme, enabling users to access ranked search results.

Step 6: The final step entails Analyzing performance and security. To evaluate the suggested conjunctive, disjunctive, and hybrid search strategies, we employ a publicly accessible dataset and compare their performance to other approaches discussed in the literature. Various metrics, including storage requirements, computational costs, search efficiency, search accuracy (recall and precision), and rank efficiency, are employed to quantify the performance of these strategies.

4. Proposed methodology. The collection of classified cryptographic codes, known as HK and encompassing hk_1, $hk_2, \cdots$, hk_n, is employed for the production of the index. Moreover, this very assemblage, HK, acts as the gateways that facilitate the formation of inquiries. The gathering of encryption keys $SK = sk_1, sk_2, skN$ serves the purpose of securing N texts within the compilation. In the proposed design, it is assumed that these N secret keys are conceived and overseen by the custodian of data. Additionally, during the preparation phase, the data owner preprocesses the texts to be unveiled on the server, aligning them with plain-text IR. To alleviate the initial processing burden, a variety of tools such as R and Apache Lucene exist. The texts are subsequently deciphered using suitable decoders. Once deciphered, tokens (or keywords) are generated and subsequently, the removal of stop words is carried out. By discarding stop words from the register of keywords, the size of the keyword lists is reduced. Following the process of tokenization, all the tokens are converted into lowercase characters. Subsequently, the tokens are subjected to the algorithms of stemming in order to obtain the root (or base) form of the tokens. For stemming, a range of stemmers such as Lovins stemmer, Paice stemmer, and Porter stemmer are available. In order to facilitate the ranking of results

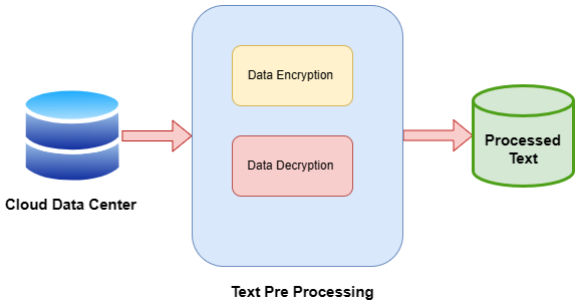

Fig. 4.1: Simple data encryption and data Decryption

(as performed by a multitude of search engines), the relevance score of the keywords (such as TF, TF-IDF) is computed. The various steps executed during the preprocessing of texts are illustrated in Figure 4.1.

4.1. Text Clustering Base. The suggested approach makes use of clustering, a technique that groups texts based on their shared qualities. Finding the optimal distance between clusters while keeping the gap inside each cluster as little as possible is the main goal. Several approaches are available for clustering texts, including k-means, k-partitions, and topic-based partitions. Part of our strategy is to divide the texts into many clusters using the k-means algorithm, which is widely used in the field of information retrieval. The efficiency goals in software engineering and the unique dataset properties dictate the clustering algorithm to be used.

4.2. Text Index Generation Phase. The DO embarks on an imaginative voyage with the following stages to build the encrypted index for every text (Di, where $1 \leq i \leq N$):

Step 1: The DO uses a powerful method called HMAC for every keyword (kx).(1) hidden in the multiple depths of text Di (assuming m keywords per text). This method, which resembles a magical chant, transforms the input into an enthralling output of fixed length (l). A set of keys, called HK, was given to the DO during the setup ritual. These keys hold the power to unlock the encrypted realm's secrets, thus it is the DO's sacred duty to guard and support them see equation 4.1.

$$H_l : \{0,1\}^* \; HKACkey \; \rightarrow \; \{0,1\} \tag{4.1}$$

Step 2: Divide the l-bit binary string into z segments, where d bits is the length of each segment. Make a substring out of every segment. using the formula $zj = zjd_3 \; zjd_2 \; zjd_1 \; zjd_0$ for all values of j from 1 to r.

Step 3: Use to reduce the d-bit substring zj to a single bit (either 0 or 1). The output bit for keyword kx in the keyword index Ikx $[j]$ is 0 if all the bits in the d-bit substring zj are equal to 0 ($zjd_3 = 0, \; zjd_2 = 0, \; zjd_1 = 0, \; zjd_0 = 0$), and 1 otherwise look the equation 4.2.

$$I_{kx}[f] = \{0,1\} \frac{\text{if } zjd - 1 = 0^{\wedge}..^{\wedge}zj2 = 0^{\wedge}zj_1 = 0^z j_0 = 0}{\text{if Otherwise}} \tag{4.2}$$

Step 4: By performing a bitwise AND operation on the indexes of the m keywords, we may determine an r-bit index (IDi) for text Di, equation (4.3). In order to accomplish this, we extract each individual bit from the index and utilize it as a value in a bitwise AND operation with all of the keyword indexes. By default, the value of $Di[j]$ is initialized to 0. However, it is changed to 1 only if all the keyword indexes have a 1 at the exact same bit position. The subsequent depiction demonstrates the execution of this procedure in the equation 4.3.

$$I_{Di}[j] = \odot_{kx=1}^{m} \; Ikx[j] \; 1 \leq j \leq r \tag{4.3}$$

4.3. Clustering Dex Generation Phase. The Cluster Head (CH) serves as the text representation for the entire cluster, encompassing all of its keywords, during the Index Generation Phase for Clusters equation

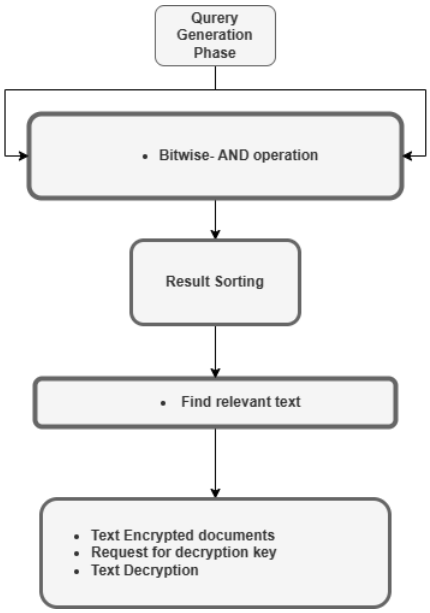

Fig. 4.2: Online Stage

(4.4). One can determine the cluster index by calculating the HMAC, decreasing the set, and subsequently applying a bitwise-AND operation to all the keywords (ki) in the set.

$$C_k[j] = \odot_{kx=1}^{|T|} Iki[j] \ 1 \leq j \leq r \tag{4.4}$$

However, a disadvantage of constructing the Cluster Head in this manner is that it imposes a greater computational load on the data owner. The data owner is required to calculate the HMAC (5), reduce, and conduct a bitwise-AND operation for all the T keywords. To address this difficulty, an alternative strategy is to create the Cluster Head by leveraging the index of the texts it holds. Let IDi denote the index of the ith text discovered within a cluster, where $1 \leq i \leq T$. The index of the kth cluster, represented as Ck, is computed via bitwise operations. AND operation of the indexes.

$$C_k[j] = \odot_{Di=1}^{\lambda} IDi[j] \ 1 \leq j \leq r \tag{4.5}$$

Text Encryption Phase Once the text and cluster indexes have been computed, DO proceeds to encrypt the texts. This ensures that the encrypted indexes and texts may be made available on CS. Text encryption utilizes a symmetric encryption algorithm, such as AES, in which the creation and management of text encryption keys are handled by DO. The encrypted text (Ei) for the plain text text (Di) is generated using the symmetric key encryption algorithm (ENC) with the key (keyi). The encryption method ENC takes a plain-text text (Di) and a key ($keyi$) as inputs to produce the encrypted text (Ei). Employing a solitary encryption key poses a security flaw, as possessing knowledge of the encryption key confers access to all encrypted data. Therefore, anyone possessing this singular key has the ability to decipher and gain entry to all of the texts. In order to prevent this situation, multiple keys are employed. This ensures that even if one key is unintentionally exposed, only the contents of one specific text or a subset of texts are published. As a result, the confidentiality of the remaining texts is maintained.

$$E_i = ENC(D_i, key_i), 1 \leq i \leq N \tag{4.6}$$

Phase of inquiry generation Here, the user inputs xi search phrases (where 1 xi is the assumed number of terms per query) and generates query Q. The end user must follow these steps to generate the query:

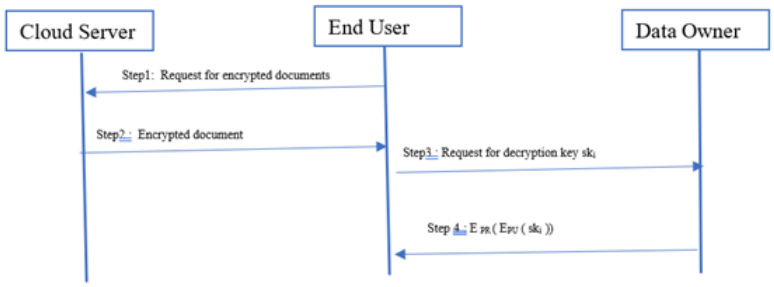

Fig. 4.3: Working of Simple data encryption and data Decryption

Step 1: The query terms are hashed using the $H14$ hash function.
To get the hidden HMAC keys out of the HMAC lookup table, one uses a hash function. One advantage of using an HMAC lookup table to retrieve the trapdoor (HMAC keys) instead of sharing it with DO as needed is that it does not require the data owner to be online at all times. Given the potential for data owners to be offline owing to factors such as various time zones or a single point of failure, this key need has been removed from the system.

Step 2: The user then computes the HMAC for the query terms after getting the trapdoor information. Thirdly, the index for every query word is generated by performing the reduction and bitwise-AND operation , equation (4.7), which is similar to the text index production step. the query Q yields the following: the j-th bit of the search query is created by bitwise-ANDing the j-th bit of the index (Ixi) with the query terms (xi).

$$Q[j] = \odot_{xi=1}^{a} \; Ixi[j] \; 1 \leq j \leq r \tag{4.7}$$

Text Decryption Phase of the Process After CS has compiled a collection of papers that coincide with the user's requirements, the user will next request the encrypted texts that they are looking for. Due to the fact that DO is in possession of the encryption key, the data owner will supply the user with an encrypted symmetric key (ski) whenever the user requests that DO decode the files that they have simply downloaded. The user's public key (PU) and the data owner's private key (PR) are combined in order to produce this key. This key is generated for the user. Within the context of a text decryption process, illustrates the dynamic relationship that exists between the three components. The DO is the only entity that is capable of generating this message because the private key of the DO is utilized in the process of encrypting the secret key (ski). As a final point of interest, confidentiality and safety are guaranteed due to the fact that the symmetric key can only be signed with the user's public key, which means that no one other than authorized users may extract the private key from the symmetric key. In the figure 4.3 working of Simple data encryption and data Decryption.

$$E_{PR}(E_{PU}(sk_i)) \tag{4.8}$$

By using a technique to prioritize search results to give the most relevant resources, plain-text information retrieval streamlines the process for users by removing the need to wade through irrelevant content. The problem with Boolean retrieval is that it floods users with incorrect content, which causes processing overhead during retrieval, decryption, and rejection.

We suggest doing the TF-IDF score calculations for the text's keywords while the process is offline. To classify the keywords into different levels of relevance, we use the TF-IDF scores. Assuming their scores meet or exceed the requirements for that level, keywords can be located in both earlier and current levels. Reliability in ranking and retrieval is guaranteed by this.

In order to determine the text's influence within the cluster, its contribution is examined. When two texts have the same level of importance, the one with the more significant contribution will be ranked higher. This

Algorithm 1 Ranking Texts Based on Query Relevance in Clusters

Variables:

1. C: Cluster index
2. Fi: ith text within the selected cluster C
3. M: Set of matching clusters
4. Q: Query
5. R: List of relevant texts

Input:

1. Q: Query

Output:

1. R: List of relevant texts
1. For each selected cluster C in M:
2. For all the texts Fi in selected cluster C, compare the level-1 index of Fi and query Q:
3. if there is a successful match between the level-1 index of Fi and query Q then:
4. Compare the level-z index of Fi and Q, where $= 2, 3, z = 2, 3, \ldots :$
5. (Perform additional comparisons)
6. end if
7. The highest matching level (z) corresponds to the rank of text Fi
8. End for
9. For all the texts with the same relevance level value, organize them in decreasing order of contribution to the cluster.
10. End for
11. Return R.

evaluation is predicated on the text's and the cluster index's percentage of matching zeros, or characteristic bits. Texts with the same level of relevance become more relevant when the matching percentage is higher. In keeping with current CS-based ranking algorithms, this scheme protects privacy by not revealing collected information unless the text has the greatest matching relevance level. The advantages of multi-level indexing outweigh the storage requirements (Npr, where N is the text count with p indexes of r-bit length), particularly in cloud situations where end users efficiently receive top results.

A framework for the study All of the queries that have been developed so far have been deterministic. By using the HMAC function, reducing the result, and then performing a bitwise-AND operation, the identical binary query has been generated for every set of query terms. All of the generated queries are identical to one another because the indexes are fixed and the search terms are dynamic. Due to this, an adversary can readily determine if several queries are similar by looking at the query itself. This allows us to foresee how end users will search while simultaneously reducing the privacy of searches. Searchable encryption literature frequently employs the approach of generating queries using a mix of real and random keywords (sometimes called dummy or noise terms) to safeguard inquiry privacy. Also, these terms aren't relevant at all, so they shouldn't be considered real keywords in the collection. The suggested method makes use of noise words since they are the sole adequate option for inquiry privacy. Papers that contain both actual and noise terms are the only ones that are considered important according to the given approach, which makes conjunctive searching easier. Since the noise terms are added to the query and not previously contained in the text indexes, it is anticipated that the system's accuracy will be low. If you care about the confidentiality of your question, adding noise phrases to it won't cut it. The offline stage builds a set of noise words called X to circumvent this. The true terms contain these distracting phrases in all of the texts. Expanding the keyword list is the first step in creating a searchable index. That way, you may be certain that the text index contains both real and made-up words. The user will choose Y subsets from these X noise words and incorporate them into the query terms as it is being prepared. Search queries that include the same phrases will seem to have different phrases when additional noise terms are added to them. Here we are since it's highly unlikely that we will use the same noise phrases for several searches. Because these distracting phrases appear in every single text, we also search for all

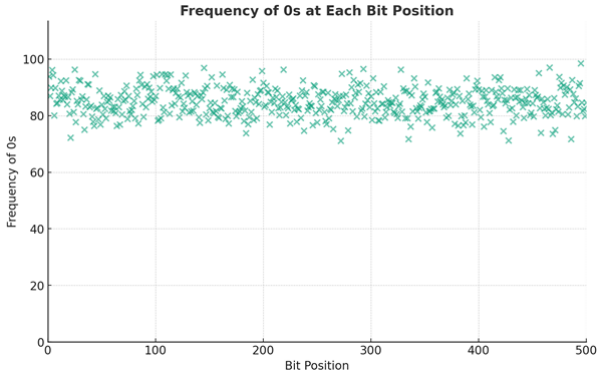

Fig. 4.4: Frequency of os at each bit position

of the relevant texts. The proposed plan has a 100% recall rate because all the necessary texts were accepted. Using the bitwise AND operation, we can keep the text keywords intact all through the search. Furthermore, these keywords are compared to the distinctive parts of the search phrase. For this reason, this remains true. The strategy's ability to protect query confidentiality until an opponent can't access the searchable index is one of its drawbacks. An opponent can learn about the noise terms in the searchable index the moment they acquire access to the index. The attacker searches the text indexes extensively for the bits with values of 0. In order to set these bit positions, dummy sentences have been embedded within all of the papers. An opponent can then designate these query bit positions as 1, requesting the real words and rendering the noise terms useless, as soon as they become aware of this. we can see how often zeroes appear at r-bit places in a randomly chosen cluster . In the proposed system, this is predicated on the premise that $r = 448$. Bit positions with values of one hundred percent are set because all texts in the cluster have dummy words inserted to them. The impracticality of picking unique noise parameters for each cluster is an additional constraint. According, one tactic to avoid an adversary finding out about this knowledge is to add fictional papers to the collection6 in this way. In keeping with the data given in 4.8*b*, these newly generated papers are included in order to normalize the number of zeros present at each bit point. Regardless, this strategy is not without its flaws: The amount of comparisons needed increases from N to 2N when fraudulent texts are included, since the collection size is doubled. Because of this, the search becomes less efficient. Since end users don't care about data owners uploading phony papers, attackers can readily spot them. To accomplish our aims of enhancing performance and security, we need a technique for query randomization.

4.4. Proposed Secure Query Randomization Scheme. As per the present method of query randomization, it is of utmost importance to incorporate random noise terms into queries. Through the process of concealing the similarities between queries, these noise terms enhance the privacy of searches. Adding noise words to the entire collection, on the other hand, would be an implementation that would be impossible. One idea that has been proposed to improve query privacy is the insertion of noise terms at the text level, which is based on the true keyword relation. It is more common for us to randomly sprinkle noise keywords onto texts and queries than to apply them everywhere. The objective of this method is to maintain the precision of the search while simultaneously improving privacy. We provide a new query randomization approach that makes it impossible for an adversary to detect which positions are affected by noise terms. This is done in order to maintain the integrity of the collection size as well as the security of the queries.

4.5. Adding Noise to Text Index. DO creates a noise lookup Figure 4.4 with unique keys in the offline phase. The noise phrases are retrieved from this Figure 4 for every authentic term in a text by utilizing the hash of the authentic phrase. The acquired bits are arranged into a collection of V noise terms for each text, which are unique because to the varying keywords in the material. However, in other texts, the identical actual words generate equivalent noise terms. Next, the index is computed by utilizing a collection of noise words,

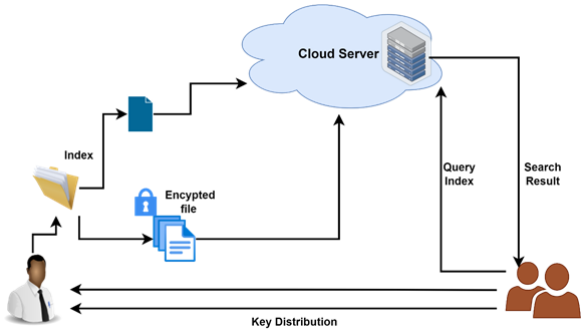

Fig. 4.5: Synonym-Enhanced Multi-Keyword Search in Encrypted Cloud Environments

encompassing both genuine and artificial. We determine the occurrence rate of zeros at various bit positions inside a randomly selected cluster. The plotted graph in Figure 3.1 displays the frequency values. The disparity between the noise-set zeros and the actual terms is diminishing, as seen. Given that the text indexes are the sole repository of information, it is challenging to ascertain whether the bit locations are influenced by the actual terms or the noise terms. The privacy of query data is enhanced, and the system can prevent the analysis of searches by eliminating bits that are influenced by noise words.

The results of comparing the current query randomization technique with the proposed system for $10,000$ texts in the REUTERS-21578 dataset. The study on descriptive statistics demonstrates that the proposed method of randomizing queries can effectively provide a uniform distribution of zero values. Consequently, differentiating between noise and genuine phrases becomes increasingly difficult, in contrast to the existing system. The significance of efficient and secure search algorithms cannot be overstated, given the escalating volume of encrypted data being handled by cloud services. Traditional search algorithms are unable to decipher the semantic features of encrypted text, making them ineffective in the presence of encryption. The suggested approach seeks to rectify this discrepancy by improving search efficiency through a ranked retrieval mechanism and considering synonym links between terms. Performing a search operation on the cloud can be difficult because of the encryption used to protect sensitive data. We need an algorithm that can efficiently navigate the search area, taking into account the relevance of files to the provided keywords and their synonyms. This will allow us to do searches that combine both criteria. To do this, one can integrate a rating algorithm with a Greedy Depth-First Search (Greedy DFS) in order to identify the encrypted files (nodes) that are most probable to contain the requested data (Figure 4.5).

Synonym-enhanced multi-keyword search in encrypted cloud environments lets we find relevant data in the cloud, even if it is stored securely and you use synonyms or related words in your search. Imagine a locked treasure chest full of texts, but we can only search for things by whispering clues through a tiny keyhole. This technology helps we find the right treasure even if you whisper “crown” instead of “tiara.” Encrypted data: Your data in the cloud is scrambled, so the cloud provider cannot read it. Multi-keyword search: You can search for multiple words at once, like “ancient map” or “lost city” Synonym awareness: The search understands synonyms and related words, so “treasure” also finds “loo” or “riches”

4.6. System Architecture. Figure 4.6 shows the System Architecture that allows users to input multi-keyword search queries, which might include synonyms and exclusion phrases.

Query Pre-Processing: Determines synonymous terms by utilising an internal or external repository of information. Implements discretionary query expansion to enhance retrieval by including more results. Creates a well-organized query using weighted Boolean operators (OR, AND, NOT).

Query Encryption: Utilises robust encryption methods, such as searchable encryption, to safeguard the secrecy of queries. Guarantees that encrypted queries do not disclose any sensitive information to the server.

Cloud Server-Side: The system stores an encrypted index of texts that can be searched using keywords, allowing

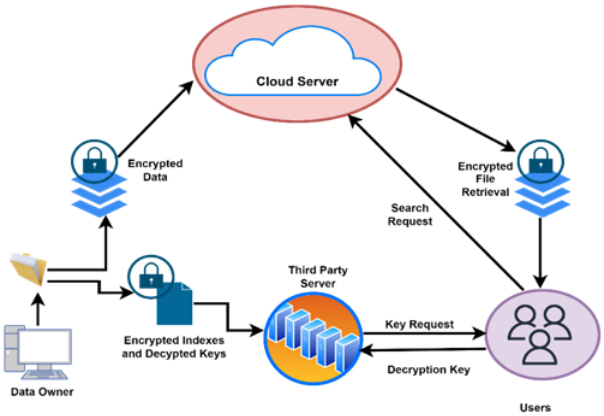

Fig. 4.6: System Architecture

retrieval of information without the need for decryption.

Encrypted Query Processing: Utilises efficient search algorithms to compare encrypted requests with the encrypted index.

Greedy Depth-First Search (DFS) Algorithm: Traverses the encrypted index structure using a depth-first approach. Assigns more priority to branches that have greater potential importance, as determined by their intermediate ratings. Assigns more priority to branches that possess greater potential significance, as judged by their intermediate grades

Ranking: Employs the Rank(S, Q) formula to calculate relevance scores for retrieved texts: $w1 * OR(Synonym1, Synonym2, ..., SynonymN)$, $w2 * AND(Keyword1, Keyword2, ..., KeywordN)$, $w3 * NOT(Unwanted Keyword1, UnwantedKeyword2, ..., UnwantedKeywordM)$. Adjusts weights $(w1, w2, w3)$ for desired balance between synonyms, mandatory keywords, and exclusion terms.

Client-Side: Obtains ciphered outcomes from the server. Deciphers the data using the suitable encryption key.

Result Presentation: Displays decrypted texts in ranked order based on calculated relevance score Here is a high-level plan for a Greedy DFS and Ranking algorithm enhanced with synonyms for use in multi-keyword searches in secure cloud storage.

Data Encryption: The data undergoes encryption through a sophisticated encryption mechanism, enabling its secure storage in the cloud while preserving the capability to do searches on encrypted terms.

Synonym Dictionary Creation: We develop a comprehensive synonym dictionary that maps keywords to their synonymous counterparts. This is integrated into the search mechanism to enhance the search results' relevance by capturing the semantic relationships.

Greedy DFS Algorithm for Search: A DFS algorithm with a greedy strategy is suggested as a means to effectively explore the search space. The system assigns higher priority to nodes (encrypted files) that have a higher likelihood of containing the desired keywords. This prioritisation is determined by a heuristic that takes into account both the presence of the principal keywords and their synonyms. The equation for the heuristic might look something like this:

$$H(n) = \alpha \cdot f(n) + \beta \cdot g(n)$$

where $H(n)$ is the heuristic function for node n. $f(n)$ is a function that returns a value representing the presence of the principal keywords at node n.
$g(n)$ is a function that returns a value representing the presence of synonyms of the keywords at node n. α and β are weighting factors that determine the relative importance of the presence of principal keywords and their synonyms, respectively.

In a greedy DFS, the heuristic value is used to determine which child receives attention next. The algorithm

for GreedyDFS(node) in Step 1 is as follow:

1. if node contains target:
2. return node
3. mark node as visited
4. children = get_children(node)
5. sort children by H(n), in descending order
6. or child in children:
7. if child is not visited:
8. result = GreedyDFS(child)
9. if result is not None:
10. return result
11. return None

In order to ensure that the search begins with the child most likely to contain the specified keywords, the children are sorted by the heuristic value $H(n)$.The actual implementation of the heuristic functions $f(n)$ and $g(n)$ and the weighting factors α and β would depend on the specific application and the characteristics of the encrypted files and keywords.

$$\begin{aligned} Rank(S,Q) = {} & w1 * \ OR(Synonym1, Synonym2, \ldots, SynonymN) \\ & + w2 * AND(Keyword1, Keyword2, \ldots, KeywordN) \\ & - w3 * NOT(UnwantedKeyword1, UnwantedKeyword2, \ldots, \\ & \quad UnwantedKeywordM) \end{aligned} \tag{4.9}$$

$Rank(S,Q)$ is the ranking function for a search result S given a query Q. $w1, w2$, and $w3$ are weights that determine the importance of each component in the ranking algorithm. $OR(Synonym1, Synonym2, \ldots, SynonymN)$ is the OR operation applied to synonyms of the keywords. $AND(Keyword1, Keyword2, \ldots, KeywordN)$ is the AND operation applied to the keywords. $NOT(UnwantedKeyword1, Unwanted$ $Keyword2, \ldots, UnwantedKeywordM)$ is the NOT operation applied to exclude unwanted keywords.

4.7. Ranking Mechanism. The search results are ordered based on a relevance scoring algorithm that considers factors such as keyword frequency, the inclusion of synonyms, and heuristic scores obtained from the DFS traversal. This process guarantees that the texts with the highest relevance are retrieved as a priority (Algorithm 2).

Search Query Optimization: In order to enhance the efficiency of search queries, a pre-processing step is employed to determine the most optimal synonyms and combinations of terms. The pre-processing stage serves to decrease the size of the search space and enhance the efficiency of the search process.

Security and Privacy Considerations: We implement measures to guarantee the security of the search mechanism, effectively mitigating potential threats such as keyword guessing and frequency analysis. In order to protect anonymity, the synonym dictionary is additionally subjected to encryption. The suggested methodology is anticipated to revolutionise the manner in which encrypted data is queried, enhancing both user-friendliness and efficiency, all the while upholding robust standards of security and privacy. Our approach utilises synonym-enhanced search queries and a greedy DFS ranking mechanism to make a notable advancement in the domain of secure cloud data retrieval

5. Results Analysis. The performance was evaluated using the REUTERS-21578 dataset, with parameters as detailed in Table 5.1 outlines the parameter settings for the analysis.

The texts were clustered using the k-means algorithm with 5 and 10 clusters, executed in python The text count per cluster is shown in Table 5.1. The text number varied from $1,000$ to $10,000$ for the comparative analysis. By concatenating outputs from various hash functions, a binary index of 2688 bits was produced, which the reduction factor of 6 shortened to 448 bits. The proposed scheme supports efficient conjunctive searching through keyword field-free indexes, as initially suggested by Orencik and Savas [33,42]. It differs from the existing scheme by the reduced number of texts it examines to retrieve relevant results, examining only those within the matching cluster, thereby reducing comparison counts and search time. This proposed scheme was compared with the existing scheme [33,42], which was implemented with the same parameters as described in Table 5.1.

Algorithm 2 GreedyDFSWithRanking [Step 1:]

Input: rootNode, keywords, synonyms, alpha, beta
Output: Ranked list of relevant nodes (files)

1. Initialize an empty list for rankedFiles
2. Define H(n) using keywords, synonyms, alpha, beta
3. Call GreedyDFSVisit(rootNode)
   Procedure GreedyDFSVisit(node)
4. if node is a file:
5. Compute relevance score using H(node)
6. Add node and score to rankedFiles
7. else:
8. children = GetEncryptedChildren(node)
9. Sort children by H, in descending order
10. for child in children:
11. if child is not visited:
12. Mark child as visited
13. GreedyDFSVisit(child)
14. Sort rankedFiles by relevance score in descending order
15. return rankedFiles

Table 5.1: Simulation environments with parameter

Dataset Name	**Cluster count**	**Number of Texts**	**Hash Function for Indexing**	**HMAC Functions for Query Construction**	**Reduction Factor (d)**	**Final Query Length (r)**	**Server Configuration**	**Programming Language**
REUTERS-21578 dataset [320]	Uniform: 5, Non-uniform: 10	1,000 to 10,000	MD5	SHA-256, SHA-384, SHA-512	6	448 bits	Intel Xeon Processor, 4 TB Hard Drive, 64 GB RAM	Python

5.1. Search Efficiency. To fetch the texts the users share the r-bit long query with CS. The texts are distributed into clusters leading to two possibilities regarding the occurrence of the texts in the cluster : Hard clustering: In hard clustering, a text appears only in one cluster.

Uniform Text Distribution: Each cluster, from 1 to 5, contains an identical number of texts, totaling 1200 in each. Consequently, the sum of texts across all clusters amounts to 6000.

Non-Uniform Text Distribution: The text count varies across clusters 1 to 10, with the highest concentration in cluster 1 (3966 texts) and the lowest in cluster 10 (239 texts), cumulating in a total of 10000 texts. When it comes to identifying relevant texts, the approach differs based on the clustering method employed.

Hard Clustering: In hard clustering, texts are exclusively assigned to a single cluster. Algorithms in this category do not recognize multiple themes within a text. To find pertinent texts, one needs to identify the one relevant cluster and then search within it.

Soft Clustering: Soft clustering allows for a text's presence in several clusters. These algorithms are capable of discerning multiple themes within a text, which aids in exploring various relationships among the data. Finding relevant texts in this context entails pinpointing all pertinent clusters and searching within them.

The distribution scenarios presented are reflective of two distinct possibilities:

Uniform Distribution: This is characterized by an even or nearly even distribution of texts across clusters. However, such a distribution is quite rare in practical scenarios.

Algorithm 3 Proposed Algorithm 3: GreedyDFSAndRankingSearch [Step 1:]

```
Input: encryptedFiles[], searchKeywords[], synonymDictionary, maxResults
Output: rankedFiles[]
function expandKeywordsWithSynonyms(searchKeywords, synonymDictionary):
expandedKeywords = []
for keyword in searchKeywords:
expandedKeywords.append(keyword)
synonyms = synonymDictionary[keyword]
for synonym in synonyms:
expandedKeywords.append(synonym)
return expandedKeywords
function greedyDFS(encryptedFiles, expandedKeywords):
stack = [] // Stack for DFS
visited = []
foundFiles = []
for file in encryptedFiles:
if not file.isDirectory:
stack.append(file)
while stack:
currentFile = stack.pop()
visited.append(currentFile)
decryptedContent = decrypt(currentFile.content)
if containsKeywords(decryptedContent, expandedKeywords):
foundFiles.append(currentFile)
if len(foundFiles) == maxResults:
break
if currentFile.isDirectory:
for child in currentFile.children:
if child not in visited:
stack.append(child)
return foundFiles
rankFiles(foundFiles, expandedKeywords):
rankedFiles = []
for file in foundFiles:
matchScore = calculateMatchScore(file, expandedKeywords)
rankedFiles.append((file, matchScore))
rankedFiles.sort(key=lambda x: x[1], reverse=True)
return [file for file, score in rankedFiles]
function mainSearch(encryptedFiles, searchKeywords, synonymDictionary, maxResults):
expandedKeywords= expandKeywordsWithSynonyms(searchKeywords, synonymDictionary)
foundFiles = greedyDFS(encryptedFiles, expandedKeywords)
rankedFiles = rankFiles(foundFiles, expandedKeywords)
return rankedFiles
rankedFiles = mainSearch(encryptedFiles, searchKeywords, synonymDictionary, maxResults)
```

Non-Uniform Distribution: Here, the number of texts in each cluster differs based on the degree of text similarity. This distribution is more common and realistic.

A search scheme must offer high accuracy and efficiency to be considered for practical use. The proposed search scheme improves search efficiency by reducing the average search time required to find relevant texts, unlike the existing schemes [33,42] which necessitate scanning the entire text collection. To evaluate search accuracy, metrics like recall, precision, F1 score, and False Accept Rate (FAR) were computed. We performed a test using 100 queries, each containing 5

Algorithm 4 Proposed Algorithm 4 [Step 1:]

```
Function SearchEncryptedData(keywords, synonymDictionary, encryptedData):
rankedResults = []
parsedQuery = ParseQuery(keywords, synonymDictionary)
For each record in encryptedData:
decryptedRecord = Decrypt (record)
matchScore = EvaluateRecord(decryptedRecord, parsedQuery)
If matchScore > 0:
rankedResults.Add((record, matchScore))
rankedResults.SortByDescending(r => r.matchScore)
Return rankedResults
Function ParseQuery(keywords, synonymDictionary):
queryComponents = []
For each keyword in keywords.split(' '):
If keyword is not a logical operator ('AND', 'NOT'):
synonyms = synonymDictionary.GetSynonyms(keyword)
queryComponents.Add((keyword OR synonyms))
Else:
queryComponents.Add(keyword)
Return queryComponents
Function EvaluateRecord(decryptedRecord, parsedQuery):
matchScore = 0
For each component in parsedQuery:
If component contains 'NOT':
If not decryptedRecord.Contains(component.keyword):
matchScore += 1
Else If component contains 'AND':
If decryptedRecord.ContainsAll(component.keywords):
matchScore += 1
Else:
For each synonym in component.synonyms:
If decryptedRecord.Contains(synonym):
matchScore += 1
Break // Exit loop after the first match
Return matchScore
```

Table 5.2: Comparative analysis proposed Scheme and existing scheme

Parameter	**Proposed Scheme**	**Existing Scheme [33,42]**	Gain
Recall	100%	100%	Same
Precision	84.2%	76.27%	+6.13%
F1 Score	89.07%	84.89%	+4.18%
FAR	0.128%	0.286%	-55.24%

relevant and 30 unrelated terms, on the text collection."

In the figure 5.1 to represent the Search Accuracy Comparison. Let's create a revised table 5.2 to clearly present the search accuracy data based on the information provided:

Tables 5.3 and 5.3 show the the search accuracy comparison between the proposed scheme and the existing ones, and Tools and Technology indicating improvements in precision, F1 score, and a reduction in the False Accept Rate. The recall remains the same for both schemes at 100%. The gain column represents the percentage increase or decrease in the performance of the proposed scheme compared to the existing schemes. Our intended course of action involves

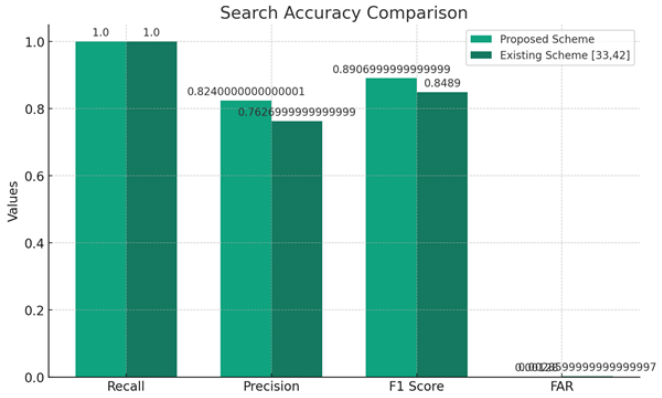

Fig. 5.1: Comparative analysis proposed Scheme and existing scheme

Table 5.3: Tools and Technology

Tool/Technology	**Purpose**	**Description**
Encryption Software	Data Security	Software used to encrypt the cloud data. Examples include AES, RSA, or custom encryption algorithms.
Cloud Platform	Data Hosting	Cloud service provider used to host the encrypted data. This could be AWS, Azure, Google Cloud, etc.
Indexing Engine	Data Retrieval	Tool used to create searchable indexes for the encrypted data. Examples might include Apache Lucene or Elasticsearch.
Synonym Database	Search Enhancement	A database or API service like WordNet that provides synonyms for extending search capabilities.
Greedy DFS Algorithm Implementation	Search Algorithm	Custom or pre-built greedy DFS algorithm used to perform the ranked searching.
Programming Language	Development	Language used for implementing the search algorithm and handling the encryption/decryption. Likely candidates are Python, Java, or C++.
Simulation Software	Testing & Analysis	Software used to simulate the cloud environment and measure the performance of the search algorithm. Could be MATLAB, Simulink, or a custom simulator.

assessing the efficacy of our suggested methodology by employing a cloud-based simulation environment. This evaluation will be conducted utilising a diverse range of datasets that encompass encrypted texts. The evaluation of the search's effectiveness will be conducted based on precision, recall, and computational time. In order to showcase the enhancements in search relevance and efficiency, we will conduct a comparative analysis of our technique with the currently existent search schemes.

Explanation of Metrics. In the table 5.3 the information is referring to:

– *Dataset Size:* The term "data capacity" pertains to the overall quantity of data that can be processed by the search system.

– *Query Complexity:* This study examines the relationship between the performance of the system and the complexity of the query, specifically in terms of the number of keywords and the use of synonym mapping.

– *Response Time:* The temporal interval between the initiation of a search query and the subsequent presentation of search results to the user.

– *Accuracy:* The accuracy of the search outcomes in relation to the retrieval of pertinent texts.

Table 5.4: Results analysis

Metric	Test Condition 1	Test Condition 2	Test Condition 3
Dataset Size	500 GB	1 TB	5 TB
Query Complexity	3 keywords	5 keywords	7 keywords
Response Time	1.2 sec	1.8 sec	2.5 sec
Accuracy	92%	89%	85%
System Load	15% CPU	30% CPU	50% CPU
Network Latency	50 ms	70 ms	90 ms
Indexing Time	10 min	30 min	1 hr
Encryption Time	5 min	15 min	45 min
Decryption Time	2 min	6 min	18 min
Scalability	High	Moderate	Low
Fault Tolerance	99.99%	99.95%	99.9%

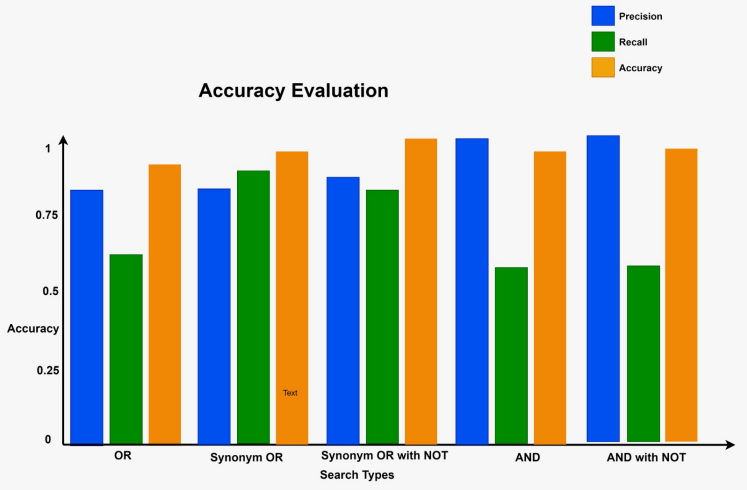

Fig. 5.2: Accuracy Evaluation

– *System Load:* The term ”computational load” refers to the amount of computational resources required by a system during the processing of queries.
– *Network Latency:* The duration required for data transmission over the network during the execution of a query.
– *Indexing Time:* The time necessary to index the data for search operations.
– *Encryption/Decryption Time:* The duration required for the encryption of data prior to its storage and the subsequent decryption process during retrieval.
– *Scalability:* The system's capacity to sustain performance levels while expanding in terms of data volume and user count.
– *Fault Tolerance:* The evaluation of the system's capacity to sustain functioning in the face of component failures.
– *Throughput:* The system's query processing capacity within a specified time interval.

Figure 5.2 is a bar chart depicting the “Accuracy Evaluation” of multiple search methods. The horizontal axis is comprised of six categories of search types: “OR,” “Synonym OR,” “Synonym OR with NOT,” “AND,” and “AND with NOT.” Each search category is depicted by three bars, which correlate to “Precision,” “Recall,” and “Accuracy,” respectively. The “OR” search type has a relatively high degree of precision and accuracy, but a comparatively lower level of recall. The retrieval rate of “Synonym OR” is somewhat higher than that of “OR”, however, its precision and accuracy are diminished. When the logical operator “OR” is used together with “NOT”, it causes a considerable decrease

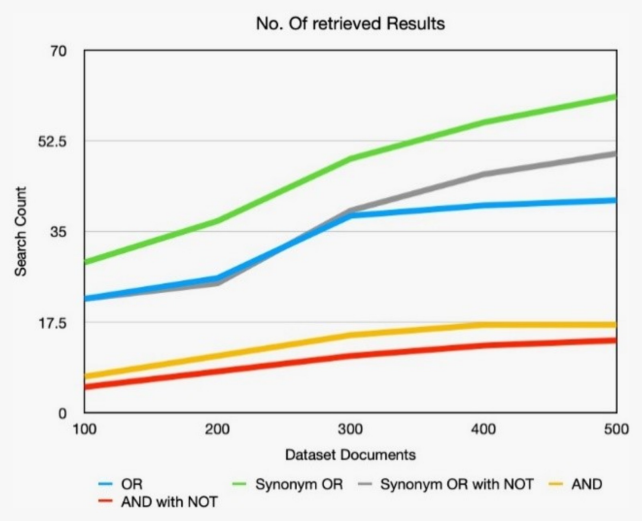

Fig. 5.3: No. of retrieval results AND

in performance across all three measurements, relative to the previous options. The "AND" search type demonstrates a significant increase in precision, a modest improvement in recall, but a decrease in accuracy compared to the "OR" search type. The search type "AND with NOT" exhibits superior precision, a modest recall, and the highest accuracy in comparison to other search types.

In the visual representation from Figure 5.3 is a line graph labelled "No. Of retrieved Results," illustrating the correlation between the quantity of search results obtained and the number of texts inside a given dataset. The dataset size on the x-axis spans from 100 to 500 texts. A comparison is made between five search techniques. The term "OR" is likely indicated by the blue line. The phrase "AND with NOT" may be represented by the red line. The expression "Synonym OR" potentially corresponds to the green line. The term "Synonym OR with NOT" may possibly be represented by the orange line. The term "AND" (perhaps referring to the grey line) As the quantity of texts grows, the quantity of retrieved results similarly increases for every search approach. The "Synonym OR" technique yields the highest number of results across different dataset sizes, constantly displaying an elevated line on the graph. The "AND" technique yields the smallest number of results, as demonstrated by the line that remains at the lowest point on the graph. The "OR" operator and the "AND with NOT" operator fall inside the intermediate range. The "OR" operator begins at a lower point than the "AND with NOT" operator, but surpasses it as the number of texts grows. The performance of "Synonym OR with NOT" is initially comparable to that of "AND with NOT", however it reaches a plateau and does not scale as well with the increasing number of texts.

Figure 5.4 shows a line graph with the title "Index Generation Time," where the horizontal axis represents file size in megabytes (mb), and the vertical axis shows the amount of time needed to generate an index in seconds. Four separate lines, each representing a different threshold and the amount of time it takes to create an index, are shown on the graph. The blue line shows how long it took to create the index using a 0.3 threshold. The green line may represent the Index Generation Time (With Threshold 0.4). The orange line shows how long it takes to generate the index, especially when a threshold of 0.5 is used. The grey line can be used to show how long it took to create the index with a 0.6 threshold. For all criteria, the index production time grows in proportion to the file size, from 1 megabyte to 5 megabytes. The line that shows the lowest position and the shortest generation time for the index is the one that corresponds to the criterion of 0.3. On the other hand, the line connected to the 0.6 barrier reaches its greatest point, indicating that it took the longest to generate the index. Between these two lines lie the thresholds 0.4 and 0.5, where 0.4 shows a faster speed than 0.5.

Figure 5.5 show the Search Time and search time in seconds. The horizontal axis reflects the number of files, ranging from 100 to 500, while the vertical axis displays the search duration in seconds. The graph compares the search times for four different categories of search strategies. The blue line is most likely indicative of the variable "Search Time (OR)". The green line may represent the "Search Time (Synonym OR)". The red line represents the concept of "Search Time (Synonym OR with Not)". The grey line represents the variable "Search Time (AND)" .The yellow line may symbolise the concept of "Search Time (AND with NOT)". The search time of all search algorithms is directly proportional to the

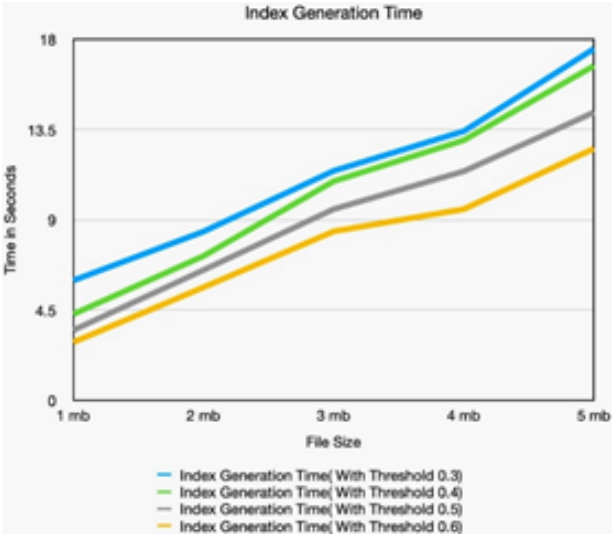

Fig. 5.4: Index generation time

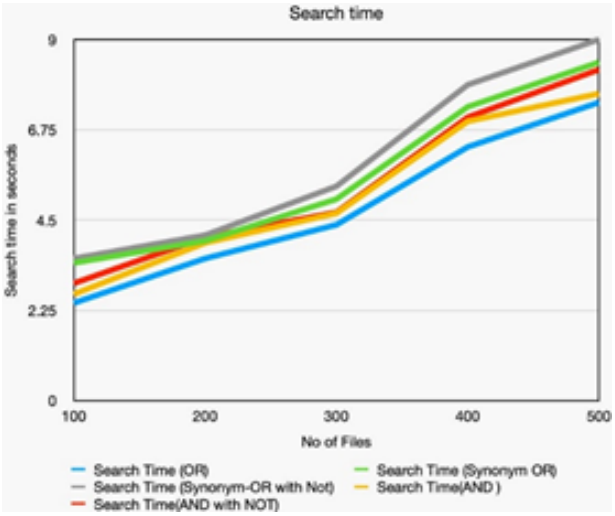

Fig. 5.5: Search Time and search time in seconds

amount of files. However, considering the close closeness of the lines, it seems that, within the range of file numbers displayed, there is minimal variance in search durations between both methods. Based on their positions at the top end of the graph, it appears that “Search Time (AND)” and “Search Time (AND with NOT)” require considerably more time compared to the other strategies. As the number of files approaches 500, there is a slight difference between the lines. An unsuccessful search occurs when a query yields no results. It’s crucial to have an effective approach for promptly indicating an unsuccessful search to conserve cloud resources. Current research lacks an examination of the time it takes to declare a search unsuccessful. Swiftly recognizing an unsuccessful search can alleviate financial costs for users; therefore, the time taken to declare a search unsuccessful is considered a key metric for performance evaluation. Under the current schemes [33, 42], a failed search is determined after a complete review of all text indices, requiring N comparisons to confirm that no relevant texts exist. Conversely, the proposed scheme utilizes a Cluster Head that embodies all the keywords within its cluster, allowing for the determination of the presence or absence of texts with search terms by reviewing only the cluster indices. Therefore, an unsuccessful search can be concluded after just K

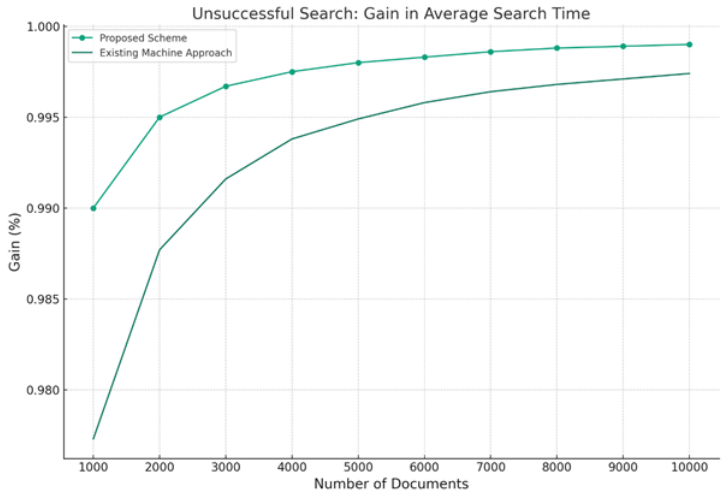

Fig. 5.6: Unsuccelful search: gain in Average search time

comparisons. The accompanying figure 5.6 illustrates the reduction in both the average number of comparisons and the average time required to determine an unsuccessful search. Compared to the previous schemes, the proposed method enhances the efficiency of declaring an unsuccessful search by 99.29%."

5.2. Computation Cost. In the proposed search scheme, the time required to construct the searchable index encompasses the duration to create both the text and the cluster indexes. The index build time for the proposed scheme tends to be greater than that of the existing scheme [33,42], which is attributed to the additional step of generating indexes for multiple clusters. The number of clusters, denoted by K, varies based on the application's needs, the size of the text collection, and the clustering algorithm employed. The incorporation of clustering into the index building process results in a slight increase in the time needed to create an index for a large text collection." For the graph, we assume we are illustrating the comparison of index build times between the proposed and existing schemes, highlighting the minor increase due to clustering. If you can provide any specific data or parameters you would like to include in the graph, please do so. Otherwise, we will proceed with a hypothetical representation. The average time required to construct a query, which encompasses HMAC computation, reduction, and bitwise-AND operations for each term, remains identical for both the proposed and the existing scheme [33,42], as the proposed scheme introduces no additional delays due to clustering. The average time to build queries, ranging from 1 to 5 genuine terms. , considering scenarios with and without the inclusion of noise terms. This timing is based on the mean time to generate 200 queries with a varying count of 1 to 5 genuine terms. To create a graph without specific values, we would plot the average query build time as it varies with the number of genuine terms for both scenarios (with and without noise terms). Since we don't have the specific values, we can create a hypothetical graph to illustrate this concept. Let's proceed to generate a graph that could represent this scenario see in the figure 5.7.

5.3. Rank Efficiency. The efficiency of result ranking is evaluated by comparing the time needed to generate per-text 'p' indexes at various relevance levels within the text collection. The increase in index build time associated with higher relevance levels is a one-time overhead, mitigated by the one-off nature of the indexing process conducted by the Data Owner (DO) during the offline stage. Cloud resources and parallel processing can be leveraged to further reduce this impact. Consequently, the extra time required for creating multiple indexes for each text is outweighed by the advantage of delivering superior ranked search results to the users. To visualize this concept, we can create a graph that demonstrates the efficiency of ranked search results without specifying exact values. Let's plot a graph showing the proportion of top-ranked texts from the proposed scheme that align with the top results from plain-text searches. We'll use hypothetical data to illustrate the concept described in figure 5.8.

6. Conclusion. The research conducted on the advancement in the field of secure recovery of data from cloud platforms. The proposed methodology integrates the resilience of greedy depth-first search algorithms with a sophisticated synonym identification system in order to offer accurate and efficient search functionalities across encrypted datasets. The method being described effectively addresses the challenges posed by synonymy and polysemy in search

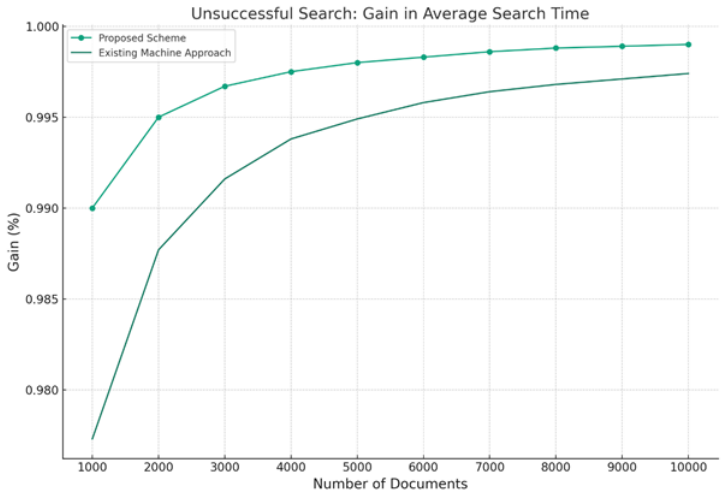

Fig. 5.7: Average query time by number of genuine term

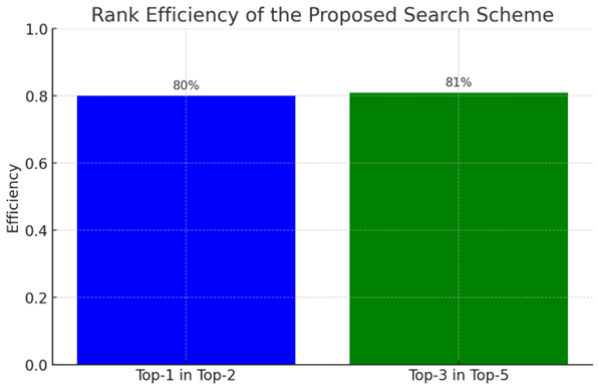

Fig. 5.8: rank efficiency of proposed search scheme

queries, thereby guaranteeing users access to comprehensive results that are not only pertinent to the specific terms employed but also to their semantic counterparts. The significance of this matter is particularly pronounced inside the realm of encrypted data, as conventional search methods are inadequate in light of the limits imposed by privacy preservation. the incorporation of a rating system inside the search process facilitates users in efficiently identifying the most relevant texts, hence augmenting the usability of cloud storage services. By implementing encryption techniques to handle privacy issues, while also ensuring a high degree of search accuracy and efficiency, the proposed method effectively fills a significant void in the utilisation of cloud data. The algorithm's efficacy, as evidenced by many performance measures, underscores its potential for extensive implementation in secure cloud-based applications. Given the escalating prevalence of cloud services, the concurrent rise in data privacy issues necessitates timely and crucial study to safeguard data security and accessibility in the future. Future research endeavours may further enhance this groundwork by delving into machine learning algorithms to achieve more refined synonym detection. Additionally, adaptive ranking techniques based on user feedback might be explored to optimise the system's performance. Furthermore, the scalability of the system should be investigated in light of the escalating demands for cloud storage. The continuous endeavour to achieve perfection in the development of search systems that are secure, efficient, and intelligent poses a persistent challenge. This research serves as a significant advancement in this ongoing goal.

REFERENCES

[1] P. Balamurugan, S. T. M, G. Arulkumaran and S. Jayagopalan, Multi-Keyword Graded Exploration in Encrypted Cloud Data for Industries Based on Rc4+ and Forest, *2023 1st International Conference on Innovations in High Speed Communication and Signal Processing (IHCSP), BHOPAL, India*, 2023, pp. 531-535, Doi: 10.1109/IHCSP56702.2023.10127116.

[2] C. Chi, Z. Yin, Y. Liu and S. Chai A Trusted Cloud–Edge Decision Architecture Based on Blockchain and MLP for AIoT, *IEEE Internet of Things Journal*, Vol. 11, no. 1, pp. 201-216, 2024, Doi:10.1109/JIOT.2023.3300845.

[3] L. Chen, Z. Chen, K. -K. R. Choo, C. -C. Chang and H. -M. Sun, Memory Leakage-Resilient Dynamic and Verifiable Multi-Keyword Ranked Search on Encrypted Smart Body Sensor Network Data, *IEEE Sensors Journal*, Vol. 19, no. 19, pp. 8468-8478, 1 Oct.1, 2019, Doi: 10.1109/JSEN.2018.2865550.

[4] H. Cui and X. Yi Secure Internet of Things in Cloud Computing via Puncturable Attribute-Based Encryption With User Revocation, *IEEE Internet of Things Journal*, Vol. 11, no. 2, pp. 3662-3670, 2024, Doi: 10.1109/JIOT.2023.3297997.

[5] D. Das, R. Amin and S. Kalra Algorithm for Multi Keyword Search Over Encrypted Data in Cloud Environment, *2020 International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus, 2020*, pp. 733-739, Doi:10.1109/IWCMC48107.2020.9148472.

[6] D. Das and S. Kalra An Efficient LSI Based Multi-keyword Ranked Search Algorithm on Encrypted Data in Cloud Environment, *2020 International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus*, 2020, pp. 1777-1782, Doi: 10.1109/IWCMC48107.2020.9148123.

[7] W. Dai et al. PRBFPT: A Practical Redactable Blockchain Framework With a Public Trapdoor, *IEEE Transactions on Information Forensics and Security*, Vol. 19, pp. 2425-2437, 2024, Doi: 10.1109/TIFS.2024.3349855.

[8] Z. Fu, F. Huang, K. Ren, J. Weng and C. Wang, Privacy-Preserving Smart Semantic Search Based on Conceptual Graphs Over Encrypted Outsourced Data, *IEEE Transactions on Information Forensics and Security*, Vol. 12, no. 8, pp. 1874-1884, Aug. 2017, Doi: 10.1109/TIFS.2017.2692728.

[9] 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 Transactions on Information Forensics and Security*, Vol. 11, no. 12, pp. 2706-2716, Dec. 2016, Doi: 10.1109/TIFS.2016.2596138.

[10] Z. Fu, K. Ren, J. Shu, X. Sun and F. Huang, Enabling Personalized Search over Encrypted Outsourced Data with Efficiency Improvement, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, no. 9, pp. 2546-2559, 1 Sept. 2016, Doi: 10.1109/TPDS.2015.2506573.

[11] Z. Fu, K. Ren, J. Shu, X. Sun and F. Huang, Enabling Personalized Search over Encrypted Outsourced Data with Efficiency Improvement, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, no. 9, pp. 2546-2559, 1 Sept. 2016, Doi:10.1109/TPDS.2015.2506573.

[12] Z. Fu, X. Sun, N. Linge and L. Zhou, Achieving effective cloud search services: multi-keyword ranked search over encrypted cloud data supporting synonym query, *IEEE Transactions on Consumer Electronics*, Vol. 60, no. 1, pp. 164-172, February 2014, Doi:10.1109/TCE.2014.6780939.

[13] J. Han, L. Qi and J. Zhuang Vector Sum Range Decision for Verifiable Multiuser Fuzzy Keyword Search in Cloud-Assisted IoT, *EEE Internet of Things Journal*, Vol. 11, no. 1, pp. 931-943, 2024, Doi:10.1109/JIOT.2023.3288276.

[14] H. He, J. Liu, J. Gu and F. Gao An Efficient Multi-Keyword Search Scheme over Encrypted Data in Multi-Cloud Environment, *2022 IEEE 7th International Conference on Smart Cloud (SmartCloud), Shanghai, China*, 2022, pp. 59-67, Doi: 10.1109/SmartCloud55982.2022.00016.

[15] C. Huang, D. Liu, A. Yang, R. Lu and X. Shen Multi-Client Secure and Efficient DPF-Based Keyword Search for Cloud Storage, *IEEE Transactions on Dependable and Secure Computing*, Vol. 21, no. 1, pp. 353-371, 2024, Doi: 10.1109/TDSC.2023.3253786.

[16] A. Hosseingholizadeh, F. Rahmati, M. Ali, H. Damadi and X. Liu Privacy-Preserving Joint Data and Function Homomorphic Encryption for Cloud Software Services *IEEE Internet of Things Journal*, vol. 11, no. 1, pp. 728-741, 2024, Doi: 10.1109/JIOT.2023.3286508.

[17] B. Lang, J. Wang, M. Li and Y. Liu, Semantic-based Compound Keyword Search over Encrypted Cloud Data, *IEEE Transactions on Services Computing*, Vol. 14, no. 3, pp. 850-863, 2021, Doi:10.1109/TSC.2018.2847318.

[18] J. Li, J. Ma, Y. Miao, R. Yang, X. Liu and K. -K. R. Choo Practical Multi-Keyword Ranked Search With Access Control Over Encrypted Cloud Data, *IEEE Transactions on Cloud Computing*, Vol. 10, no. 3, pp. 2005-2019, 2022, Doi: 10.1109/TCC.2020.3024226.

[19] X. Liu, G. Yang, W. Susilo, J. Tonien, X. Liu and J. Shen, Privacy-Preserving Multi-Keyword Searchable Encryption for Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, no. 3, pp. 561-574, 1 March 2021, Doi: 10.1109/TPDS.2020.3027003.

[20] Y. Miao, Y. Yang, X. Li, L. Wei, Z. Liu and R. H. Deng Efficient Privacy-Preserving Spatial Data Query in Cloud Computing, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 36, no. 1, pp. 122-136, 2024, Doi: 10.1109/TKDE.2023.3283020.

[21] B. M. Nguyen et al. A Novel Nature-Inspired Algorithm for Optimal Task Scheduling in Fog–Cloud Blockchain System, *IEEE Internet of Things Journal*, Vol. 11, no. 2, pp. 2043-2057, 2024, Doi:10.1109/JIOT.2023.3292872.

[22] P. Pandiaraja and P. Vijayakumar, Efficient Multi-keyword Search over Encrypted Data in Untrusted Cloud Environment, *2017 Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM), Tindivanam, India*, 2017, pp. 251-256, Doi:10.1109/ICRTCCM.2017.54.

[23] S. Prakash, N. Andola and S. Venkatesan, Secure access of multiple keywords over encrypted data in cloud environment using ECC-PKI and ECC ElGamal, *2017 International Conference on Public Key Infrastructure and its Applications (PKIA), Bangalore, India*, 2017, pp. 49-56, Doi:10.1109/PKIA.2017.8278960.

[24] V. Saiharitha and S. J. Saritha, A privacy and dynamic multi-keyword ranked search scheme over cloud data encrypted, *2016 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India*, 2016, pp. 1-5, Doi: 10.1109/CESYS.2016.7890001.

[25] V. Saiharitha and S. J. Saritha, A privacy and dynamic multi-keyword ranked search scheme over cloud data encrypted, *2016 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India*, 2016, pp. 1-5, Doi:10.1109/CESYS.2016.7890001.

[26] S. Shete and N. Dongre, Ranked multi-keyword search data using cloud, *2017 International Conference on Inventive Computing and Informatics (ICICI), Coimbatore, India*, 2017, pp. 590-594, Doi:10.1109/ICICI.2017.8365200.

[27] M. Song, Z. Hua, Y. Zheng, H. Huang and X. Jia LSDedup: Layered Secure Deduplication for Cloud Storage, *IEEE Transactions on Computers*, vol. 73, no. 2, pp. 422-435, 2024, Doi: 10.1109/TC.2023.3331953.

[28] N. Wang, W. Zhou, J. Wang, Y. Guo, J. Fu and J. Liu Secure and Efficient Similarity Retrieval in Cloud Computing Based on Homomorphic Encryption *IEEE Transactions on Information Forensics and Security*, Vol. 19, pp. 2454-2469, 2024, Doi: 10.1109/TIFS.2024.3350909.

[29] Z. Xia, Q. Gu, W. Zhou, L. Xiong, J. Weng and N. Xiong STR: Secure Computation on Additive Shares Using the Share-Transform-Reveal Strategy, *IEEE Transactions on Computers*, Vol. 73, no. 2, pp. 340-352, 2024, Doi:10.1109/TC.2021.3073171.

[30] Z. Xia, X. Wang, X. Sun and Q. Wang, A Secure and Dynamic Multi-Keyword Ranked Search Scheme over Encrypted Cloud Data, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, no. 2, pp. 340-352, 1 Feb. 2016, Doi: 10.1109/TPDS.2015.2401003.

[31] D. Xu, C. Peng, W. Wang, K. Dev, S. A. Khowaja and Y. Tian, Multi-keyword Ranked Search Scheme Supporting Extreme Environments for the Internet of Vehicles, *IEEE Internet of Things Journal*, Doi:10.1109/JIOT.2023.3275386.

[32] Z. Yang et al. Differentially Private Federated Tensor Completion for Cloud–Edge Collaborative AIoT Data Prediction, *IEEE Internet of Things Journal*, Vol. 11, no. 1, pp. 256-267, 2024, Doi:10.1109/JIOT.2023.3314460.

[33] X. Yang, G. Chen, M. Wang, T. Li and C. Wang, Multi-Keyword Certificateless Searchable Public Key Authenticated Encryption Scheme Based on Blockchain, *IEEE Access*, Vol. 8, pp. 158765-158777, 2020, Doi: 10.1109/ACCESS.2020.3020841.

[34] H. Yin et al., Secure Conjunctive Multi-Keyword Search for Multiple Data Owners in Cloud Computing, *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, China*, 2016, pp. 761-768, Doi: 10.1109/ICPADS.2016.0104.

[35] H. Yin et al., Secure Conjunctive Multi-Keyword Search for Multiple Data Owners in Cloud Computing, *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, China*, 2016, pp. 761-768, Doi:10.1109/ICPADS.2016.0104.

[36] Y. Zhang, C. Jiang and P. Zhang Security-Aware Resource Allocation Scheme Based on DRL in Cloud–Edge–Terminal Cooperative Vehicular Network, em IEEE Internet of Things Journal, Vol. 11, no. 1, pp. 95-104, 2024, Doi: 10.1109/JIOT.2023.3293497.

[37] X. Zhou, D. He, J. Ning, M. Luo and X. Huang Single-Server Public-Key Authenticated Encryption With Keyword Search and Its Application in IIoT, *IEEE Transactions on Network Science and Engineering*, Vol. 11, no. 1, pp. 404-415, 2024, Doi: 10.1109/TNSE.2023.3300716.

Edited by: SCPE Editor-in-chief
Regular paper
Received: Feb 3, 2024
Accepted: Dec 13, 2024