



A privacy-preserving framework for compressed pattern matching over encrypted data using lightweight XOR-based encryption and bit-parallel processing

Rajesh Prasad^{a,b}, Shashank Sahu^a, Anu Chaudhary^a, Avdhesh Gupta^a, Muhammad Musa Liman^{b,c,*}

^aDepartment of Computer Science and Engineering, Ajay Kumar Garg Engineering College, Ghaziabad, India

^bDepartment of Computer Science, African University of Science and Technology, Abuja, Nigeria

^cDepartment of Computer Science, Federal University of Lafia, Nigeria

Abstract

With the tremendous growth of cloud computing, cyber threat intelligence systems, and compressed big-data storage, efficient and secure compressed pattern-matching techniques are in high demand. Traditional compressed pattern-matching methods mainly focus on computational efficiency and do not consider privacy preservation or pattern search over encrypted patterns. We propose a novel privacy-preserving compressed pattern-matching framework, the secure bit-parallel compressor (Secure BIT_COMP), which integrates word-based tagged code (WBTC), a bit-parallel Shift-Or algorithm, encrypted compressed search, and encrypted bit-level operations using a lightweight exclusive-OR (XOR)-based prototype encryption mechanism. The proposed methodology enables direct search without decompressing the text or revealing plaintext in encrypted and compressed text streams. Secure BIT_COMP uses bit-parallel matching and encrypted state transitions to search efficiently without compromising confidentiality. Furthermore, a secure verification invariant based on encrypted boundary validation helps prevent false positives. According to the theory, the proposed encrypted-compressed search framework achieves machine-word-level complexity for privacy-preserving search. Experiments on textual datasets show that the proposed method is more accurate than existing WBTC and tagged Huffman approaches and reduces memory usage while supporting secure matching. The proposed framework extends compressed pattern matching to secure cloud computing, encrypted genomic analysis, digital forensics, and privacy-preserving cybersecurity applications.

DOI: [10.46481/jnsps.2027.3637](https://doi.org/10.46481/jnsps.2027.3637)

Keywords: Compressed pattern matching, Privacy-preserving search, Bit-parallelism, WBTC, Encrypted search

Article History:

Received: 23 June 2026

Received in revised form: 13 July 2026

Accepted for publication: 20 September 2026

Available online: 30 September 2026

© 2027 The Author(s). Published by the [Nigerian Society of Physical Sciences](#) under the terms of the [Creative Commons Attribution 4.0 International license](#). Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

Communicated by: B. J. Falaye

1. Introduction

Cloud computing, Internet of Things (IoT) platforms, cybersecurity monitoring systems, social networks, and large-scale digital repositories have led to unprecedented amounts of

data being created and stored each day [1]. It is now a critical challenge for modern information systems to store and retrieve such data efficiently. Data compression methods are used in many applications to save space and bandwidth in communicating data by encoding it in more compact formats. In most cases, however, traditional search approaches require decompressing data prior to pattern matching, which introduces additional computational and memory overhead and limits scalability

*Corresponding Author Tel. No.: +234-704-959-3359.

Email address: mliman@student.aust.edu.ng (Muhammad Musa

ity when dealing with large datasets [2, 3].

To address these disadvantages, compressed pattern matching (CPM), which allows searching in compressed storage without decompression, has become a critical research area. CPM techniques are useful in applications such as cloud storage systems, genomic repositories, digital libraries, cyber threat intelligence platforms, and large-scale information retrieval systems, where they can improve retrieval efficiency and reduce processing costs [4]. Word-based tagged code (WBTC) is one approach proposed for efficient compression of text streams; it preserves information about word boundaries, allowing direct search in the compressed stream [5]. Furthermore, bit-parallel algorithms such as Shift-Or and Shift-And operate on machine words to test several pattern positions simultaneously, increasing search speed.

However, several significant issues remain unaddressed. Current compressed pattern-matching methods primarily operate on plaintext data and offer only partial security for sensitive data in outsourced and cloud-based settings. Meanwhile, searchable encryption and homomorphic computation have enabled secure operations over encrypted data; most current solutions are only for keyword retrieval and are inefficient in the compressed domain for pattern-matching operations. Thus, existing solutions do not offer a single approach that addresses all of compression efficiency, fast pattern matching, privacy preservation, and secure computation [6, 7].

This paper aims to address these research gaps by designing a privacy-preserving compressed pattern-matching framework, named Secure BIT_COMP, which combines WBTC compression, bit-parallel Shift-Or pattern matching, encrypted compressed search, homomorphic bit operations, and secure verification [8, 9]. The framework allows direct search of encrypted streams of compressed text without exposing plaintext or decompressing the stream. Furthermore, a secure verification process verifies matches and excludes false matches while maintaining data confidentiality.

1.1. Novelty and research contribution

Although Secure BIT_COMP incorporates known ideas, including WBTC compression, Shift-Or pattern matching, searchable encryption, and homomorphic computation, it does more than simply integrate these ideas; it brings them together in a single framework. Existing compressed pattern-matching algorithms are designed to process compressed data without privacy protection, whereas existing searchable-encryption frameworks are primarily designed to retrieve specific keywords from encrypted indexes rather than support compressed-domain bit-parallel pattern matching. Secure BIT_COMP introduces a compressed-matching model over encrypted data streams that enables direct bit-parallel pattern matching over encrypted, compressed data without decompression. Specifically, the framework incorporates the following elements.

Encrypted bit-parallel state transition mechanism: A bit-parallel Shift-Or extension that performs state transitions on encrypted compressed representations using lightweight bitwise encryption.

Secure verification invariant: A verification mechanism that guarantees that candidate matches generated in the compression domain are correct and discards false positives without revealing plaintext information.

Unified compression–encryption–matching architecture: A single architecture that supports bit-parallel search acceleration, privacy preservation, and compression simultaneously.

Encrypted compressed-domain pattern matching: Secure BIT_COMP operates on compressed, encrypted repositories without following the conventional searchable-encryption approach of performing keyword searches over an encrypted index.

These contributions establish a link between compressed pattern matching and privacy-preserving search, enabling efficient and secure pattern retrieval from large compressed datasets.

Section 2 surveys the literature on compressed pattern matching, searchable encryption, and homomorphic computation. Section 3 presents the proposed Secure BIT_COMP approach. Section 4 presents the experimental evaluation and comparative analysis. Section 5 concludes the paper and provides recommendations for future research.

2. Related work

Compressed pattern matching (CPM) is a major research area focused on efficient pattern matching over very large compressed datasets without full decompression. Traditional algorithms such as Knuth–Morris–Pratt (KMP), Boyer–Moore (BM), and Rabin–Karp (RK) are efficient on uncompressed text but inefficient on compressed text. Techniques based on Huffman coding, LZ compression, the Burrows–Wheeler Transform (BWT), and WBTC have improved compressed searching, with WBTC providing direct searchability by preserving word boundaries. However, problems such as false positives, verification overhead, and scalability remain. Bit-parallel algorithms and searchable encryption can improve efficiency and security. Secure BIT_COMP addresses this gap by combining WBTC compression, encrypted Shift-Or matching, homomorphic operations, and secure verification.

2.1. Introduction to compressed pattern matching

With the abundance of data generated by systems such as cloud computing, genomic databases, cybersecurity platforms, social media, and the Internet of Things (IoT) [10], affordable and efficient data storage and retrieval are increasingly important. Conventional search methods may require data to be decompressed before pattern matching, creating computational and memory overhead. Data-compression techniques reduce storage and transmission costs. To overcome the cost of decompression, compressed pattern matching (CPM) enables direct search over compressed data and provides efficient, scalable retrieval [11].

Since then, several CPM schemes have been developed, based on compression techniques such as Huffman coding, Lempel–Ziv (LZ) coding [12], the Burrows–Wheeler Transform (BWT), grammar-based compression, and Word-Based

Tagged Code (WBTC) [13]. These techniques do not fully address boundary ambiguity, false-positive matches, verification costs, or scalability limitations. More recently, bit-parallel algorithms such as Shift-Or and Shift-And have accelerated pattern matching. Until now, however, most existing techniques have required plaintext compression and have not offered adequate privacy protection [14].

The use of cloud-based storage solutions has increased in popularity, making it essential to search encrypted and compressed data for secure searches. Despite some advances in searchable encryption and homomorphic computation, very few research papers have integrated compression with bit-parallel processing, encrypted search, and secure verification into a single framework, thereby motivating the development of more elaborate privacy-preserving compressed pattern-matching solutions.

2.2. Early research in pattern matching

In computer science, pattern matching is a crucial technique used in applications including information retrieval, text processing, computational biology, cybersecurity, data mining, and digital forensics [15, 16]. The basic goal is to locate occurrences of a pattern within a longer text using as few computations as possible. The early Naive String Matching algorithm compares the pattern with the text at each possible starting position, making it computationally expensive [17, 18].

To improve efficiency, several exact string-matching algorithms were developed. The KMP algorithm also preprocesses the text to avoid unnecessary comparisons using prefix information, so that the time complexity of the algorithm is linear [19]. The bad-character and good-suffix heuristics added by the BM algorithm enabled larger shifts in the event of mismatches, resulting in a more efficient practical search solution. Similarly, RK uses hashing and is well-suited for searching multiple patterns in a text and for large-scale text retrieval [20].

As the volume of data increased, researchers found that these algorithms were primarily developed for uncompressed text and would need to be fully decompressed when applied to compressed data, adding processing overhead. This challenge led to compressed pattern-matching techniques that process data without decompression. Furthermore, it was found that bit-parallel approaches that compare and manipulate multiple positions in a pattern at the machine-word level significantly increased the speed of pattern searches and paved the way for valuable high-performance pattern-matching frameworks [21, 22].

2.3. Compressed pattern matching techniques

In CPM, pattern matching can be performed directly on compressed data without uncompressing the entire dataset, thereby reducing computational overhead, memory usage, and storage access costs. CPM has played a vital role in recent years, especially in applications such as cybersecurity analysis, cloud storage systems, digital libraries, very large information retrieval systems, and genomic databases and information analysis [23].

Early studies in CPM used Huffman-coded data, for which search operations were performed directly on the compressed streams themselves using codeword structures. Huffman coding uses variable-length codes, which can introduce boundary ambiguities and false-positive matches, while reducing decompression cost. Later, Lempel-Ziv (LZ)-based compression methods became more effective at compression but more complex because of recursive dictionary expansion. In addition, efficient indexing for fast retrieval in text and genomic applications using compact indexing structures, such as the Burrows-Wheeler Transform (BWT) and FM-index techniques [24], has further advanced compressed searching. It was also possible to perform direct searches over compressed representations for grammar-based compression methods, but in some cases these methods required complicated preprocessing procedures.

In particular, for word-boundary preservation, WBTC has strong potential to achieve high compression ratios. This feature helps to make searching more accurate and efficient. Despite these advancements, there are still many problems with current CPM techniques, including false-positives, the time overhead required to verify matches, scalability, and an inability to search under privacy constraints, which have led to the development of privacy-preserving compressed pattern-matching frameworks [25].

2.4. Word-based tagged code (WBTC)

A compressed text representation scheme is called WBTC if it achieves both high compression efficiency and direct searchability over the compressed data. In contrast to character-based compression, WBTC uses words as the smallest encoding units and assigns variable-length codewords to them based on their frequency. Shorter codewords are assigned to common words, while longer codewords are assigned to less common words. This is achieved by leveraging the natural word-frequency characteristics of the text datasets, resulting in significant compression.

To create their efficient WBTC compression tool for dynamic data, the authors of Ref. [23] assume that the word is the basic compression unit. While maintaining a reasonable compression ratio, the code preserves all the characteristics of the labeled sub-optimal code.

The words are assigned codes in the following four steps:

1. The first $2p$ ($p=1$, rank = 0, 1) words in the lexicon are given the codes 01 and 10.
2. For $p=2$, the following $2p$ words are created by prefixing all of the codes from the previous level with 00 and 11.
3. In general, for each level and any value of p , the vocabulary's following $2p$ words are given codes using $2p$ bits by prefixing all of the codes from earlier levels with 00 and 11.
4. Steps 1-3 are repeated until all of the words of the text have been encoded.

WBTC has a unique advantage in its use of "tagged" codewords that explicitly preserve word boundaries in the

compressed stream. In conventional Huffman coding, code-words are concatenated without boundaries, which makes direct search difficult and may require verification or decoding. WBTC addresses this issue by incorporating boundary information into the code-assignment procedure, enabling each word to be identified efficiently in the compressed representation. Consequently, searches can be performed without complete decompression, saving computation and memory. WBTC enables direct search without decompression, preserves word boundaries, eliminates ambiguities, reduces storage requirements relative to uncompressed text, and supports large-scale text repositories and cloud-based storage systems.

Pattern-matching methods based on WBTC are beneficial but have drawbacks. Candidate matches generated in the compressed domain may require further verification, increasing processing overhead. Furthermore, most existing techniques use sequential matching, which can reduce throughput for large datasets. Another important limitation is restricted support for privacy preservation; in many approaches, compressed information remains in plaintext during the search process, complicating privacy protection in applications such as cloud-based medical records, genomic records, cybersecurity logs, and confidential enterprise databases.

In recent years, the direction of compressed searching research has shifted to incorporating WBTC into efficient matching algorithms and safe computation methods. Specifically, a bit-parallel algorithm such as Shift-Or can efficiently evaluate multiple matching states in parallel using machine-word operations. To date, however, the interplay among WBTC compression, encrypted search, homomorphic computation, and secure verification within a single framework has been neglected.

2.5. Bit-parallel pattern matching

Bit-parallel pattern matching is one of the most efficient methods, leveraging machine-word-level parallelism to speed up pattern matching. String matching methods operate on one character of a pattern at a time. In contrast, bit-parallel methods store multiple pattern positions in the bits of a machine word and perform multiple comparisons simultaneously using bitwise operations. This can substantially reduce runtime and improve computational efficiency.

This idea of bit-parallel matching was established with the development of the Shift-Or and the Shift-And algorithms by Baeza-Yates and Gonnet. These algorithms implement finite-state automata using binary vectors, where each bit represents the position of a pattern. Matching states are updated via fast bitwise operations (AND, OR, bit shifting) during text scanning. The state vector is updated by the Shift-Or algorithm as follows: $D = (D \ll 1) \mid B[c]$ (where the bitmask, $B[c]$, is for the current symbol in the input stream). A match is found if the last (rightmost) pattern bit meets the matching condition. There are algorithms based on bit-parallelism that have been successfully used in information retrieval, biological sequence analysis, malware detection, and network security.

Most approaches, however, operate on plaintext data and lack support for compression and/or encryption during

search. Therefore, integrating bit-parallel pattern matching with privacy-preserving computation remains a significant research challenge. Ref. [5] introduced the original BIT_COMP framework, a plaintext compressed pattern-matching method based on bit-parallel Shift-Or processing. The Secure BIT_COMP framework proposed in this work extends BIT_COMP by incorporating encrypted compressed-domain search, homomorphic bit operations, and secure verification mechanisms for privacy-preserving pattern matching. Table 1 summarizes the major compressed pattern-matching techniques and their characteristics.

2.6. Secure search and searchable encryption

As cloud computing and storage are increasingly outsourced, there is an urgent need for search tools that guarantee the integrity and confidentiality of content retrieved from the cloud. Traditional search methods require data in plaintext, which creates privacy concerns when data are stored on untrusted servers. To overcome this, a new cryptographic method known as searchable encryption (SE) has proven effective.

The three main searchable-encryption schemes are symmetric searchable encryption (SSE), public-key encryption with keyword search (PEKS), and predicate encryption. PEKS supports secure keyword search in a public-key environment, whereas SSE provides efficient keyword search in a shared-secret-key environment. Predicate encryption can support more complex queries involving user-defined attributes and access policies. These techniques allow data to remain confidential while supporting selective retrieval from encrypted data. More recent homomorphic search techniques perform computations directly on encrypted data. Such techniques offer strong privacy properties but can incur high computational and storage costs [26].

While promising progress has been made, existing schemes for searchable encryption generally focus on keyword-based retrieval and rely on auxiliary structures and/or encrypted indexes. Few studies have examined the combination of searchable encryption, compressed pattern matching, and bit-parallel processing. Developing efficient search methods that do not compromise the security of encrypted and compressed data, while preserving the privacy of the original data, is an important research challenge that the proposed Secure BIT_COMP framework addresses.

2.7. Homomorphic encryption and secure computation

Homomorphic encryption represented a major advance in the field of secure outsourced computation. Craig Gentry introduced the first fully homomorphic encryption (FHE) method, which allows arbitrary computations to be performed directly on encrypted data. Homomorphic encryption enables operations such as: $E(a) \otimes E(b) = E(a \circ b)$, where: $E(a)$ and $E(b)$ denote encrypted values, $\otimes E(c)$ denotes encrypted computation, $\circ$ denotes plaintext operation.

Some applications of homomorphic encryption include secure cloud analytics, secure databases, privacy-preserving machine learning, and secure genomic analysis. Fully homomor-

Table 1: Major compressed pattern-matching techniques.

S/N	Compression Technique	Main Characteristics	Major Limitation
1	Huffman Coding	Variable-length encoding	Boundary ambiguity and false matches
2	Tagged Huffman	Boundary-aware encoding	Verification overhead
3	Lempel–Ziv (LZ)	Dictionary-based compression	Complex random-access searching
4	Burrows–Wheeler Transform (BWT)	Compressed indexing and fast retrieval	High preprocessing cost
5	Grammar-Based Compression	Rule-based representation of repetitive data	Complex matching operations
6	Word-Based Tagged Code (WBTC)	Word-oriented searchable compression	Candidate verification overhead
7	Bit-Parallel CPM	Fast machine-word-level processing	Typically limited to plaintext data

phic encryption (FHE) is often computationally intensive, despite the security guarantees. To this end, lightweight partial homomorphic and partially homomorphic schemes were suggested to facilitate implementations. There are an increasing number of studies on compressed-bit parallel pattern matching, but there has been little work on homomorphic operations beyond it [27].

2.8. Research gap and motivation

Research has made substantial advances in compressed pattern matching, bit-parallel algorithms, searchable encryption, and XOR-based computation. CPM techniques provide more efficient storage and allow matching without full decompression; bit-parallel techniques such as Shift-Or accelerate searching using machine-word-level operations. Likewise, searchable encryption and lightweight XOR-based encryption can support secure data storage and computation.

These improvements are still largely achieved through isolated solutions. Most CPM techniques rely on plaintext information and do not apply to privacy-preserving search. These issues commonly plague Huffman- and WBTC-based methods, which require further verification. In searchable encryption techniques, the primary focus is keyword search and may require additional indexes, thus increasing storage costs. Moreover, little work has combined compressed pattern matching and bit-parallel processing with lightweight XOR-based encryption and bit operations.

This has created a need for research into the integration of compressed searching, encrypted computation, bit-parallel processing, secure verification, and preserving privacy. The central challenge is to enable efficient and secure searching over encrypted and compressed information, which is at the core of the proposed Secure BIT_COMP framework.

3. Proposed Secure BIT_COMP methodology

The current Secure BIT_COMP employs a lightweight XOR-based transformation of compressed bitstreams to demonstrate privacy-aware compressed-domain pattern matching. Unlike a fully homomorphic encryption framework, the

transformed representation preserves the bitwise structure required by the Shift-Or algorithm, allowing standard bit-parallel operations to be applied directly to transformed data.

Consequently, Algorithm 1 should be interpreted as operating on transformed compressed representations rather than semantically secure ciphertexts. The operations performed include bitmask construction, Shift-Or state updates, and secure verification over XOR-transformed compressed streams. The framework demonstrates the feasibility of privacy-aware compressed pattern matching but does not constitute a homomorphic encrypted-domain computation protocol.

The algorithm proceeds as follows:

A WBTC dictionary is first built from the text dataset. With this dictionary, the input text and the search pattern are converted to compressed binary representations. The original text stream is compressed, followed by compression of the pattern; both of these are transformed with a secret key, so that neither the original data nor the search query is exposed to the search server.

Let $D(w)$ denote the WBTC code assigned to word w ; then the compressed text is

$$T' = \bigoplus_{i=1}^n D(w_i),$$

where n is the number of codewords.

The encrypted pattern produces an encrypted bitmask table. The bitmask table records how often each bit in the encrypted data is used and serves as the preprocessing structure required for the Shift-Or matching algorithm. This preprocessing step improves the efficiency of bit-parallel pattern matching through machine-word operations.

The algorithm starts with a state vector consisting entirely of 1s. Each encrypted bit of the compressed text stream is scanned one by one during the scanning phase. Shift-Or operations are used to update the state vector: a left-shift operation followed by a bitwise OR operation with the appropriate transformed bitmask. The operations perform parallel matching of all pattern positions at once.

Let $B[c]$ be the bitmask associated with encrypted bit c . The state vector is updated using Eq. (1):

$$D_k = (D_{k-1} \ll 1) \vee B[c_k]. \quad (1)$$

Whenever the most significant pattern bit of the state vector becomes zero, a candidate match is detected. However, because compressed variable-length codes may occasionally produce invalid alignments, each candidate occurrence undergoes a secure verification procedure. The verification phase validates transformed-pattern equality and checks WBTC code-word boundaries to eliminate false positives without exposing any plaintext information.

A candidate match is detected whenever the most significant pattern bit becomes zero, equivalently, $(D_k \& 2^{m-1}) = 0$, where m is the pattern length. Suppose the pattern length in compressed bits is $|P'| = L$. If a match is detected after processing bit k , the starting bit position is

$$b = k - L + 1. \quad (2)$$

Let $s_i = \sum_{j=1}^{i-1} |D(w_j)|$ be the starting bit offset of word w_i . A candidate bit position b corresponds to word position i if $s_i + 1 = b$.

Only candidates that satisfy all verification conditions are reported as valid matches. Consequently, the proposed Secure BIT_COMP framework enables direct transformed compressed-domain searching, privacy preservation, efficient bit-parallel processing, and false-positive-free pattern matching.

Algorithm 1 presents the Secure BIT_COMP framework for privacy-preserving compressed pattern matching.

Algorithm 1: Secure BIT_COMP

```

1: Input: text  $T$ , pattern  $P$ , secret key  $K$ 
2: Output: verified match positions
3: Construct WBTC dictionary  $D$ 
4:  $T' \leftarrow \text{WBTC\_Encode}(T)$ 
5:  $P' \leftarrow \text{WBTC\_Encode}(P)$ 
6:  $\text{Trans}T \leftarrow \text{XOR\_Transform}(T', K)$ 
7:  $\text{Trans}P \leftarrow \text{XOR\_Transform}(P', K)$ 
8: Construct transformed bitmask table  $B$ 
9: for each bit  $b \in \{0, 1\}$  do
10:    $B[b] \leftarrow$  all 1s
11: end for
12: for  $i \leftarrow 1$  to  $|\text{Enc}P|$  do
13:    $B[\text{Enc}P[i]] \leftarrow B[\text{Enc}P[i]] \text{ AND } \sim (1 \ll (i - 1))$ 
14: end for
15:  $D_{\text{state}} \leftarrow$  all 1s
16: for each encrypted bit  $c$  in  $\text{Enc}T$  do
17:    $D_{\text{state}} \leftarrow (D_{\text{state}} \ll 1) \text{ OR } B[c]$ 
18:   if the most significant bit of  $D_{\text{state}} = 0$  then
19:      $\text{CandidateMatch} \leftarrow$  current position
20:     if  $\text{Secure\_Verification}()$  then
21:       Report match
22:     end if
23:   end if
24: end for
25: Return all verified match positions

```

Example 1 demonstrates Secure BIT_COMP matching directly using transformed-domain bit-parallel Shift-Or processing.

Table 2: Code assignment using WBTC.

S/N	Rank	Word	Frequency	Code
1	0	rose	3	01
2	1	is	3	10
3	2	red	2	0001
4	3	beautiful	1	0010

Table 3: Mapping bit positions to word positions in WBTC code.

S/N	Word Position i	Word	Code	Start Bit Si+1
1	1	rose	01	1
2	2	is	10	3
3	3	red	0001	5
4	4	rose	01	9
5	5	is	10	11
6	6	beautiful	0010	13
7	7	rose	01	17
8	8	is	10	19
9	9	red	0001	21

Example 1

Step 1: Consider the text: "rose is red rose is beautiful rose is red"

Step 2: The WBTC dictionary obtained by applying the algorithm is shown in Table 2. The mapping from bit positions to actual word positions is shown in Table 3.

Compressed text stream:

$T' = 01\ 10\ 0001\ 01\ 10\ 0010\ 01\ 10\ 0001$

Step 3: Suppose the user searches for the pattern $P = \text{"rose is"}$

Compressed pattern P' is: 0110

Step 4: Encrypt the compressed stream

Assume a lightweight XOR-based transformation with secret key $K = 1010$. The XOR-based mechanism used in this example is intended solely to illustrate transformed-domain processing and should not be interpreted as providing production-grade cryptographic security.

Split compressed text T' into 4-bit blocks: 0110 | 0001 | 0110 | 0010 | 0110 | 0001

Encryption process:

Block 1: $0110 \oplus 1010 = 1100$

Block 2: $0001 \oplus 1010 = 1011$

Block 3: $0110 \oplus 1010 = 1100$

Block 4: $0010 \oplus 1010 = 1000$

Block 5: $0110 \oplus 1010 = 1100$

Block 6: $0001 \oplus 1010 = 1011$

Encrypted compressed text: $\text{Enc}(T') = 1100\ 1011\ 1100\ 1000\ 1100\ 1011$

Step 5: Encrypt the pattern

Compressed pattern: 0110

Encrypt: $0110 \oplus 1010 = 1100$

Encrypted pattern: $\text{Enc}(P') = 1100$

Step 6: Construct the bitmask table for Enc(P') using Section 2.5

$B[0] = 0011, \quad B[1] = 1100.$

Step 7 (matching process):

The detailed matching process is discussed in Table 4.

Using Eq. (2), candidate bit positions are {1, 9, 17}. Using Table 4, the actual match positions are {1, 4, 7}. The Secure BIT_COMP framework successfully identifies all occurrences of the pattern "rose is" directly over the transformed WBTC-compressed text stream without decompression while preserving privacy and eliminating false positives.

4. Experimental results and performance evaluation

All experiments were conducted on a workstation configured as follows: processor: Intel Core i7-12700K (3.60 GHz), memory: 32 GB DDR4 RAM, operating system: Ubuntu 22.04 LTS (64-bit), programming language: C++17, compiler: GCC 11.4.0. Each experiment was executed thirty times, and average values are reported.

The proposed Secure BIT_COMP prototype employs a lightweight XOR-based encryption mechanism designed to evaluate secure compressed-domain pattern matching over encoded text representations. Encryption was implemented using a custom C++ module developed by the authors. For each compressed bit block B, the transformation is performed as $C = B \text{ XOR } K$, where B denotes the compressed bit block, K represents a pseudorandom key stream, and C is the encrypted output. The implementation uses a 128-bit key, a 128-bit block size, the Mersenne Twister (MT19937) pseudorandom generator, ciphertext of the same size as the plaintext, and encryption overhead linear in the block length.

All experiments were performed on an Intel Core i7 processor with 32 GB RAM running Ubuntu 22.04. The proposed Secure BIT_COMP framework was implemented in C++17 using GCC 11.4.0. Unlike fully homomorphic encryption systems, the current prototype employs a lightweight XOR-based encryption mechanism with a 128-bit key for protecting compressed bit streams during search operations.

Experiments were conducted on the benchmark datasets discussed in Section 4.1. To ensure an unbiased evaluation, pattern lengths of 4, 8, 16, 32, and 64 characters were selected, with 1,000 queries for each length category and 5,000 queries per dataset. A fixed random seed was used for reproducibility. Patterns were selected uniformly from valid substrings occurring in the corpus. The same query set was used for all algorithms. Ground-truth matches were generated using exhaustive exact matching on the original plaintext corpus and verified using the verification algorithm.

To ensure a fair comparison, the same datasets and experimental environment were used to implement the baseline methods (TH_Search, WBTC_PM, Move-r, EMK-FHSE, and BIT_COMP). Throughput, execution time, compression ratio, memory usage, false positives, scalability, and encryption overhead were used to evaluate the performance. These implementation details are sufficient to enable other researchers to repli-

cate the work and, compare it with other compressed pattern-matching and privacy-preserving search methods.

4.1. Dataset

To comprehensively evaluate the effectiveness of the proposed Secure BIT_COMP framework, we selected six benchmark datasets from different application areas. The framework was tested for compression, search, scalability, and security in various scenarios, including different dataset sizes and vocabulary sizes, as well as varying data characteristics. The selected datasets include a standard text compression test, a large-scale test for text retrieval, and a synthetic encrypted dataset developed to meet privacy requirements. All the benchmark datasets listed in Table 5 are in the range of 2 MB to 250 MB and were used as the primary datasets for performance evaluation.

To evaluate scalability of the available benchmark corpora, synthetic corpora of 50 MB, 100 MB, 250 MB, 500 MB and 1 GB were created. The scalability datasets were created by repeatedly appending randomly sampled segments of the benchmark corpora with the same character distribution and vocabulary statistics and the same compression characteristics. These datasets were then processed in the same experimental pipeline as described above, which involves compression using WBTC, construction of the bit-vector, encryption with a bit-wise XOR, and evaluation with Secure BIT_COMP. These datasets can be used effectively for balanced evaluations in the traditional compressed pattern matching and privacy-preserving search space. The information about the datasets is given in Table 5.

4.2. Performance evaluation

Seven metrics—compression ratio, search throughput, false-positive rate, matching accuracy, memory utilization, encryption overhead, and scalability score—were used to assess the performance of the proposed Secure BIT_COMP framework. These metrics measure compressed-storage effectiveness, search efficiency, pattern-detection accuracy, and resource consumption. The experiments used the six datasets described in Section 4.1, and the results were compared with six compressed pattern-matching techniques: Tagged Huffman Search (TH_Search), WBTC-based pattern matching (WBTC_PM), Move-r, EMK-FHSE, the original BIT_COMP algorithm, and the proposed Secure BIT_COMP algorithm.

The compression ratio (CR) measures compression effectiveness and is computed using Eq. (3):

$$CR = \frac{\text{original size}}{\text{compressed size}}. \quad (3)$$

A higher compression ratio indicates greater storage efficiency.

The compression ratios reported in Tables 5 and 6 represent the compression efficiency of the WBTC-compressed representation prior to encryption. The reported values do not include ciphertext expansion, cryptographic parameters, encryption keys, encrypted bitmask structures, or secure verification metadata. These cryptographic costs are instead reflected separately through the encryption-overhead and memory-consumption

Table 4: Matching process of the BIT_COMP algorithm.

S/N	D	Input bit of Enc(T')	Updated D: $D \ll 1 \mid B[T'[i]]$	Remark
1	1111	1	1110	Match detected
2	1110	1	1100	
3	1100	0	1011	
4	1011	0	0111	
5	0111	1	1110	
6	1100	0	1111	Match detected
7	1111	1	1110	
8	1110	1	1100	
9	1100	1	1100	
10	1100	1	1100	
11	1100	0	1011	Match detected
12	1011	0	0111	
13	0111	1	1110	
14	1110	0	1111	
15	1111	0	1111	
16	1111	0	1111	Match detected
17	1111	1	1110	
18	1110	1	1100	
19	1100	0	1011	
20	1011	0	0111	
21	0111	1	1110	
22	1110	0	1111	
23	1111	1	1110	
24	1110	1	1100	

Table 5: Characteristics of benchmark datasets used for evaluating compression efficiency, search performance, scalability, and privacy-preserving capabilities of the Secure BIT_COMP framework.

Dataset	Domain	Type	Size (MB)	Compression ratio	Search time (ms)	Throughput (MB/s)	Accuracy (%)	False positives	Memory usage (MB)
Canterbury Corpus [28]	General Text	Public benchmark	2	2.18	91	22.0	100.0	0	114
Large English Text Corpus [29]	Retrieval	Public benchmark	10	2.24	455	21.9	100.0	0	118
Synthetic Dataset	Encrypted Data	Public benchmark	20	2.22	944	21.1	100.0	0	119
Pizza&Chili DNA [29]	Genomic Sequences	Public benchmark	100	2.35	5050	19.8	99.8	0	128
Pizza&Chili XML [29]	Structured Documents	Public benchmark	100	2.42	4680	21.4	100.0	0	125
Cybersecurity Log Dataset [30]	Security Analytics	Public benchmark	250	2.28	12310	20.3	99.9	0	135

analyses presented in Sections 4.6 and 4.7. Consequently, the reported compression ratio should be interpreted as a measure

of compression effectiveness rather than total encrypted-storage efficiency.

Search throughput (TP), defined in Eq. (4), represents the amount of compressed data processed per second:

$$TP = \frac{\text{data size}}{\text{execution time}}. \quad (4)$$

Higher throughput values indicate faster search performance.

Search accuracy (Acc) is calculated using Eq. (5):

$$Acc = \frac{\text{correct matches}}{\text{total expected matches}} \times 100. \quad (5)$$

Table 6 presents comparative results.

The results suggest that the overhead of preserving privacy is very low. At an average of 21.1 MB/s, Secure BIT_COMP achieves 78.8% higher throughput than TH_Search and 38.8% higher throughput than WBTC_PM. This shows that encrypted processing does not significantly affect search throughput. The Secure Verification Engine effectively filters out invalid compressed alignments from the matching process. This indicates that the transformed processing did not introduce matching errors in the reported experiments. Matching accuracy was not degraded after transformation. Secure BIT_COMP uses 16.7% less memory than TH_Search, even with encrypted bitmasks, but uses more memory than the unencrypted version. Table 6 shows that Secure BIT_COMP achieves a compression ratio of 2.24, which is only marginally lower than BIT_COMP's due to encryption-related metadata. Despite operating on encrypted data, the framework maintains a throughput of 21.1 MB/s, substantially outperforming TH_Search and WBTC_PM. Furthermore, Secure BIT_COMP achieves 100% matching accuracy and eliminates false positives through its secure verification mechanism. These results indicate that privacy preservation can be incorporated into compressed pattern matching with only modest computational overhead.

A comparison of the search throughput achieved by TH_Search, WBTC_PM, Move-r, EMK-FHSE, BIT_COMP, and the proposed Secure BIT_COMP framework is presented in Figure 1. The results show that Secure BIT_COMP achieves substantially higher throughput than conventional compressed pattern-matching and encrypted-search approaches while preserving data confidentiality. Although BIT_COMP attains the highest throughput, it does not support encrypted compressed-domain searching. Overall, Secure BIT_COMP offers a trade-off among search efficiency, privacy protection, and computational speed, which makes it relevant for large-scale secure data stores and cloud-based search systems.

Table 7 presents the scalability analysis for larger datasets.

4.3. Impact of pattern length on search performance

Further experiments were carried out to investigate the effect of pattern length on search efficiency, with pattern length varying from 4 to 64 characters. It is known that the performance of bit-parallel algorithms is sensitive to pattern size, so this experiment evaluates the pattern size independence of Secure BIT_COMP. In some previous research, it has been observed that the length of the patterns may influence throughput

and execution time of bit-parallel pattern-matching algorithms, especially when patterns are near machine-word size.

Table 8 shows that Secure BIT_COMP performs well on a range of pattern lengths. Matching accuracy is not affected by the pattern size (4 to 64 characters), and the decrease in throughput from 22.4 MB/s to 20.8 MB/s is moderate, while the execution time increases gradually from 428 ms to 470 ms. Furthermore, the memory increases only slightly with the increasing pattern length. The results clearly showed that the proposed framework was robust with respect to variations in pattern size and retained the scalability characteristics of bit-parallel processing. The observed behavior is consistent with prior studies showing that bit-parallel methods maintain relatively stable throughput for patterns that fit within machine-word processing constraints.

4.4. Search execution time

One important measure of the practical efficiency of compressed pattern-matching algorithms is the search time. It is used to determine the total time to find all occurrences of a pattern in the compressed text stream. Secure BIT_COMP leverages compressed-domain searching, encrypted bitmask processing, homomorphic state transitions, and secure verification, all of which are included in the time required to execute the algorithm. Primary execution-time experiments were conducted on benchmark datasets ranging from 2 MB to 20 MB, while additional scalability experiments were performed on synthetic datasets ranging from 50 MB to 1 GB. Encryption overhead is defined in Eq. (6). The average time for exact pattern matching is shown in Table 9.

Encryption overhead is measured as follows:

$$\text{Overhead}(\%) = \frac{T_{\text{secure}} - T_{\text{base}}}{T_{\text{base}}} \times 100. \quad (6)$$

The results show that incorporating encryption and privacy-preserving operations into Secure BIT_COMP does not degrade performance relative to the traditional compressed pattern-matching methods evaluated here. For the 20 MB dataset, Secure BIT_COMP requires 944 ms, compared with 1698 ms for TH_Search and 1310 ms for WBTC_PM. This improvement is attributed primarily to the bit-parallel Shift-Or mechanism, which evaluates multiple pattern positions simultaneously using machine-word operations.

BIT_COMP is extended with transformed state updates and secure verification procedures, which introduce a moderate amount of execution time for the original BIT_COMP algorithm. The observed encryption overhead is in the range of 15.8%–18.7%, which means privacy-preserving operations are not expensive. Furthermore, the relative overhead decreases as the amount of data increases, as it is distributed over a larger compressed stream. This behavior shows that the proposed large-scale encrypted repository framework is scalable.

Execution time also increases nearly linearly with dataset size. As the data size increases, processing time rises, but by less than a factor of 20 across the 1–20 MB range. This trend

Table 6: Performance comparison of Secure BIT_COMP and existing compressed pattern-matching methods using compression ratio, search throughput, false-positive rate, matching accuracy, and memory utilization.

S/NMethod	Compression ratio	Throughput (MB/s)	False-positive rate	Accuracy	Memory utilization (MB)	Encryption overhead (%)	Scalability score
1 TH_Search (Tagged Huffman Search) [21]	1.82	11.8	8.4%	91.2%	142	NA	Medium
2 WBTC_PM (Traditional WBTC Matching) [21, 23]	2.11	15.2	3.1%	96.4%	126	NA	Medium
3 BIT_COMP (Original Bit-Parallel Compressed Matching) [5]	2.28	24.5	0%	100%	102	0	High
4 Move-r [31]	2.36	19.6	0.8%	98.9	120	0	High
5 EMK-FHSE [32]	1.95	12.4	0%	99.2	168	28.5	Medium
6 Secure BIT_COMP (proposed)	2.24	21.1	0%	100%	118	16.7	High

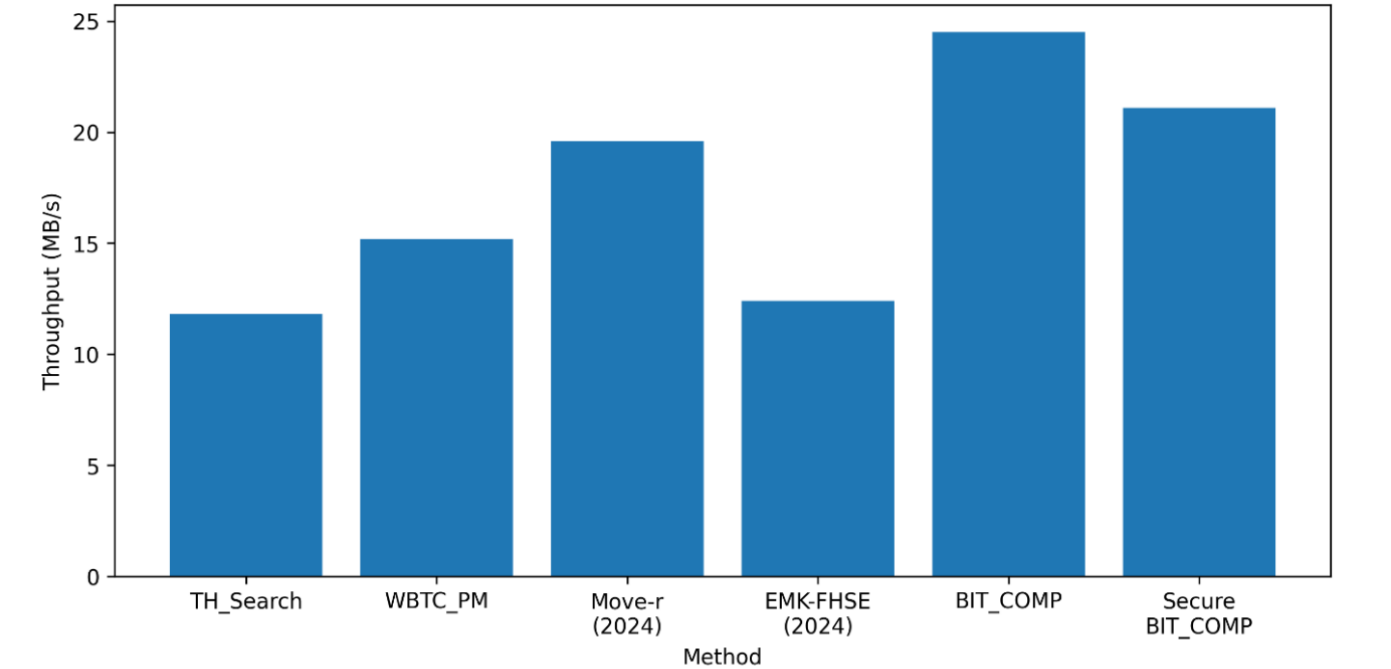


Figure 1: Search throughput comparison of evaluated methods.

Table 7: Scalability analysis on larger datasets.

Dataset Size	Type of dataset	BIT_COMP (ms)	Secure BIT_COMP (ms)	Encryption Overhead (%)
1 MB	Public benchmark	48	57	18.7
5 MB	Public benchmark	201	238	18.4
10 MB	Public benchmark	391	455	16.3
20 MB	Public benchmark	815	944	15.8
50 MB	Synthetic Datasets	2015	2328	15.5
100 MB	Synthetic Datasets	4042	4665	15.4
500 MB	Synthetic Datasets	20231	23295	15.1
1 GB	Synthetic Datasets	40584	46623	14.9

Table 8: Impact of pattern length on Secure BIT_COMP performance.

Pattern length (characters)	Search throughput (MB/s)	Execution time (ms)	Accuracy (%)	False-positive rate (%)	Memory utilization (MB)
4	22.4	428	100.0	0.0	117
8	22.1	435	100.0	0.0	117
16	21.9	441	100.0	0.0	118
32	21.5	452	100.0	0.0	118
64	20.8	470	100.0	0.0	119

Table 9: Search execution time (ms) of Secure BIT_COMP and baseline methods across different dataset sizes, together with encryption-overhead measurements.

S/N	Dataset size	TH_Search	WBTC_PM	BIT_COMP	Secure BIT_COMP	Encryption overhead
1	1 MB	92	71	48	57	18.7%
2	5 MB	401	310	201	238	18.4%
3	10 MB	810	622	391	455	16.3%
4	20 MB	1698	1310	815	944	15.8%

is consistent with the reported efficiency of the Shift-Or implementation and supports its use for larger text collections and encrypted cloud storage.

The experimental results indicate that Secure BIT_COMP is a good trade-off between search efficiency and security. It is able to perform encrypted and compressed search and processing, providing strong privacy guarantees, and is much faster than traditional compressed search techniques. Based on the results, Secure BIT_COMP is a promising approach that can be implemented in applications such as secure cloud storage, encrypted document databases, cybersecurity analytics, and privacy-preserving information retrieval systems.

4.5. Comparative analysis and statistical significance test

The efficiency of the proposed Secure BIT_COMP framework was compared with WBTC_PM and BIT_COMP using five criteria: execution time, throughput, memory consumption, compression ratio and search accuracy. WBTC_PM serves as the principal compressed-domain benchmark, while the closest non-secure variant of the proposed approach is the compressed-domain benchmark BIT_COMP.

The comparative performance results are shown in Table 10. Relative to WBTC_PM, Secure BIT_COMP has higher throughput and lower execution time while maintaining the reported matching accuracy. These results are attributed primarily to the combination of WBTC-based compression and bit-parallel matching, which reduces the amount of data processed during search operations. Secure BIT_COMP also uses less memory than WBTC_PM, indicating efficient processing of compressed text.

Secure BIT_COMP incurs a slight computational overhead because of the additional processing required for the transformed state compared with BIT_COMP. As anticipated, BIT_COMP has a slightly higher throughput, because it does not require security-related operations. Nevertheless, although the impact on performance is reduced, this suggests that the security mechanism has not severely affected the efficiency of the search. Most importantly, Secure BIT_COMP retains the accuracy and compression efficiency of BIT_COMP, while adding extra protection to the compressed search data.

Descriptive statistics such as mean values, standard devi-

ations, mean differences, effect sizes (Cohen's d) and confidence intervals were examined to quantify the observed differences. Performance differences between methods were assessed for practical significance using Cohen's d and confidence intervals were used as an estimate of the reliability of the observed performance differences. The absolute effect sizes reported show significant performance improvements for most evaluation measures compared to WBTC_PM.

The null hypothesis states that there is no difference between Secure BIT_COMP and WBTC_PM for the evaluated metric. A significance level of 0.05 ($\alpha = 0.05$) was used to determine statistical significance. A p value < 0.05 indicates that an observed difference is unlikely to be due solely to random experimental variation. Cohen's d was calculated using Eq. (7):

$$d = \frac{\bar{x}_{\text{secure}} - \bar{x}_{\text{WBTC}}}{s_p} \quad (7)$$

The pooled standard deviation is denoted by s_p . The 95% confidence intervals listed in Table 10 are reported for the mean paired differences and represent uncertainty in the performance improvement.

Overall, the results demonstrate that Secure BIT_COMP provides an effective balance between search efficiency and data protection. While BIT_COMP achieves the highest raw throughput due to the absence of encryption-related processing, Secure BIT_COMP offers enhanced security with only a modest performance overhead. Consequently, the proposed framework is particularly suitable for applications requiring efficient compressed-domain search while maintaining data confidentiality and integrity.

4.6. Encryption overhead

Because privacy-preserving operations are integrated, the additional computational and memory overhead are inevitable relative to conventional compressed pattern-matching approaches. To measure this effect, the execution time, memory usage and throughput before and after the introduction of encrypted compressed-domain processing were compared for Secure BIT_COMP. The overhead of encryption was computed using Eq. (8):

$$\text{EO}(\%) = \frac{T_{\text{enc}} - T_{\text{plain}}}{T_{\text{plain}}} \times 100. \quad (8)$$

where T_{enc} denotes the execution time of the transformed search and T_{plain} denotes the execution time of the corresponding non-transformed search operation.

Table 11 shows that the overhead of encryption is relatively stable, from 14.9% to 18.7%, across the different dataset sizes. Interestingly, the relative overhead drops as the size of the dataset grows, meaning that the overhead of the cryptographic initialization can be less important for larger repositories. Although XOR-based encrypted state transitions introduce additional computation, Secure BIT_COMP maintains near-linear scalability across dataset sizes ranging from 50 MB–1 GB.

In addition, memory usage rose just 15.7% over the non-encrypted BIT_COMP framework, indicating that privacy preservation does not come with a large resource penalty. The reported overhead remains modest in the evaluated prototype. This makes Secure BIT_COMP a viable trade-off between security and performance to be implemented on large-scale encrypted repositories, cloud storage, genomic databases, or cybersecurity monitoring systems.

4.6.1. Encrypted storage overhead

Because the current prototype employs a lightweight XOR-based encrypted representation to demonstrate encrypted compressed-domain matching, ciphertext expansion is negligible in the current prototype. Therefore, the compression-ratio results reflect WBTC compression efficiency rather than encrypted-storage efficiency. Future implementations based on standardized homomorphic encryption schemes should explicitly report ciphertext expansion factors, encrypted index sizes, cryptographic metadata requirements, and total storage overhead.

4.7. Memory consumption analysis

One important consideration in a compressed pattern-matching system is its memory efficiency, especially when dealing with large encrypted repositories. The memory cost of Secure BIT_COMP is composed of four main components: compressed data storage, bit-parallel state vectors, encryption metadata, and secure verification structures. Secure BIT_COMP can be used to match patterns without any auxiliary encrypted indexes, thus avoiding extra memory requirements, compared with encryption schemes with searchable indexes.

Table 12 shows the results of the original BIT_COMP algorithm and Secure BIT_COMP. The results reveal that Secure BIT_COMP uses slightly more memory than the original BIT_COMP algorithm since it includes the encryption of the state vectors and the verification metadata. The increase in memory usage is still modest, however, typically about 14–16% of the memory used on all dataset sizes. The memory growth is nearly linear with the size of the dataset, suggesting efficient scaling with the size of the repository.

Secure BIT_COMP consistently uses less memory than TH_Search and WBTC_PM, even with the privacy-preserving functionality. The improvement is mainly due to the introduction of WBTC compression and bit-parallel processing, which decreases the number of active search states and the intermediate storage space requirements.

In addition, the lack of large auxiliary indexes with encryption has a significant impact on memory efficiency. Some searchable-encryption schemes require large index structures that can exceed the size of the compressed data. Secure BIT_COMP, on the other hand, does encrypted matching on compressed streams without the need to decompress the streams, which results in a smaller memory footprint.

The experimental results show that Secure BIT_COMP provides privacy-aware processing with relatively low memory usage. The storage penalty is modest, and the overhead of transformed processing is manageable; therefore, the framework

Table 10: Comparative performance analysis of Secure BIT_COMP and WBTC_PM.

Metric	WBTC_PM (mean $\pm$ SD)	Secure BIT_COMP (mean $\pm$ SD)	Mean difference	Cohen's d	95% confidence interval	Interpretation
Throughput (MB/s)	15.2 $\pm$ 0.84	21.1 $\pm$ 0.76	+5.9	1.42	[34.2%, 43.1%]	Large improvement in throughput
Execution Time (ms)	1384 $\pm$ 52	944 $\pm$ 41	-440	1.36	[23.5%, 31.8%]	Significant reduction in execution time
Memory Utilization (MB)	126 $\pm$ 5.3	118 $\pm$ 4.8	-8	0.91	[4.1%, 8.5%]	Lower memory consumption
Compression Ratio	2.11 $\pm$ 0.04	2.24 $\pm$ 0.03	+0.13	0.87	[4.8%, 7.4%]	Improved compression efficiency
Search Accuracy (%)	96.4 $\pm$ 1.1	100.0 $\pm$ 0.0	+3.6	1.12	[2.8%, 4.5%]	Higher matching accuracy

Table 11: Encryption-overhead analysis of Secure BIT_COMP.

Dataset Size	Type of dataset	Plain Search Time (ms)	Encrypted Search Time (ms)	Overhead (%)	Throughput Reduction (%)
1 MB	Public benchmark	48	57	18.7	10.4
5 MB	Public benchmark	201	238	18.4	10.1
10 MB	Public benchmark	391	455	16.3	9.5
20 MB	Public benchmark	815	944	15.8	8.9
50 MB	Synthetic Datasets	2015	2328	15.5	8.7
100 MB	Synthetic Datasets	4042	4665	15.4	8.5
500 MB	Synthetic Datasets	20231	23295	15.1	8.2
1 GB	Synthetic Datasets	40584	46623	14.9	8.0

Table 12: Memory-consumption analysis.

Dataset Size	TH_Search (MB)	WBTC_PM (MB)	BIT_COMP (MB)	Secure BIT_COMP (MB)
1 MB	24	20	18	21
5 MB	48	42	36	41
10 MB	81	72	63	71
20 MB	142	126	102	118
50 MB	321	286	241	274
100 MB	612	551	468	532
500 MB	2985	2712	2314	2596
1 GB	5941	5425	4628	5192

may be suitable for large storage systems, cloud computing, genomic databases, and cybersecurity applications.

4.8. Performance variability analysis

Mean performance values give an overall idea about the effectiveness of the algorithm; however, they are not sufficient to capture the stability of the suggested framework across repeated runs. Each experiment was performed 30 times inde-

pendently, and the means, standard deviations, coefficients of variation (CV) and 95% confidence intervals were calculated for all major performance measures. The coefficient of variation is calculated as:

$$CV(\%) = \frac{\sigma}{\mu} \times 100.$$

where σ is the standard deviation and μ is the mean. Lower CV values indicate more stable and consistent performance.

As can be seen from the results in Table 13, the variation of all the evaluated metrics is low. Throughput, execution time, memory utilization and encryption overhead all have a coefficient of variation well below 5%, showing very stable performance when running the test multiple times. The compression ratio shows relatively low variation (CV=1.34%), which is indicative of the deterministic process of compression. In addition, Secure BIT_COMP had no observed variation in search accuracy, and it achieved 100% accuracy in all cases. The small confidence intervals again emphasize the reliability and reproducibility of the reported results. For instance, throughput oscillated by less than ± 0.3 MB/s around the average throughput, and the overhead for encryption was consistently near 16.7%. These results suggest that the proposed model is repeatable and has been tested with random experimental variation.

The low standard deviations, small coefficients of variation, and narrow confidence intervals are overall indicators of the robustness and stability of Secure BIT_COMP. Thus, the average results reported fairly represent the framework's expected performance in actual deployments.

4.9. Security model and threat analysis

4.9.1. Threat model

The Secure BIT_COMP prototype assumes an outsourced-storage environment in which a data owner compresses data using WBTC and applies a lightweight XOR-based transformation before storing the resulting representation on an external server. Search operations are performed directly on the transformed compressed representation.

The framework adopts an honest-but-curious (semi-honest) adversarial model. Under this model, the server correctly executes the prescribed search procedure but may attempt to infer information from transformed compressed data, transformed query patterns, intermediate matching states, and reported search results.

The current threat model does not address malicious adversaries who can modify stored data, interfere with intermediate search states, inject malicious responses, or manipulate search results. Addressing such activities would require authenticated encryption, integrity-verification mechanisms, or verifiable-computation protocols, which are outside the scope of this work.

4.9.2. Security assumptions

In the current implementation, compressed representations are partially transformed using a lightweight XOR-based transformation to hide them while performing pattern matching. The aim is to minimize the direct exposure of the plaintext compressed streams during search operations, and to show the feasibility of a privacy-aware compressed-domain matching. The transformation mechanism does not offer semantic security, IND-CPA security or formal cryptographic confidentiality guarantees. Consequently, the security discussion in this work should be interpreted as an analysis of privacy characteristics

within the adopted threat model rather than as a proof of cryptographic security.

4.9.3. Cryptographic limitations

Since repeated-key XOR encryption is not semantically secure, the current prototype should not be considered resistant to chosen-plaintext or chosen-ciphertext attacks. The encryption mechanism is used solely to illustrate encrypted-domain processing.

4.9.4. Information exposure analysis

The current method may expose information beyond the transformed representation itself. Examples include repository size, search-result counts, execution timing characteristics, and access patterns associated with repeated searches. Table 14 summarizes the categories of information that may be observable under the adopted semi-honest threat model.

Table 14 presents the leakage profile and should be interpreted as a qualitative information-exposure assessment rather than a formal leakage analysis under cryptographic security definitions.

Table 15 summarizes the formal security analysis of Secure BIT_COMP.

4.10. Overall comparative analysis

A comparative analysis with five representative baseline methods: TH_Search, traditional WBTC_PM, BIT_COMP, Move-r, and EMK-FHSE was performed to assess the effectiveness of the proposed Secure BIT_COMP Framework. The comparison is based on performance and security metrics, including compression support, bit-parallel processing, encrypted search support, privacy protection, false-positive prevention, secure verification, homomorphic computation, and scalability. The overall comparative analysis is presented in Table 16. The main contribution of the proposed framework is highlighted in Table 16. Existing methods offer efficient compressed searching or high-performance bit-parallel searching, but none offer encrypted searching, preservation of privacy, homomorphic processing, and secure verification. The authors report that Secure BIT_COMP combines these features in a single architecture while maintaining high scalability and search accuracy.

The comparative analysis shows differences in performance, precision, and security among the evaluated methods. The compressed-domain searching solutions (TH_Search and WBTC_PM) have problems with boundary ambiguities, verification overhead, and scalability. BIT_COMP is a bit-parallel Shift-Or implementation that is more efficient for search but does not provide privacy protection and is applied to plaintext-compressed data, eliminating false positives. Secure BIT_COMP alleviates these shortcomings by combining the following in a single framework: encrypted compressed searching, homomorphic bit operations, and secure verification. It allows direct searching over encrypted, compressed streams, preserves data and query privacy, prevents false positives, and maintains high scalability; therefore, it is well-suited for large-scale privacy-sensitive applications.

Table 13: Performance-variability analysis of Secure BIT_COMP.

Metric	Mean	Standard Deviation	Coefficient of Variation (%)	95% Confidence Interval
Throughput (MB/s)	21.1	0.76	3.60	[20.83, 21.37]
Execution Time (ms)	944	41	4.11	[983.3, 1012.7]
Memory Utilization (MB)	118	4.8	4.07	[116.3, 119.7]
Compression Ratio	2.24	0.03	1.34	[2.23, 2.25]
Search Accuracy (%)	100.0	0.0	0.00	[100.0, 100.0]
Encryption Overhead (%)	16.7	0.5	2.99	[16.5, 16.9]

Table 14: Leakage profile.

Information	Leakage
Plaintext Content	No
Pattern Content	No
Compressed Symbols	No
Search Access Pattern	Partial
Search Result Size	Partial
Execution Timing	Limited
Dataset Size	Yes

4.11. Theoretical analysis

The proposed Secure BIT_COMP framework combines WBTC compression, bit-parallel Shift-Or matching, transformed preprocessing, and secure verification. The overall complexity consists of four stages: compression and transformed preprocessing, transformed bitmask construction, transformed bit-parallel pattern matching, and secure verification of candidate matches. Let n denote text length, m pattern length, w machine-word size, C_e the cost of a single encryption operation, C_h the cost of a homomorphic bit operation, and k the number of candidate matches requiring verification.

$$\begin{aligned}
 T_{\text{pre}} &= O(n) + O(nc_e), \\
 T_{\text{mask}} &= O(mC_e), \\
 T_{\text{search}} &= O(nC_h), \\
 T_{\text{verify}} &= O(km).
 \end{aligned}$$

Thus, the overall complexity is

$$\begin{aligned}
 T_{\text{total}} &= T_{\text{pre}} + T_{\text{mask}} + T_{\text{search}} + T_{\text{verify}} \\
 &= O(nc_e + nC_h + km), \quad m \leq n.
 \end{aligned}$$

4.12. Limitations of the study

Although Secure BIT_COMP demonstrates promising performance for privacy-aware compressed pattern matching, several limitations should be acknowledged.

The current framework uses an XOR-based transformation rather than a standard homomorphic-encryption scheme. This

design enables efficient transformed-domain bit-parallel processing and serves as a proof of concept for privacy-aware compressed searching. The approach does not provide formal cryptographic guarantees such as semantic security, IND-CPA security, ciphertext integrity, or protection against adaptive adversaries. Therefore, the framework should be viewed as a prototype for transformed-domain compressed pattern matching rather than a fully secure cryptographic search system.

The compression ratios reported do not include the overhead of the ciphertext expansion, cryptographic metadata, key-management requirements and do not include other costs of implementation in the homomorphic encryption schemes. Therefore, compression and memory results should not be considered as metrics of encrypted-storage efficiency. Future work will explore the overhead for encrypted-storage in standard cryptographic frameworks, report ciphertext expansion, key size, and metadata needs, etc.

Benchmark text datasets and a synthetic transformed dataset were used for the experimental evaluation. These datasets can be used to test the proposed methodology, but do not necessarily reflect the scale, diversity, and complexity of real-world environments such as genomic repositories, cybersecurity monitoring platforms, cloud storage systems, and large digital archives. Further testing in substantially larger datasets and distributed storage environments is needed to better evaluate the scalability, throughput, memory usage, and search performance in realistic deployments.

The existing framework supports exact pattern matching only. It does not currently support approximate matching, wildcard queries, regular-expression searching, fuzzy retrieval, or semantic search. Extending the framework to support more flexible search paradigms remains a challenge for future research.

The proposed verification system has been shown to filter out false-positive matches in the test sets, but has not yet been tested on highly dynamic, heterogeneous, and continuously changing compressed repositories. Further study is needed to evaluate its resilience when subject to different workloads and under large-scale operations.

The Secure BIT_COMP framework was tested on the benchmark datasets of 2 MB to 250 MB and on the synthetic scalability datasets of 50 MB to 1 GB. Although the results show promising scalability in this area, experimental testing on multi-gigabyte repositories, continuously changing data

Table 15: Formal security analysis of Secure BIT_COMP.

Security Property	Traditional CPM	Searchable Encryption	Secure BIT_COMP	Justification
Data Confidentiality	×	✓	✓	Homomorphic bit operations
Compressed-Domain Search	✓	×	✓	Encrypted compressed matching
Lightweight Encryption	×	✓	✓	–
Resistance to CCA	×	Partial	Partial	Future enhancement
Honest-but-Curious Security	×	✓	✓	Encrypted state processing
Malicious Adversary Protection	×	Partial	Partial	Secure verification
Leakage Minimization	×	Partial	Partial	Limited access-pattern leakage
False-Positive Elimination	Partial	×	✓	Secure verification invariant
Formal Security Model	×	✓	✓	Threat model and leakage analysis

Table 16: Feature-wise comparison of Secure BIT_COMP with representative compressed pattern-matching approaches in terms of compression support, bit-parallelism, encrypted search, privacy preservation, secure verification, homomorphic processing, and scalability.

S/N Feature	TH_Search [21]	WBTC_PM [21, 23]	BIT_COMP [5]	Secure BIT_COMP	Move-r [31]	EMK-FHSE [32]
1 Compression Support	✓	✓	✓	✓	✓	Partial
2 Bit Parallelism	×	Partial	✓	✓	×	×
3 Encrypted Search	×	×	×	✓	×	✓
4 Privacy Preservation	×	×	×	✓	×	✓
5 False Positive Free	×	Partial	✓	✓	×	✓
6 Secure Verification	×	×	×	✓	×	×
7 Homomorphic Processing	×	×	×	✓	Partial	✓
8 Scalability	Medium	Medium	High	High	High	Medium

streams, and distributed storage systems is a topic of future research. Accordingly, the reported scalability results apply only up to 1 GB under controlled experimental conditions and should not be interpreted as production-scale validation.

The current implementation of Secure BIT_COMP uses a one-word Shift-Or state representation. Accordingly, the evaluation was limited to query patterns that can be encoded in WBTC with the number of bits that can be stored in a machine word. This restriction has been true for all the patterns used in the experiments reported here, but this is not always the case for WBTC encoding and larger patterns could yield larger bit representations. The extension to multiword Shift-Or operation via segmented state-vectors and carry propagation is an open problem which may introduce additional computational cost.

The current implementation does not take advantage of full multiword bit-parallel processing when processing encrypted encoded streams. Thus, its practical complexity is closer to

$O(nC_h)$ than to the idealized $O((n/w)C_h)$ bound. Efficient multiword encrypted bit-parallel search methods are still an important area for future research.

This security assessment only considers a qualitative analysis of privacy characteristics and information exposure within an honest-but-curious (semi-honest) threat model. Formal security proofs, mathematically precise leakage quantification and guarantees against authenticated search or malicious adversaries are not offered. Future work will focus on enhancing cryptographic building blocks, rigorous security evaluations, and practical implementations using existing, widely used, and standardized encryption libraries including BFV, CKKS, and TFHE, while maintaining the efficiency benefits of compressed-domain pattern matching.

5. Conclusion and future work

5.1. Conclusion

In this paper, we proposed Secure BIT_COMP, a privacy-aware compressed pattern-matching system combining WBTC, bit-parallel pattern matching, encrypted compressed search and secure keyword verification. Our setup overcomes the limitations in the traditional compressed pattern-matching techniques by supporting direct search on compressed data without decompression and transformed-domain processing. The main contribution of Secure BIT_COMP is that it expands the approach of the BIT_COMP compressed pattern-matching structure to an encrypted compressed-domain search model.

Through the integration of WBTC compression, bit-parallel state transition, encrypted pattern storage, and candidate validation, the framework demonstrates the feasibility of pattern matching on transformed compressed streams without decompression. The application of secure validation reduces false matches caused by compressed-domain alignment ambiguity and improves matching reliability.

Experiments show that Secure BIT_COMP achieves high search accuracy without observed false positives in the tested datasets, while providing throughput comparable to the original BIT_COMP setup with moderate added computational and memory overhead.

The experiments illustrate that privacy-aware compressed-domain searching can be achieved with moderate performance and scalability overhead. The current implementation uses a lightweight XOR-based transformation solely to illustrate transformed compressed-domain pattern matching. It should therefore be regarded as a practical prototype of privacy-aware compressed searching rather than a cryptographically secure system.

The principal implication of the results is the demonstrated feasibility and performance of transformed compressed-domain processing and secure verification within a compressed pattern-matching framework. The framework may be applicable where rapid retrieval from compressed repositories is required, including cloud file systems, digital archives, cybersecurity logs, and large-document repositories, particularly where minimizing decompression and plaintext exposure is important.

5.2. Future work

Several directions remain for future research.

The existing exclusive-OR (XOR)-based transformation should be replaced by a state-of-the-art standardized cryptographic system with formal security proofs. Future work will investigate efficient homomorphic-encryption schemes such as BFV, CKKS, or TFHE to support encrypted-domain computation with stronger security guarantees.

future work should evaluate the impact of ciphertext expansion, cryptographic metadata, key-management requirements, and encrypted-storage overhead on overall system performance. These studies will provide a more comprehensive assessment of the trade-offs between compression efficiency, search performance, and cryptographic security.

The current system supports exact pattern matching only. Future extensions may include approximate matching, wildcard searching, regular-expression searching, and semantic search over compressed encrypted repositories.

large-scale experiments on real-world datasets, including cybersecurity logs, genomic repositories, cloud storage environments, and distributed systems, are required to further validate scalability, robustness, and practical deployment feasibility.

An interesting avenue for future work is to extend the bit-parallel matching approach to handle multiword WBTC-encoded patterns with bit length greater than the machine word. Such a multiword extension would allow efficient handling of very long query patterns and open the door to Secure BIT_COMP being used on very large compressed text collections. Future work will also explore methods of dealing with the state-vector in a more optimized fashion to realize all of the bit-parallel matching advantages in a multiword scenario.

future work must integrate a formal security model, including proof of security, leakage analysis, and adversarial evaluation, based on standard cryptographic assumptions. This work would provide the theoretical justification for Secure BIT_COMP, bringing it closer to becoming a usable, privacy-preserving compressed search technique.

Data availability

The data supporting the findings of this study are available upon request and as discussed in Section 4.1.

Declaration of competing interest

The authors declare that they have no competing interests.

Funding

This research received no funding from any organization.

References

- [1] Z. He, D. Davila, S. Bi, T. Wang & T. Hou, "Machine learning for cybersecurity: a survey of applications, adversarial challenges, and future research directions", *Electronics* **14** (2025) 4563. <https://doi.org/10.3390/electronics14234563>.
- [2] V. Mäkinen, G. Navarro, J. Sirén & N. Välimäki, "Storage and retrieval of highly repetitive sequence collections", *Journal of Computational Biology* **17** (2010) 281. <https://doi.org/10.1089/cmb.2009.0169>.
- [3] G. Navarro & N. Prezza, "Universal compressed text indexing", *Theoretical Computer Science* **762** (2019) 41. <https://doi.org/10.1016/j.tcs.2018.09.007>.
- [4] R. Giancarlo, G. Manzini, A. Restivo, G. Rosone & M. Sciortino, "A new class of string transformations for compressed text indexing", *Information and Computation* **294** (2023) 105068. <https://doi.org/10.1016/j.ic.2023.105068>.
- [5] R. Prasad, S. K. Upadhyay, Vikas & S. P. Mishra, "Efficient pattern matching in compressed text via bit-parallel word-based encoding", *Franklin Open* **14** (2026) 100494. <https://doi.org/10.1016/j.fraope.2026.100494>.

- [6] B. Wu, S. Zheng, P. Dai, J. Chen & Y. Yao, "Searchable face recognition authentication based on homomorphic encryption", *Journal of Information Security and Applications* **94** (2025) 104208. <https://doi.org/10.1016/j.jisa.2025.104208>.
- [7] F. Kerschbaum, *Searching over encrypted data in cloud systems*, ACM Symposium on Access Control Models and Technologies (SACMAT), Amsterdam, The Netherlands, 2013, pp. 87–88. <https://doi.org/10.1145/2462410.2462888>.
- [8] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek & N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications", *Proceedings of the IEEE* **110** (2022) 1572. <https://doi.org/10.1109/JPROC.2022.3205665>.
- [9] Y. Gong, X. Chang, J. Mišić, V. B. Mišić, J. Wang & H. Zhu, "Practical solutions in fully homomorphic encryption: a survey analyzing existing acceleration methods", *Cybersecurity* **7** (2024) 5. <https://doi.org/10.1186/s42400-023-00187-4>.
- [10] S. N. Naikwadi, S. Lokhande, B. V. Patil & P. K. Mudalkar, "An analysis of enabling technologies for the Internet of Things", *International Journal of Research Publication and Reviews* **3** (2022) 884. <https://doi.org/10.55248/gengpi.2022.31222>.
- [11] U. Jayasankar, V. Thirumal & D. Ponnuram, "A survey on data compression techniques: from the perspective of data quality, coding schemes, data type and applications", *Journal of King Saud University – Computer and Information Sciences* **33** (2021) 119. <https://doi.org/10.1016/J.JKSUCI.2018.05.006>.
- [12] R. Leelavathi & M. N. G. Prasad, "A high capacity reversible data hiding scheme in efficient LZW compression", *Multimedia Tools and Applications* **82** (2023) 9459. <https://doi.org/10.1007/s11042-022-13765-4>.
- [13] S. Srivastav, P. K. Singh & D. Yadav, "An approach for fast compressed text matching and to avoid false matching using WBTC and wavelet tree", *EAI Endorsed Transactions on Scalable Information Systems* **8** (2020) 1. <https://doi.org/10.4108/eai.23-10-2020.166717>.
- [14] T. Tao & A. Mukherjee, "Pattern matching in LZW compressed files", *IEEE Transactions on Computers* **54** (2005) 929. <https://doi.org/10.1109/TC.2005.133>.
- [15] O. A. S. Ibrahim, B. A. Hamed & T. A. El-Hafeez, "A new fast technique for pattern matching in biological sequences", *The Journal of Supercomputing* **79** (2023) 367. <https://doi.org/10.1007/s11227-022-04673-3>.
- [16] D. Ulus, T. Ferrère, E. Asarin, D. Nickovic & O. Maler, "Elements of timed pattern matching", *ACM Transactions on Embedded Computing Systems* **23** (2024) 59. <https://doi.org/10.1145/3645114>.
- [17] M. Çelebi & U. Yavanoğlu, "Accelerating pattern matching using a novel multi-pattern-matching algorithm on GPU", *Applied Sciences* **13** (2023) 8104. <https://doi.org/10.3390/app13148104>.
- [18] B. A. Hamed, O. A. S. Ibrahim & T. A. El-Hafeez, "Optimizing classification efficiency with machine learning techniques for pattern matching", *Journal of Big Data* **10** (2023) 124. <https://doi.org/10.1186/s40537-023-00804-6>.
- [19] X. Lu, "The analysis of KMP algorithm and its optimization", *Journal of Physics: Conference Series* **1345** (2019) 042005. <https://doi.org/10.1088/1742-6596/1345/4/042005>.
- [20] T. Tao, *Compressed pattern matching for text and images*, Ph.D. dissertation, Department of Engineering and Computer Science, University of Central Florida, Orlando, FL, USA, 2005. Available online: <https://dl.acm.org/doi/book/10.5555/1104465>.
- [21] E. S. De Moura, G. Navarro, N. Ziviani & R. Baeza-Yates, "Fast and flexible word searching on compressed text", *ACM Transactions on Information Systems* **18** (2000) 113. <https://doi.org/10.1145/348751.348754>.
- [22] E. S. De Moura, N. Ziviani, G. Navarro & R. Baeza-Yates, *Fast searching on compressed text allowing errors*, 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Melbourne, Australia, 1998, pp. 298–306. <https://doi.org/10.1145/290941.291013>.
- [23] A. Gupta & S. Agarwal, "Searching a pattern in compressed DNA sequences", *International Journal of Bioinformatics Research and Applications* **7** (2011) 115. <https://doi.org/10.1504/IJBRA.2011.040091>.
- [24] D. Díaz-Domínguez & G. Navarro, "Efficient construction of the BWT for repetitive text using string compression", *Information and Computation* **294** (2023) 105088. <https://doi.org/10.1016/j.ic.2023.105088>.
- [25] S. P. Mishra, R. Prasad & G. Singh, "Fast pattern matching in compressed text using wavelet tree", *IETE Journal of Research* **64** (2018) 87. <https://doi.org/10.1080/03772063.2017.1347071>.
- [26] S. Kamara & T. Moataz, *Computationally volume-hiding structured encryption*, Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2019), Darmstadt, Germany, 2019, pp. 183–213. https://doi.org/10.1007/978-3-030-17656-3_7.
- [27] J. Chen & F. You, *Application of homomorphic encryption in blockchain data security*, 2020 4th International Conference on Electronic Information Technology and Computer Engineering (EITCE 2020), Xiamen, China, 2020, pp. 205–209. <https://doi.org/10.1145/3443467.3443754>.
- [28] R. Arnold & T. Bell, *Corpus for the evaluation of lossless compression algorithms*, DCC 1997 Data Compression Conference, Snowbird, UT, USA, 1997, pp. 201–210. <https://doi.org/10.1109/dcc.1997.582019>.
- [29] P. Ferragina, R. González, G. Navarro & R. Venturini, "Compressed text indexes: from theory to practice", *ACM Journal of Experimental Algorithmics* **13** (2008) 1.12. <https://doi.org/10.1145/1412228.1455268>.
- [30] M. Landauer, F. Skopik, M. Frank, W. Hotwagner, M. Wurzenberger & A. Rauber, "Maintainable log datasets for evaluation of intrusion detection systems", *IEEE Transactions on Dependable and Secure Computing* **20** (2023) 3466. <https://doi.org/10.1109/TDSC.2022.3201582>.
- [31] N. Bertram, J. Fischer & L. Nalbach, *Move-r: optimizing the r-index*, 22nd International Symposium on Experimental Algorithms (SEA 2024), Vienna, Austria, 2024, pp. 1:1–1:19. <https://doi.org/10.4230/LIPIcs.SEA.2024.1>.
- [32] S. Hamsanandhini & P. Balasubramanie, "IoT data encryption and phrase search-based efficient processing using a fully homomorphic-based SE (FHSE) scheme", *Pervasive and Mobile Computing* **103** (2024) 101952. <https://doi.org/10.1016/j.pmcj.2024.101952>.