Searchable Encryption in Mobile Cloud Setting

Piphattra Sreekongpan, Pichaya Arechep, Thanapoom Thanyasukpaisal
School of ICT, Sirindhorn International Institute of Technology, Thammasat University, Pathum Thani, Thailand
6422772077@*g.siit.tu.ac.th*, 6422780138@g.siit.tu.ac.th, 6422780674@g.siit.tu.ac.th

***Abstract*—**In the age of digital transformation, mobile cloud environments require secure and efficient data access solutions, as the need to store and process large volumes of data continues to grow. Searchable encryption provides a promising method to ensure data security and privacy; however, high computational demands have limited its applicability, particularly in resource-constrained environments like mobile cloud computing. This paper proposes a novel approach to enhance the scalability and efficiency of searchable encryption by integrating optimized search algorithms, such as tree-based search, and extracting keywords for faster retrieval. Additionally, the use of blockchain technology is explored to bolster data integrity and transparency through a distributed ledger of access and modification history. Our findings highlight the potential of combining advanced searchable encryption techniques with blockchain for secure and efficient data storage solutions in mobile cloud environments, paving the way for practical applications in both academic and commercial sectors.

Keywords—Searchable Encryption, Multiple Keyword Search, Bloom filter, Auxiliary index, Keyword Search, Blockchain, Smart Contract, Data tree

I. INTRODUCTION

In the healthcare domain, managing personal health records (PHRs) securely and efficiently is paramount. PHRs contain sensitive patient data that necessitates robust security measures to guarantee confidentiality and integrity. Traditional data storage solutions often lack the functionalities for granular access control and data redaction, posing privacy concerns. Additionally, searching for specific health information within a large dataset of PHRs can be a cumbersome and time-consuming process.

This paper proposes a novel approach for secure and searchable PHR management that leverages the combined strengths of redactable blockchain and searchable encryption. Our system ensures the confidentiality of patient data while enabling authorized users to efficiently search and retrieve relevant health information. The proposed architecture utilizes a redactable blockchain to store the cryptographic hashes of encrypted PHRs. This allows for data modification and redaction while maintaining the integrity of the blockchain ledger. Searchable encryption techniques are implemented to facilitate efficient keyword-based searches on the encrypted PHRs, enabling authorized users to locate specific health information without decrypting the entire dataset.

This work contributes to the field of secure PHR management by introducing a system that offers the following key benefits:

Enhanced Data Security: The redactable blockchain guarantees the immutability and authenticity of the stored PHR data. Additionally, searchable encryption protects the confidentiality of patient information even at rest.

Fine-grained Access Control: The system enforces granular access control mechanisms, ensuring that only authorized users can access specific health records.

Efficient Search Functionality: The implementation of searchable encryption techniques allows for efficient keyword-based searches on the encrypted PHRs, streamlining the retrieval process for authorized users.

Data Redaction Capabilities: The redactable blockchain empowers authorized entities to redact sensitive data from PHRs while preserving the integrity of the overall system.

This paper presents the design and implementation details of our proposed system. We evaluate the system's performance through extensive simulations and analyze its effectiveness in securing and facilitating searches on PHR data. The results demonstrate the feasibility and efficacy of our approach for secure and searchable PHR management in the healthcare domain.

II. RELATED WORK

Extensive research in the field of searchable encryption has established a foundational basis for our project, providing critical insights and a range of methodological approaches..

Fugkeaw et al. [1] proposed a secure and efficient searchable encryption scheme for cloud data warehouses, integrating partial homomorphic encryption, inverted indexes, and B+Tree indexing. Their approach also incorporates blockchain technology to automate search permission verification, user authentication, and result validation, ensuring scalability and immutability. The use of B+Tree indexing reduces the search space, improving performance and resource efficiency. Their system demonstrated effective support for concurrent OLAP queries. Future work will focus on achieving fully forward security for keyword updates in searchable encryption.

Yang et al. [2] introduced a dual traceable distributed attribute-based encryption (ABKS) scheme, integrating zero-knowledge proof for malicious data source tracing and white-box traitor tracing for users leaking secret keys. Their scheme, secure under the IND-CKCPA model, ensures anonymity and traceability for both data sources and users. They also proposed updatable and transferable message-lock encryption (UT-MLE), enabling block-level ciphertext updates and ownership transfers, with proven security properties. Combining DT and UT-MLE, they developed the ABSE with ownership transfer system (DTOT), demonstrating enhanced efficiency and robust security features through extensive experiments.

Al-Badarneh et al. [3] evaluated the performance of two Bloom join algorithms with varying Bloom filter sizes using Hadoop Pseudo Distributed Mode. Their study demonstrated that the optimal Bloom filter size depends on the input dataset and that tuning the size significantly impacts performance. The authors found that Map-side Bloom join is more efficient under specific conditions, such as when physical memory is available, while Reduce-side Bloom join is more general and can perform better with certain filter sizes. They recommended smaller Bloom filter sizes when memory constraints exist, and found that small to medium filter sizes in Reduce-side joins yielded better performance in terms of elapsed time.

Hou et al. [4] addressed the challenge of maintaining confidentiality in remote forensic investigations by enabling multiple keyword searches over encrypted data.

Cai et al. [5] proposed a secure and reliable content search mechanism for encrypted files stored in decentralized storage services. They addressed the challenge of minimizing overhead by preserving file and index locality in searchable encryption. Recognizing the increased threats in decentralized settings, where both clients and service peers may engage in malicious behaviors, the authors incorporated fair payment mechanisms using verifiable searchable encryption and blockchain. This ensures that service peers are compensated only for correct services. Their implementation, based on Python and Ethereum smart contracts, demonstrates the efficiency of their encrypted search service and the feasibility of their fairness design through experimental results.

Yang et al. [6] introduced the concept of searchable encryption and multi-key searchable encryption (MKSE), providing a general description of MKSE along with an analysis of its functions and processes. They presented a concrete MKSE model based on the bilinear map and BDHI assumption, avoiding the strict requirements of ideal random oracles by generalizing security to the BDHI problem and elliptic curves. The authors proposed a new, efficient MKSE scheme where users can protect their privacy through encryption while performing keyword retrieval without decryption. The scheme allows file sharing for keyword-based retrieval, with security analysis addressing potential attacks. Their work emphasizes the growing importance of MKSE as a research area and highlights the need for further exploration in searchable encryption methods.

Li et al. [7] proposed a personalized search scheme for mobile cloud environments, focusing on efficient and secure updates. Their scheme achieves key security features, including confidentiality, trapdoor unlinkability, forward and backward security, collusion resistance, and hidden access policies. Through extensive experiments, they demonstrated that their approach outperforms existing schemes in terms of functionality, computation, and communication overhead. The authors suggest future work to explore the dynamic nature of searchable encryption and to evaluate their proposal on real-world cloud platforms.

III. **PRELIMINARIES**

This section outlines the background of our project, covering key concepts such as blockchain technology, Auxiliary trees, Bloom filters, and Trapdoors. It provides a brief explanation of blockchain's role as a decentralized ledger, Bloom filters for efficient search optimization, Auxiliary trees for data organization, and Trapdoors for implementing one-way functions.

A. BLOCKCHAIN

Blockchain technology is a secure, decentralized, and immutable ledger that records and tracks digital data in a transparent and tamper-resistant way. It utilizes advanced public key encryption and cryptographic hashing methods. Each data block, once validated, becomes cryptographically linked to the previous one, creating a chain. A typical blockchain consists of the cryptographic hash of the preceding block, a timestamp, a nonce, and transaction details. Blockchain networks also support smart contracts, which are self-executing programs that run on the blockchain.

B. BLOOM FILTER

A Bloom filter is a probabilistic data structure used for efficient membership testing, where it allows for fast queries at the expense of permitting some false positives, but guaranteeing no false negatives. This makes it particularly suitable for applications such as data warehousing, caching, and searchable encryption. The structure of a Bloom filter consists of a bit array, denoted as B, of length m, initially set to 0, with each bit representing either 0 or 1. Additionally, a set of k independent hash functions, $H = \{h_1, h_2, ..., h_k\}$, maps input elements to specific positions within the bit array.

When inserting an element x into the filter, the element is hashed with each of the k hash functions, marking the corresponding positions in the array as 1. To check whether an element y is present in the set, the filter hashes y using the same k functions and verifies if all the corresponding bit positions are set to 1. If any bit is 0, the element is definitely not in the set, but if all bits are 1, the element is likely present, though this could result in a false positive. The false positive probability P_{fp} is given by the formula,

$$P_{fp} = (1 - e^{-kn/m})^k$$

where n is the number of elements inserted into the Bloom filter, m is the size of the bit array, and k is the number of hash functions used. To minimize the false positive rate, the optimal number of hash functions, k_{opt}, is given by,

$$k_{opt} = \frac{m}{n} ln2$$

One of the main advantages of Bloom filters is their space efficiency, particularly when handling large datasets. Bloom filters require a relatively small, fixed amount of memory compared to traditional data structures, making them suitable for environments with limited resources. However, a key limitation is that elements cannot be deleted from the filter, and there is always a probability of false positives. Despite these drawbacks, Bloom filters remain an effective solution for scenarios where membership queries need to be performed quickly and efficiently.

C. AUXILIARY TREE

The Auxiliary Search Tree is introduced as a supplementary data structure that improves the efficiency of operations on more complex or primary data structures. Typically tree-like, it is designed to optimize the primary structure's operations, especially in hierarchical or dynamic data contexts, ensuring faster or more efficient algorithms.

In this project, Auxiliary Search tree was implemented for structuring keywords into their own categories for efficient lookup. Each keyword is mapped to a list of corresponding PIDs containing it.

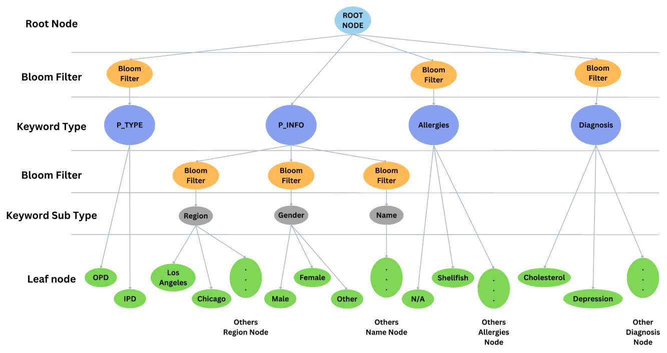


Fig. 1. Auxiliary Tree

The tree is organized into levels, describe as follow

1). Root Node: The Root Node is the entry point for queries, directing them to appropriate branches based on their type.
2). Bloom Filter(First Level): The Bloom Filter at this level quickly checks the presence of the keyword in the tree. If absent, the query is discarded, optimizing the search.
3). Keyword Type: The Keyword Type level categorizes the keyword into types such as P_TYPE, P_INFO, Allergies, and Diagnosis.
4). Bloom Filter(Second Level): A second Bloom Filter validates the presence of the keyword within the specific Keyword Type before proceeding to the Keyword Sub Type level.
5). Keyword Sub Type: The Keyword Sub Type level divides the P_INFO category into more specific subtypes, such as Region, Gender, and Name, for finer query categorization.
6). Leaf Node: The Leaf Node holds the final results, containing a mapping lists of Patient IDs (PIDs) that match the query criteria.

IV. **OUR PROPOSED SCHEME**

We propose searchable encryption in mobile cloud settings in the aspect of health patients records. This section presents the system overview and details of the construct of our proposed scheme.

A. System model

Comprehensive three-layered model designed for secure medical data processing, storage, and retrieval. The model integrates blockchain technology, smart contracts, and the InterPlanetary File System (IPFS) to ensure data integrity, confidentiality, and accessibility. Figure 1 presents the overall system model of our proposed system.

The following subsections provide a detailed overview of each layer and its associated processes.

1. Data Processing Layer

Objective: To process raw medical data for secure storage and subsequent retrieval.

Procedure

1. Keyword Extraction: The Data Uploader (DU) extracts pertinent keywords from raw medical records to facilitate indexing and searching.
2. Keyword Hashing: Extracted keywords undergo a hashing process to ensure anonymity and security during transmission.
3. Index Storage: Hashed keywords are stored in a cloud-based search structure for efficient indexing.
4. Record Encryption: Medical records are encrypted to maintain patient privacy.
5. Decentralized Storage: Encrypted medical files are uploaded to an IPFS-based cloud server, ensuring decentralized and resilient storage.

2. Cloud Layer

Objective: To manage the storage, search, and validation of encrypted medical data utilizing blockchain and auxiliary search structures.

Procedure

1. Query Hashing: The search engine hashes the user's query for secure processing.
2. Search Mechanism: An auxiliary search tree employing a Bloom filter processes the hashed query to identify potential matches. The search supports multiple keywords search and Boolean operations.
3. Hash Retrieval: Upon finding a match, the corresponding hash value is retrieved.
4. Blockchain Indexing: The system utilizes the retrieved hash to locate the specific index on the blockchain where the file resides.
5. Smart Contract Execution: Smart Contract 1 manages blockchain indexes and facilitates the retrieval of relevant encrypted files from IPFS.
6. Data Transmission: The encrypted file is securely transmitted to the mobile layer.

3. Mobile Layer

Objective: To provide authorized users with secure access to medical data.

Procedure:

1. Trapdoor Generation: A mobile user (MU) generates a "trapdoor," a secure token designed for accessing encrypted data.
2. User Authentication: Smart Contract 2 authenticates the user by validating the trapdoor.
3. Data Retrieval: Upon successful authorization, the system processes the search query and returns relevant encrypted medical records to the user.

Key Technological Components

a. Blockchain Technology: Ensures secure, immutable storage of indexes and facilitates data access management through smart contracts.

b. Smart Contracts: Two distinct smart contracts are employed,
   - Smart Contract 1: Manages indexing and storage operations.
   - Smart Contract 2: Handles user authentication and access control.

c. InterPlanetary File System (IPFS): Provides a decentralized file storage solution, ensuring high availability and integrity of encrypted medical records.

d. Bloom Filter and Auxiliary Search Tree: Enables efficient, probabilistic searching of hashed keywords while maintaining privacy.

Operational Workflow

1. Medical records undergo processing, encryption, and storage within the IPFS infrastructure.
2. Keywords are subjected to hashing and subsequently indexed on the blockchain.
3. Authorized users can conduct searches and retrieve encrypted files through a secure, authenticated process that leverages smart contracts and a Bloom filter-based search structure.

This model presents a robust, privacy-preserving approach to the storage and access of medical data across distributed systems. It places significant emphasis on user authentication and data confidentiality, addressing key concerns in the realm of digital health information management.

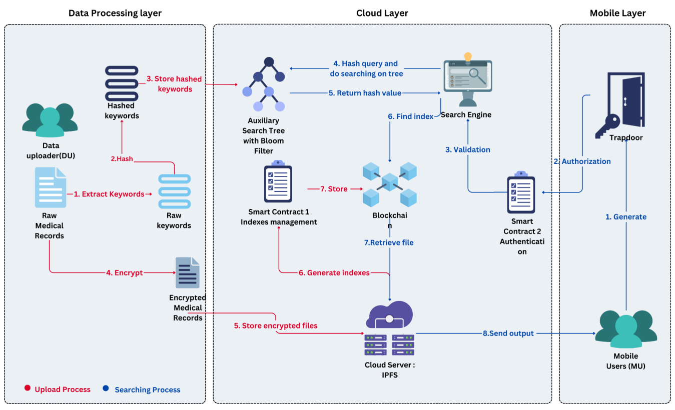


Fig. 2. Overall System model

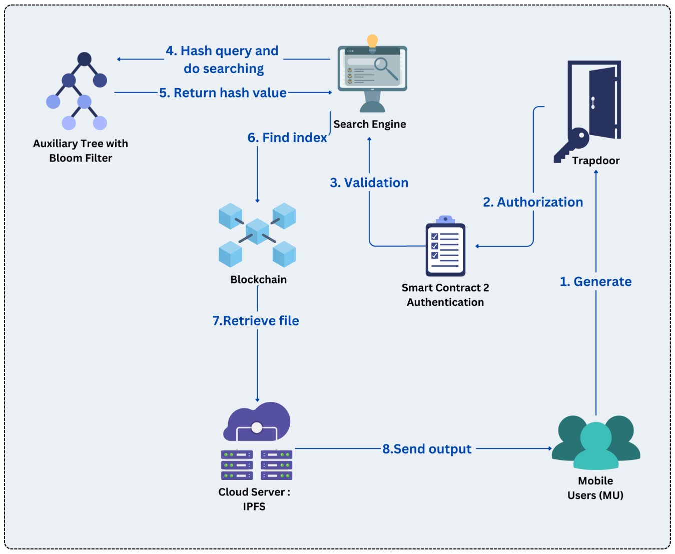


Fig. 3. Searching model

B. *Methodology*

TABLE 1: NOTATIONS USED IN OUR MODEL

Notation	Description
H_{Pw}	Hashed version of the user's password
Pw_U	User's password
PID	Patient IDs
CID	Content Identifiers
$HPID$	Hashed version of a Patient ID
$PubAddr_U$	Public blockchain wallet address of the user.
$BF_i \cdot contains(H_{Kwd_i})$	Bloom filter check for the presence of the keyword i

Step 1. User Authentication

Before performing any operation, the user U must authenticate using their public blockchain wallet address ($PubAddr_U$) and their registered password (Pw_U). Once user input is received, the server compares the password hash with stored hash password on Blockchain.

$$H_{Pw} = SHA256(Pw_U)$$
$$Auth(PubAddr_U, H_{Pw}) \rightarrow \{True,\ False\}$$

If the authentication result is $True$, the server creates a credential session for the user, granting them access to the system's functionalities.

Step 2. User Search Query

The user search process employs hashing to preserve the integrity of the input text and support multiple keyword searches and Boolean operations (e.g., AND, OR, NOT). Unlike encryption, hashing is a one-way function, which means the original input cannot be retrieved once hashed. However, it is used to match keywords efficiently in the system.

1) The user creates a search query Q comprising keywords. Where $Q = \{Kwd_1, Kwd_2, ..., Kwd_n\}$, hash the query and then send it to the server.

$$H_{Kwd_i} = SHA256(Kwd_i),\ \forall Kwd_i \in Q$$

1.1) Handling PID range searching, and specific PIDs

While the system primarily leverages Bloom filters for keyword-based searches, it also supports efficient retrieval of:

Individual PIDs: Users can directly query for a specific PID by including it in the search query. The system efficiently retrieves the corresponding data if the PID exists within the IPFS.

$$Q = \{PID_n\}$$

PID Ranges: The system can handle queries specifying a range of PIDs. For instance, a search query like "1-100" would retrieve data associated with PIDs from 1 to 100 (inclusive).

$$Q = \{PID_n - PID_k\}$$

2) Before traversing the auxiliary tree, the server uses Bloom filters to determine which nodes contain the hashed keywords. Nodes without matches are skipped, reducing unnecessary processing.

$$BF_i \cdot contains(H_{Kwd_i}),\ \forall i \rightarrow \{True,\ False\}$$

If $BF_i \cdot contains(H_{Kwd_i}) \rightarrow True$, proceed to search the corresponding auxiliary tree node. Skip nodes where BF check fails.

3) The server retrieves all Patient IDs (PIDs) mapped to the searched keywords. For queries with Boolean operators, the server computes the final list of PIDs by combining, intersecting, or excluding results according to the operation logic. For each valid H_{Kwd_i}, retrieve the set of PIDs

$$PID_{Kwd_i} = \{PID_1, PID_2, \ldots, PID_k\}$$

For handling Boolean Operation, our search model combine results from multiple PID_{Kwd} set based on the logic.

3.1) Boolean Search Operations

After identifying the relevant PIDs, the system refines the search results using Boolean operations. These operations, including AND, OR, and NOT, are applied to the sets of PIDs retrieved from the inverted index at each node.

AND Operation

$$PID_{result} = \bigcap_{i=1}^{n} PID_{Kwd_i}$$

OR Operation

$$PID_{result} = \bigcup_{i=1}^{n} PID_{Kwd_i}$$

NOT Operation

$$PID_{result} = PID_{Kwd_1} \setminus \bigcup_{i=2}^{n} PID_{Kwd_i}$$

4) The server hashes the finalized PIDs and queries the blockchain to retrieve Content Identifiers (CIDs) associated with them.

$$HPID_j = SHA256(PID_j),\ \forall PID_j \in PID_{result}$$
$$CID_{HPID_j} = Blockchain.query(HPID_j)$$

5) The server sends the search results, including the retrieved CIDs, back to the client for display and download encrypted medical records from IPFS.

$$\{CID_{HPID_1}, CID_{HPID_2}, \ldots, CID_{HPID_k}\}$$

Phase 0 : Data Uploading Phase

The secure storage of sensitive files begins with a structured process to ensure confidentiality, integrity, and accessibility within a decentralized environment. The initial step involves preparing raw data files, typically in .csv format, containing structured information such as IDs, names, and attributes like gender or region. These files undergo keyword extraction to generate metadata, enabling efficient search operations later.

To safeguard confidentiality, the files are encrypted using AES (Advanced Encryption Standard). A user-provided key is hashed via MD5 to generate a 16-byte encryption key, which is used to encrypt the file content after applying PKCS7 padding for block alignment. The encrypted file is then saved with a .dat extension, obfuscating the original format for added security.

The encrypted files are uploaded to IPFS (InterPlanetary File System), a decentralized storage network. During this process, each file is assigned a unique Content Identifier (CID), ensuring its integrity and enabling retrieval without reliance on a centralized server. This combination of AES encryption and IPFS storage provides a secure, scalable, and decentralized solution for data storage while preserving data confidentiality and accessibility.

Algorithm 1: AES Encryption

```
1: FUNCTION enct_file(input_path,output_path, user_key):
2: INPUT:
3:         input_path ← Path to the raw file
4:         output_path ← Path to save the encrypted file
5:         user_key ← User-provided key
```

```
6: OUTPUT: Encrypted file saved at output_path
7: BEGIN:
8:   GENERATE ENCRYPTION KEY:
9:        key ← MD5(user_key)
10:  CREATE AES CIPHER:
11:       cipher ← AES.new(key, ECB mode)
12:  READ RAW DATA:
13:       file_data ← READ_FILE(input_path)
14:  PAD DATA TO BLOCK SIZE:
15:       padded_data ← PAD(file_data)
16:  ENCRYPT DATA:
17:       encrypted_data ← cipher.encrypt(padded_data)
18:  WRITE ENCRYPTED DATA:
19:      WRITE_FILE(output_file_path, encrypted_data)
20:  PRINT "File encrypted and saved to:", output_path
21: END
```

Phase 1 : Keywords, Auxiliary Tree Generation Phase

In the Keywords and Auxiliary Tree Generation Phase, raw medical CSV files are processed to extract meaningful keywords, which are then mapped into a hierarchical structure called the Auxiliary Generation Tree for efficient organization and retrieval. These keywords, derived from specific fields like medical conditions or procedures, are hashed using a cryptographic algorithm such as SHA-256 to ensure privacy while maintaining integrity for indexing and search purposes.

Algorithm 2: Keywords Extraction

```
1: BEGIN
2: DEFINE columns_to_extract = ['PID', 'Name', 'Gender', 'Type', 'Allergies', 'Region', 'Diagnosis']
3: FOR each file IN input_directory DO
4:    IF file ENDS WITH '.csv' THEN
5:        SET data = read_csv(file)
6:        IF 'Patient ID' IN data.columns THEN
7:            RENAME column 'Patient ID' TO 'PID'
8:        END IF
9:        SET extracted_data = data[columns_to_extract]
10:       WRITE extracted_data TO output_path
11:   END IF
12: END FOR
13: END
```

This algorithm extracts specific columns from CSV files containing patient data (e.g., PID, Name, Gender). The required columns are validated, and any missing columns are discarded. The data is then written to text files for further processing. This process ensures that the extracted data is clean and ready for the next phases of the workflow.

Phase 2 : Search Index Generation Phase

This phase establishes the foundation for efficient and secure data retrieval by creating a search index that maps hashed Patient IDs (hPIDs) to their respective Content Identifiers (CIDs).The primary objective of this phase is to securely generate, store, and manage mappings between hPIDs and CIDs on a blockchain. By leveraging blockchain technology, the mappings are immutable and tamper-proof, enhancing trust and data integrity.

Algorithm 3: Smart Contract for Mapping, Storing, and Retrieve CID (Solidity)

```
1: BEGIN
2:    DECLARE mapping HPidToCid
3:    DECLARE event MappingAdded(HPid, cid)
4:
5:    FUNCTION addMapping(HPid, cid)
6:      HPidToCid[HPid] = cid
7:      EMIT MappingAdded(hashedPid, cid)
8:    END FUNCTION
9: END
```

The search index is generated by mapping hashed PIDs (hPIDs) to their corresponding Content Identifiers (CIDs) obtained from Phase 0. These mappings are uploaded as blockchain transactions to ensure secure, immutable storage.

Algorithm 4: Uploading transactions

```
1: BEGIN
2:    SET ganache_url
3:    CREATE web3 instance WITH ganache_url
4:    SET contract_address
5:    SET abi_file_path
6:    SET contract to web3.eth.contract
7:          with contract_address and contract_abi
8:    SET private_key
9:    SET account =
      web3.eth.account.from_key(private_key).address
10:   FUNCTION hash_pid(pid):
11:      RETURN SHA256 hash of pid
12:   END FUNCTION
13:
14:   FUNCTION send_transaction(hpid, cid):
15:      SET nonce =
         web3.eth.get_transaction_count(account)
16:      SET transaction to
         contract.functions.addMapping(hpid, cid).
         build_transaction with gas and nonce
17:      SET signed_txn =
         web3.eth.account.sign_transaction
         (transaction, private_key)
18:      SET tx_hash =
        web3.eth.send_raw_transaction
        (signed_txn.raw_transaction)
19:     SET receipt =
        web3.eth.wait_for_transaction_receipt(tx_hash)
20:     PRINT "Transaction successful: " + tx_hash.hex()
21:   END FUNCTION
22:   CALL send_transaction with hpid and cid
23: END
```

The process involves uploading transactions to a blockchain using the Web3.py library. It interacts with a smart contract to map a hashed patient ID (hPIDs) to a Content Identifiers (CIDs). The steps include setting up a Web3 connection, specifying the contract's address and ABI, and signing the transaction with a private key. After signing, the transaction is sent to the blockchain, and the code waits for a confirmation of its success.

Key functions include,

- hash_pid: Hashes the patient ID using SHA-256.
- send_transaction: Sends the transaction to the smart contract to map the patient ID to the customer ID.

Phase 3: Search Query Phase

In this phase, the system enables users to perform secure and efficient searches by leveraging hashed queries, auxiliary tree searching, and parallel processing. This phase is critical for translating user inputs into actionable queries that interact with both auxiliary tree structures and blockchain-stored data.

Algorithm 5: User Authentication

```
1: FUNCTION authenticate_user(user_address, password):
2:    TRY
3:       SET password_hash =
         call hash_password function with password
4:       SET stored_hash =
         call auth_contract.functions.getStoredHash
         (user_address)
5:       IF stored_hash == password_hash THEN
6:          RETURN call
            auth_contract.functions.authenticate
            (user_address, password_hash)
7:       ELSE
8:          RETURN False
9:       END IF
10:   EXCEPT Exception as e
11:      RETURN False
12: END FUNCTION
```

This algorithm provides a secure method for user authentication using blockchain-based credentials. It compares a hashed password entered by the user with a stored hash on the blockchain. If the hashes match, the user is authenticated; otherwise, access is denied. The algorithm leverages smart contract functionality for decentralized authentication.

Algorithm 6: Search Auxiliary

```
1: FUNCTION search_pids_in_range(start, end):
2:    RETURN list of PIDs from start to end (inclusive)
3: END FUNCTION
4: FUNCTION hash_keyword(keyword):
5:    RETURN SHA256 hash of the keyword
6: END FUNCTION
7: FUNCTION search_auxiliary_with_bloom(app, query):
8:    SET hashed_query = call hash_keyword with query
9:    SET matched_files = empty list
10:   FOR each bloom_filename IN files_in_directory
(bloom_output_path) DO
11:      IF bloom_filename ENDS WITH ".bloom" THEN
12:         SET bloom_filepath =
            join bloom_output_path with bloom_filename
13:         TRY
14:            OPEN bloom_filepath FOR reading
15:            LOAD bloom_filter from the file
16:            IF hashed_query IN bloom_filter THEN
17:               APPEND
                  bloom_filename (without ".bloom")
                  to matched_files
18:            END IF
19:         EXCEPT Exception
20:            CONTINUE
21:         END TRY
22:      END IF
23:   END FOR
24:   IF matched_files IS EMPTY THEN
25:      RETURN empty set
26:   END IF
27:   FUNCTION search_file(file_path):
28:      SET results = empty set
29:      TRY
30:         OPEN file_path FOR reading
31:         READ lines from the file
32:         FOR each line i IN lines DO
33:            IF hashed_query IN line THEN
34:               IF next line exists (i + 1) THEN
35:                  ADD PIDs from next line (split by ", ")
                     to results
36:               END IF
37:               BREAK
38:            END IF
39:         END FOR
40:      EXCEPT Exception
41:         RETURN empty set
```

```
42:         END TRY
43:         RETURN results
44:     END FUNCTION
45:     SET results = empty set
46:     WITH ThreadPoolExecutor AS executor
47:         SET file_paths = list of full paths for matched_files
48:         FOR each file_path IN file_paths DO
49:             SET file_results = call search_file(file_path)
50:             ADD file_results to results
51:         END FOR
52:     RETURN results
53: END FUNCTION
```

This algorithm enables efficient searching for specific keywords across multiple files using Bloom filters. First, it hashes the search query and checks against Bloom filter files. If a match is found, a parallel search is performed on the relevant files to retrieve associated patient IDs (PIDs). The parallel processing improves the performance of large-scale searches.

Phase 4: Data Retrieval Phase

This phase focuses on securely retrieving data files from the decentralized IPFS network based on the Content Identifier (CID) provided during the search query phase. It ensures that users can access the associated files in a seamless and efficient manner while maintaining security and resource management.

Algorithm 7: Download file from IPFS

```
1: FUNCTION download_file():
2:     IF "username" NOT IN session THEN
3:         REDIRECT to home page ("/")
4:     END IF
5:     SET cid = get 'cid' from request.args
6:     IF cid IS NOT PROVIDED THEN
7:         RETURN HTTP code 400
8:     END IF
9:     SET ipfs_url
10:    SET tmp_file_path
11:    TRY
12:        SET response =
           make a GET request to ipfs_url with stream=True
13:        IF response.status_code != 200 THEN
14:            RETURN HTTP code 500
15:        END IF
16:        OPEN tmp_file_path for writing in binary mode
17:        FOR each chunk in response content DO
18:            WRITE chunk to tmp_file
19:        END FOR
20:        RETURN send_file response with tmp_file_path, as
attachment, and filename "{cid}.dat"
21:    EXCEPT Exception
22:        RETURN HTTP code 500
23:    FINALLY
24:        START new thread to REMOVE
           tmp_file_path after delay
25: END FUNCTION
```

This algorithm handles the process of downloading a file from the IPFS network using a provided Content Identifier (CID). It first checks for user authentication, retrieves the CID from the request, and forms the IPFS URL. If the CID is missing, the algorithm returns an error. A GET request is made to the IPFS URL, and the response is written to a temporary file. Once the file is saved, it is sent to the client as a downloadable attachment. In case of any errors during the process, the algorithm returns a server error. Finally, a background thread is used to clean up the temporary file after a delay, ensuring proper resource management.

V. IMPLEMENTATION AND EXPERIMENT

This section describes the implementation of our mobile search encryption services through the prototype design, the experiments to test its functional modules and the cost of gas used to execute the smart contract.

A. Prototype System

The prototype system for our searchable encryption service uses HTML for the front-end interface and Python Flask for backend logic. User authentication is handled via Smart Contracts 2 on the Ethereum blockchain, while IPFS ensures decentralized and secure file storage. Bloom filters and parallel processing to optimize search queries.

Figure 4 illustrates the user flow, starting with login verification through Smart Contract 2 “AuthContract.Sol” and Once the user is verified, the server creates a session for the user to use services, then proceeds to the search portal. Users can feed the search query, Finally the search results display intersected PIDs with their corresponding CIDs, retrievable via IPFS. The system integrates multiple technologies, such as blockchain, IPFS, AES encryption, and Bloom filters, into a cohesive architecture to deliver efficient and privacy-preserving search services.

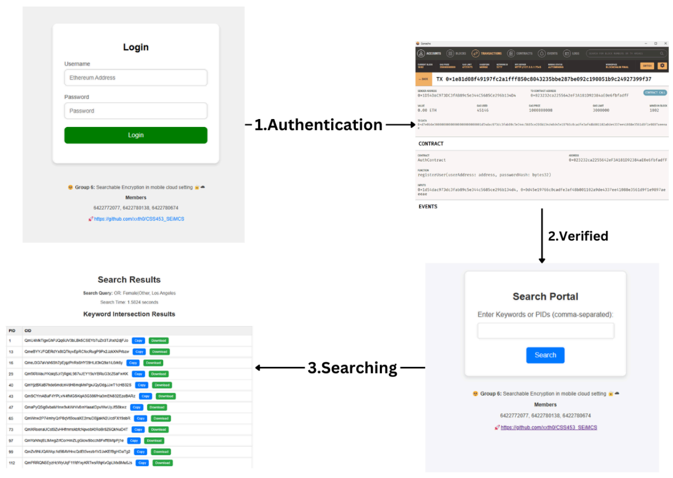
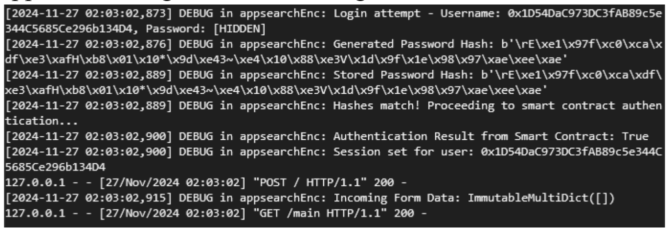


Fig. 4. Services of our project

B. Experiment

In this section, we demonstrate the login authentication process using smart contracts. When a user provides credentials, the server hashes the password using the Keccak algorithm and interacts with a smart contract deployed on the blockchain. The smart contract validates the hashed password against the stored hash on-chain. Upon successful validation, the server grants the user access and establishes a session for application usage, as shown in Figure 4.

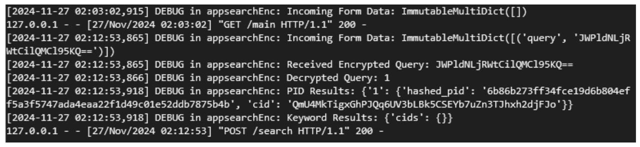


```
[2024-11-27 02:03:02,873] DEBUG in appsearchEnc: Login attempt - Username: 0x1D54DaC973DC3fAB89c5e
344C5685Ce296b134D4, Password: [HIDDEN]
[2024-11-27 02:03:02,876] DEBUG in appsearchEnc: Generated Password Hash: b'\rE\xe1\x97f\xc0\xca\x
df\xe3\xafH\xb8\x01\x10*\x9d\xe43~\xe4\x10\x88\xe3V\x1d\x9f\x1e\x98\x97\xae\xee\xae'
[2024-11-27 02:03:02,889] DEBUG in appsearchEnc: Stored Password Hash: b'\rE\xe1\x97f\xc0\xca\xdf\
xe3\xafH\xb8\x01\x10*\x9d\xe43~\xe4\x10\x88\xe3V\x1d\x9f\x1e\x98\x97\xae\xee\xae'
[2024-11-27 02:03:02,889] DEBUG in appsearchEnc: Hashes match! Proceeding to smart contract authen
tication...
[2024-11-27 02:03:02,900] DEBUG in appsearchEnc: Authentication Result from Smart Contract: True
[2024-11-27 02:03:02,900] DEBUG in appsearchEnc: Session set for user: 0x1D54DaC973DC3fAB89c5e344C
5685Ce296b134D4
127.0.0.1 - - [27/Nov/2024 02:03:02] "POST / HTTP/1.1" 200 -
[2024-11-27 02:03:02,915] DEBUG in appsearchEnc: Incoming Form Data: ImmutableMultiDict([])
127.0.0.1 - - [27/Nov/2024 02:03:02] "GET /main HTTP/1.1" 200 -
```

Fig. 5. The Process of login authentication by comparing hashed passwords using Smart Contract 2 to grant user sessions.

Once the login is successful, the user is directed to the search portal, where they can perform various types of searches. The portal supports Boolean search, multi-keyword search, specific PID search, and range-based PID search.

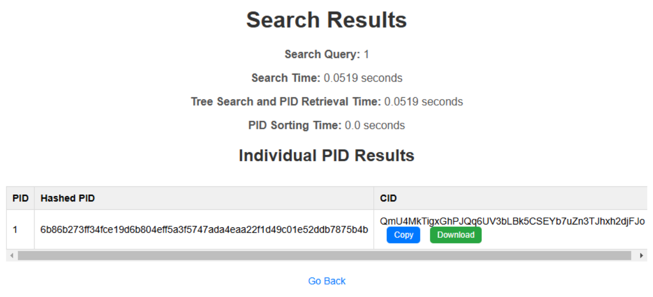
When a user initiates a search, the client hashes the search keywords locally to ensure data privacy and transmits the hashed keywords to the server. The server processes the hashed queries by interacting with the auxiliary data structures and the blockchain via Smart Contract 1 to retrieve relevant PIDs and CIDs. The query is shown in Figure 5.



```
[2024-11-27 02:03:02,915] DEBUG in appsearchEnc: Incoming Form Data: ImmutableMultiDict([])
127.0.0.1 - - [27/Nov/2024 02:03:02] "GET /main HTTP/1.1" 200 -
[2024-11-27 02:12:53,865] DEBUG in appsearchEnc: Incoming Form Data: ImmutableMultiDict([('query', 'JWPldNLjR
WtCilQMCl95KQ==')])
[2024-11-27 02:12:53,865] DEBUG in appsearchEnc: Received Encrypted Query: JWPldNLjRWtCilQMCl95KQ==
[2024-11-27 02:12:53,866] DEBUG in appsearchEnc: Decrypted Query: 1
[2024-11-27 02:12:53,918] DEBUG in appsearchEnc: PID Results: {'1': {'hashed_pid': '6b86b273ff34fce19d6b804ef
f5a3f5747ada4eaa22f1d49c01e52ddb7875b4b', 'cid': 'QmU4MkTigxGhPJQq6UV3bLBk5CSEYb7uZn3TJhxh2djFJo'}}
[2024-11-27 02:12:53,918] DEBUG in appsearchEnc: Keyword Results: {'cids': {}}
127.0.0.1 - - [27/Nov/2024 02:12:53] "POST /search HTTP/1.1" 200 -
```

Fig. 6. The search query of a user performing a search for PID 1.

After the server processes the search, the results are displayed to the user, showing the corresponding PID and its associated CID. Each result includes a "Copy" button for quick access to the CID and a "Download" button to retrieve the associated file directly from the IPFS (InterPlanetary File System) as seen in Figure 7.



Search Results

Search Query: 1

Search Time: 0.0519 seconds

Tree Search and PID Retrieval Time: 0.0519 seconds

PID Sorting Time: 0.0 seconds

Individual PID Results

PID	Hashed PID	CID
1	6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b	QmU4MkTigxGhPJQq6UV3bLBk5CSEYb7uZn3TJhxh2djFJo Copy Download

Go Back

Fig. 7. The front-end interface of the search result

Next, the performance testing, We compared the “Typical tree base structure” to the “Bloom Filter + Parallel Search Model”. Key metrics include the Tree Search and PID Retrieval Time and Blockchain Search Time, measured in milliseconds.

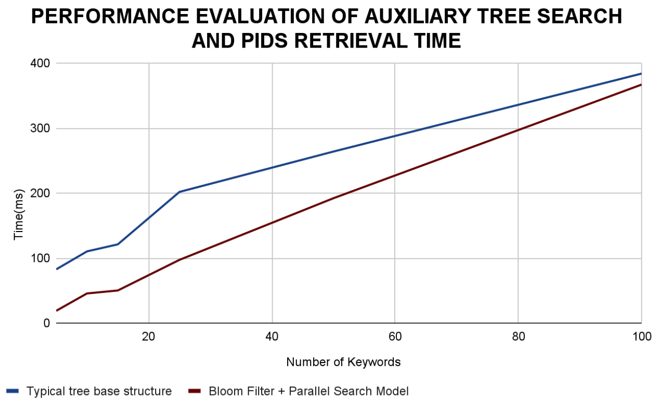


Fig. 8. The retrieval time of both the Typical tree base structure Model and the New Model varies as a function of the number of keywords.

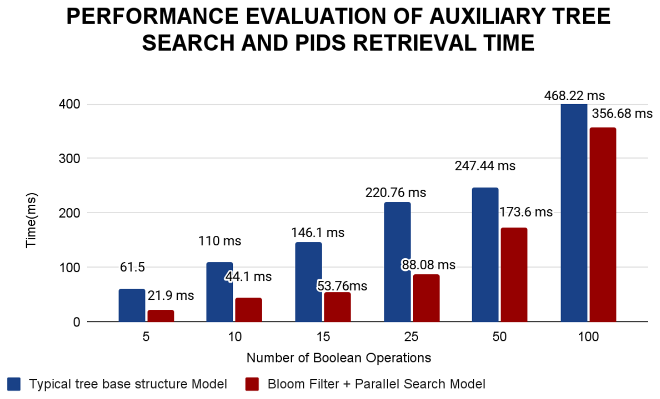


Fig. 9. The retrieval time of both the Typical tree base structure Model and the New Model varies as a function of the number of Boolean Operations.

These graphs compare the tree search and PID retrieval times between two models: the Typical tree base structure, which utilizes a pure Auxiliary Tree, and the new model, which incorporates both Bloom Filters and Parallel Searching. The retrieval time of both models varies with the number of keywords. As the number of keywords retrieved increases, the retrieval time rises. However, the new model, with Bloom Filters and Parallel Searching, exhibits a more efficient increase in retrieval time, highlighting its scalability and enhanced performance compared to the original model.

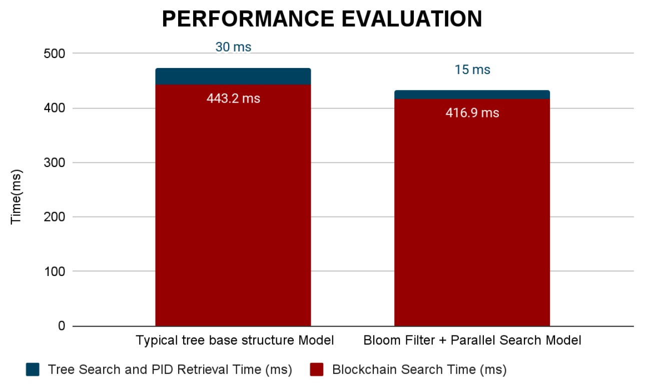


Fig. 10. Graphical representation of performance evaluation between Typical tree base structure model and new model with Bloom Filter, Parallel Search implementation.

This graph illustrates the time taken for searching and retrieving PIDs with the same settings. The data shows that the majority of the time is spent during the Blockchain search, which cannot be optimized. However, the new model demonstrates reduced retrieval time significantly compared to the Typical tree base structure model.

B. Processing Cost occurred in Blockchain

To assess the efficiency of the smart contracts implemented via blockchain technology, we focus on the associated gas costs. In our experiments, we simulated the blockchain network's gas fees necessary to execute these contracts. These contracts are designed to store PIDs, authenticate users.

A gas limit of 6721975 Gwei was enforced in this experiment. A total of 1,000 PIDs were randomly generated, each paired with a randomly selected set of keywords.

Process	Gas Consumption (Gwei)	Cost (USD)
Adding each PID	97,082	0.378
Authentication	45,146	0.176
Create contract address	920,620	3.581

We evaluate the cost of deploying and utilizing the smart contract by calculating the gas fees associated with key operations. These operations include adding each PID, user authentication, which is user registration to the AuthContract smart contract, and creating contract addresses, which is AuthContract and StoreCIDs smart contract. The gas cost is expressed in Gwei, and we convert it to USD for a clearer understanding of the actual costs incurred.

The evaluation results show that the gas cost for adding each PID is 97,082 Gwei, equivalent to approximately 0.378 USD. For user authentication, the gas cost is 45,146 Gwei, roughly equal to 0.176 USD. Creating a contract address incurs a gas cost of 920,620 Gwei, corresponding to around 3.581 USD.

The evaluation results provide valuable insights into the advantages of deploying and using the smart contract in a practical setting. It enhances security,and privacy and offers a robust and efficient solution for decentralized data storage and retrieval.

VI. **Conclusion**

In this paper, we have proposed an innovative approach for secure and efficient health data management using a combination of searchable encryption, blockchain technology, and decentralized storage. By leveraging Bloom filters, Auxiliary Trees, and parallel searching, we demonstrate significant improvements in search efficiency, particularly when handling encrypted medical records in mobile cloud environments.

Our model ensures data confidentiality and integrity by utilizing AES encryption for file security, hashing for keyword privacy, and blockchain for tamper-proof record keeping. The Bloom filter reduces unnecessary searches by eliminating irrelevant data early in the query process, while the parallel search mechanism enhances performance by distributing the query load across multiple processing threads.

The experimental results show that our system outperforms the original model, which relies solely on an Auxiliary Tree, particularly in terms of retrieval time and search efficiency. Although the blockchain search remains a bottleneck, the inclusion of Bloom filters and parallel processing significantly reduces the overall processing time for queries with multiple keywords.

In conclusion, the integration of advanced encryption techniques, blockchain-based indexing, and optimized search strategies offers a robust and scalable solution for managing encrypted health data. While there is still room for optimization in the blockchain search phase, our approach contributes valuable insights into the future of secure, privacy-preserving data management systems in mobile cloud environments.

Future work will focus on further reducing the overhead in blockchain search operations and enhancing the system's ability to handle larger datasets with even higher efficiency. This paper paves the way for more practical and secure applications in the fields of digital health and cloud computing.

REFERENCES

[1] S. Fugkeaw, L. Hak, and T. Theeramunkong, "Achieving Secure, Verifiable, and Efficient Boolean Keyword Searchable Encryption for Cloud Data Warehouse," *IEEE Access*, vol. 12, pp. 49848 - 49864, Mar. 2024, doi: 10.1109/ACCESS.2024.3383320.

[2] Y. Yang and R. H. Deng, "Dual Traceable Distributed Attribute-Based Searchable Encryption and Ownership Transfer," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 247 - 262, Jan.–Mar. 2023, doi: 10.1109/TCC.2021.3090519.

[3] A. Al-Badarneh and H. Najadat, "Performance Evaluation of Bloom Filter Size in Map-Side and Reduce-Side Bloom Joins," in *Proc. 2017 8th Int. Conf. Inf. Commun. Syst. (ICICS)*, Irbid, Jordan, Apr. 2017, doi: 10.1109/IACS.2017.7921965.

[4] S. Hou, T. Uehara, and S. M. Yiu, "Privacy Preserving Multiple Keyword Search for Confidential Investigation of Remote Forensics," in *Proc. 2011 3rd Int. Conf. Multimedia Inf. Netw. Secur. (MINES)*, Shanghai, China, Nov. 2011, pp. 2162-8998, doi: 10.1109/MINES.2011.90.

[5] C. Cai, J. Weng, X. Yuan, and C. Wang, "Enabling Reliable Keyword Search in Encrypted Decentralized Storage With Fairness," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 1, pp. 131 - 144, Jan.–Feb. 2021, doi: 10.1109/TDSC.2018.2877332.

[6] J. Yang and Z. Liu, "Multi-Key Searchable Encryption Without Random Oracle," in *Proc. 2014 Int. Conf. Intell. Netw. Collaborative Syst. (INCoS)*, Salerno, Italy, Sep. 2014, doi: 10.1109/INCoS.2014.143.

[7] H. Li and D. Liu, "Personalized Search Over Encrypted Data With Efficient and Secure Updates in Mobile Clouds," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 1, pp. 97 - 109, Jan.–Mar. 2018, doi: 10.1109/TETC.2015.2511457.