Implementation of A Blockchain-based Searchable Encryption for Securing Contact Tracing Data

*Zheng Yao Ng, Iftekhar Salam**

School of Computing and Data Science, Xiamen University Malaysia, Malaysia
swe1904870@xmu.edu.my, iftekhar.salam@xmu.edu.my

Abstract

We developed a blockchain-based multi-keyword searchable encryption scheme for securing COVID-19 contact tracing data. In this scheme, we used AES-GCM to encrypt the contact tracing data, guaranteeing that only authorized users can perform decryption. Our scheme employed a Ciphertext-Policy Attribute-based searchable encryption to encrypt the search index, ensuring only users with appropriate attributes can perform search operations. The scheme supports dynamic updates of the search index. The blockchain-based storage with smart contract ensures immutability and non-repudiation of storage and retrieval. Overall, the evaluation of the scheme shows that it works efficiently without compromising the security goals. This is one of the first works to implement a solution for secure storage and search of contact tracing data with blockchain-based searchable encryption. Compared to the existing searchable contact tracing schemes, it provides more features and maintains efficiency even if a large search index is used.

Keywords: Contact tracing, Attribute-based searchable encryption, Blockchain, Smart contract, Privacy preserving encryption

1 Introduction

The COVID-19 epidemic changed the economy and lifestyles worldwide. The government of many nations implemented systematic control techniques using mobile applications like TraceTogether [1], PriLok [2], Pronto-C2 [3], DP3T [4], BeepTrace [5], and MySejahtera [6]. These applications operate by processing personal information such as names, ID card numbers, current locations, and recently visited locations. The collected data is used to monitor COVID-19 virus-infected clusters. Therefore, the security of the information possesses a high priority. Currently, in Malaysia, only a limited number of research addressing the security of confidential data stored in the cloud have been published. Several of the existing works highlight the risk posed to privacy when data is stored in the cloud without adequate security measures. Jung et al. [7] found that 70% of location data exposes owners' precise details such as residence, workplace, hobbies, and routines. Thus such data may cause severe violations of the security and privacy of the owner. Aside from that, on October 2021, the Ministry of Health of Malaysia (MOH) reported that a third party had misused the MySejahtera register functions API to send fake emails and SMS to users. This incident may have resulted in the collection of certain information by a third party and raised concerns among the public regarding the system's security. Cloud storage security should also be a concern, along with the security issues related to the application. Most users maintain "trusted-but-curious" attitudes towards cloud storage; however, numerous cases [8-9] have shown that even without an active adversary, the database may be accidentally leaked. Hence, appropriate and effective practices must be applied to sensitive information before outsourcing to the cloud.

The confidentiality of contact tracing data can be protected using encryption algorithms before outsourcing the data to the cloud. However, the system's usability will be affected if the cloud users cannot search over the encrypted data. The naïve way for a receiver to search for a particular desired data requires downloading and decrypting the whole encrypted dataset and then searching for the desired data in plaintext. This will significantly decrease the efficiency of the data analysis or management process if a complete encrypted dataset needs to be downloaded before every operation. Hence, searchable encryption schemes were introduced to tackle the lack of efficiency in the searchability of encrypted data. Incorporating searchable encryption with blockchain might be a viable solution to address the security of the data without affecting functionality and efficiency. Therefore, in this work, we implement a searchable encryption scheme using blockchain for contact tracing data and assess its viability to assure different security goals while maintaining efficiency. This is an extended version of the work reported at ISPEC 2022 [10].

1.1 Related Work

The COVID-19 pandemic has recently brought attention to contact tracing. Only a few schemes address the problem of searchable contact tracing. Many contact tracing systems use blockchain for security features [3, 5, 11]. These schemes focus on maintaining privacy but do not include search functionality. A few existing schemes propose to process contact tracing data in a searchable, cryptographically privacy-preserving manner. Tahir et al. [12] implemented a blockchain-based searchable encryption scheme using Hyperledger Fabric (HLF) for secure contact tracing. In this

*Corresponding Author: Iftekhar Salam; E-mail: iftekhar.salam@xmu.edu.my
DOI: 10.53106/160792642024032502007

scheme, users register with the application server, uploading encrypted data to the blockchain. When a COVID-19 case is confirmed, the hospital notifies the enterprise, triggering decryption of patient and close contact data. However, a single entity, the system developer, handles all encryption and decryption, raising privacy concerns. This can result in undesirable situations where the third party misuses user data. Also, under this system, authorized personnel are determined by the organization. Company administrators determine authorization to view contact tracing data. If a malicious person is enrolled in the system with an insider's help, the system will not be subject to access policies, compromising data confidentiality. Also, this scheme only supports searching for a single keyword.

A surveillance system with searchability on encrypted COVID-19 data was proposed by Nabil et al. [13]. The similarity values between visitor and patient embedding vectors are computed in this work without requiring data retrieval from the cloud. This work uses an offline key distribution center to distribute the cryptographic keys. Each party uses a separate key. The keys must be updated frequently to avoid security problems. This work prohibits the server from learning any data that helps the computation process. However, no emphasis was placed on the integrity of the results or data used in the computations. Moreover, this scheme does not support distributed storage of contact tracing data.

Since the searchable encrypted contact tracing data over the blockchain network is linked for data analysis, most schemes tend to directly process the data once it's retrieved from the smart contract. Consequently, little attention is paid to the retrieval of data for organizational review. This leads to a situation where only the results of the analysis can be viewed by the organization, but not the underlying data used for the analysis. Based on the review of the existing schemes, multi-keyword search functionality is more desirable than single-keyword search as COVID-19 cluster analysis typically involves large datasets. So, using a single keyword scheme is less efficient for the scheme mentioned above. Also, centralized storage should be avoided to combat data misusage [14].

1.2 Our Contribution

We adopted and implemented Guo et al.'s Attribute-Based Searchable Encryption (ABSE) [15]. Their scheme supports multi-owner to multi-user attribute-based single-keyword searching. Each owner generates their own attribute sets and corresponding public keys for the encryption of the data, and each receiver generates a search token based on their attribute. However, Guo et al.'s scheme may cause unstandardized secret key generation on the receiver side since each owner may possess different attribute sets. Furthermore, the dynamic update is not supported within their scheme. This may affect the effectiveness of the system in a multi-owner writing scenario. In our work, we proposed several tweaks to Guo et al.'s scheme, such as:

i. incorporating an Attribute Authority Center (AAC) to address the unstandardized key generation of multiple data owners,
ii. enabling a dynamic update to ensure that the search index is not rebuilt each time new data is uploaded,
iii. enabling multi-keyword search functionality, and
iv. incorporating with blockchain where the search index is uploaded to the smart contract and the encrypted data is uploaded to the InterPlanetary File System (IPFS).

We employed Advanced Encryption Standard (AES) for encrypting contact tracing data and Attribute-Based Searchable Encryption (ABSE) for encrypting the search index, ensuring confidentiality. We also used blockchain and smart contracts to store/access encrypted search indexes and retrieve files, respectively. We used InterPlanetary File System (IPFS) to outsource the encrypted data. The usage of blockchain enables us to log the access of files and search index, thus achieving non-repudiation. The immutability property of blockchain also ensures the integrity of the stored content. Evaluation shows our system is efficient, with search efficiency scaling linearly to the index size, and low deployment costs for smart contracts.

2 Overview of the System Framework

The proposed framework assumes that the data should only be accessible by the government and other government-authorized organizations such as research companies and hospitals. The contact tracing datasets include name, identity card number, gender, check-in date, time, and visited location. Four keywords are used to identify the datasets: the organization that provides the data, the location area coverage for the contact tracing data, and the month and year of the collected data. Since the contact tracing datasets are confidential and not available publicly, the dataset used in this paper is randomly generated dummy data following the type of values that the MySejahtera app uses.

Table 1 shows a sample of the data generated, whereas Table 2 shows the sample of keywords for the data. Each dataset has four keywords to be registered for the corresponding data owner. The sample from Table 1 shows the content datasets within a plaintext form data file associated with the keywords in Table 2.

Table 1. Sample datasets of generated contact tracing data

Name	IC number	Gender	Date	Time	Visited location
Daron Robe	000-33-4851	Female	05-02-22	7:28 AM	Kip Mall
Lucas Smith	254-75-5861	Male	06-02-22	6:24 AM	Aeon Mall
Matt Thompson	471-32-3650	Male	04-02-22	1:30 PM	Aron Mall
Kieth Ulyatt	872-61-2400	Male	08-02-22	5:29 AM	KFC

Table 2. Sample keywords for the data

Organization	Location	Month	Year
OrganizationABC	Salak Tinggi	February	2022

If the data requester intends to search, they need to search for keywords set by the data owner. For example, in this case, to retrieve the datasets from Table 1, the data requester can search either one of the four keywords: "OrganizationABC", "Salak Tinggi", "February", and "2022" in order to get the file associated with Table 2. The more keywords the data requester includes, the more specific the search results will be returned to the data requester. For example, if only "February" is searched, all datasets associated with "February" will be returned. They could be datasets with keywords "February", "2021", or "OrganizationBCC", "February". Hence, if the data requester wishes for a specific set of datasets as the results, more keywords must be entered into the system.

2.1 High-Level Overview of the Proposed System

The system mainly consists of four parties, Data Owner (DO), Data Requester (DR), Attribute Authority Center (AAC), and Cloud Service Platform (CSP). Figure 1 shows a high-level overview of the proposed system. A brief discussion of these entities are discussed below:

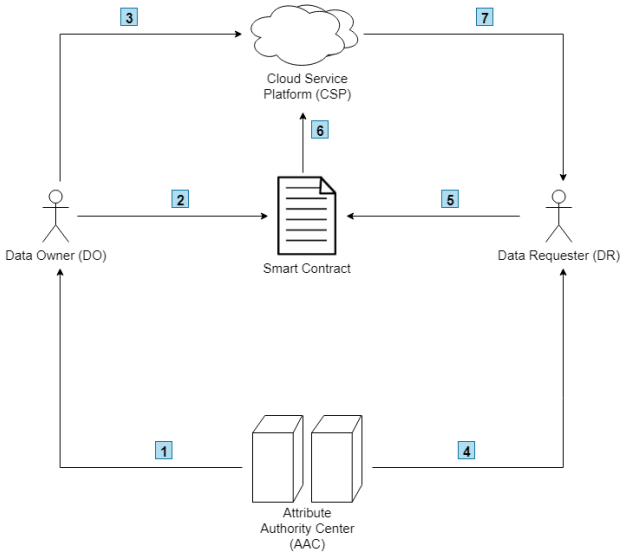


Figure 1. High-level overview of the proposed system

i. Data Owner (DO): The DO, typically a local hospital or center, collects contact tracing data and uploads the initial data to the system. The DO encrypts and uploads contact tracing data to the CSP, along with the encrypted search index and access policy to the smart contract.
ii. Data Requester (DR): The DR, which may be a government department or research center, can request encrypted data from the system. Each DR has specific attributes, and decryption is possible only if their attributes align with the access policy set by the Data Owner (DO).
iii. Attribute Authority Center (AAC): AAC is the entity responsible for generating and distributing the public and private keys for all system users. Depending on the system, AAC could be a trusted third party or an internal department of the organization.
iv. Cloud Service Platform (CSP): CSP is responsible for storing the encrypted data. The results returned from the smart contract can be used to retrieve the desired files.

The DO and DR use the smart contract to interact with the system. DO uploads the index to the smart contract within the blockchain network, whereas DR interacts with the smart contract to retrieve the encrypted data that matches the desired keywords.

2.2 Workflow of the System

The workflow of the system is demonstrated as a sequence diagram in Figure 2.

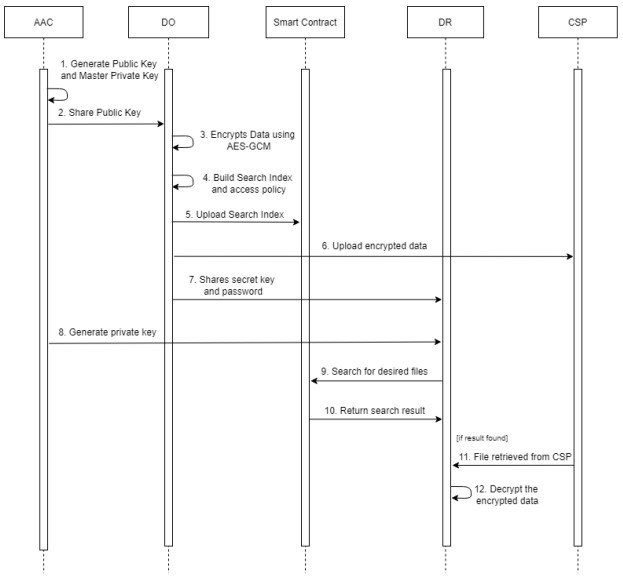


Figure 2. Sequence diagram of the proposed system

The steps involved in the sequence diagram include:

i. AAC generates the system parameters and a master private key. Then, the system parameters are distributed to DO through a secure communication channel.
ii. DO generates an encrypted keyword search index with an access policy and encrypts the contact tracing data with AES-GCM. The encrypted contact tracing data will be outsourced to CSP for storing purposes.
iii. DO stores the encrypted search index into a smart contract and deploys it onto the blockchain network for fair payment and searching purposes.
iv. AAC generates a private key based on the attributes of DR and distributes it to that particular DR through a secure communication channel.
v. DR generates a search token, deploys a smart contract for search-related functions and sends it to the blockchain to make a transaction. The search operation will proceed only if a sufficient fee is given.
vi. After executing the search operation, a list of files with the desired keywords will be retrieved from the

CSP.

vii. The result is sent to DR; upon receiving the encrypted data file, DR decrypts the encrypted files.

3 Construction of the Scheme

For our implementation, the datasets are encrypted via Advanced Encryption Standard (AES), whereas the search index is encrypted via a modified version of Guo et al.'s [15] Attribute-Based Searchable Encryption. The detailed mathematical description of the scheme can be found from the referred article [15].

AES-Galois/Counter Mode (AES-GCM) [16] is chosen particularly for this system due to its efficiency and security. It utilizes an authentication tag during the decryption process to determine whether the user can perform decryption. During the decryption operation, the results will not be displayed to the user if the generated tag does not match the authentication tag stored within the encrypted dataset.

The search index is secured using a modified version of Guo's asymmetric encryption scheme [15] for enhanced security. This encrypted search index is stored in a smart contract, while the encrypted datasets are stored in the InterPlanetary File System (IPFS), a distributed network for data storage and sharing. Unlike a centralized authority like CSP, IPFS reduces the risk of data abuse by authorities. With content stored across multiple nodes, IPFS ensures availability and accuracy, even in the face of malicious actions. The IPFS desktop, chosen for this implementation, offers a user-friendly interface, making it easier for users to manage content, link to services, and track connected peers compared to other command-line-operated IPFS versions.

3.1 System Setup

Attribute-Based Encryption is used for the encryption of the search index, and AES-GCM has been used for the file encryption. For prime number mapping purposes, the JPBC library [17] has been imported to generate keys for attribute-based encryption. The public key was generated using a preset properties file provided by JPBC for bilinear prime mapping. Then, a list of attribute sets that contains all the possible attributes for the DR has been inputted into AAC to generate a master private key. After initializing the required system parameters, the public key is distributed to the data owner to encrypt the search index. Algorithm 1 shows the key generation process.

Algorithm 1. Key generation algorithm

Input: *All possible user attributes*
Output: *Public Key and Master Private Key*
1. *Initialize JPBC for bilinear mapping*
2. *Set new Element for each parameter*
3. ***for each*** *attribute*
4. *Get the grouping in master private key*
5. ***for each*** *attribute*
6. *Get the grouping for each attribute in public key*
7. *Perform multiplicative operations for each element*
8. ***end***

3.2 File Encryption

The data owner generates a secret key and distributes it to the data requester beforehand. The secret key is used in AES-GCM for encryption and decryption. The data owner can set a password during the encryption; hence, the data requester must also know the password to decrypt the file. The file encryption process is shown in Algorithm 2.

Algorithm 2. File encryption via AES-GCM

Input: *Contact tracing data in plaintext form and password*
Output: *Encrypted contact tracing data*
1. *Initialize AES-GCM for encryption using the* ***javax.crypto library***
2. *Convert the plaintext file into a byte*
3. *Generates a random salt and IV*
4. *Encrypts the byte file with salt, IV, and password*
5. ***if*** *file encryption succeeds* ***then***
6. *Writes encrypted bytes into the same directory*
7. ***end***

First, AES-GCM is initialized using javax.crypto library. After the initialization, it reads the plaintext file into byte form for encryption. A random salt and IV alongside the password are generated for the encryption. Upon successful encryption, the encrypted bytes are written into a file and stored within the directory of the file. Next, the data owner generates a search index and encrypts it using AES-GCM but with a different preinstalled key within the system. The key differs from the key used for file encryption, and with the aid of using a password during encryption, the adversary will not be able to encrypt/decrypt the file easily, even if the keys were to be learnt. After successful encryption, the encrypted data are uploaded to the IPFS.

3.3 Index Generation

The encrypted search index for the particular file is then uploaded to the smart contract. The smart contract holds an array class mapped to a SearchIndex structure. The uploaded search index is added to the array class for searching. The structure consists of seven variables, which are summarized in Table 3. Algorithm 3 shows the process of index encryption and upload. Smart contract stores the SearchIndex structure and maps it with an incremental count representing its location within the structure.

Table 3. Summary of search index structure

Index structure	Explanation
id	The unique ID number for each file
ipfsHash	The hash values for the file within IPFS
organization	The organization that owns the data
location	The location area of the collected data
month	The month of the data collection
year	Year of the data collection
token	Token generated from access policy

Algorithm 3. Index generation and uploads

Input: *Keyword Index and access policy token*
Output: *Encrypted search index in smart contract*
1. *Convert the keyword index into an encrypted index*
2. *Pass the access policy into ABSE*
3. *Generates a file token based on the access policy*
4. *Upload the encrypted index along with token to smart contract*
5. *Smart contract stores the encrypted index and token within a new position within the SearchIndex structure array*
6. *A unique ID corresponding to the position has been generated for the particular file position*
7. ***End***

After the keyword index encryption, ABSE generates a file token by comparing the access policy and attribute set. Algorithm 4 shows the generation of the file token. The token and encrypted index are then uploaded to the smart contract. Smart contract stores the SearchIndex structure and maps it with an incremental value to represent its location within the structure array.

Algorithm 4. File token generation

Input: *Access policy and attribute*
Output: *File token for authentication purpose*
1. *Initialize JPBC using* ***it.unisa.dia.gas.jpbc library***
2. ***for each*** *attribute in the system*
4. ***if*** *the attribute is within the access policy*
5. *Multiply the token with corresponding public parameters with the attribute*
4. ***else***
5. *Multiply the token with corresponding public parameters without the attribute*
6. *Perform multiplicative operations for each element in accordance with the scheme*
7. ***End***

The search index is dynamically updated by adding keywords to the structure. Each time a new file structure is added, the unique id increments by one, representing the file's position for variable retrieval. Using the *addIndex*() function within the smart contract, a search index that supports dynamic updates for the addition of records is implemented. It is fulfilled by adding another value into the structure, which is essentially adding another column to the array. Hence, the search index does not require to be rebuilt each time a new index is added.

3.4 Searching and Access Control

The data requester first enters their attribute set and desired keywords to generate a search token. A transaction is then invoked to retrieve five string arrays from the smart contract to the client. First, the comparison between searched keywords and four string arrays, organization[], location[], month[] and years[], is performed. If the searched keyword for the corresponding array is not null and contained within the array, the keyword's position is stored within an array, position[]. Once every array has been compared, the last array retrieved from the smart contract, token[], is used to verify whether the data requester has met the requirements to access the encrypted data. This is because if the token were to verify first before searching for the file with the desired keywords, the system needs to run through all of the tokens retrieved from the smart contract, where the size is essentially the size of the whole search index. This significantly impacts the efficiency of the system. If the search token of the data requester matches the retrieved token, then the data requester can view the encrypted data. This process is repeated for each file that satisfies the desired keywords. After that, the ipfsHash is retrieved from the smart contract and used to download the file. Algorithm 5 shows the process of searching and access control.

Algorithm 5. Searching and access control

Input: *Desired keywords in plaintext and attribute set of the data requester*
Output: *Hash value for encrypted file associated with desired keywords*
1. *Generate search token based on the attribute sets*
2. *Retrieve organization[], location[], month[], year[] and token[] from smart contract,*
3. ***for each array***
4. ***if*** *desired keywords* ***NOT*** *null*
5. *Check for searched keyword locations*
6. *Store the result within results[]*
7. *Remove duplicate values within results[]*
8. ***for*** *each element in results[]*
9. *Compare search token with token[results]*
10. ***if*** *(search token != token[results])* ***then***
11. *Remove the element from results[]*
12. *Retrieve ipfsHash from smart contract using results[]*
13. *Download the encrypted data using the ipfsHash*
14. ***end***

Table 4 shows an example of the organization array retrieved from the smart contract. If the data requester's desired keywords for the organization value is "OrgA", then the file hash value associated with the keyword "OrgA" is stored in ID 0 and 3 in the structure. Then, the token in the corresponding position determines whether the data requester fits the access policy. If the data requester is permitted access, they could request the ipfsHash value from the smart contract and get access to the encrypted file. If the data requester searches for two keywords, the same operation will be applied for both arrays, and their results will be compared.

Table 5 shows an example of a location array retrieved from the smart contract. If the data requester desired keywords for organization value is "OrgA" and the location value is "Klang", the only matched location would be location 0. Hence, only a token from location 0 must be computed for access control. The same logic applies to scenarios where more keywords were to be added within the system. After getting the IPFS hash value, the data can be downloaded and decrypted to retrieve the desired data.

Table 4. Example of organization array

[0]	[1]	[2]	[3]
OrgA	OrgB	OrgC	OrgA

Table 5. Example of location array

[0]	[1]	[2]	[3]
Klang	Nilai	Klang	Sepang

3.5 File Decryption

Once the index of the file is retrieved from the search results, a transaction can be made to retrieve the IPFS hash value for the file. The value is used to download the encrypted data from IPFS and decrypt it using AES-GCM. The data requester needs to input the password and secret key. If both inputs match the values used during encryption, then only the decryption is performed successfully. Algorithm 6 shows the decryption of the encrypted file retrieved from IPFS.

Algorithm 6. File decryption

Input: *Password and secret key*
Output: *Printed decrypted file in plaintext form*
1. *Initialize AES-GCM for decryption using* ***javax.crypto library***
2. Buffers the IV, Salt and Ciphertext
3. Reads and initialize the secret key
4. Upload encrypted index & token to smart contract
5. ***if*** *password invalid*
6. Decryption process failed.
7. else
8. Decrypts the encrypted file using the secret key
9. Print the decryption output into a file.
10. end

3.6 User Interfaces

For this implementation, Java Swing is used to implementing user interfaces for the implemented scheme. A GUI form is prepared within the IntelliJ IDE Swing UI Designer tool. As multiple types of actors are expected to interact with the system, a card panel is used to demonstrate the interfaces of different types of users rather than opening multiple individual windows, which may cause the procedure to be untidy if all of the windows are operated within a single device. JTextField is used to get the input of the user, and JButton is used to create an event listener to perform the functions for the particular sections.

Figure 3 depicts the key generation center interface where AAC generates public and master private keys based on entered attribute sets. The navigation bar facilitates user interface navigation. In Figure 4, the data owner interface allows file encryption with AES-GCM by entering the file path and password. After encryption and upload, the data owner adds the file to the search index. Figure 5 shows the data requester interface, enabling attribute set input for secret key generation. Users can search for files, download encrypted files from IPFS, and perform decryption.

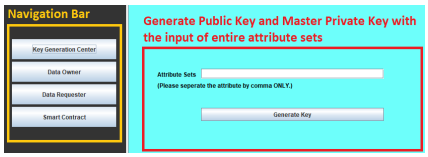


Figure 3. Key generation center interface

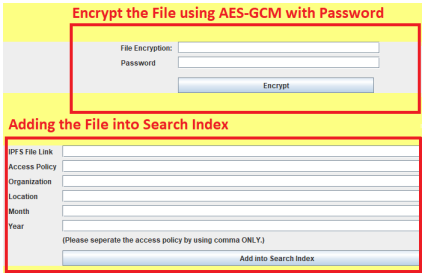


Figure 4. Data owner interface

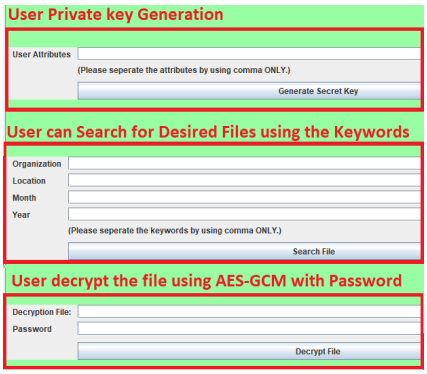


Figure 5. Data requester interface

4 Analysis of the Implemented Scheme

In this section, we first evaluate the system regarding its security and efficiency. Next, we discuss the results of the functionality tests. Lastly, we present a comparison between the implemented and existing systems.

4.1 Security Analysis

In the implemented scheme, the security of the contact tracing data is being prioritized over the searching efficiency and computational overhead. Hence, several security goals are inspected.

4.1.1 Confidentiality of Contact Tracing Data

The CSP/IPFS cannot extract meaningful data from outsourced datasets. Even if an adversary gains access to the encrypted file, they cannot directly read datasets or perform common attacks, such as known ciphertext attacks, to recover the original file.

We used AES-GCM for the encryption of datasets, along with a password chosen by the data owner. The encryption process has been performed with a symmetric key generated by the data owner and a password. After the encryption has been done, the encrypted file will then only be uploaded to the IPFS storage network. Hence, the knowledge of both values, the key and the password, must be known for the file decryption operation to produce the original file. An adversary without the knowledge of these values will not be able to decrypt the file and learn any meaningful data. Aside from that, the pattern of the ciphertext block will be randomized completely. Additionally, the authentication tag will detect any modification of the data. Hence, the confidentiality and integrity of the outsourced contact tracing data are assured.

4.1.2 Non-repudiation and Immutability of Data

Storing a search index on the local server or outsourcing data to the CSP may risk the confidentiality and integrity of data as the CSP may view or modify the content. We used a smart contract for the search index and stored the outsourced data in the IPFS. This allows any accidental or deliberate modification attempts to be identified. Any access to the data will be logged so that the data requester will not be able to deny their transactions.

In this scheme, the search index is uploaded to the Ethereum blockchain, which possesses immutability properties and distributed ledgers. Therefore, anything stored within the blockchain network must check its validity with the network participants before any transactions can be made. Any tampers made to the data within the network can be easily identified. Aside from the search index, the outsourced data stored within IPFS possess the same properties. Any amendments to the content of the stored data in the IPFS will significantly affect the hash values within the blockchain network. When the value for one participant has been changed, the other nodes will not be affected. Hence, any form of amendment will be recognized immediately.

4.1.3 Access Control of the Data Requester

The unauthorized data requester cannot access the data content even if there are matching files with their searched keywords. The attribute set of the data requester determines whether they are permitted to retrieve the encrypted file from the smart contract. The data requester's attribute sets must also be entirely matched with the access policy.

In this scheme, the entire attribute variables set by the organization have been buffered into key generation during the setup stage. A master private key that contains all attribute variables has been created. Each time the data requester generates a private key based on their attributes, the entire attribute sets will be used along with the master private key to determine which attributes the data requester possesses and which do not. This way, the private keys generated with different attributes cannot be used to produce a pattern for attributes the data requester does not own, as the master private key is required for the generation. Hence, a data requester without attributes that match the access policy cannot retrieve the item from the smart contract.

4.2 Performance Evaluation

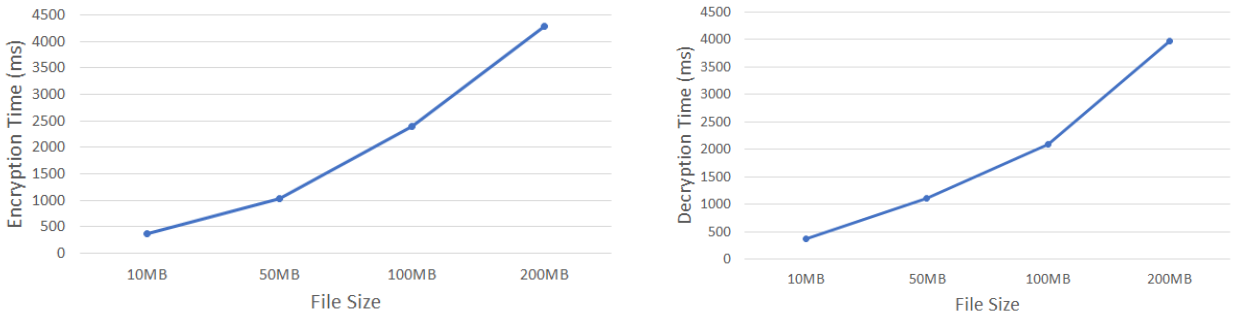
We used dummy data to assess the performance, as contact tracing datasets are not publicly available due to their sensitive nature. Multiple aspects, including encryption and decryption, search, and cost, are considered in the evaluation.

4.2.1 Encryption Efficiency

We used the speed of the encryption operations over different datasets with different sizes, e.g., 10 MB, 50 MB, 100 MB and 200 MB, to assess the encryption efficiency. We utilised the System.currentTimeMillis() function before and after the encryption and determined the time to complete the operation by calculating the interval. Figure 6(a) illustrates a slight curve in the time taken for the encryption operation to be completed when the data size increases from 10 MB to 200 MB. This indicates a potentially exponential growth in the time required for the encryption operation when the file size increases. This is mainly due to the trade-off of achieving stricter security goals. The same pattern is observed for the decryption time, as illustrated in Figure 6(b).

4.2.2 Index Searching Efficiency

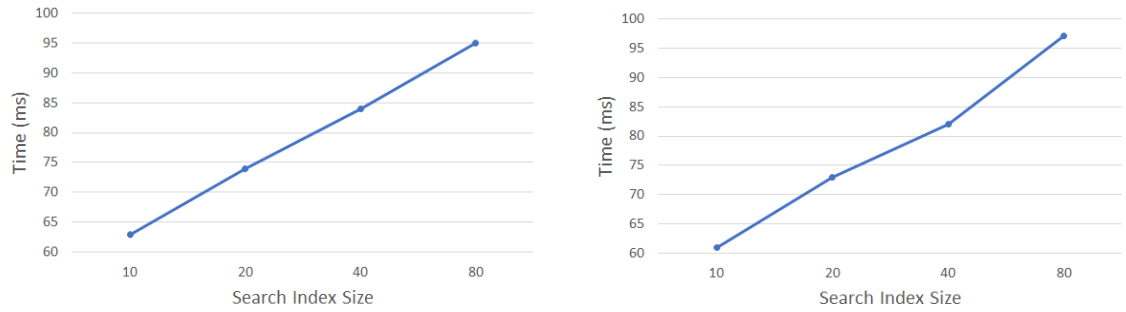
We evaluated the search efficiency according to the time required for the system to search through the index to get the desired file. This process includes comparing each keyword array and computing token to see whether the data requester can retrieve the encrypted file from IPFS. The search index is filled with keywords and up to 80 files' search index. The search query will be encrypted in the designed application to reduce the workload on the blockchain server and the network data transmission time. Figure 7(a) shows the time to search through the search index for a single keyword. The time taken to complete the search operation is directly proportional to the search index size. Figure 7(b) shows the time to search through the search index for two keywords. It is observed that there is no considerable difference between the time for searching one keyword or multiple keywords. Hence, this scheme supports the search without significantly increasing the search time.



(a) Encryption time

(b) Decryption time

Figure 6. Time taken for encryption and decryption



(a) Single keyword (b) Two keywords

Figure 7. Time taken for the searching operation

4.2.3 Ethereum Gas Price

Deploying the smart contract for search index storage and access costs 1,131,872 gas, amounting to 5,659,360 gwei (0.00565936 ETH) as of July 2, 2022, with one gas priced at five gwei. The cost-effectiveness is achieved by moving complex algorithms to a designated application rather than embedding them in the smart contract. The deployment cost overhead is directly proportional to the initial data load. For instance, deploying a smart contract with 500 search indexes incurs an overhead of 565,936,000 gas, equivalent to 2.82968 ETH—a relatively low, one-time cost. Figure 8 illustrates the transaction costs of the smart contract deployment.

Figure 8. Transaction cost for deploying the smart contract

Aside from the deployment costs, the search index upload and retrieval and IPFS hash retrieval require gas. Table 6 shows the estimated costs for each function within the smart contract in ETH. The amount of ETH needed is low for each transaction as adding the index only requires approximately 6 USD per transaction and 0.14 USD per search. To address the limitation posed by currency within the system, the designed application can be integrated into web browsers and coupled with an officially recognized and highly reputable extension, such as Metamask, widely trusted by Ethereum web developers. This approach ensures that researchers can access the system from anywhere around the world.

Table 6. Estimated ETH costs for each function

Functions	Transaction costs (Gas)	ETH
Add index	1131860	0.0056593
Get index	26227	0.000131135
Get IPFS	41830	0.00020915

4.3 Testing of Functionalities

Any misuse of the sensitive information provided by contact tracing data will result in a breach of privacy for the particular person. Hence, the system must be working as intended. Assuming multiple Data Owner (DO) has uploaded their file to IPFS and search index to the smart contract, and a user / Data Requestor (DR) from government officials such as authorities from the Ministry of Health, a series of testing is conducted to ensures that the implemented system is working as intended. The keyword is not stored in encrypted form to differentiate the keywords from each other to facilitate the testing process. However, note that it would be encrypted for the actual system.

4.3.1 Test 1: DO Encrypts Dataset

Assuming the data owner encrypts the datasets, they must input the targeted file path and password into the system. Then AES encryption operation is performed, and the encryption results are shown within the targeted file path with its file name categorized. Figure 9 shows the system input and output for this test scenario. The encrypted file is then uploaded to IPFS. Figure 10 shows that the uploaded data within the IPFS network is encrypted and working as expected.

4.3.2 Test 2: DO Uploads Search Index

Assuming the encrypted file is uploaded to IPFS, the data owner creates a search index for the file by entering the file link, keywords, and access policy. If the input of access policy attributes is valid, the index is uploaded successfully. An error message is displayed if one of the elements within the access policy does not exist within the system's attribute sets. Figure 11 shows the complete attribute sets within the system. The input is "*DataProvider, DataAnalytics, Dep1, Dep2, Dep3, SeniorExecutive, JuniorExecutive*". Figure 12 and Figure 13 show the outcome of successful and failed test cases. Notice that "*Dep1*" in Figure 12 is a valid input for the access policy as the attribute exists within the complete attribute sets, and "*Dep*" in Figure 13 is not a valid input. Therefore, due to incorrect input in the access policy, an error message is displayed for the test case of Figure 13. On the other hand, for the test case of Figure 12, as expected, the index is successfully uploaded. After the search index is built successfully, it is uploaded to the smart contract. Figure 14 displays the information stored within the smart contract after the successful upload.

Figure 9. Functionality of encryption function

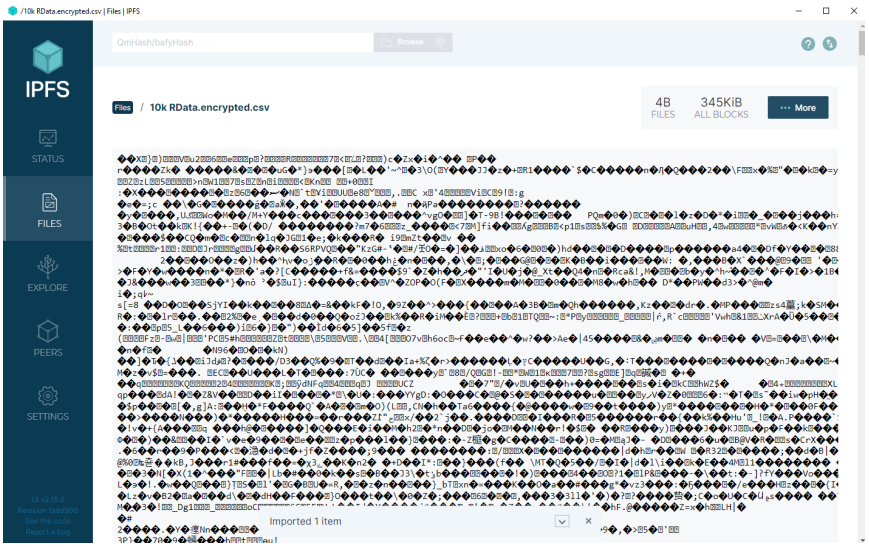


Figure 10. Uploaded file in IPFS in encrypted format

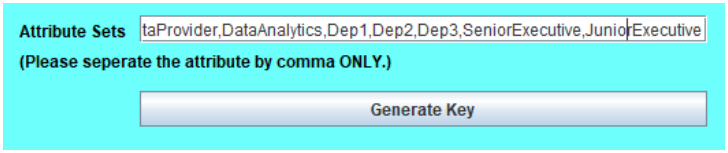


Figure 11. Total attributes within the system

Figure 12. The index is successfully uploaded

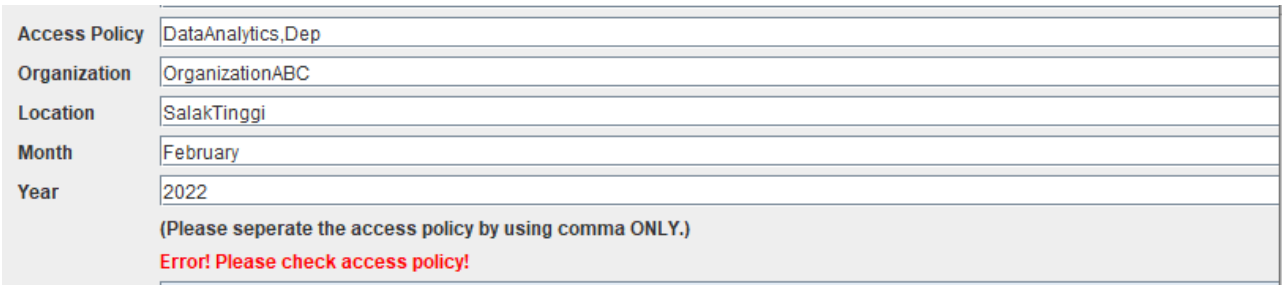


Figure 13. An error message pop due to incorrect input in access policy

Figure 14. The search index is successfully uploaded onto smart contract

4.3.3 Test 3: DR Searched for Existing Keywords

Figure 15 shows the search index saved into the smart contract. These are retrieved from the system after the data requester calls the searching function. Index[0] can be retrieved by "*DataAnalytics*, *Dep1*" attributes; Index[1] can be retrieved by "*DataAnalytics*, *Dep2*" attributes; Index[2] can be retrieved by "*Dep2*, *SeniorExecutive*" attributes.

For searching, the data requester first generates their private key corresponding to their attribute set. Afterwards, they can search for their desired keyword file. The data requester with the "*DataAnalytics*, *Dep1*" attribute is being tested in this test case. Based on the indices specified above, the expected outcome is that the data requester can retrieve only Index [0].

After the keyword searching, authentication functions will check whether the data requester is permitted for the file. Accordingly, the corresponding position for the file hash will be returned to the data requester. Figure 16 shows the result of the searched keyword "2022". Notice from Figure 15 that for the keyword "2022", the desired files are in Index[0] and Index[2]. Therefore, this is a valid request; however, based on the access policy defined above, the data requester with attributes "*DataAnalytics*, *Dep1*" can only retrieve the file in Index[0]. In this scenario, the data requester's attribute does not match the access policy of Index [2], and this is not retrieved. Hence, only Index [0] is retrieved as a result. Figure 16 shows the interfaces for key generation for the data requester and the searching functions for this test case, which shows that the file with the searched keyword that matches the access policy was found. As expected, the test returned Index[0] as the output of the file location for this test scenario. In our implementation, the search function reveals the index of the desired file in a .txt file and is then used to retrieve the IPFSHash value from the smart contract.

4.3.4 Test 4: DR Search with Incorrect Attributes Set

The same search index from Figure 15 is used for this test case. Assuming the data requester attribute set is "*Dep1*, *SeniorExecutive*", the expected outcome for the test case is data requester can retrieve no results. A similar search query is used with the keyword "2022". As the attribute set of the data requester does not fulfill any matched keyword file, no results should be received. Figure 17 shows the output of the search functions for the test case works as expected, where no results have been found when the attribute of the data requester is set to be "*Dep1*, *SeniorExecutive*", even though there are two files that satisfy the desired keywords.

Figure 15. Saved search index within smart contract

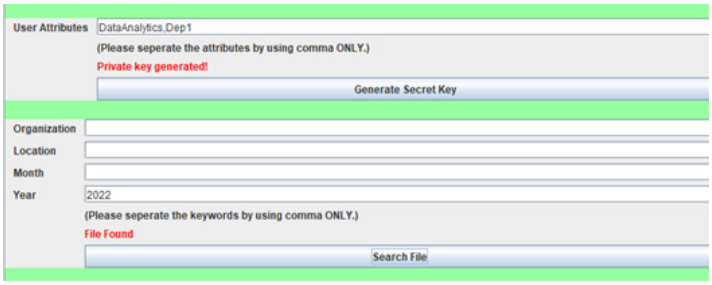


Figure 16. Generation of private key and search functions

Figure 17. The output for test case 4

4.3.5 Test 5: DR Decrypts with Correct Password

Following test case 3, the data requester can decrypt the file through the system after retrieving the encrypted file from Index[0] via the hash value taken from the smart contract. The file will be decrypted successfully when the correct password is input into the system. For the test scenario, the correct password is "Tester1". Figure 18 shows the retrieval of IPFS value from the smart contract. As shown in the figure, the link to download the file with Index[0] is retrieved from the IPFS, and the data requester can download the encrypted data from it.

After the data requester downloads the file, they can decrypt the encrypted file using the correct password. Figure 19 shows the input of the file location and the password. As expected, Figure 20 shows the successful decryption output results for the correct password.

4.3.6 Test 6: DR Decrypts with Incorrect Password

In this scenario, using the same encrypted file from test case 5, the data requester inputs the incorrect password for the decryption process. Figure 21 shows the input for decryption operations, whereas Figure 22 shows the output of the decryption operations. As shown in Figure 21, the incorrect password "Tester2" is the password field input. The expected result in this scenario is that the decryption results will not be shown to the data requester. Even though Figure 21 shows that the decryption operation has been performed; however, the content of the decrypted file is empty due to an incorrect authentication tag. Hence, it is proven that without the correct password, the output of the decryption operation will not be meaningful. Hence, it is proven that using the wrong password will not return the decryption results to the data requester.

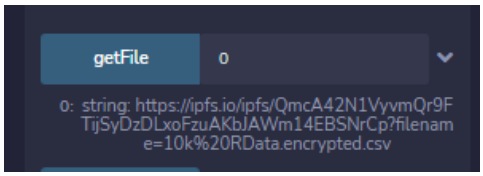


Figure 18. Retrieving data from smart contract index[0]

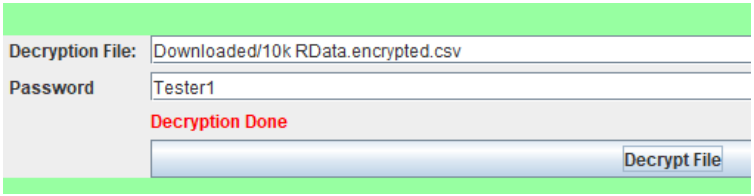


Figure 19. Input for the decryption operations

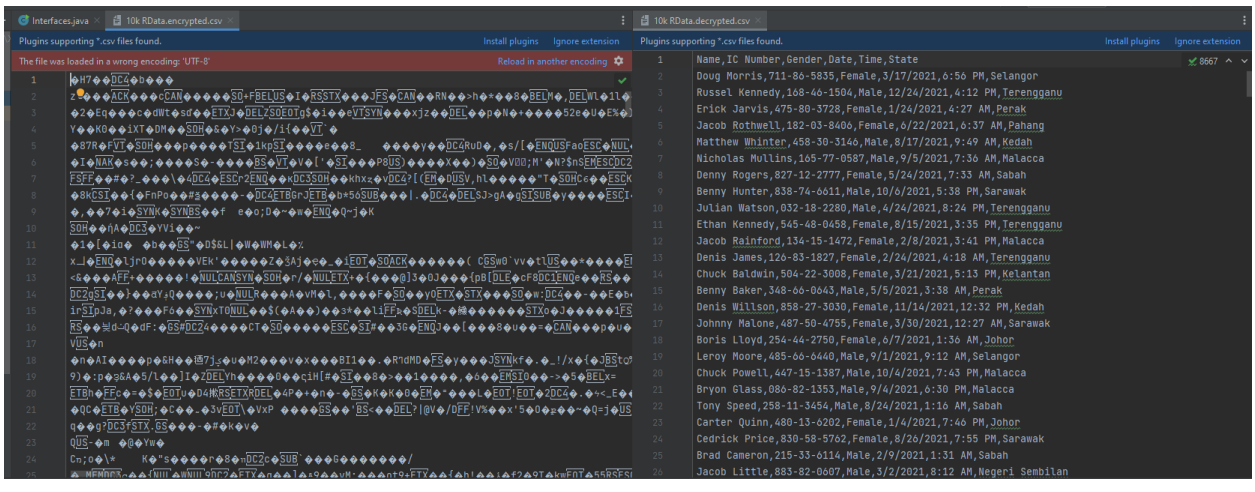


Figure 20. The encrypted file is successfully decrypted

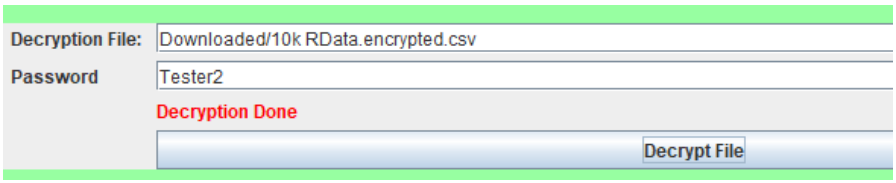


Figure 21. Incorrect password input for decryption

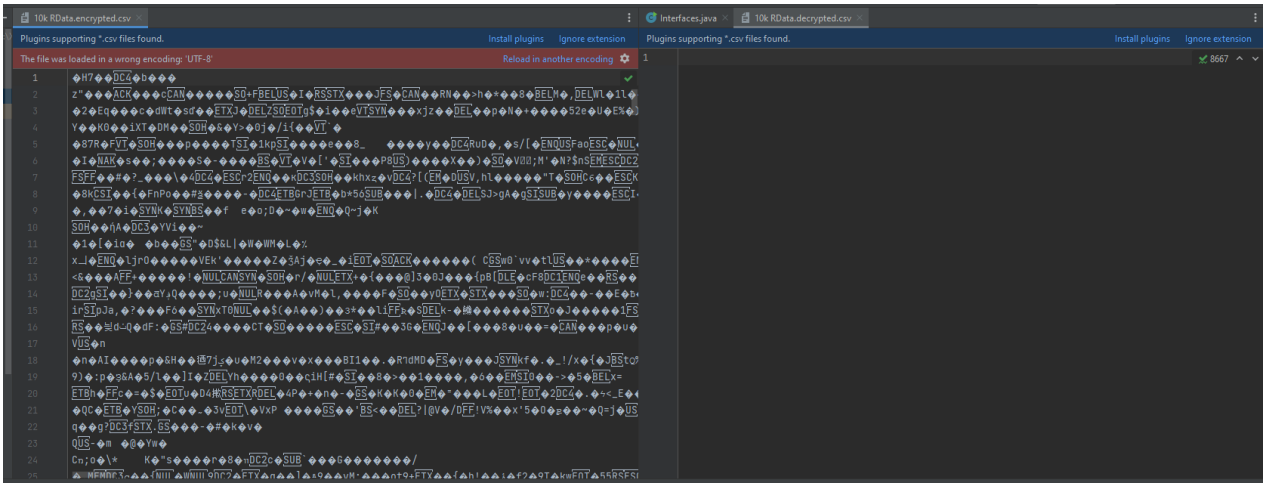


Figure 22. The decrypted file does not contain any words due to failed decryption process

4.4 Comparison with Existing Schemes

A set of features has been compared between related work and the proposed scheme, as shown in Table 7. Our implementation utilizes attribute-based encryption, whereas the rest schemes use symmetric searchable encryption. These schemes have no advanced access policies, as anyone with the search token can search for the encrypted contact tracing data. Our scheme will only reveal the desired search results to users with authorized attributes with the required secret key and password. Any attribute mismatch with the same search query will not return the results. For both of the existing schemes, the search index is reconstructed each time a new file is added for the related works discussed in Section 1.1. Contrary to this, our system supports a dynamic update of the search index. This is achieved by adding new column and row into the search index each time a new data has been added into the index. Any modification does not require the index to be rebuilt, reducing the effort and cost of building a new search index.

Table 7. Comparison of our implementation with other existing works

Features	Schemes		
	[12]	[13]	This paper
Search index encryption	SSE	SSE	ABSE
Multi-keyword	×	✓	✓
Dynamic update	×	×	✓
Storage integrity	Search index only	Search index only	✓
Non-repudiation	✓	✓	✓
Encrypted search index	✓	✓	✓
Encrypted search query	×	✓	✓

(✓: Supported, ×: Not Supported)

Besides, the scheme proposed by Tahir et al. [12] only supports single-keyword searching. This results in an inefficient search over datasets with a large number of features. By using multi-keyword searching, our scheme greatly increases the efficiency of search, hence reducing the overall operation time. Our scheme also assures the integrity of both the search index and encrypted data. This is because the search index is stored within a smart contract, and the encrypted data is stored within IPFS. Both smart contract and IPFS could identify any modifications that are being made to the content. The search index is stored within a smart contract for all the schemes discussed in this section; hence, all these schemes support non-repudiation as the access of the search index and encrypted file will be automatically logged by the transactions. Aside from that, all schemes encrypted their search index before storing it within the smart contract. This way, the adversary cannot learn the keywords for each requested file. However, Tahir et al.'s scheme does not encrypt the search query when the data requester searches for the encrypted file using the keywords; hence, the adversary can learn the keywords of the retrieved file. Once the file's keyword has been retrieved, any file with the same keywords will reveal the encrypted search index. This issue is eliminated within our scheme as we incorporated encryption into our keyword searches.

5 Conclusion

We implemented a secure and efficient system for storing COVID-19 contact tracing data by combining searchable encryption with blockchain technology. Our approach utilizes AES encryption for data security and employs attribute-based searchable encryption to establish controlled access policies. Ciphertext-Policy Attribute-based searchable

encryption is used to encrypt the search index, ensuring that only users meeting the access policy criteria can access files. The dynamic update feature allows seamless additions to the search index within the smart contract, preventing the need for reconstruction. All access to the data are meticulously recorded through blockchain transactions, enabling swift action in enforcing compliance or policies.

The implemented scheme performs well even with a larger search index. While achieving high efficiency, the security goals of confidentiality, integrity and non-repudiation are maintained. Overall, compared to the existing schemes related to COVID-19 contact tracing data, our implemented system provides more features on searchability over encrypted contact tracing data while maintaining security goals. In conclusion, an attribute-based encryption scheme is suitable for encrypting contact tracing data that only allows a group of users that fulfilled the requirement to access and perform decryption on the encrypted data. This approach efficiently implements access control for the data while assuring the confidentiality of the data. The involvement of blockchain technology towards the data access system is feasible, and it helps to achieve several security goals such as immutability, non-repudiation, and no point-of-failure. Overall, this work has presented a scheme involving symmetric encryption, multi-keyword asymmetric searchable encryption and blockchain technology to implement a secure and efficient system for storing COVID-19 contact tracing data.

Acknowledgements

This work was funded by Xiamen University Malaysia Research Fund (XMUMRF) under the Grant no. XMUMRF/2022-C9/IECE/0032.

References

[1] Tracetogether App, Available from: https://www.developer.tech.gov.sg/products/categories/digital-solutions-to-address-covid-19/tracetogether/overview.html

[2] P. Esteves-Verissimo, J. Decouchant, M. Völp, A. Esfahani, R. Graczyk, *Prilok: Citizen-protecting Distributed Epidemic Tracing*, arXiv, June, 2020. https://arxiv.org/abs/2005.04519

[3] G. Avitabile, V. Botta, V. Iovino, I. Visconti, *Towards Defeating Mass Surveillance and Sars-cov-2: The Pronto-c2 Fully Decentralized Automatic Contact Tracing System*, Cryptology ePrint Archive, April, 2020. https://eprint.iacr.org/2020/493

[4] C. Troncoso, M. Payer, J.-P. Hubaux, M. Salathé, J. Larus, E. Bugnion, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, L. Barman, S. Chatel, K. Paterson, S. Čapkun, D. Basin, J. Beutel, D. Jackson, M. Roeschlin, P. Leu, B. Preneel, N. Smart, A. Abidin, S. Gürses, M. Veale, C. Cremers, M. Backes, N. O. Tippenhauer, R. Binns, C. Cattuto, A. Barrat, D. Fiore, M. Barbosa, R. Oliveira, J. Pereira, *Decentralized Privacy-Preserving Proximity Tracing*, arXiv, May, 2020. https://arxiv.org/abs/2005.12273

[5] H. Xu, L. Zhang, O. Onireti, Y. Fang, W. J. Buchanan, M. A. Imran, Beeptrace: Blockchain-enabled Privacy-Preserving Contact Tracing for Covid-19 Pandemic and Beyond, *IEEE Internet of Things Journal*, Vol. 8, No. 5, pp. 3915-3929, March, 2021.

[6] Mysejahtera App, Available from: https://mysejahtera.malaysia.gov.my/.

[7] G. Jung, H. Lee, A. Kim, U. Lee, Too Much Information: Assessing Privacy Risks of Contact Trace Data Disclosure on People with Covid-19 in South Korea, *Frontiers in Public Health*, Vol. 8, Article No. 305, June, 2020.

[8] Chinese Start-up Leaked 400gb of Scraped Data Exposing 200+ Million Facebook, Instagram and Linkedin Users, SafetyDetectives, Available from: https://www.safetydetectives.com/blog/socialarks-leak-report/.

[9] Threatpost, *Microsoft Leaves 250m Customer Service Records Open to the Web*, Available from: https://threatpost.com/microsoft-250m-customer-service-records-open/152086/.

[10] Z. Y. Ng, I. Salam, Blockchain-Based Multi-keyword Search on Encrypted COVID-19 Contact Tracing Data, *International Conference on Information Security Practice and Experience*, Taipei, Taiwan, 2022, pp. 75-92.

[11] Z. Hee, I. Salam, *Blockchain Based Contact Tracing: A Solution Using Bluetooth and Sound Waves for Proximity Detection*, Cryptology ePrint, February, 2022. https://eprint.iacr.org/2022/209

[12] S. Tahir, H. Tahir, A. Sajjad, M. Rajarajan, F. Khan, Privacy-Preserving Covid-19 Contact Tracing Using Blockchain, *Journal of Communications and Networks*, Vol. 23, No. 5, pp. 360-373, October, 2021.

[13] M. Nabil, A. Sherif, M. Mahmoud, W. Alsmary, M. Alsabaan, Privacy-Preserving Non-Participatory Surveillance System for Covid-19-Like Pandemics, *IEEE Access*, Vol. 9, pp. 79911-79926, May, 2021.

[14] T. L. Tan, I. Salam, M. Singh, Blockchain-Based Healthcare Management System with Two-Side Verifiability, *PLOS ONE*, Vol. 17, No. 4, pp. 1-25, April, 2022.

[15] W. Guo, X. Dong, Z. Cao, J. Shen, Efficient Attribute-Based Searchable Encryption on Cloud Storage, *Journal of Physics: Conference Series*, Vol. 1087, No. 5, pp. 1-10, October, 2018.

[16] D. McGrew, J. Viega, *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC*, NIST Special Publication 800-38D, November, 2007.

[17] A. De Caro, V. Iovino, JPBC: Java Pairing Based Cryptography, *2011 IEEE Symposium on Computers and Communications*, Kerkyra, Greece, 2011, pp. 850-855.

Biographies

Zheng Yao Ng received the bachelor's degree in software engineering from Xiamen University Malaysia in 2023. He is currently working as a Software Consultant with Fusionex Group, Malaysia. His research interest includes encryption and cryptography.

Iftekhar Salam received the B.Eng. from Multimedia University, Malaysia, in 2008, the M.S. degree from Dongseo University, South Korea, in 2011, and the Ph.D. degree from Queensland University of Technology, Australia, in 2018. He is currently an Associate Professor at Xiamen University Malaysia. His research interest includes cryptography, cryptanalysis, authenticated encryption, cryptographic protocols.