Received 5 September 2023; accepted 14 September 2023. Date of publication 18 September 2023; date of current version 4 October 2023.

*Digital Object Identifier 10.1109/OJCOMS.2023.3316922*

# Efficient Personal-Health-Records Sharing in Internet of Medical Things Using Searchable Symmetric Encryption, Blockchain, and IPFS

**ABHISHEK BISHT[1], ASHOK KUMAR DAS[1] (Senior Member, IEEE), DUSIT NIYATO[2] (Fellow, IEEE), AND YOUNGHO PARK[3] (Member, IEEE)**

[1]Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad 500032, India

[2]School of Computer Science and Engineering, Nanyang Technological University, Singapore

[3]School of Electronics Engineering, Kyungpook National University, Daegu 41566, South Korea

CORRESPONDING AUTHORS: A. K. DAS AND Y. PARK (e-mail: iitkgp.akdas@gmail.com; parkyh@knu.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2020R1I1A3058605.

**ABSTRACT** Secure storage and sharing of Personal Health Records (PHRs) in Internet of Medical Things (IoMT) is one of the significant challenges in the healthcare ecosystem. Due to the high value of personal health information, PHRs are one of the favourite targets of cyber attackers worldwide. Over the years, many solutions have been proposed; however, most solutions are inefficient for practical applications. For instance, several existing schemes rely on the bilinear pairings, which incur high computational costs. To mitigate these issues, we propose a novel PHR-sharing scheme that is dynamic, efficient, and practical. Specifically, we combine searchable symmetric encryption, blockchain technology and a decentralized storage system, known as Inter-Planetary File System (IPFS) to guarantee confidentiality of PHRs, verifiability of search results, and forward security. Moreover, we provide formal security proofs for the proposed scheme. Finally, we have conducted extensive test-bed experiments and the results demonstrate that the proposed scheme can be used in practical scenarios related to IoMT environment.

**INDEX TERMS** Healthcare system, searchable encryption, blockchain, security, Internet of Medical Things (IoMT), inter-planetary file system (IPFS) decentralized storage.

## I. INTRODUCTION

THE INTERNET of Medical Things (IoMT) is a subset of the Internet of Things (IoT), which consists of medical devices that are interconnected with each other to provide services. It has numerous advantages, such as remote patient care, reduction in the number of in-person visits to the doctors, and increase in speed and efficiency of diagnosis, amongst many others. However, such technology quickly becomes a target of cyber attacks and therefore requires proper security measures to be effective. If not secured properly, IoMT devices can breach the privacy of the patients including, but not limited to, leakage of payment details of the patients. Therefore, it is necessary to incorporate IoMT smart devices with the health infrastructure in a secure way. The work presented in this article deals with an efficient and practical scheme for storing and sharing Personal Health Records (PHRs) generated with the help of IoMT devices using blockchain technology, searchable encryption, and IPFS.

Blockchain first came into light in 2008 when Satoshi Nakamoto released the white paper [1] on Bitcoin – a decentralized digital currency. Since then researchers have been continuously working on its other use cases and have made significant progress. Blockchain acts as a distributed ledger that records transactions. However, it is not just limited to financial transactions as in Bitcoin. It can record any data represented as a transaction. The primary benefit of the blockchain is that data is immutable under cryptographic assumptions after being recorded. One will have to create a new transaction to nullify the effect of a previous transaction.

The transactions are stored in a block where a single block may contain an arbitrary number of transactions depending on the requirement and context of use. A blockchain network may contain many independent nodes forming a Peer-to-Peer (P2P) network. They interact with each other and make decisions based on an agreement protocol, known as the consensus algorithm. The goal of consensus is to reach a joint agreement between the nodes in the system, often in the presence of some faulty nodes. Several consensus algorithms, such as "Proof of Work (PoW)", "Proof of Elapsed Time (PoET)", Raft, and "Practical Byzantine Fault Tolerance (PBFT)", are in use today, each with their own pros and cons.

Searchable encryption (SE) was first introduced by Song et al. [2]. Its need arose from the fact that data needs to be stored on storage servers in encrypted form to mitigate the risks of security and privacy breaches. Nevertheless, this comes at the cost of sacrificing some desirable functionalities. A major one is searching over the stored data. SE is a solution to the above problem that lets the users encrypt their data while retaining the ability to search over it without explicitly decrypting it. SE is categorized into two types: a) *static* and b) *dynamic*. In the case of static SE, an entire batch of files is processed at once. Files cannot be added or removed once the batch has been encrypted. The only way to do so is to discard the previous encryption and create a new one with the modified file set. On the other hand, dynamic SE allows addition and removal of files efficiently without repeating the entire process. SE can further be categorized into *Symmetric* and *Asymmetric* settings based on the type of encryption used. In a symmetric setting, the data is encrypted using symmetric or private key cryptography, and in asymmetric setting the data is encrypted using asymmetric or public key cryptography. In addition, based on the number of keywords searched over the encrypted data, SE is categorized into single-keyword SE and multi-keyword SE. The latter is difficult to achieve, but it has more practical value.

Inter-Planetary File System (IPFS) [3] is a protocol and peer-to-peer file sharing network to store files in a decentralized database. It uses content-based hashing to identify a file in its storage uniquely. Data is divided into small chunks of 256 KB (known as IPFS objects) and stored on its decentralized system. It also provides versioning of the files. Whenever a file is modified, a separate copy of the modified IPFS objects is created while retaining the original ones. It means that IPFS provides immutability to the files stored on it.

### A. PROBLEM STATEMENT AND MOTIVATION

PHRs have always been high-value targets for cyber-attacks; therefore, securing them proves to be a significant challenge. The necessity of being able to share health records with selected parties, such as doctors and insurance companies, further augments the risk of health data leakage. It has led to the requirement of schemes that can not only store health records securely but also allow secure sharing of health data.

Such a scheme requires a guarantee of *confidentiality* of the PHRs, *verifiability* of correctness of the results returned to the end user[1] and *efficiency* in terms of time and space complexity. Moreover, with the increase in usage of Internet of Medical Things (IoMT), it has become crucial to securely integrate the healthcare infrastructure with IoMT.

Over the years, many solutions have been proposed in the literature for health record storage and sharing. However, some schemes leak too much information during their operations, other schemes do not guarantee the verifiability of the search results, most do not provide any proof for forward security and most are inefficient because of use of bilinear pairings which have high computational cost. Additionally, most of the schemes make use of centralized storage for PHRs which makes them unreliable in case of an attack on the centralized storage. We identified that most of the existing schemes for PHR sharing use asymmetric SE and there are not enough works that have explored the usage of symmetric SE for designing a robust PHR sharing scheme with keyword search. Therefore, by using searchable symmetric encryption, blockchain technology, and Inter-Planetary File System (IPFS), we propose a scheme for PHR sharing which is verifiable, forward secure, efficient and decentralized.

### B. RESEARCH CONTRIBUTIONS

As our main contribution through this work, we present an efficient PHR-sharing solution by combining SSE, blockchain technology, and IPFS. The existing schemes that use blockchain or IPFS are either in-efficient or lack forward security. To the best of our knowledge no existing schemes have proposed an approach for health data storage and sharing, that is verifiable, forward secure, efficient and decentralized, by combining these three.

- We develop a PHR sharing scheme that maintains confidentiality of PHRs by proving that it is semantically secure against an adaptive adversary. We prove it to be forward secure and verifiable. Confidentiality is guaranteed by using symmetric key encryption (SSE) for PHRs and other private metadata. By using the "Elliptic Curve Integrated Encryption Scheme (ECIES)" we ensure that communication channels are secure. Verifiability of the results returned to the user is guaranteed by blockchain, and use of symmetric key encryption ensures that our scheme is efficient. Finally, forward security is ensured by the design of our scheme. Subsequently, we provide rigorous proofs to support our claims.
- For the storage of PHRs, we use a decentralized content-addressable file system, IPFS. Any file stored in IPFS is uniquely identified by the hash of its contents. Our designed scheme relies on this content-hash identity (id) for its operation. Using IPFS has many advantages over centralized storage, such as greater fault tolerance,

1. It differs from data integrity, which only ensures that the data is not tampered with, after being sent and before being received.

TABLE 1. Abbreviations.

| Abbreviation | Full Form |
|---|---|
| API | Application Programming Interface |
| BN | Blockchain Node |
| CHD | Current Head Dictionary |
| D | Doctor |
| DU | Data User |
| ECDH | Elliptic Curve Diffie-Hellman |
| ECIES | Elliptic Curve Integrated Encryption Scheme |
| H | Hospital |
| HS | Hospital Server |
| IPFS | Inter Planetary File System |
| MR | Merkle Radix |
| P | Patient |
| PHR | Personal Health Record |
| REST | Representational State Transfer |
| SE | Searchable Encryption |
| SSE | Searchable Symmetric Encryption |

resistance to Denial of Service (DoS) attacks and better scalability.

- We perform extensive experiments using different consensus algorithms, namely "Practical Byzantine Fault Tolerance (PBFT)", Raft, and "Proof of Elapsed Time(PoET)" to demonstrate that our scheme is practical.

### C. PAPER OUTLINE

The rest of the paper follows the following structure. The relevant mathematical preliminaries needed to describe the proposed scheme are discussed in Section II. The related works on the existing schemes are provided in Section III. The system architecture along with the threat model are supplied in Section IV. A high-level description of the proposed scheme with its various phases, followed by a detailed description, is provided in Section V. Next, a detailed security analysis of the proposed scheme is presented in Section VI. The test-bed experiments and their extensive results are presented in Section VII. Moreover, a comparative study among the proposed scheme and other relevant schemes has been done in Section VII. Some future works are then highlighted in Section VIII. The article is finally concluded in Section IX.

## II. PRELIMINARIES

In this section, we discuss the following mathematical preliminaries that are used later in discussion of the proposed scheme. We follow the list of abbreviations provided in Table 1.

#### 1) COLLISION-RESISTANT CRYPTOGRAPHIC HASH FUNCTION

A keyed hash function is a pair of polynomial-time algorithms $(Gen, h)$ that are defined as follows [4]:

- *Gen:* It takes a security parameter $1^n$ as input and outputs a secret key $k$.
- *h:* It takes key $k$ and $x \in \{0, 1\}^n$ as inputs, and outputs a value $y = h(x) \in \{0, 1\}^{l(n)}$ as the output, called the message digest or hash value.

The hash function is said to be collision resistant, if for a polynomial time adversary $\mathcal{A}$ it is computationally infeasible to find a pair $(x, x') \in \{0, 1\}^n \times \{0, 1\}^n$ such that $h(k, x) = h(k, x')$. In practice, however, un-keyed hash functions are used, and we use the same in our work. Thus, an un-keyed hash function is a hash function $h : \{0, 1\}^* \rightarrow \{0, 1\}^{l(n)}$ which takes an arbitrary length input string $x \in \{0, 1\}^*$ and produces the hash output $y = h(x) \in \{0, 1\}^{l(n)}$.

#### 2) DICTIONARY

A dictionary is an abstract data type that is used to store key-value pairs such that every key occurs only once. It is also known as *map*, *associative array* or *symbol table*. A dictionary must support the following basic operations:

*Insertion:* It requires a key-value pair as its input and stores it in the dictionary. The key-value pair is also called an element of the dictionary. If an element with the same key exists, then its value is overridden by the new input value.

*Deletion:* It requires the element's key to be deleted as the input and removes the corresponding key-value pair from the dictionary.

*Lookup:* It requires a key as its input and returns the corresponding value. An error or default value is returned if the key is not present in the dictionary.

#### 3) STANDARD BLOCKCHAIN MODEL

It was proposed by Kosba et al. [5] to specify the security of the protocols using blockchain. It considers the blockchain as a conceptual trusted party that ensures correctness and availability, but not privacy.

#### 4) INTER-PLANETARY FILE SYSTEM (IPFS)

It is a content-based distributed file system [3] where the data is identified through the hash of the contents rather than giving them an independent identifier.

#### 5) HYPERLEDGER SAWTOOTH

It is a blockchain framework maintained by Hyperledger foundation and provides a flexible solution for developing decentralized applications. It allows the programmers to implement application logic in a variety of languages, including Python and JAVA unlike other platforms such as Ethereum, which restricts the programmer to Solidity.

Each node in a Sawtooth network runs a process known as *transaction processor*. It needs to be implemented by the application developer for processing the transactions of his application. A sawtooth node can run multiple transaction processors for different families of transactions.

#### 6) MERKLE-RADIX (MR) TREE

It is a data-structure that stores key-value pairs and is used by most of the blockchain frameworks to maintain their global state. It is a combination of the Merkle hash tree and the Radix (or Trie) tree that is used to store data and verify its integrity through the properties of the Merkle tree.

#### 7) ELLIPTIC CURVE INTEGRATED ENCRYPTION SCHEME (ECIES)

It is a hybrid encryption scheme proposed by Shoup [6]. It uses an elliptic curve key-agreement protocol, such as ECDH and a symmetric encryption scheme. The key-agreement protocol derives the shared secret for the sender which is then used to encrypt the data to be transferred. The encrypted data is sent to the receiver along with the information by which the receiver can derive the shared-secret. The receiver can then use this information to recover the original plaintext.

#### 8) AES-OCB3

The "Advanced Encryption Standard Offset Codebook v3 (AES-OCB3)" [7] is a symmetric key encryption algorithm that is the result of using the "Advanced Encryption Standard (AES)" in Offset Codebook v3 (OCB3) mode. OCB mode integrates Message Authentication Code (MAC) with the operations of AES in block-cipher mode. It eliminates the need for explicitly using an authentication mechanism.

#### 9) CONSENSUS ALGORITHMS

There are several consensus algorithms in use today. The ones we have used in our experiments are the Practical Byzantine Fault Tolerance (PBFT) [8], Proof of Elapsed Time (PoET) and Raft [9].

## III. RELATED WORKS

In this section, we present the following categories of the related works.

### *A. SEARCHABLE ENCRYPTION*

Song et al. [2] in 2000 took the first step toward constructing an efficient searchable encryption scheme. Earlier works [10] made use of Oblivious RAMs (ORAMs), and gave a complete and general solution to the problem of searchable encryption. However, ORAMs are quite expensive in terms of time and space. Therefore, the researchers started looking for alternative solutions that would be efficient. Although the work in [2] is more efficient than ORAM, it was linear in the total number of keywords in document collection and hence was unsuitable for practical applications. Continuing in this direction, Goh [11], in 2003, constructed a secure inverted index and demonstrated its application in SE through a scheme with constant search time. Their scheme, unfortunately, had a weak security model which was subsequently improved upon by Curtmola et al. [12] in 2006. They proposed two new security models – a) adaptive and b) non-adaptive, which become the standards for securing SE schemes. However, their scheme was static, and therefore it had limited practical applications.

The first dynamic searchable encryption scheme was proposed by Kamara et al. [13] in 2012 with an efficient update mechanism. In the same year, Islam et al. [14] demonstrated attacks possible due to access pattern disclosure and reinforced the need for forward secrecy in SE schemes. However, scalability was still an issue as most of the schemes supported only single keyword search. Working in this direction, Cash et al. [15] introduced a scheme that supported conjunctive keyword search along with the Boolean queries.

There was a growing need for forward security after Islam et al.'s work. It was first addressed by Stefanov et al. [16] in 2013. They attempted to find a balance between security and efficiency through their scheme. However, it relied heavily on client-side computation, which is limited in its practical applications. Another issue that needed to be addressed in SE was that none of the works had ever attempted to formalize the data leakage occurring during the operations. In 2015, Cash et al. [17], for the first time, formalized the leakages inherent in SE schemes and demonstrated the attacks possible due to them. Before their work, there was no proper study of such attacks on SE schemes. It formalized the leakages and classified the existing schemes based on four levels of leakage.

Later, it was identified that in order to save the computational costs, a server can return partial or wrong results to the data user. To address this issue, the requirement of verifiability was added to SE schemes. In 2016, Bost et al. [18] proposed the first scheme that supported verifiability by modifying Stefanov's scheme. With time, the need for more features was identified in SE schemes – one of which was backward privacy. It ensures that after deletion of data, future queries cannot be performed on it. Bost et al. [19] in 2017 introduced the first SSE scheme that supported backward privacy. In the same year, Kim et al. [20] introduced a dynamic SE scheme with a primary focus on data deletion, which previous schemes mainly had overlooked. In 2018, Etemad et al. [21] presented a forward, secure, efficient, and parallelizable dynamic SE scheme with the performance on par with previous non-forward-secure dynamic SE schemes. Recently, in 2022, Watanabe et al. [22] proposed a scheme that fixed a security flaw in the backward security of Etemad's scheme. Additionally, there are other schemes, such as the scheme proposed in [23]. It has focused on providing additional features such as data de-duplication and conjunctive queries while maintaining required properties of forward and backward security.

### *B. BLOCKCHAIN-BASED SEARCHABLE ENCRYPTION*

Hu et al. proposed the first work to make use of blockchain in searchable encryption [24] in 2018. Their work aimed to introduce a searchable symmetric encryption scheme in which the results returned by a potentially malicious server are verifiable by the data user. The authors eliminated the use of a central server for data storage and instead used Blockchain as its replacement. This scheme, however, has a significant shortcoming in that Blockchain is not designed to handle a large amount of data and using it to store large files results in impractical latency. In the same year, the work of Cai et al. [25] sought to prevent dishonesty both at client and server side regarding payment for search results by recording the logs of transactions in a public Blockchain. This scheme, however, requires explicit verification of the results by the client from the blockchain. Since then several schemes [26], [27] have been proposed that did not eliminate the central server but instead used Ethereum smart contracts to ensure fairness between client and server. The smart contract is designed in such as way that until both parties agree, the payment will not be released. To enable forward security in blockchain-based searchable schemes, Guo et al. [28] introduced a verifiable and forward-secure SSE scheme that relied on blockchain (for verifiability).

### C. APPLICATION OF BLOCKCHAIN IN PHR SHARING

After the advent of blockchain in 2008 [1], several researchers started to find its applications in other areas, including healthcare. To this end, one of the early works to use blockchain in the health sector was proposed by Azaria et al. [29] in 2016. The authors attempted to store references of medical data onto blockchain and leverage its immutability and transparency to ensure data integrity, verifiability, and data sharing. In the same year, Yue et al. [30] proposed an app called "Healthcare Data Gateway (HDG)" based on blockchain gives patients full control over their health data. In the following year, Xia et al. [31] proposed one of the early works specifically geared towards sharing medical records using blockchain. They attempted to address the access control problems associated with health records using the immutability and autonomy provided by blockchain. Improving upon Xia et al.'s scheme, Fan et al. [32] put forth a scheme that had efficient access and retrieval mechanisms and sought to address the privacy risks associated with sharing of data with third parties.

### D. APPLICATION OF SE AND BLOCKCHAIN IN PHR SHARING

In 2018, Zhang and Lin [33] proposed a scheme that used blockchain as well as searchable encryption to provide a health data sharing scheme. They used a combination of private and consortium blockchains along with public encryption with keyword search (PEKS) to build blockchain-based secure and privacy-preserving PHI sharing (BSPP) scheme. However, it is inefficient due to heavy dependency on bilinear pairing.

In 2021, Wang et al. [34] proposed a PHR sharing scheme that uses IPFS to store the PHRs. Nevertheless, it uses bilinear maps for cryptographic operations, making it inefficient. Moreover, there are some schemes [35], [36] for health recording sharing based on proxy re-encryption. Some of the schemes [37], [38], [39], [40] have used attribute-based encryption to have better access control over sharing of PHR. However, all these schemes rely on bilinear maps for security, making them inefficient. Additionally, there has not been any focus on forward security in these schemes, and many are not dynamic. Unlike the schemes discussed so far, recently, in 2021, Tang et al. [41] proposed a PHR sharing scheme based on searchable symmetric encryption (SSE) and blockchain. However, it requires the concerned hospitals to store the data on a local server, which is not a realistic assumption. A recent scheme proposed by Nie et al. [42] uses IPFS to store the encrypted cipher-texts and blockchain for meta-data. However, it is not dynamic in nature, i.e., all the health records are encrypted at a time and there is no mechanism to add the health records later.

### E. OBSERVATIONS

We observe that there are many schemes that make use of asymmetric searchable encryption for health data sharing. However, the use of symmetric searchable encryption for the same has not been explored well enough, even though the literature of SSE is quite strong. Our work tries to fill this research gap by providing a symmetric searchable encryption based health data storage and sharing scheme using blockchain technology and IPFS. The proposed scheme is able to provide confidentiality of PHRs and other private metadata, secure communication channels, verifiability using blockchain, forward security, and also efficiency using symmetric key encryption.

## IV. SYSTEM MODEL

In this section, we discuss the system architecture and threat model used in the development of the proposed s

The system model proposed in this article consists of various patients, hospitals, a blockchain network consisting of peer nodes maintained by hospitals, and an IPFS network that is accessible through the Internet.

Fig. 1 shows a diagrammatic representation of our system model. It shows two hospital instances designated as *Hospital 1* and *Hospital 2*. Each hospital contains a *hospital server*, a *Blockchain node*, several *doctor devices* and several *medical devices*. In practice there will be more number of hospitals; however, for clarity we have shown only two hospitals. Further, there are two patients designated as *Patient 1* and *Patient 2*. In practice, there will be many patients and each patient can visit any of the hospitals for diagnosis but for simplicity, we have shown only two patients. *Patient 1* visits *Hospital 1* and *Patient 2* visits *Hospital 2*. A patient is alternatively called *data owner*. Further, we have an instance of *data user* in the model. A data user can interact with any of the hospitals and data owners. The exact nature of this interaction will become clear in subsequent sections.

Fig. 1 shows the blockchain and IPFS networks. Each hospital is also a part of the consortium blockchain network.

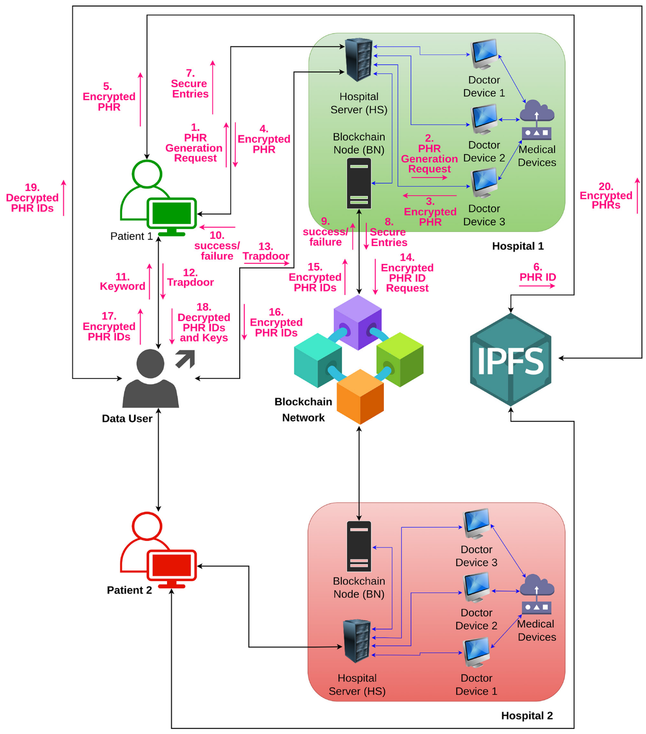


**FIGURE 1.** **System model.**

It is used to store meta-data such as the encrypted index. IPFS, on the other hand, is independent of the hospitals and used to store the actual PHRs in encrypted form. We can say that the PHRs are stored in an off-chain structure that is decentralized; thereby providing more reliability.

Apart from presenting the different entities involved in our system model, Fig. 1 also describes the network model of our system. The connector lines in the figure represent communication channels between the entities. Bidirectional arrows denote the fact that the channels are bidirectional. The channels shown using black lines are built over the Internet while the channels denoted using blue lines are built over local network of the entity in which they are contained. For instance, the communication channel between *Patient 1* and *Hospital Server* is shown using a black line, while that of *Doctor Device 1* and *Hospital Server* is shown using a blue line.

We have shown the various messages that are passed between the entities in the system during its operation. These messages are shown using the color pink and numbered in a sequential manner so as to describe the flow of information in the system. Note that the messages are not shown for Patient 2 and Hospital 2 as they would be similar to that of Patient 1 and Hospital 1. Additionally, each entity in the system generates its own pair of elliptic curve keys which are then used for secure communication with other entities using "Elliptic Curve Integrated Encryption Scheme (ECIES)".

We describe each of the entities present in the system model in detail.

- *Patient (P):* A patient is a person who visits a hospital for medical diagnosis and is also called the data owner.
- *Hospital Server (HS):* We define hospital to be a medical institution that provides services to a patient. Each hospital that is part of our system should also maintain a server (denoted by HS) that handles all the requests from patients and a blockchain node (BN) that connects the hospital to the blockchain network.
- *Doctor (D):* Every instance of a hospital has several doctors associated with it and each doctor is provided a device through which he/she can access the hospital's IT infrastructure.
- *Data User (DU):* Any entity that wants to make use of the data uploaded into the system by data owners is called a data user. A patient itself can also become a data user. Another common instance of a data user is a doctor who requires a patient's medical history for diagnosis. Moreover, insurance companies can also be categorized as data users.
- *Blockchain Node (BN):* In our scheme, every hospital in the ecosystem should be a part of the consortium blockchain network, and thus should maintain a blockchain node that takes part in the consensus mechanism.
- *IPFS:* It is used by patients to store their Personal Health Records (in encrypted form). It can be accessed through the Internet.
- *Medical Devices:* Each hospital has a set of medical devices to diagnose patients and are connected to the local network of the hospital. A few examples of such devices are "Electrocardiogram (ECG) device", "Electroencephalogram (EEG) device" and "Magnetic Resonance Imaging (MRI) device", etc. A doctor can use these devices to generate data to be included in the PHR.

## V. PROPOSED SCHEME

Till now, we have discussed about individual entities in our system. We now move ahead and describe the interactions between these entities which forms the core of our proposed scheme. First, we define the various components essential for understanding the scheme and then provide its overview followed by a detailed description.

The proposed scheme provides dynamic updates by allowing addition of new PHRs without having to rebuild the index. It is important to note that our scheme does not provide the functionality to update the contents of an individual PHR. It only allows addition of new PHRs dynamically, i.e., without having to rebuild the index. PHR records are generated by a doctor; therefore, a data owner does not need the facility to themselves update an individual PHR. Moreover, the literary works have only focused on addition of new PHRs and not on updating an individual PHR. Here, we also focus on the same. After PHR addition, a data user can search for PHRs containing specific keywords with consent from the data owner. For ease of understanding, we have provided a list of notations used to describe our scheme, in Table 2.

TABLE 2. **Notations.**

| Notation | Definition |
|---|---|
| $h(\cdot)$ | Collision-resistant hash function |
| $MK$ | Patient master key |
| $TS$ | Timestamp |
| $E_K(\cdot)$ | Probabilistic ECIES encryption using public key $K$ |
| $E_k^{SK}(\cdot)$ | Probabilistic symmetric encryption using key $k$ |
| $X[k]$ | Access an element of dictionary $X$ using key $k$ |
| $X_i$ | Access $i^{th}$ element of list $X$ |
| $\|\|$ | Concatenation |
| $U$ | Universal set for keywords |

## A. DEFINITIONS

*Definition 1 (PHR):* A *PHR* is a text file containing the diagnosis report generated by a doctor.

*Definition 2 (Keyword):* A keyword is a sequence of characters that is delimited by a special character such as a blank space.

*Definition 3 (Secure Entries Dictionary):* It is a dictionary denoted by $\Delta$ and is used to store the entries of the encrypted search index.

More details about $\Delta$ are presented in Section III.

*Definition 4 (Trapdoor):* A trapdoor is a special search token that is issued by data owner to a data user when the latter wants to search for the PHRs containing a given keyword.

The exact structure of a trapdoor is provided in Section IV.

*Definition 5 (PHR Sharing Scheme):* We define our scheme as a tuple $\Sigma = $ (*KwExt*, *SEGen*, *PHRAdd*, *TrapGen*, *Search*, *RetId*) where the algorithms are described as follows:

- $W \leftarrow$ *KwExt(PHR): KwExt* takes a *PHR* as input and returns a list $W$ of unique keywords present in the *PHR*.
- $\Delta \leftarrow$ *SEGen(W): SEGen* takes a set of unique keywords $W$ as input and returns a secure entries dictionary $\Delta$ as the output.
- $s \leftarrow$ *PHRAdd($\Delta$): PHRAdd* takes a secure entries dictionary as input and stores its content on the blockchain MR-tree. It returns the status of the operation denoted by $s$ which can be either *success* or *failure*.
- $t \leftarrow$ *TrapGen(w): TrapGen* takes a keyword $w$ as input and returns the corresponding trapdoor $t$ for search.
- $\Gamma \leftarrow$ *Search(t): Search* takes a trapdoor $t$ as input and returns a list $\Gamma$ of encrypted identifiers of PHRs containing the keyword corresponding to $t$.
- $\Upsilon \leftarrow$ *RetId($\Gamma$): RetId* takes search result $\Gamma$ as the input and returns the actual PHR identifiers $\Upsilon$.

## B. OVERVIEW

Our scheme consists of the following phases: 1) *Setup*, 2) *Patient registration*, 3) *PHR generation and addition*, and 4) *Keyword search*.

We first give an overview of the scheme in this section and then describe each phase in detail.

In the *Setup* phase, the entities select security parameters, such as public-private key pairs and other secret values. This phase is executed only once during the system initialization. The second phase, *patient registration*, is executed by a patient whenever he/she visits a hospital for the first time. After completion of this phase, the patient receives a globally unique identifier and password. The patient uses the same identifier and password when visiting any other hospital later.

The third phase, *PHR generation and addition*, is initiated by a patient when he/she requires health diagnosis. The patient retrieves the list of doctors from the hospital and selects a doctor for treatment. He/she then sends a request to the chosen doctor via *HS* to retrieve the doctor's public key. Finally, he/she sends a token to the doctor via *HS*. The token is later used to authenticate the patient when he/she visits the doctor.

Upon connecting with the doctor, the patient is authenticated using the token. The doctor diagnoses him/her only if the token provided by the patient matches the one it received earlier. Diagnosis involves collecting data from the IoMT smart devices and giving medical prescriptions based on the patient's condition. The doctor then constructs a PHR of the patient containing all this information and sends it to the patient securely.[2] Upon receiving the PHR, the patient runs *KwExt* algorithm to retrieve unique keywords from the PHR. The patient then uploads the PHR to IPFS and receives a unique content-id for it denoted by $id_{phr}$. Finally, the patient runs *SEGen* algorithm and sends the output $\Delta$ from the algorithm, to the hospital server. The hospital server sends this data to its *BN*, which submits it as an add transaction to the blockchain network. Finally, this transaction is processed and added to blockchain by the peers. A flowchart of PHR generation process from the perspective of a patient can be seen in Fig. 2.

The fourth and final phase, *Keyword Search*, starts when a data user wants to search for PHR containing a particular keyword. He/she requests for a trapdoor corresponding to the keyword to be searched from the patient. The data user sends this trapdoor to the hospital server, which, in turn, runs the search algorithm and returns search results containing encrypted ids of the PHRs containing the given keyword along with some possible dummy entries. The dummy entries are a means to provide security to our scheme (see Section IV for details). The data user can get actual PHR ids and decryption keys by sending the search results to the patient/data owner. After obtaining the decrypted PHR id from the patient, the user can then directly query IPFS and

2. Channels are secured using ECIES encryption.

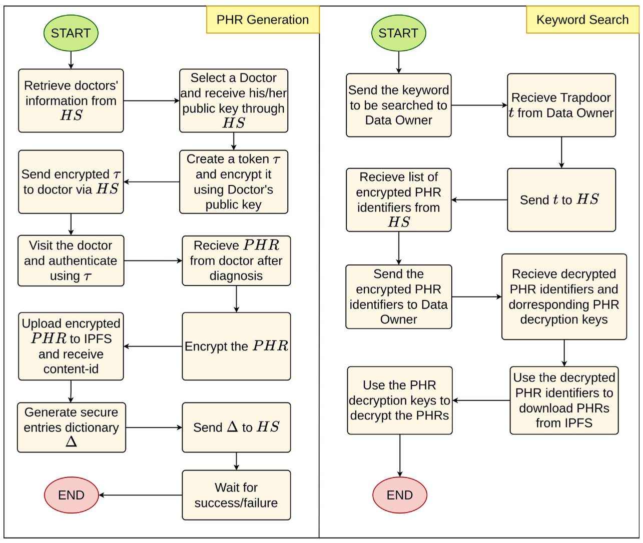

PHR Generation
START
Retrieve doctors' information from HS
Select a Doctor and receive his/her public key through HS
Create a token τ and encrypt it using Doctor's public key
Send encrypted τ to doctor via HS
Visit the doctor and authenticate using τ
Recieve PHR from doctor after diagnosis
Encrypt the PHR
Upload encrypted PHR to IPFS and receive content-id
Generate secure entries dictionary Δ
Send Δ to HS
Wait for success/failure
END
Keyword Search
START
Send the keyword to be searched to Data Owner
Recieve Trapdoor t from Data Owner
Send t to HS
Recieve list of encrypted PHR identifiers from HS
Send the encrypted PHR identifiers to Data Owner
Recieve decrypted PHR identifiers and dorresponding PHR decryption keys
Use the decrypted PHR identifiers to download PHRs from IPFS
Use the PHR decryption keys to decrypt the PHRs
END

**FIGURE 2.** **Flowcharts for PHR generation and keyword search.**

download the encrypted PHRs. Since the PHRs are stored in encrypted form, the data user first needs to decrypt them using the key sent by patient along with the actual PHR id. A flowchart of this process from the perspective of data user is also shown in Fig. 2.

## C. PHASES

We now describe the phases mentioned above, in detail.

### 1) SETUP

In this phase, the entities generate security parameters such as public-private key pairs. It has the following sub-phases.

- *Global Setup:* A non-singular elliptic curve $E_p(a, b)$ of the form: $y^2 = x^3 + ax + b \pmod p$ is chosen where $p$ is a large prime, and $a, b \in Z_p = \{0, 1, 2, \ldots, p-1\}$ such that $4a^3 + 27b^2 \neq 0 \pmod p$ is satisfied. Additionally, a generator $G$ is chosen from $E_p(a, b)$ whose order will be as large as $p$. Next, a "collision-resistant one-way cryptographic hash function, $h(\cdot)$" with output length $l$ is chosen which acts as a random oracle $\mathcal{O}$. A security parameter $n$ is also chosen. The keyword space which contains all the valid keywords that can be present in a PHR is denoted by $U$. Finally, for ECIES encryption, ECDH key exchange and probabilistic AES-OCB3 encryption [7] are chosen as the underlying components.
- *Patient Setup:* Each patient (device) randomly selects an integer $k_p \in Z_p^*$ as its private key, where $Z_p^* = \{1, 2, \ldots, p-1\}$. The corresponding public key is computed as $K_p = k_p \cdot G$ where $(\cdot)$ represents elliptic curve scalar multiplication, that is, $k_p \cdot G = G + G + \cdots + G$, $k_p$ times. In addition to public-private key pair, each patient randomly selects a master key $MK \in \{0, 1\}^n$. Further, each patient device maintains a dictionary which we term as "Current Head Dictionary ($CHD$)". It takes a keyword (sequence of characters) as its key and stores a PHR id as the corresponding value. Patient device also maintains a reverse look-up dictionary $RLD$ which maps encrypted PHR ids to actual PHR ids and the PHR decryption keys. The significance of these dictionaries will become clear in the PHR generation (see Section III).
- *Hospital Setup:* In a similar manner to a patient device, $HS$ randomly selects its private key $k_h \in Z_p^*$ and computes public key as $K_h = k_h \cdot G$. It is important to note that blockchain node, $BN$, will also use this key because $HS$ and $BN$ are maintained by the same entity (i.e., hospital). $HS$ also maintains a dictionary that we term as "User Dictionary ($UD$)" which takes the patient's unique id as the key and the patient's public key as the value. It is used to retrieve the public key of the patient for encryption when $HS$ wants to send some data to the patient.
- *Doctor Setup:* A doctor device is assumed to be maintained by each doctor. Doctor device randomly selects its private key $k_d \in Z_p^*$ and computes the public key as $K_d = k_d \cdot G$. Each doctor has a unique identity $id_{D_i}$ for the hospital where he/she is employed.
- *Data User Setup:* Any data user that wants to interact with the system must select a private key $k_u$ randomly from $Z_p^*$ and compute the corresponding public key as $K_u = k_u \cdot G$.

### 2) PATIENT REGISTRATION

Suppose a patient, $P$ wants to visit a hospital $H$. He/she registers himself/herself by providing the public key $K_p$ to $HS$ and then selecting a unique id and password. This id and password combination is used by the patient to authenticate himself or herself while communicating with $HS$. Upon completion of registration, the patient receives the public key $K_h$ of the hospital server $HS$. Note that there are other ways the patients can receive $K_h$ (for example, the hospital can list its public key on its website and use a certificate authority for its validation). However, to avoid providing unnecessary details and to simplify the scheme, we have assumed that the hospital server will simply send its public key back to the patients during registration. We have used similar ways to share public keys throughout our scheme, but if required they can listed on a public platform. It does not have any impact on the security of the scheme.

### 3) PHR GENERATION

An overview of the PHR generation process is given in Fig. 3. We describe the entire process in detail in this section.

- *Initiation:* To initiate PHR generation, a patient $P$ requests an updated list of doctors from the $HS$. The patient $P$ generates $\tau \in \{0, 1\}^*$ as a token and then selects a doctor $D$ from the list and sends a request to $D$ through $HS$ to get $D$'s public key. $P$ then encrypts $\tau$ using public key of $D$ and sends it to him/her via $HS$.

Later, when the patient interacts with $D$,[3] he/she sends the token $\tau$ to the doctor for verification. If it matches the token previously sent by the patient to the doctor, then the

3. Patient has the option to interact in-person or remotely depending on the need.

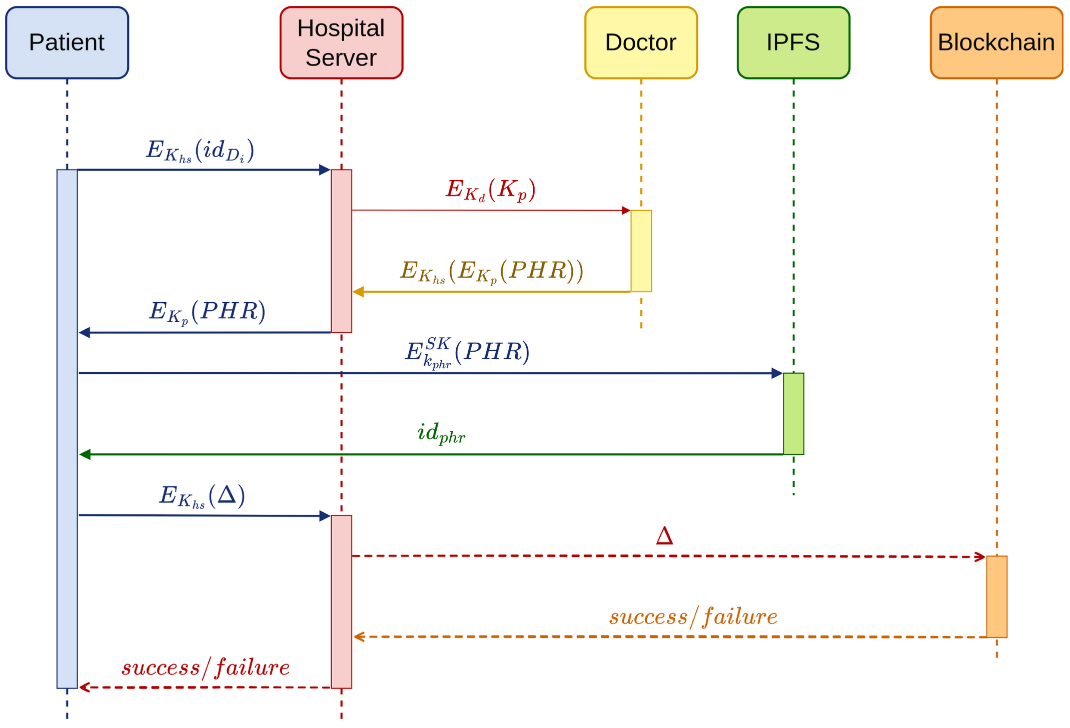


**FIGURE 3.** PHR generation sequence diagram.

**Algorithm 1:** Keyword Extraction

```
Input: PHR
Initialize a set $ks$
for each keyword $w$ in PHR do
    insert($ks$, $w$)
end
Convert $ks$ to list $l$
return $l$
```

doctor diagnoses the patient. If required, the doctor collects data from the medical devices and generates a PHR. Doctor device then encrypts the PHR using public key $K_p$ of the patient. This encrypted PHR is finally sent to the patient $P$.

- *Secure Entries Generation:* $P$ upon receiving the encrypted PHR, decrypts it using his/her private key and then runs Algorithm 2, which is an implementation of *SEGen*, over the PHR to get secure entries dictionary $\Delta$. Algorithm 2 is explained below in detail.

*PHR upload:* Algorithm 2 first generates the PHR encryption key $k_{phr} = h(MK \| TS)$, where $MK$ is $P$'s master key and $TS$ is the current time stamp. Next, the algorithm executes Algorithm 1, an implementation of *KwExt*, to generate the list of unique keywords, $W = \{w_0, w_2, \ldots, w_{n-1}\}$, present in the PHR. It then encrypts the PHR using the generated key and uploads it to IPFS, which returns a content id, denoted by $id_{phr}$. This id is encrypted as $id'_{phr} = E^{SK}_{k_{phr}}(id_{phr})$ to store it in the encrypted index. The encryption is necessary to prevent leakage of information to hospital. During implementation the encryption function $E^{SK}(\cdot)$ is replaced by AES-OCB3.

$P$ finally stores the actual PHR id $id_{phr}$ and PHR decryption key $k_{phr}$ in the reverse lookup dictionary *RLD* by using $id'_{phr}$ as the key. *RLD* is used to map an encrypted PHR id to an actual PHR id and the decryption key during keyword search (more details are provided in Section IV).

*Noise Addition:* Instead of using the original set of keywords $W$ to generate the secure entries, we first add some noise to it by merging $Q$ with $W$. $Q$ is a subset of the keyword space $U$. The new keyword set is denoted by $V$. This addition of noise is required in order to prevent attacks against our scheme (see Section II).

*Encrypted Search Index:* Before going any further into explanation of Algorithm 2, we need to understand the structure of the encrypted search index and how it is stored in the Blockchain. The encrypted search index can be considered as a dictionary with a unique keyword $w$ as the key and a linked list as the corresponding value. The linked list stores the encrypted identifiers of the PHRs that contain keyword $w$. This encrypted search index is partially stored on the patient device and partially in the blockchain's Merkle-Radix (MR) tree. Specifically, the key-set of this dictionary is stored on the patient's device in the form of *CHD* dictionary; while the value-set, which is essentially a set of linked lists, is stored in the MR-tree. This division is essential for the security of the search index, because generation of trapdoor is dependent on the key-set. If we store it on the blockchain, anyone can publicly access it and generate the trapdoor corresponding to a keyword. This would defeat the purpose of having a trapdoor.

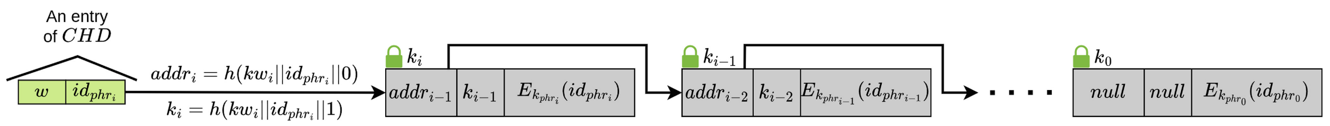

An entry of $CHD$
$w$
$id_{phr_i}$
$addr_i = h(kw_i||id_{phr_i}||0)$
$k_i = h(kw_i||id_{phr_i}||1)$
$k_i$
$addr_{i-1}$
$k_{i-1}$
$E_{k_{phr_i}}(id_{phr_i})$
$k_{i-1}$
$addr_{i-2}$
$k_{i-2}$
$E_{k_{phr_{i-1}}}(id_{phr_{i-1}})$
$k_0$
$null$
$null$
$E_{k_{phr_0}}(id_{phr_0})$

FIGURE 4. A single entry of the conceptual encrypted index.

**Algorithm 2:** Secure Entries Generation

```
Input: $PHR, Q$
$TS \leftarrow$ Current Timestamp
$k_{phr} \leftarrow h(MK \parallel TS)$
$W \leftarrow \mathrm{KwExt}(PHR)$
Upload $E^{SK}_{k_{phr}}(PHR)$ to IPFS and receive $id_{phr}$
$id'_{phr} \leftarrow E^{SK}_{k_{phr}}(id_{phr})$
$RLD[id'_{phr}] = (id_{phr}, k_{phr})$
$V \leftarrow Q \cup W$
for $0 \le i < |V|$ do
    $kw \leftarrow h(MK \parallel V_i)$
    $addr \leftarrow h(kw \parallel id_{phr} \parallel 0)$
    $k \leftarrow h(kw \parallel id_{phr} \parallel 1)$
    if $CHD[V_i] = null$ then
        $addr^{prev} \leftarrow null$
        $k^{prev} \leftarrow null$
    else
        $id^{prev}_{phr} \leftarrow CHD[V_i]$
        $addr^{prev} \leftarrow h(kw \parallel id^{prev}_{phr} \parallel 0)$
        $k^{prev} \leftarrow h(kw \parallel id^{prev}_{phr} \parallel 1)$
    end
    if $V_i \in W$ then
        $c \leftarrow E^{SK}_{k}(addr^{prev}, k^{prev}, id'_{phr})$
    else
        $r \overset{<DOLLAR>}{\leftarrow} \{0, 1\}^{l_{id_{phr}}}$
        $c \leftarrow E^{SK}_{k}(addr^{prev}, k^{prev}, E^{SK}_{k_{phr}}(r))$
    end
    $\Delta[addr] \leftarrow c$
    $CHD[V_i] \leftarrow id_{phr}$
end
return $\Delta$
```

Fig. 4 shows one entry of this conceptual dictionary. Recall that $CHD$ is a dictionary maintained by the client device. Specifically, $CHD[w]$ stores the id of the PHR that contains $w$ in it and was added most recently to the system. This PHR identifier forms the head node of the linked list associated with keyword $w$ along with some dummy nodes (discussed in part - Node Generation ). As mentioned earlier, the linked-list corresponding to key $w$ is used to store the identifiers of all the PHRs that contain $w$. Therefore, there is a linked list corresponding to every unique keyword contained in the set of PHRs of a patient. Each node of a linked list is stored individually in the MR-tree of the blockchain and has a unique MR-tree address. A node is linked to the next one by storing the MR-tree address of the next node in the current node. We have used a linked-list over other data structures, such as an array, because by design it ensures forward security (details are provided in proof of Theorem 2).

*Node Generation:* As discussed earlier, each keyword present in $V$ should have a corresponding linked list. The next part of the algorithm generates the nodes for these linked lists. For each keyword $w \in V$ a new node for the linked list corresponding to $w$ will be created. This node is added to the front of the linked list and becomes the new head node. Adding the node from front, rather than back, is required to ensure forward security. In $i^{th}$ iteration of the *for* loop present in the algorithm, a new head node for the linked list corresponding to keyword $V_i$ is created. Specifically, for $0 \le i < |V|$, the algorithm generates $kw$, $addr$ and $k$ as follows: $kw = h(MK \parallel V_i)$; $addr = h(kw_i \parallel id_{phr} \parallel 0)$; $k = h(kw \parallel id_{phr} \parallel 1)$. Here, $kw$ is an intermediate expression, $addr$ is the MR-tree address where the node will be stored and $k$ is the key which will be used to encrypt the contents of the node.

Before constructing the new head node, the address and decryption key of the current head node are required. Therefore, the algorithm obtains the PHR id that forms the current head node from $CHD$ as $id^{prev}_{phr} = CHD[V_i]$. This id is then used to generate the MR-tree address of the new head node and the decryption key for the contents of new head node as follows:

$$addr^{prev} = h\Big(kw \parallel id^{prev}_{phr} \parallel 0\Big)$$
$$k^{prev} = h\Big(kw \parallel id^{prev}_{phr} \parallel 1\Big).$$

If $CHD$ does not contain an entry for $V_i$, then $addr^{prev}$ and $k^{prev}$ are set to *null* value.

Finally, the algorithm calculates $c$. If the keyword $V_i$ is present in the PHR, then $c$ is computed as follows.

$$c = E^{SK}_{k}\Big(addr^{prev}, k^{prev}, id'_{phr}\Big).$$

Otherwise, the value of $id'_{phr}$ is replaced by encryption of a random value $r$ of same length as $id_{phr}$ as follows.

$$c = E^{SK}_{k}\Big(addr^{prev}, k^{prev}, E^{SK}_{k_{phr}}(r)\Big).$$

The condition above is necessary to find the correct PHR identifiers during keyword search. Details are present in Keyword Search Section IV.

Now, $c$ is the new head node of the linked list corresponding to the keyword $V_i$. It contains three values: a) address of the previous head, b) key of the previous head, and c) encrypted PHR id or encrypted random value. Also note

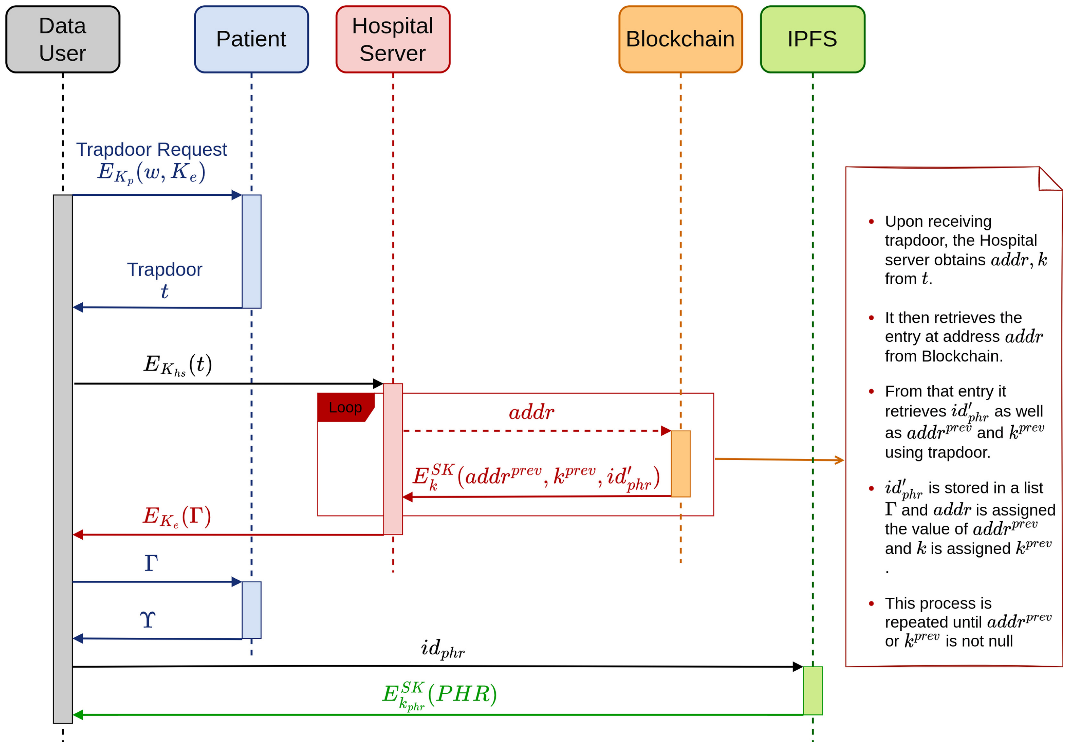

Data User
Patient
Hospital Server
Blockchain
IPFS
Trapdoor Request
$E_{K_p}(w, K_e)$
Trapdoor
$t$
$E_{K_{hs}}(t)$
Loop
$addr$
$E_k^{SK}(addr^{prev}, k^{prev}, id'_{phr})$
$E_{K_e}(\Gamma)$
$\Gamma$
$\Upsilon$
$id_{phr}$
$E^{SK}_{k_{phr}}(PHR)$
Upon receiving trapdoor, the Hospital server obtains $addr, k$ from $t$.
It then retrieves the entry at address $addr$ from Blockchain.
From that entry it retrieves $id'_{phr}$ as well as $addr^{prev}$ and $k^{prev}$ using trapdoor.
$id'_{phr}$ is stored in a list $\Gamma$ and $addr$ is assigned the value of $addr^{prev}$ and $k$ is assigned $k^{prev}$.
This process is repeated until $addr^{prev}$ or $k^{prev}$ is not null

**FIGURE 5.** Search process.

that, in Fig. 4 we have not shown dummy nodes for the sake of simplicity. However, in practice there will be dummy nodes mixed with the valid nodes.

All new head nodes denoted by $\{c_i \mid i \in \{0, 1, 2, \ldots, |V| - 1\}\}$ are stored in dictionary $\Delta$ by using $addr_i$ as the key. The algorithm returns $\Delta$ as its output.

• *Uploading to Blockchain:* After generating $\Delta$ Patient $P$ sends it to $HS$ which forwards it to the "Blockchain Node ($BN$)". $BN$ then retrieves each key values pair $(addr, c)$ and stores $c$ at address $addr$ of the MR-tree, and records this transaction in a block. Upon completion of this operation, $BN$ notifies $HS$ about the successful execution of the transaction, which in turn shares the success response with $P$.

### 4) KEYWORD SEARCH

An overview of the keyword search process is given in Fig. 5. This section describes the search process in detail.

• *Trapdoor Generation:* Suppose a data user ($DU$) wants to search for a PHR of a patient $P$ that contains a specific keyword $w$. $DU$ sends the keyword $w$ to $P$ to receive a trapdoor $t$ corresponding to $w$. For this, $P$ runs Algorithm 3 which calculates $kw = h(MK \parallel w)$. It then retrieves the PHR id stored in head node of the linked list corresponding to $w$ as follows:

$$id_{phr}^{head} = CHD[w]$$

The algorithm then calculates the address and key of the head node as follows:

$$addr = h\Big(kw \parallel id_{phr}^{head} \parallel 0\Big)$$

**Algorithm 3:** Trapdoor Generation

```
Input: $w$
$kw \leftarrow h(MK \parallel w)$
if $CHD[w] = null$ then
    $id_{phr}^{head} \leftarrow null$
else
    $id_{phr}^{head} \leftarrow CHD[w]$
end
$addr \leftarrow h(kw \parallel id_{phr}^{head} \parallel 0)$
$key \leftarrow h(kw \parallel id_{phr}^{head} \parallel 1)$
$t \leftarrow addr \parallel key$
return $t$
```

$$k = h\Big(kw \parallel id_{phr}^{head} \parallel 1\Big)$$

$addr$ is the MR-tree address of the head node of the linked list corresponding to $w$ and $k$ is the symmetric key using which contents of the head node were encrypted. The algorithm returns $t = addr \parallel k$ as its output which is then sent by $P$ as the trapdoor to $DU$.

• *Search:* $DU$ then sends the trapdoor $t$ and its own public key to $HS$ after encrypting it using $K_h$.[4] $HS$ then executes Algorithm 4 which directly queries the state of the blockchain's MR-tree using REST API. Specifically, using the trapdoor $t$, $HS$ obtains the address $addr$ and key $k$ required to obtain the head node of the linked list. $HS$

4. $K_h$ can be obtained directly from hospital.

**Algorithm 4:** Keyword Search

```
Input: $t$
$addr, key \leftarrow$ Parse $t$
Initialize a list $\Gamma$
while $True$ do
    Send a request to the blockchain node to get data
      at $addr$
    Store the response in $res$
    if $res.code = ok$ then
        $C \leftarrow res.content$
        $addr^{prev}, k^{prev}, x \leftarrow D_{key}(C)$
        $Append(\Gamma, x)$
        if $addr^{prev} = null$ then
            break
        end
        $addr \leftarrow addr^{prev}$
        $key \leftarrow k^{prev}$
    else
        return $(not\text{-}found, null)$
    end
end
$c \leftarrow E_{K_h}(\Gamma)$
return $(ok, c)$
```

**Algorithm 5:** Retrieve Identifiers

```
Input: $\Gamma$
Initialize list $\Upsilon$
for $0 \leq i < |\Gamma|$ do
    if $RLD$ contains $\Gamma_i$ then
        $id_{phr}, k_{phr} \leftarrow RLD[\Gamma_i]$
        $append(\Upsilon, (id_{phr}, k_{phr}))$
    end
end
return $\Upsilon$
```

sends a request using the blockchain's REST API to retrieve data at the address *addr* of the MR-tree. The response of the request is $C = E_k^{SK}(addr^{prev}, k^{prev}, x)$, where $x$ can be either $id'_{phr}$ or $E_{k_{phr}}^{SK}(r)$. $x$ is appended to a list $\Gamma$ by the *HS*. $addr^{prev}$ and $k^{prev}$ are then used as the new values to obtain the previous node. This process continues until $addr^{prev}$ or $k^{prev}$ is not null. After completion of Algorithm 4, *HS* sends $\Gamma$ to *DU* after encrypting it using *DU*'s public key. Note that, for simplicity, we assume that HTTP protocol is being used for communication. Therefore, we have used some of the HTTP status codes like "ok" and "not-found" in the algorithm to denote the status of responses.

- *PHR Identifier Retrieval:* *DU* sends $\Gamma$ to *P* in order to obtain the actual PHR identifiers. *P* runs Algorithm 5 to retrieve the actual identifiers of PHRs that contain *w*. The algorithm uses reverse lookup dictionary *RLD* for this purpose. If $\Gamma_i$ denotes an element in $\Gamma$, then *P* checks if an entry corresponding to $\Gamma_i$ exists in *RLD*. If it exists, then actual PHR identifier and decryption key are retrieved as $id_{phr}, k_{phr} \leftarrow RLD[\Gamma_i]$ and are appended to list $\Upsilon$ along with the PHR decryption key $k_{phr}$. Finally, the list $\Upsilon$ is sent to *DU* which can then retrieve the encrypted PHR from IPFS and decrypt them.

## VI. SECURITY ANALYSIS

For proving the security of our scheme, we use the real-ideal simulation paradigm [43] that is used in most of the previous searchable encryption works [12], [13], [21] and prove that our scheme provides *semantic security* against an *adaptive adversary* in a semi-honest setting. First, we define the threat model under which our system is secure. This is followed by definition of information leakage of our scheme in the form of leakage functions. We then show that a simulator that takes these leakage functions as its inputs is indistinguishable from the real protocol for the adversary. This is followed by a proof that our scheme provides *forward secrecy* and is also *verifiable*.

### A. THREAT MODEL

We evaluate security our proposed scheme by considering hospital as the adversary. All the important data in the system flows through the hospital, therefore, if we can show that our scheme is secure against an adversary with capabilities of hospital, we can be sure that it is secure against any adversary with lesser capabilities.

We consider a hospital to be *honest-but-curious*. It executes the protocol honestly, but can try to learn about the information it is unauthorized to access. It can do so by using the data leaked to the hospital server or the Blockchain node during execution of the protocol. We collectively represent this leakage in the form of leakage-functions which are described in Section VI-B.

We assume a doctor to be trusted, because he/she is the entity which generates the contents of the PHRs. Moreover, by securing the communication between any two entities in the system using ECIES encryption, we can ensure that a doctor is not impersonated by an adversary.

### B. SECURITY DEFINITIONS

*Definition 6 (Search Pattern):* A search pattern describes whether a keyword previously searched for is searched again. Formally, if $m$ is the number of search queries that have been made, the search pattern is a matrix $Z$ of dimensions $m \times m$ such that the value $Z_{i,j}$ is 1 if $w_i = w_j$; otherwise, it is 0. Here, $w_i$ is the keyword searched during $i^{th}$ query of a data user.

*Definition 7 (Leakage):* The leakage for $\Sigma$ is defined as a tuple $L = (L_{SEGen}, L_{PHRAdd}, L_{Search})$ where $L_{SEGen}$, $L_{PHRAdd}$ and $L_{Search}$ are leakages that occur during the secure entry generation, addition and search, respectively. These leakages are inherent in all the efficient searchable encryption schemes present till now. Note that, our scheme does not leak anything during the setup phase, because there is no interaction between the entities during this phase. Each entity performs the setup offline, and as a result, there is no possibility of any leakage except through physical means.

We define $L_{SEGen}$, $L_{PHRAdd}$ and $L_{Search}$ as follows:

- $\mathbf{L_{SEGen}}$: *SEGen* takes a PHR as input and gives secure entries dictionary $\Delta$ as the output. In *SEGen*, the length (or size) of the PHR is leaked during its upload to the IPFS. Since the PHR is encrypted, no other information is leaked but the length can still be figured out from the ciphertext. We denote the length of the PHR by $m$. Thus, $L_{SEGen}$ is defined as a function that takes a PHR as input and outputs $m$.
- $\mathbf{L_{PHRAdd}}$: During the addition of a PHR, the only information that is leaked by our scheme is the number of entries in $\Delta$. This is because we give only $\Delta$ to the *HS*. Since the key is output of a hash function and the value is output of a probabilistic encryption function, *HS* can only figure out its size (the number of entries in $\Delta$) but not the actual content. Thus, $L_{PHRAdd}$ takes a set of these keywords $W$ as input and outputs $\alpha$.
- $\mathbf{L_{Search}}$: During search corresponding to a keyword $w$, the information that is leaked to the server is the number of files containing the keyword $w$, denoted by $\beta$, and search pattern $Z$. Formally, $L_{Search}$ takes a keyword $w$ as input and outputs $(\beta, Z)$.

*Definition 8 (Adaptive Semantic Security for* $\Sigma$*):* We define security of our scheme similar to [21]. Let $n \in \mathbf{N}$ be a security parameter and $m = poly(n)$ be a positive integer, where $\mathbf{N}$ is the set of natural numbers. For simplicity of proof we assume that patient acts as a data owner as well as a data user. This does not affect the security of the scheme as data user is considered trusted.

Consider the experiments $Ideal_{\mathcal{F},\mathcal{S},\Psi}$ and $Real_{\Sigma_{\mathcal{F}},A,\Psi}$ executed between stateful adversary $A$ and stateful simulator $\mathcal{S}$ using leakage functions $L_{PHRAdd}$ and $L_{Search}$.

- $Ideal_{\mathcal{F},\mathcal{S},\Psi}$: Environment $\Psi$ asks the data owner to run SEGen, Add or Search operation by giving it the required information. For SEGen, $\Psi$ gives a PHR as input. For *Add*, $\Psi$ gives secure-entries to be added to blockchain as input and for *Search* it gives a keyword. Data owner sends the operation to ideal functionality $\mathcal{F}$. $\mathcal{F}$ gives the corresponding leakage to $\mathcal{S}$ which returns either abort or continue to $\mathcal{F}$. For SEGen, it returns abort or secure entries dictionary. For Add, it returns abort or "done" and for Search, it returns abort or list of encrypted PHR/dummy identifiers. $\Psi$ observes the output and finally returns a bit $b$ as the output.
- $Real_{\Sigma_{\mathcal{F}},A,\Psi}$: Environment $\Psi$ asks the data owner to run *SEGen*, *Add* or *Search* operation and provides the required information. For *SEGen*, it provides a PHR. For *Add*, it provides a secure entries dictionary and for *Search* it provides a keyword. Data Owner executes the corresponding operation in presence of a real world adversary $A$. Data Owner outputs either abort or secure entries dictionary for SEGen. For Add it outputs abort or 'Done' and for Search, it outputs abort or list of encrypted PHR/dummy identifiers. $\Psi$ observes the data owner's output and finally returns a bit $b$ as the output.

We say that a scheme $\Sigma_F$ emulates an ideal functionality $\mathcal{F}$ if for all real world PPT adversary $A$, there exists a PPT simulator $\mathcal{S}$ such that for all polynomial time environments $\Psi$ there exists a negligible function $negl(n)$ such that:

$$|Pr\big[Real_{\Sigma_{\mathcal{F}},A,\Psi}(n) = 1\big] - Pr\big[Ideal_{\mathcal{F},\mathcal{S},\Psi}(n) = 1\big]| \\ \leq negl(n)$$

*Definition 9 (Forward Security):* A scheme $\Sigma = (KwExt$, *SEGen*, *PHRAdd*, *TrapGen*, *Search*, *RetId*$)$ is forward secure [16], if the leakage that occurs due to previous *Add* and *Search* operations do not reveal any partial information about a newly added PHR except what is revealed by leakage $L$.

*Definition 10 (Verifiability):* A scheme $\Sigma = (KwExt$, *SEGen*, *PHRAdd*, *TrapGen*, *Search*, *RetId*$)$ is said to be verifiable, if the results returned to the user through the *Search* operation are valid.

## C. SECURITY CLAIMS

### 1) ADAPTIVE-SEMANTIC SECURITY

The correctness of Theorem 1 ensures that our scheme is confidential against a PPT adversary in semi-honest model. Once proven secure against hospital, we can be sure that our scheme is also secure against any other adversary that does not have access to all the leaked information.

*Theorem 1:* If $h$ is a hash function and $E^{SK}(\cdot)$ is a CPA secure encryption scheme, then our PHR sharing scheme $\Sigma$ is secure based on Definition 8 under random oracle model.

*Proof:* To prove that the real game and ideal game are indistinguishable by any PPT distinguisher, we provide an implementation of PPT Simulator $\mathcal{S}$ that takes only the information provided by leakage functions as input to simulate patient behaviour in a way that is indistinguishable from real patient. $\mathcal{S}$ initializes two dictionaries *LenDict* and *ResDict*. *LenDict* uses a keyword as the key and stores an integer as the value. *ResDict* uses a keyword as the key and stores a list as the value.

For SEGen, $\mathcal{S}$ returns a secure-entries dictionary by executing the $S_{SEGen}$ routine described in Fig. 6. The routine takes $\alpha = |V|$ as the input. $\mathcal{S}$ obtains $V$ using $W \cup Q$, where $W$ can be easily obtained from the PHR and $Q$ is known publicly. The routine simply creates $\alpha$ number of random entries for the dictionary $\Delta$ and returns it as the output. The key is a randomly chosen from $\{0, 1\}^l$ and the value is probabilistic encryption of $0^{(l+n+l_{id_{phr}})}$. Here, $l$ is the output length of hash function, $n$ is the security parameter and $l_{id_{phr}}$ is the length of PHR identifier. The key and value of each entry are indistinguishable based on random oracle model and the CPA security of $E^{SK}$, respectively. Thus, $\Psi$ is unable to distinguish between the real and ideal games using the *SEGen* operation.

During Add $\mathcal{S}$ simply returns 'Done' because the addition of a PHR in real game does not return anything besides abort or 'Done'.

```
$\mathcal{S}_{\mathbf{SEGen}}(\alpha)$
1 :  Initialize dictionary $\Delta$
2 :  for $1 \leq j \leq \alpha$
3 :    $addr \xleftarrow{\$} \{0,1\}^l$
4 :    $k \xleftarrow{\$} \{0,1\}^l$
5 :    $\Delta[addr] = E_k^{SK}(0^{(l+n+l_{id_{phr}})})$
6 :  endfor
7 :  return $\Delta$
$\mathcal{S}_{\mathbf{Search}}(\beta, \mathbf{w})$
1 :  if $LenDict[w] = null$
2 :    $LenDict[w] \leftarrow \beta$
3 :    Initialize $ResDict[w]$ as an empty list
4 :    for $0 \leq i < \beta$
5 :      $k \leftarrow \{0,1\}^n$
6 :      $append(ResDict[w], E_k^{SK}(0^{l_{id_{phr}}}))$
7 :    endfor
8 :  endif
9 :  if $\beta \neq LenDict[w]$
10 :   for $0 \leq i < (\beta - LenDict[w])$
11 :     $append(ResDict[w], E_k^{SK}(0^{l_{id_{phr}}}))$
12 :   endfor
13 :   $LenDict[w] \leftarrow \beta$
14 : endif
15 : return $ResDict[w]$
```

**FIGURE 6.** Implementation of simulator.

During Search $\mathcal{S}$ must return a list of encrypted PHR/dummy identifiers. For this $\mathcal{S}$ executes the routine described in Fig. 6. It takes an input $\beta$ which is the length of the list of encrypted PHR/dummy identifiers. This information is provided to $\mathcal{S}$ by the leakage function $L_{Search}$. $\mathcal{S}$ uses *LenDict* to keep track of the length of the list that is to be returned as output when keyword $w$ is searched. When $w$ is searched for the first time, and the leakage corresponding to it is $\beta$ then a new entry $(w, \beta)$ is created in *LenDict*. *ResDict* also gets a new entry with $w$ as the key and a list containing $\beta$ encryptions of $0^{l_{id_{phr}}}$ ($l_{id_{phr}}$ is the length of the PHR identifier, which depends on the hash function used by IPFS). This list is returned as the output. Each entry of the list is indistinguishable based on the CPA-Security of $E^{SK}$. When $w$ is searched again, the same list is returned. Now, for the case when $w$ is searched after addition of one or more PHR, the simulator just adds the number of entries equal to the difference between $\beta$ and *LenDict*$[w]$. *LenDict*$[w]$ is then updated to $\beta$ and *ResDict*$[w]$ is returned. Thus, in every case, $\Psi$ gets consistent results and it is therefore unable to distinguish the ideal game from real game. ■

### 2) FORWARD SECURITY

It is an important property of any SE scheme. It ensures that the data leaked during the operations of the proposed scheme do not reveal any partial information about the PHRs that will be added in the future. For example, if an adversary knows that a PHR contains a certain keyword, he/she must not be able to determine if a PHR that is added later contains the same keyword or not.

*Theorem 2:* Our scheme $\Sigma$ is forward-secure in semi-honest model under the random oracle model.

*Proof:* When a new PHR is added, the address is computed as $h(kw \parallel id_{phr} \parallel 0)$ (see Section III). Here, $id_{phr}$ is computed by taking the hash of the contents of the PHR and therefore is unique for each PHR. Thus, even for a keyword that occurs multiple times in different PHRs, the address generated will always be a random value output by the random oracle $h$. Further, new nodes are added to the head of the linked list in the existing encrypted index. This ensures that upon addition of a new PHR id to the same list does not require any modification in the encrypted index. Even in an extreme scenario where an adversary has once queried all the keywords whose entries are in the encrypted index,[5] the adversary will not be able to figure out which keywords are present in the newly added PHR. Hence, our scheme is forward secure. ■

### 3) VERIFIABILITY

Verifiability refers to the fact that the results returned by the hospital to the user are correct. For example, an adversary can return only partial results to the data user in order to save computational cost. Out of total number of PHRs containing a given keyword, the server may return only a fraction of those. A data sharing scheme must be safeguarded against such types of attacks.

*Theorem 3:* Our scheme $\Sigma$ is verifiable under the standard blockchain model.

*Proof:* According to the standard blockchain model [5], a blockchain is trusted for correctness and availability, but not privacy. Since a hospital server is part of a consortium blockchain network and is honest but curious, we can be sure that once the hospital receives a search request, it will forward it to the blockchain node. The blockchain node will return the correct search results from the distributed ledger by using Algorithm 4 because it is a trusted for correctness under the standard blockchain model. Therefore, the results received by the user will always be guaranteed to be valid. ■

## D. SECURITY AGAINST ATTACKS

### 1) QUERY RECOVERY USING FILE INJECTION

In 2016, Zhang et al. [44] proposed a query recovery attack that could retrieve all the query made by the user. This devastating attack works on the schemes where the adversary is

5. This can happen when the adversary has some pre-knowledge about the patient's disease.

able to inject files of their own choice and the query pattern is leaked to the adversary during search. This attack is prominent in cases such as application of SE over emails where the adversary can just send an email containing keywords of their own choice to the data owner.

In our case, however, this attack is resisted by the use of dummy entries (see Algorithm 2). The attack relies on the knowledge of a subset of keyword space (denoted by $Q$) and the access pattern (the set of identifiers of files that contain the keyword being searched), however, our scheme hides the access pattern from the hospital by adding dummy entries from keyword space in the secure entries dictionary. Now, if the adversary somehow manages to inject a file with keywords of their own choice, then when a data user searches for a keyword $w$ from $Q$, the search results (which consists of encrypted PHR/dummy identifiers) returned do not actually reveal if the PHR contains $w$ or not. This is because the result returned can be a dummy encryption. The data user will have to query the data owner to get the actual results. This way the adversary won't be able to learn the access pattern better than guessing.

#### 2) CONTENT RECOVERY USING KNOWN KEYWORD SET

This is an attack specific to SE schemes that allows addition of single files to the system instead of adding as a batch. If the adversary knows a keyword set $W' \subset U$, which contains some or all of the keywords in the keyword set $W$ of the PHR, then after addition of a PHR the adversary can immediately search for all the keywords in $W'$ and determine the content of the PHR with high accuracy. This attack is very devastating if $W'$ and $W$ have large number of keywords in common.

Our scheme provides protection against this attack by merging a large enough set of keywords $Q \subseteq U$ with the set of keywords $W$ present in the PHR (see Algorithm 2). After getting the search results, the data user has to request data owner to execute Algorithm 5 over the search results to get the actual PHR identifiers. This ensures that an adversary cannot gets the actual PHR identifiers unless he/she impersonates the data user. The impersonation will be possible only if the adversary can steal the private key of the data user. Therefore, just as in previous attack, the adversary will not be able to determine if the search results obtained are valid or just some dummy entries. Thus, our scheme is secure against PHR recovery using known keyword set attack.

## VII. PERFORMANCE EVALUATION

In this section, we present the results of evaluation of the proposed scheme under different conditions, and also provide a comparative study with the existing competing schemes.

### A. THEORETICAL ANALYSIS

In this section, we provide an asymptotic analysis of various algorithms in our scheme.

TABLE 3. Communication cost.

| Phase | | Cost (in bytes) |
|---|---|---|
| Add | PHR request | $32 + m$ |
| | PHR upload | $m + 32$ |
| | Secure entries upload | $80\|\Delta\|$ |
| Search | Trapdoor request | $\|w\| + 33 + 48$ |
| | Search request | $48 + (32 + 80)\|\Gamma\| + 32\|\Gamma\|$ |
| | Identifier retrieval | $2 \times 32\|\Gamma\|$ |
| | PHR retrieval | $32 + m$ |

- *Keyword Extraction:* This algorithm iterates over all the keywords present in the PHR. Therefore, it has time complexity of $O(|W|)$, where, $|W|$ is the number of keywords present in the PHR.
- *Secure Entries Generation:* The most time consuming operation in this algorithm is the for-loop which iterates over the set of keywords $V$. Therefore, the complexity is $O(|V|)$.
- *Trapdoor Generation:* All the operations present in this algorithm have constant execution time. Therefore, the time complexity is $O(1)$.
- *Keyword Search:* The most time consuming operation here is the while-loop which runs till the complete list of nodes is extracted. Hence, the time complexity is $O(|\Gamma|)$ where $\Gamma$ is the set identifiers corresponding to a keyword in the encrypted index.
- *Retrieve Identifiers:* In this algorithm, the most time consuming operation is the for-loop which iterates over each element of $\Gamma$. Hence, the time complexity is $O(|\Gamma|)$.

In Table 3, we show the total communication overhead of our scheme. Here, 32 is the size of a key of the index $\Delta$, or the size of the PHR identifier, 33 is the size of public key, 48 is the size of trapdoor, 80 is the size of each entry in $\Delta$ or the size of contents of a node in encrypted index and $m$ is the size of PHR.

### B. TEST-BED SETUP AND ORGANIZATION

Fig. 7 shows the organization of our test-bed setup. We used four desktop systems shown in Fig. 9 and a laptop shown in Fig. 10 as the computing hardware for our simulation. Each of the desktops had an Intel core i5 12400 processor, 16 GB RAM, and 1TB SSD, and the laptop had an Intel core i7-9750H processor, 16 GB RAM, and 256 GB SSD. Each of the desktops ran Ubuntu Server 22.04 LTS and had Docker (version 20.10.22) installed on it. Docker was used to simulate the Hyperledger Sawtooth nodes and IPFS nodes. The laptop used Ubuntu Desktop 22.04 LTS as the Operating System (OS). It ran an instance of the hospital server, doctor device, and patient device. All the systems were part of the same LAN. The entire test-bed setup took around 4000 lines

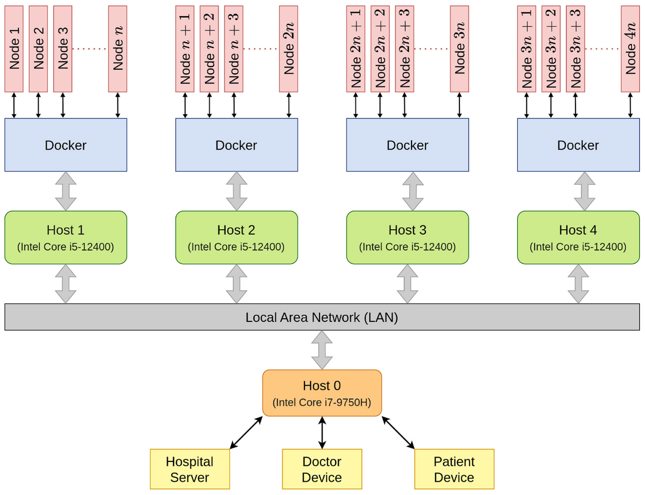
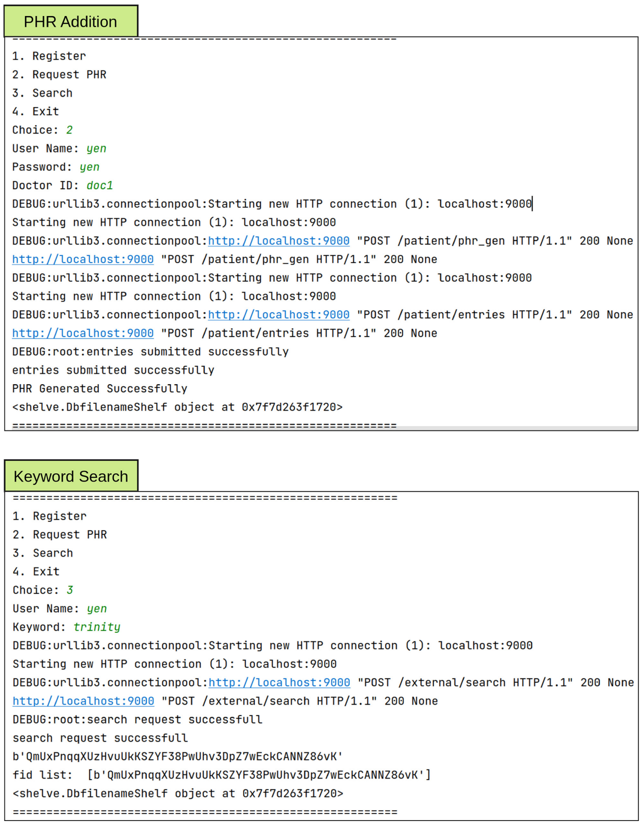


FIGURE 7. **Experimental setup organization.**



```
=========================================================
1. Register
2. Request PHR
3. Search
4. Exit
Choice: 2
User Name: yen
Password: yen
Doctor ID: doc1
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): localhost:9000
Starting new HTTP connection (1): localhost:9000
DEBUG:urllib3.connectionpool:http://localhost:9000 "POST /patient/phr_gen HTTP/1.1" 200 None
http://localhost:9000 "POST /patient/phr_gen HTTP/1.1" 200 None
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): localhost:9000
Starting new HTTP connection (1): localhost:9000
DEBUG:urllib3.connectionpool:http://localhost:9000 "POST /patient/entries HTTP/1.1" 200 None
http://localhost:9000 "POST /patient/entries HTTP/1.1" 200 None
DEBUG:root:entries submitted successfully
entries submitted successfully
PHR Generated Successfully
<shelve.DbfilenameShelf object at 0x7f7d263f1720>
=========================================================
```

```
=========================================================
1. Register
2. Request PHR
3. Search
4. Exit
Choice: 3
User Name: yen
Keyword: trinity
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): localhost:9000
Starting new HTTP connection (1): localhost:9000
DEBUG:urllib3.connectionpool:http://localhost:9000 "POST /external/search HTTP/1.1" 200 None
http://localhost:9000 "POST /external/search HTTP/1.1" 200 None
DEBUG:root:search request successfull
search request successfull
b'QmUxPnqqXUzHvuUkKSZYF38PwUhv3DpZ7wEckCANNZ86vK'
fid list:  [b'QmUxPnqqXUzHvuUkKSZYF38PwUhv3DpZ7wEckCANNZ86vK']
<shelve.DbfilenameShelf object at 0x7f7d263f1720>
=========================================================
```

FIGURE 8. **Logs outputs.**

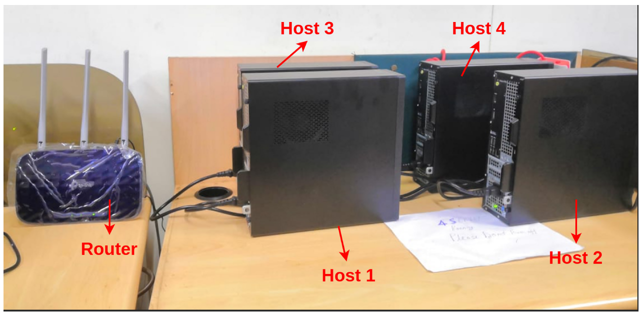


FIGURE 9. **Experimental setup 1.**

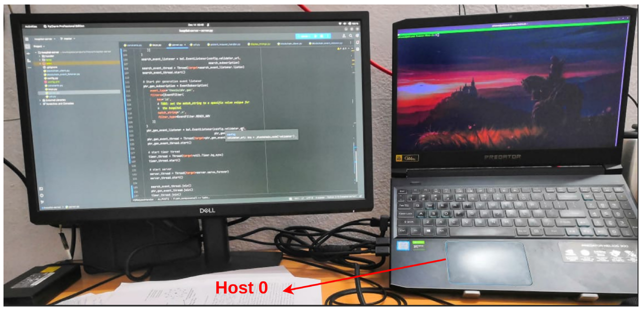


FIGURE 10. **Experimental setup 2.**

TABLE 4. **Timings of core operations.**

| Operation | Time (ms) |
|---|---|
| SHA-256 | 0.0014 |
| AES-OCB3 Encryption | 0.0161 |
| AES-OCB3 Decryption | 0.0149 |
| ECIES Encryption (ECDH + AES-OCB3) | 0.3425 |
| ECIES Decryption (ECDH + AES-OCB3) | 0.6801 |

of code written mainly in Python (version 3.10.6). We used the *cryptography* Python library for various cryptographic operations. Fig. 8(c) shows the output logs during execution of our scheme.

## C. EXPERIMENTAL RESULTS

We used Secure Hash Algorithm (SHA-256) as the replacement of the hash function $h$. For ECIES, we use the combination of ECDH key exchange and Advanced Encryption Standard (AES), wherein ECDH uses the standard SECP256-R1 curve specified by the "National Institute of Standards and Technology (NIST)" and AES uses offset codebook version 3 (OCB3) mode. For symmetric key encryption and decryption, AES in OCB3 mode is used. Table 4 shows the timings of the core operations that have been used. The timings shown in this table were calculated by taking the average over timings of 1000 executions for an input size of 256 bytes, and the hardware used was Intel Core i7-9750H @ 2.60 Hz, 16 GB RAM, and 256 GB solid-state drive (SSD).

Next, we present the extensive simulation results for our scheme by presenting the results of six experiments that we carried out after the implementation of the proposed scheme. For the experiments, we have taken a minimum number of unique keywords in a PHR to be 200. This value is based on the Heaps' law [45] for English text (with $K = 3.67$

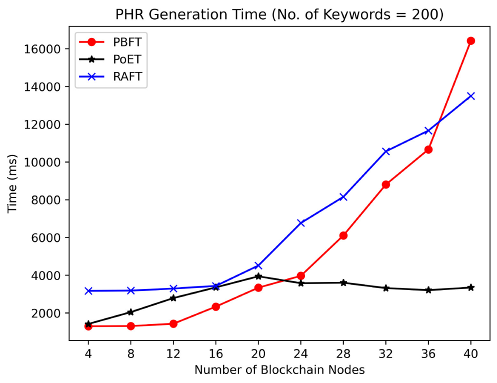


FIGURE 11. Experiment 1: results.

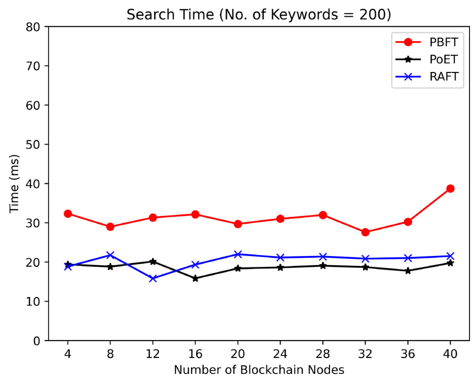


FIGURE 12. Experiment 2: results.

and $\beta = 0.69$),[6] and on the fact that a standard A4 page on an average contains 350 – 450 words. Moreover, due to limitations of computational hardware available to us, the maximum number of blockchain nodes that we were able to use for the experiments was 40. Beyond that number, other factors such as context switching would influence the actual results to a large extent.

### 1) EXPERIMENT 1

For this, we used a PHR with 200 unique keywords and recorded the overall time taken to add it to the system by varying the number of nodes in the blockchain network. We recorded the observations for three different state-of-the-art consensus algorithms: 1) PBFT, 2) PoET, and 3) Raft, and presented the results in Fig. 11. For the number of nodes less than 20, PBFT performs better than the other two algorithms. On further increasing the number of nodes, the performance of PBFT and Raft declines exponentially. However, for a larger number of nodes, PoET works quite well.

### 2) EXPERIMENT 2

We added a PHR containing 200 unique keywords to the system for this. We then searched for a keyword and recorded the time taken by the search algorithm against the number of blockchain nodes. Fig. 12 shows the results using PBFT, PoET, and Raft. It can be seen that the increase in the number of nodes does not affect the search time. This is because our search algorithm directly queries the state of the blockchain using REST API and thus does not has to be processed as a transaction by the blockchain nodes.

### 3) EXPERIMENT 3

In this experiment, we fixed the number of blockchain nodes to 8 and then recorded the time taken to complete PHR addition against the number of keywords in the PHR. Fig. 13 shows the results using PBFT, PoET, and Raft. The results show that the addition time increases linearly with the number of keywords. However, PBFT performs better than the other two algorithms.

6. Note: $\beta$ here is not related to the scheme, but is a part of the formula in Heaps' law.

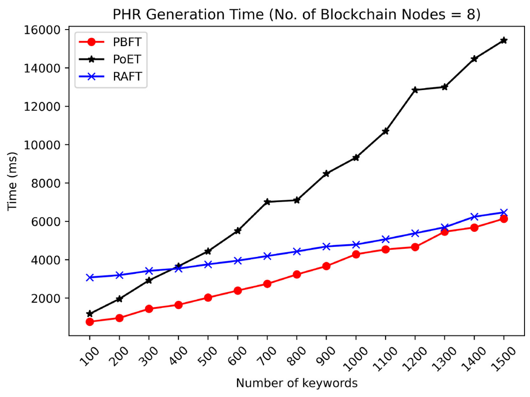


FIGURE 13. Experiment 3: results.

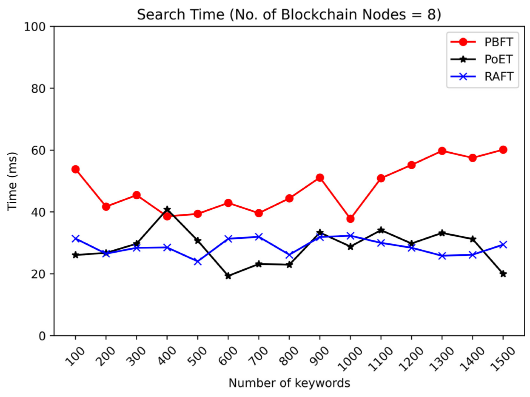


FIGURE 14. Experiment 4: results.

### 4) EXPERIMENT 4

We fixed the number of blockchain nodes to 8, added a PHR, and recorded the time to search a keyword against the number of unique keywords in the PHR. For this experiment, we have taken a list of size one by adding only one PHR to obtain consistent results. The reason is that the asymptotic complexity of the search is linear in terms of entries in the linked list corresponding to the keyword being searched. So, having a linked list of different sizes will affect the search results, eventually leading to inconsistent figures. Fig. 14 shows the results obtained using PBFT, PoET, and Raft, and it can be concluded that search time is not affected by the

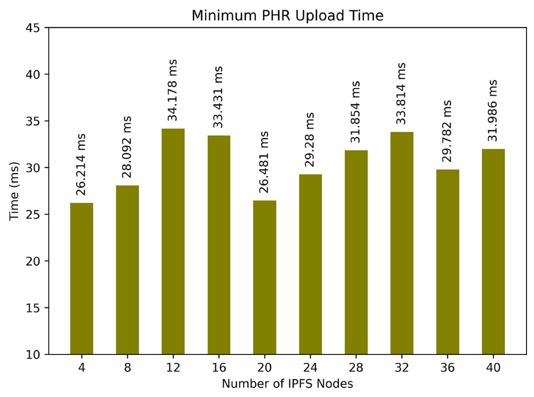


FIGURE 15. Experiment 5: results.

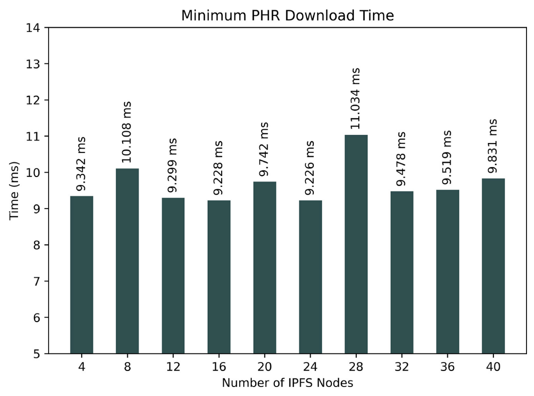


FIGURE 16. Experiment 6: results.

number of keywords present in the PHR. This is because the MR-tree used by sawtooth has a constant lookup time.

### 5) EXPERIMENT 5

In this experiment, we measured the effect of change in the number of IPFS nodes on PHR upload time. Fig. 15 shows that it remains unaffected by the change.

### 6) EXPERIMENT 6

Here, we measured the effect of the change in the number of IPFS nodes on PHR download time. Fig. 16 shows that it remains unaffected by the change.

### D. COMPARATIVE ANALYSIS

Although there is a plethora of work on SSE using blockchain, there are only few works on its application in PHR sharing. However, as discussed in Section III-D, many PHR-sharing schemes are based on public key-based searchable encryption. A strict comparison is not possible due to the different cryptographic parameters used by these schemes. Nevertheless, in Table 5, we have tried to compare these schemes with our scheme based on the expected features from a PHR sharing scheme. An exception to the above is a scheme proposed by Tang et al. [41], which is based on symmetric searchable encryption and uses a technique similar to our scheme for adding PHRs to the system. However,

TABLE 5. Comparison of security and functional features.

| Scheme | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ |
|---|---|---|---|---|---|---|---|---|
| [34] | Asymmetric | ✓ | ✓ | × | ✓ | ✓ | ✓ | × |
| [38] | Asymmetric | ✓ | ✓ | × | ✓ | ✓ | ✓ | × |
| [26] | Asymmetric | ✓ | × | × | × | ✓ | ✓ | × |
| [41] | Symmetric | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| Ours | Symmetric | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**Note:** $A_1$: Encryption Type, $A_2$: Confidentiality, $A_3$: Dynamic, $A_4$: Forward Secure, $A_5$: Verifiable, $A_6$: Blockchain Based, $A_7$: Decentralized Storage, $A_8$: Efficient

TABLE 6. Time complexity comparison.

| Scheme | Setup | Build | Add | Search |
|---|---|---|---|---|
| [34] | $O(1)$ | – | $O(\|W\| + \|A\|)$ | $O(\|W\| + \|A\|)$ |
| [38] | $O(1)$ | – | $O(\|W\|)$ | $O(\|A\|$ |
| [26] | $O(1)$ | $O(\|D\| \cdot \|W\|)$ | – | $O(\|D\|)$ |
| [46] | $O(1)$ | – | $O(\|W\|$ | $O(\|\Gamma\|)$ |
| Ours | $O(1)$ | – | $O(\|V\|)$ | $O(\|\Gamma\|)$ |

it relies on the local storage of hospitals, unlike our scheme, which uses the decentralized storage, and thus, it has limited practical applications. Our scheme, though primarily based on symmetric searchable encryption, uses elliptic curve cryptography for communication between entities and, therefore, can be categorized as a hybrid scheme.

In Table 6, we show a comparative analysis on the time complexities of various operations involved in the proposed scheme and other existing schemes. For the setup phase, the time complexity for each scheme is constant. However, the asymmetric schemes will be comparatively slower because of the use of bilinear pairings.

All other schemes than the scheme proposed in [26], which generates index only once at the beginning, can dynamically add PHRs as and when required. $W$ represents the set of unique keywords in a PHR and $D$ is the set of PHRs used during the build phase. $A$ presents in complexities of some of the schemes is related to attribute based encryption, where it denotes the set of attributes. In case of our scheme, $V$ denotes the super-set of keyword set $W$ and $\Gamma$ is the set of file identifiers corresponding to a single keyword in the index. For most of the patients, the PHRs will not be very large in number, and therefore, comparatively our scheme is quite efficient during search.

## VIII. FUTURE WORKS

In this section, we list the following future research directions that we would like to work in our future study.

### A. SUPPORT FOR PARALLELIZATION

The addition of PHR in our scheme supports parallelization. As seen in “Uploading to blockchain” part of Section V-C3, the addition of (*addr*, *val*) pairs to the state of the blockchain are independent of each other. Therefore, it can be modified to leverage parallelization, significantly decreasing PHR addition time. However, the algorithm must run as a part of a transaction processor. Therefore, all the blockchain peers must have parallelization capability; otherwise, a single peer without parallelization would create a bottleneck for the entire system.

### B. CONJUNCTIVE MULTI-KEYWORD SEARCH SUPPORT

Our scheme can be modified to support multi-keyword search, but at the expense of more leakage. The modified scheme would require the PHR ids to be encrypted using a deterministic algorithm. The search algorithm executed by the hospital server can take a set of keywords as input and obtain the linked lists corresponding to the keywords. Subsequently, it can find the common encrypted PHR ids and return them to the user. Moreover, the process to obtain the linked lists can be made parallel because each can be obtained independently of the other.

## IX. CONCLUSION

With the growing dependency of healthcare on IoMT infrastructure, we require a secure system that not only provides confidentiality and privacy to the patients' health data, but also allows its secure sharing with other parties in the healthcare ecosystem. We attempted to provide a solution by proposing a novel scheme for PHR sharing that is dynamic, efficient and practically implementable. Our design considered various entities into consideration, including IoMT smart devices, and provided a carefully designed protocol for their interactions. We proved the proposed scheme to be forward secure, verifiable and semantically secure against an adaptive adversary. Moreover, we based our scheme on top of a decentralized file system (IPFS) in order to make our scheme fault tolerant and robust against attacks targeting centralized systems. Additionally, we provided various insights into how this scheme can be extended to support parallelization and conjunctive multi-keyword search.

## REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, cryptography mailing list." Mar. 2009. [Online]. Available: https://metzdowd.com

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp.Security Privacy (S&P)*, Berkeley, CA, USA, 2000, pp. 44–55.

[3] J. Benet. "IPFS-content addressed, versioned, P2P file system." 2014. [Online]. Available: https://arxiv.org/abs/1407.3561.

[4] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton, FL, USA: CRC Press, 2020.

[5] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Security Privacy (S&P)*. San Jose, CA, USA, 2016, pp. 839–858.

[6] V. Shoup. "A proposal for an ISO standard for public key encryption." 2001. [Online]. Available: https://eprint.iacr.org/2001/112

[7] T. Krovetz and P. Rogaway, "The OCB authenticated-encryption algorithm," IETF, Fremont, CA, USA, RFC 7253, May 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7253

[8] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.

[9] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf. (Usenix ATC)*, Philadelphia, PA, USA, 2014, pp. 305–319.

[10] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996. [Online]. Available: https://doi.org/10.1145/233551.233553

[11] E.-J. Goh. "Secure indexes." 2003. [Online]. Available: https://eprint.iacr.org/2003/216

[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Security (CCS)*, Alexandria, VA, USA, 2006, pp. 79–88.

[13] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, 2012, pp. 965–976.

[14] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, 2012, pp. 1–15.

[15] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. Annu. Cryptol. Conf. (CRYPTO)*, Santa Barbara, CA, USA, 2013, pp. 353–373.

[16] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," 2013. [Online]. Available: https://eprint.iacr.org/2013/832

[17] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. 2nd ACM Conf. Comput. Commun. Security (CCS)*, 2015, pp. 668–679.

[18] R. Bost, P.-A. Fouque, and D. Pointcheval, "Verifiable dynamic symmetric searchable encryption: Optimality and forward security." 2016. [Online]. Available: https://eprint.iacr.org/2016/062

[19] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Dallas, TX, USA, 2017, pp. 1465–1482.

[20] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Dallas, TX, USA, 2017, pp. 1449–1463.

[21] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans, "Efficient dynamic searchable encryption with forward privacy," in *Proc. Privacy Enhanc. Technol.*, vol. 2018, 2017, pp. 20–25.

[22] Y. Watanabe, K. Ohara, M. Iwamoto, and K. Ohta, "Efficient dynamic searchable encryption with forward privacy under the decent leakage," in *Proc. 11th ACM Conf. Data Appl. Security Privacy (CODASPY)*, Baltimore, MD, USA, 2022, pp. 312–323.

[23] L. Chen, J. Li, and J. Li, "Towards forward and backward private dynamic searchable symmetric encryption supporting data deduplication and conjunctive queries," *IEEE Internet Things J.*, vol. 10, no. 9, pp. 17408–17423, Oct. 2023.

[24] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Honolulu, HI, USA, 2018, pp. 792–800.

[25] C. Cai, J. Weng, X. Yuan, and C. Wang, "Enabling reliable keyword search in encrypted decentralized storage with fairness," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 1, pp. 131–144, Jan.-Feb. 2021.

[26] Y. Zhang, R. H. Deng, J. Shu, K. Yang, and D. Zheng, "TKSE: Trustworthy keyword search over encrypted data with two-side verifiability via blockchain," *IEEE Access*, vol. 6, pp. 31077–31087, 2018.

[27] H. Li, H. Tian, F. Zhang, and J. He, "Blockchain-based searchable symmetric encryption scheme," *Comput. Elect. Eng.*, vol. 73, pp. 32–45, Jan. 2019.

[28] Y. Guo, C. Zhang, and X. Jia, "Verifiable and forward-secure encrypted search using blockchain techniques," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dublin, Ireland, 2020, pp. 1–7.

[29] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, Vienna, Austria, 2016, pp. 25–30.

[30] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, "Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control," *J. Med. Syst.*, vol. 40, no. 10, p. 218, 2016.

[31] Q. Xia, E. B. Sifah, A. Smahi, S. Amofa, and X. Zhang, "BBDS: Blockchain-based data sharing for electronic medical records in cloud environments," *Information*, vol. 8, no. 2, p. 44, 2017.

[32] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, "MedBlock: Efficient and secure medical data sharing via blockchain," *J. Med. Syst.*, vol. 42, no. 8, pp. 1–11, 2018.

[33] A. Zhang and X. Lin, "Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain," *J. Med. Syst.*, vol. 42, no. 8, pp. 1–18, 2018.

[34] Y. Wang, A. Zhang, P. Zhang, Y. Qu, and S. Yu, "Security-aware and privacy-preserving personal health record sharing using consortium blockchain," *IEEE Internet Things J.*, vol. 9, no. 14, pp. 12014–12028, Jul. 2022.

[35] Y. Wang, A. Zhang, P. Zhang, and H. Wang, "Cloud-assisted EHR sharing with security and privacy preservation via consortium blockchain," *IEEE Access*, vol. 7, pp. 136704–136719, 2019.

[36] M. Alsayegh, T. Moulahi, A. Alabdulatif, and P. Lorenz, "Towards secure searchable electronic health records using consortium blockchain," *Netw.*, vol. 2, no. 2, pp. 239–256, 2022.

[37] H. Qian, J. Li, Y. Zhang, and J. Han, "Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation," *Int. J. Inf. Security*, vol. 14, no. 6, pp. 487–497, Nov. 2015. [Online]. Available: https://doi.org/10.1007/s10207--014-0270--9

[38] J. Sun, L. Ren, S. Wang, and X. Yao, "A blockchain-based framework for electronic medical records sharing with fine-grained access control," *PLoS ONE*, vol. 15, no. 10, 2020, Art. no. e0239946.

[39] S. Niu, L. Chen, J. Wang, and F. Yu, "Electronic health record sharing scheme with searchable attribute-based encryption on blockchain," *IEEE Access*, vol. 8, pp. 7195–7204, 2019.

[40] S. Wang, D. Zhang, and Y. Zhang, "Blockchain-based personal health records sharing scheme with data integrity verifiable," *IEEE Access*, vol. 7, pp. 102887–102901, 2019.

[41] X. Tang, C. Guo, K.-K. R. Choo, Y. Liu, and L. Li, "A secure and trustworthy medical record sharing scheme based on searchable encryption and blockchain," *Comput. Netw.*, vol. 200, Dec. 2021, Art. no. 108540.

[42] X. Nie, A. Zhang, J. Chen, Y. Qu, and S. Yu, "Time-enabled and verifiable secure search for blockchain-empowered electronic health record sharing in IoT," *Security Commun. Netw.*, vol. 2022, Dec. 2022, Art. no. 1103863. [Online]. Available: https://doi.org/10.1155/2022/1103863

[43] Y. Lindell, *How to Simulate It–A Tutorial on the Simulation Proof Technique*. Cham, Switzerland: Springer, 2017, pp. 277–346. [Online]. Available: https://doi.org/10.1007/978--3-319--57048-8_6

[44] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. 25th USENIX Conf. Security Symp.*, 2016, pp. 707–720.

[45] H. S. Heaps, *Information Retrieval, Computational and Theoretical Aspects*. Cambridge, MA, USA: Academic Press, 1978.

**ABHISHEK BISHT** received the Bachelor of Technology degree in computer science from the University of Petroleum and Energy Studies, Dehradun, India, in 2021. He is currently pursuing the M.S. degree in computer science and engineering with the Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad, India. His research interests are cyber security, searchable encryption, IoT security, and blockchain technology.

**ASHOK KUMAR DAS** (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering, the M.Tech. degree in computer science, and the M.Sc. degree in mathematics from Indian Institute of Technology Kharagpur (IIT Kharagpur), India. He is currently a Full Professor with the Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad, India. He was also a Visiting Faculty with the Virginia Modeling, Analysis and Simulation Center, Old Dominion University, Suffolk, VA, USA. He has authored over 370 papers in international journals and conferences in the above areas, including over 310 reputed journal papers. His Google Scholar H-index is 80 and i10-index is 237 with over 18 300 citations. His research interests include cryptography, system and network security, blockchain, security in Internet of Things, Internet of Vehicles, Internet of Drones, cloud/fog computing, intrusion detection, AI/ML security, and post-quantum cryptography. He was a recipient of the Institute Silver Medal from IIT Kharagpur. He has been listed in the Web of Science (Clarivate$^{\text{TM}}$) Highly Cited Researcher 2022 in recognition of his exceptional research performance. He is/was on the editorial board of IEEE SYSTEMS JOURNAL, *Journal of Network and Computer Applications* (Elsevier), *Computer Communications* (Elsevier), *Journal of Cloud Computing* (Springer), *Cyber Security and Applications* (Elsevier), *IET Communications*, *KSII Transactions on Internet and Information Systems*, and *International Journal of Internet Technology and Secured Transactions* (Inderscience). He also severed as one of the Technical Program Committee Chairs of the first International Congress on Blockchain and Applications, Avila, Spain, June 2019, International Conference on Applied Soft Computing and Communication Networks, October 2020, Chennai, India, and second International Congress on Blockchain and Applications, L'Aquila, Italy, October 2020.

**DUSIT NIYATO** (Fellow, IEEE) received the B.Eng. degree from the King Mongkuts Institute of Technology Ladkrabang, Thailand, in 1999, and the Ph.D. degree in electrical and computer engineering from the University of Manitoba, Canada, in 2008. He is a Professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests are in the areas of sustainability, edge intelligence, decentralized machine learning, and incentive mechanism design.

**YOUNGHO PARK** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic engineering from Kyungpook National University, Daegu, South Korea, in 1989, 1991, and 1995, respectively. He is currently a Professor with the School of Electronics Engineering, Kyungpook National University. From 1996 to 2008, he was a Professor with the School of Electronics and Electrical Engineering, Sangju National University, South Korea. From 2003 to 2004, he was a Visiting Scholar with the School of Electrical Engineering and Computer Science, Oregon State University, USA. His research interests include computer networks, multimedia, and information security.