Blockchain-Enabled Secure and Smart HealthCare System

Debendranath Das, Indian Statistical Institute, debendra_r@isical.ac.in, India
Amudhan Muthaiah, Indian Institute of Technology, amudhanmuthaiah@yahoo.in, India
Sushmita Ruj, CSIRO's Data61 & UNSW Sydney, sushmita.ruj@csiro.au, Australia

Abstract—

Technology has evolved over the years, making our lives easier. It has impacted the healthcare sector, increasing the average life expectancy of human beings. Still, there are gaps that remain unaddressed. There is a lack of transparency in the healthcare system, which results in inherent trust problems between patients and hospitals. In the present day, a patient does not know whether he or she will get the proper treatment from the hospital for the fee charged. A patient can claim reimbursement of the medical bill from any insurance company. However, today there is minimal scope for the Insurance Company to verify the validity of such bills or medical records. A patient can provide fake details to get financial benefits from the insurance company. Again, there are trust issues between the patient (i.e., the insurance claimer) and the insurance company. Blockchain integrated with the smart contract is a well-known disruptive technology that builds trust by providing transparency to the system. In this paper, we propose a blockchain-enabled *Secure and Smart HealthCare System*. Fairness of all the entities: patient, hospital, or insurance company involved in the system is guaranteed with no one trusting each other. Privacy and security of patient's medical data are ensured as well. We also propose a method for privacy-preserving sharing of aggregated data with the research community for their own purpose. Shared data must not be personally identifiable, i.e, no one can link the acquired data to the identity of any patient or their medical history. We have implemented the prototype in the Ethereum platform and Ropsten test network, and have included the analysis as well.

Index Terms—Blockchain, Healthcare, Electronic Health Record, Data Security, Privacy, Fairness.

✦

1 INTRODUCTION

The use of IoT devices in the healthcare sector generates a massive amount of patient's medical data [1], [2]. The EHRs (Electronic Health Records) are typically stored in some databases. However, these data are sensitive and cannot be made public. Tampering of medical data has the potential to jeopardize a person's life. Access to medical records must not compromise the privacy of a person. A third party might use such sensitive data to inflict harm or sell it to other parties. Blockchain is a distributed immutable verifiable append-only ledger, which is maintained by a peer-to-peer network [3], [4]. A block in a blockchain contains a set of transactions or data. Every block in the chain is connected to the previous one by a secure hash function, ensuring immutability. Immutability guarantees that once the transaction or data is written in some block in the blockchain, it cannot be tampered. Another essential property of Blockchain is its decentralized and distributed nature. There is no central authority that can solely handle the data in the blockchain. In a permissionless Blockchain model, anyone can download and store the ledger locally and then take part in the consensus protocol to sync the state of their blockchain with others. The following salient features of blockchain make it an attractive technology for addressing the security and privacy issues in IoT based applications[5]:

– Decentralization: The lack of central authority ensures robustness and overcomes the problem of a single point of failure. Moreover, any participant can join the system and

function without being controlled by any single party.

– Anonymity: The inherent anonymity is well-suited for most IoT use cases, where the identity of the users must be kept private.

– Security: Blockchain realizes a secure network over untrusted parties that are desirable in IoT with many heterogeneous devices. An immutable ledger also helps to achieve non-repudiation.

– Data immutability: Blockchain technology is expected to improve data record management and brings fairness and auditability in the system.

Besides this, smart contracts come into the picture with the advent of Ethereum[6]. Contracts are meant for agreement between two or more entities. Smart contracts introduce a set of advantages, such as cost reduction, speed, precision, efficiency, and transparency. In many applications, a smart contract helps to eliminate the role of a trusted third party and brings fairness to the system.

We have designed a patient-centric healthcare system where the medical data can be accessed by a party provided the patient has given consent to do so. An access control matrix stored in Blockchain disallows any sort of malicious intervention. The digital footprint of the medical data is stored in the blockchain to provide integrity, immutability, and digital signature also helps to provide accountability as well. The hospital authority cannot extort any arbitrary amount from a patient by providing wrong treatment or denying treatment. In that case, the hospital authority will be penalized. On the other hand, if a patient claims that

he/she has received an invalid medical report or denies receiving treatment, the logic encoded in the smart contract prevents such behavior by penalizing the patient.

Transcript of any interaction between patient and insurance company or database owner prevents any party from wrongly ascribing blame to an honest party. Such audit trails help in resolving any dispute by seeking legal help or mutual settlement.

1.1 Related Work

We discuss the state-of-the-art blockchain solution for the healthcare system. Xia et al. [7] had proposed a cloud-based blockchain platform for sharing files with untrustworthy parties seeking access to medical files. Accessing continuous stream of patient data was enabled using blockchain architecture in [8]. To reduce the overhead of data stored in Blockchain, Zhang et al. [9] proposed a blockchain-based solution where data is stored offline and the verification part of the associated data is put on the blockchain.

Griggs et al. [10] proposed a solution where the data from wearables act as the oracle to the smart contracts in the blockchain. The data preservation system in the work proposed by Li et al. [11] basically contains two programs - the data access program and the blockchain interaction program.

Dias et al. [12] proposed a blockchain-based access control method for EHR. The policy is a relation between an EHR and the associated data keepers. Liang et al. [13] proposed a three-layer architecture for securely storing and retrieving data. The continuous data from the wearables are stored in the cloud and their corresponding hashes are kept in the blockchain.

Yup et al. [14] designed the *healthcare data gateway*, a blockchain approach for healthcare intelligence to address the privacy of users by proposing a data access control for privacy. Zhang et al. [15] proposed PSN-based healthcare by designing two protocols for authentication and sharing of healthcare data. Liang et al. [16] proposed a mobile-based healthcare record sharing system using blockchain. They designed a secure user-centric approach to provide access control and privacy using a channel formation scheme. Jiang et al. [17] designed a medical data exchange system using blockchain by developing off-chain and on-chain verification for the security of the system's storage.

Fan et al. [18] proposed an improved consensus mechanism to obtain enhanced security and privacy of medical data. Wang and Song [19] proposed an attribute-based encryption mechanism to design a secure medical record sharing system. [20] presented an electronic health record management system using an attribute-based signature scheme for multiple users in the blockchain network. Sun et al. [21] designed a distributed attribute-based signature scheme for medical systems based on blockchain and proposed a blockchain-based record sharing protocol.

Zhang and Poslad [22] proposed an access control policy for electronic medical records with finer granular access. Yang and Li [23] proposed an architecture for securing EHR based on distributed ledger technology. Performance of Hyperledger Fabric, a permissioned blockchain network, was analyzed in [24]. Gorenflo et al. [25] proposed a performance optimization for Hyperledger blockchain framework. A detailed analysis of these papers have been provided in [26].

TABLE 1: Drawbacks of State-of-the-Art Healthcare Systems

Related Works	Drawbacks
[14], [16], [22], [23]	Lack implementation and do not evaluate System's Performance
[7], [15], [16], [22], [23]	Lack Proper Framework
[11], [15]	Do not provide Data Access Control Policy
[14], [16], [22], [23], [24]	No Algorithm is given for the Protocol
[14], [15], [17], [21], [22], [23], [24], [25]	Lack EHR Sharing Protocol
[7], [24]	Scalability Problem
[16], [20]	Interoperability Problem
[7]	Key Management Problem
[8], [17], [21]	Performance and Fairness problem
[18], [21], [25]	Require High Storage, Power and/or Computation Cost

Although state-of-the-art tries to enhance the security of the healthcare system using Blockchain Framework, these have certain drawbacks. We have summarized it in Table 1. In our model, we have addressed most of the existing problems by proposing a decentralized, distributed healthcare system using a permissionless blockchain framework. We have implemented our protocol and evaluated the system's performance. Our model also ensures the privacy of patients' medical data (using Access Control Policy), data security, and fairness of various entities involved in the system.

1.2 Contribution of the Paper

In this paper, we define and discuss in detail a new blockchain-based architecture for a patient-centric Healthcare system. Our system is robust, but at the same time privacy-preserving. We briefly discuss the novelty of the design:

- We propose a *Secure and Smart HealthCare System* which coordinates the interaction between patient and hospital while a patient is getting treated; between patient, the medical database owner, and insurance company when the patient needs to claim any reimbursement and between medical database owner and the research community while collecting the data.
- The proposed system ensures that no one has access to the patient's data stored in the medical database until and unless the patient has granted permission to do so.
- A patient cannot be overcharged for seeking treatment from a hospital. Simultaneously, a hospital has to start the treatment within a specified period of time. If they fail to do so, the patient can withdraw any deposit made.
- A patient can either retain the data with himself or store it in a medical database, which is assumed to be trusted/semi-trusted. The database owner will impose a check on the validity of the data provided by the patient before storing it in the database that prevents storing any spurious data.
- Fairness is guaranteed for both the patient and the insurance company when the former places a request for a claim based on an insurance policy purchased. An insurance company can verify whether the claim is valid.
- Patient's data remains private and gets shared among those who have been granted legitimate access. Data cannot be tampered by any external agent, any malicious

behavior can be detected using the digital fingerprint of the data recorded in Blockchain.

– Medical data provided to the research community by the system for study or analysis will not compromise the patient's privacy.

– We have implemented the prototype in the Ethereum platform and Ropsten test network, and have evaluated the performance. Code for the protocol is available on GitHub*.

1.3 Organization

The rest of the paper is structured as follows -
Section 2 briefly discusses the basic building blocks. In Section 3, the system model and high-level view of our construction are presented. We have discussed our major procedures, implementation, and technical details in Section 4. In Section 5, we have addressed our security claims. Section 6 presents the results of our proposed system. Finally, we have concluded the paper in Section 7.

2 PRELIMINARIES

In this section, we discuss the building blocks and other proposed methodologies that we have used in our system. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$.

2.1 Basic Cryptographic Primitives

– **Encryption Scheme:** Encryption Scheme serves the purpose of privacy or confidentiality of the messages exchanged between a sender and recipient. We have used two types of encryption - *Private Key Encryption* or *Symmetric Key Encryption (SKE)* and *Public Key Encryption* or *Asymmetric Key Encryption (ASKE)*.

– **Digital Signature Scheme:** Digital Signature (DS) serves the purpose of user authentication in the system. Obtaining the DS of an entity, any recipient can verify if the message originated from the intended sender or not.

– **Hash Function:** A Hash function ensures data integrity in the system. It can be defined as $H : \{0,1\}^* \rightarrow \{0,1\}^k$ that maps messages of arbitrary length to a fixed size message digest of length k. An ideal hash function must satisfy the following three properties: *One Way or Pre-image Resistant*, *Second Pre-image Resistant* and *Collision Resistant*.

– **Merkle Tree:** Merkle Trees are binary trees, which are used to prove the membership of data belonging to a set. The leaf node comprises the data present in the set, and the output of a leaf node is the hash of the data. For every non-leaf node, the output is the hash of the concatenated children node's outputs. The output of the root node is referred to as the *Merkle Root*. The acronym MR is used to denote the Merkle Tree Root, as shown in Figure 1.
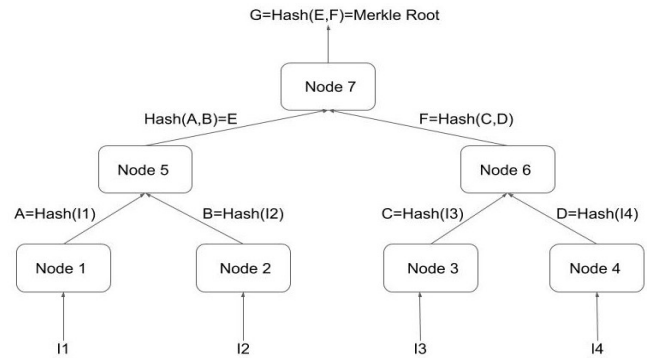
---

* https://github.com/Debendranath/Blockchain-Enabled-Secure-and-Smart-Healthcare-System

Fig. 1: *Merkle Tree*

2.2 Other Building Blocks

– **Blockchain:** A Blockchain (BC) is an immutable, decentralized, distributed verifiable ledger. Once the information is put into the BC, it can not be tampered with by any means later. As the name suggests, it consists of a chain of blocks, where each block in turn consists of a series of transactions. The blocks are typically *hash-linked* to one another, i.e, if a transaction or data inside any block is altered, the hash of that particular block gets changed. A mismatch in the hash value gets reflected in the succeeding blocks. Hence, to prevent such detection, the hash needs to be altered in subsequent blocks as well. In practice, it is not possible to do (or can be done with negligible probability), and that is the foundation of the immutability in Blockchain. The most commonly used consensus mechanism is Proof-of-Work (PoW) in Bitcoin as well as Ethereum. Other alternate consensus algorithms like Proof-of-Stake, Proof-of-Burn, Proof-of-Authority are still being explored. In our work, we use a Proof-of-Work (PoW) based consensus mechanism.

– **Smart Contract:** Smart Contracts (SC) are self-executing contracts, where the terms of an agreement between parties are encoded using a programming language(e.g, solidity). The smart contract also helps to build trust in the system by eliminating the role of intermediaries or middle man. Blockchain integrated with smart contracts can be used for developing decentralized applications which can address certain real-world problems in a secure and trustworthy manner.

3 HIGH LEVEL VIEW OF THE PROPOSED SYSTEM

3.1 System Model

The major actors or parties involved in our proposed system, as shown in Figure 2, are as follows -

1. **Patient (P)/User**
2. **Hospital Authority (HA)**
3. **Insurance Company (IC)**
4. **Database Owner (DBO)**
5. **Research Community (RC)**

We need to ensure that every party communicates with other parties in a secure and trustworthy manner. There are some fixed protocols or rules that are rigorously defined in the system, and every communicating party should follow
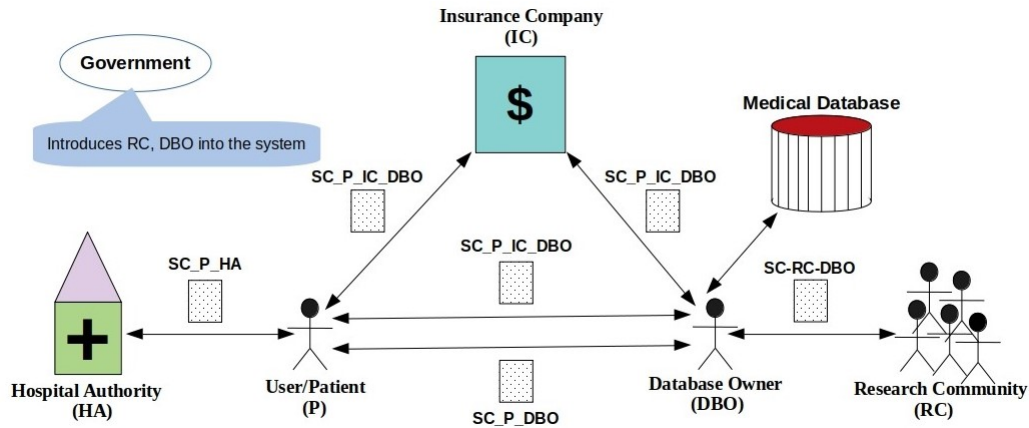
Fig. 2: *Healthcare System Model*

this protocol. The protocol suite is written in the form of functions inside the smart contracts

In our system, we have the following smart contracts.

– Smart Contract between Patient (P) and Hospital Authority (HA): **SC_P_HA**
– Smart Contract between Patient (P), Insurance Company (IC) and Database Owner (DBO): **SC_P_IC_DBO**
– Smart Contract between Patient (P) and Database Owner (DBO): **SC_P_DBO**
– Smart Contract between Research Community (RC) and Database Owner (DBO): **SC_RC_DBO**

We also have one additional smart contract, named **SC_Registration**, where the entities of the systems can register themselves before participating in the protocol. Therefore, we have five smart contracts in total to build up the entire system. In the next few subsections, we will discuss how these different actors interact among themselves in the system. Before that, let us state some crucial facts or assumptions of our system.

3.2 Objective

We intend to propose a *patient-centric hospital management system* which realizes the following objectives:

– *Fairness*: An honest party will never lose money even if the rest of the parties are malicious and try to cheat and claim money without providing the desired service or data.
– *Privacy*: Any party cannot view a patient's data until and unless it gets the consent from the patient.
– *Data Security*: Data of a patient stands protected and cannot be tampered.

3.3 Assumptions

a) In our system, every single party has a unique ID. A smart contract generates these IDs at the time of registration in the system. One unique ID corresponds to a particular *PublicKey* To be a part of the system, it is mandatory to register first. There is no option to bypass the *Registration Phase* and get the benefits from the system.
b) In case of the patient claims for medical insurance to reimburse the medical expenditure, he/she must

keep the files(related to bills) in the medical repository. Access to these data is strictly restricted. The access control regulation is in the hands of the patient (i.e., data owner). If the patient grants permission to the third party (say, hospital/ Insurance Company, etc.) for accessing his/her records, then only they can do so. The patient can also revoke permission from these entities. The access control matrix is stored in the Blockchain.

c) In the proposed system, every patient maintains their medical files digitally, i.e., Electronic Health Records (EHR), and these records are stored in a medical database, owned by the Government (or Trusted Third Party Vendor). Thus, the Medical Data Repository Owner or simply the Database Owner (DBO) is considered semi-trusted in our system. A patient can also keep their records to their private storage locally or can appeal to the Database vendor for removing his/her records in case of privacy concerns, which essentially supports the European General Data Protection Regulation (GDPR). As the data is stored in plain text form, DBO can have open access to the stored data. However, this assumption can be removed by adding the scope of handling encrypted data in the database. The model can be changed accordingly in that case. The DBO's activity log is maintained in the blockchain. In case DBO misbehaves, then it can be questioned and penalized accordingly.

d) In our system, the blockchain is mainly used to store the following - hash of the records (EHRs) /payment bills /money receipts, timestamps of asset creation, signatures of issuing authority (as well as the signature of the data owner sometimes) and some other auxiliary pieces of information (context-specific) for supporting accountability, integrity, authentication, and fairness.

e) Database Owner (DBO) or Research Community (RC) must satisfy certain prerequisite conditions to be a part of the system and appeal to the Government expressing their interest. The conditions or criteria may vary for different Governments of various countries. If all the necessary criteria get satisfied by them, the Government introduces DBO or RC into the system.

3.4 Communication Protocol between Patient and Hospital

When a patient comes to a hospital with medical issues, the hospital analyzes the patient's problems. After preliminary scanning, the hospital generates an approximate cost of the treatment (i.e., estimated cost). Hospital authority and patient both need to lock this amount to the smart contract (SC_P_HA), ensuring fairness in the protocol.

In case the hospital needs to check the patient's medical history from the medical repository; the hospital authority asks the patient to grant proper access permission. The hospital can read the patient's medical history if permission is granted. The patient can revoke access permission from the hospital at any instant of time if needed. Information related to this access control (who has access right to whose data) is stored in the blockchain.

Once the estimated cost is locked by the patient, the next obvious task is to start the treatment of the patient within some fixed time window. If the hospital fails to start treatment, then the patient can unlock his or her money after the time window. The hospital invokes a specific function to register the timestamp of starting the treatment.

The patient's testimonials, reports, prescriptions, etc. (i.e., medical files) are generated by the hospital. But these documents are not yet transferred to the patient immediately due to security reasons, which is addressed in *Section 5*. With the help of some cryptographic computations and fair exchange protocol, as shown in Figure 3, the hospital sends the medical files to the patient. The hospital stores the following crucial attributes as the metadata in the blockchain.

1) Merkle root of the file
2) Merkle root of the encrypted file
3) Signature of the hospital on Merkle root of the file.
4) Signature of the hospital on H(Patient ID || Date of Report || Merkle root of Encrypted File).

Two Merkle roots are constructed: one with the file chunks as the input, termed as M_1 and the other having encrypted chunks of the same file as input, denoted as M_2. These Merkle roots are used for verification done by the protocol on behalf of the patient and for verification to be done by other associated entities like Insurance Company and Database Owner. The signature on the Merkle root of the file makes the hospital accountable for the file it has encrypted. The last one in the list, signature on the hash of the patient's attributes and the hash of the encrypted file, adds much more accountability, giving the patient the chance to raise a complaint if the hospital has misbehaved.

Upon receiving these file attributes offline, the patient verifies and gives consent. If the patient finds there is a mismatch in the file attributes, he/she can withdraw the locked amount. If the attributes match, the patient invokes a function to give consent and sign on the file.

In between, the hospital is required to provide the final medical bill to the patient. We assume that the cost of the final bill is not greater than that of the estimated cost bill. The patient has two options - either give consent or raise a dispute for an overcharge of the treatment. With the latter option, the hospital and patient come to a mutual conclusion on the price through offline communication. And this pathway involves two additional transactions to be done by the parties before agreeing to the revised final bill.

Receiving the patient's consent on the final bill, the hospital sends the key to the patient for decrypting the encrypted file that was sent earlier. If the Merkle root of the decrypted file does not match with the one in the contract, the patient raises a complaint by providing a *Proof of Misbehavior*. Once the complaint is verified and the counterparty is found to be malicious, he or she is penalized accordingly. Vice versa, if a wrong complaint has been raised, the party gets penalized.

If the patient neither raises a complaint nor gives consent for the medical report within the timeout period, the hospital withdraws the locked money (patient's and hospital's) and aborts the protocol. In all the functions, there is a timeline check to ensure that each process in the protocol is done within the allocated time window. Also, at each stage of the protocol, both parties are given functions to abort from the protocol to avoid indefinite waiting if one of the parties stops responding. The aforesaid communication model between patient and hospital is depicted in Figure 3.

3.5 Communication among Patient, Insurance Company and Database Owner

We assume that every *Medical Insurance Company* maintains a database locally, where various policy-related information is stored. For simplicity, we can assume that the insurance company maintains a relational table, having schema Policy (policyNo, termsAndcondition, price), policyNo being the primary key. This database is publicly readable.

For policy buying (or claiming) purposes, buyers, or insurance companies would undergo some transactions that are kept in blockchain for future accountability purposes, bringing fairness into the system. Let us denote policyNo, termsAndCondition, and price as P, $T\&C\_P$, and V respectively. Buying an insurance policy is a two-phase process. In phase one, the buyer specifies his/her interest to the insurance company for a particular policy (P). The buyer locks the policy price (V) in the smart contract and also puts the hash of the terms and condition file ($H(T\&C\_P)$) in the blockchain. In phase two, the insurance company also needs to mention the hash of the terms and conditions file and the policy price to the contract within a fixed time window. If the insurance company responds within this time limit by specifying $H(T\&C\_P)$, and policy price on the blockchain, then smart-contract verifies if these two hash values (i.e, the hash value mentioned by the buyer in phase one and so by the insurance company in phase two) are the same. The contract also checks for the mismatch between the policy price locked by the user and the price mentioned by the insurance company. If they match, then the policy price (V) is transferred to the insurance company's account, and the system provides a unique policy ID ($poID$) to the user for future references. All the necessary information corresponding to this $poID$ (e.g., buyer and seller info, the hash of $T\&C\_P$, timestamp of buying, etc.) are stored in the blockchain. From now on, the user/buyer is a legitimate person to get the benefits of the policy upon satisfying the terms and conditions of the policy. In case the two hashes do not match, or the insurance company does not reply
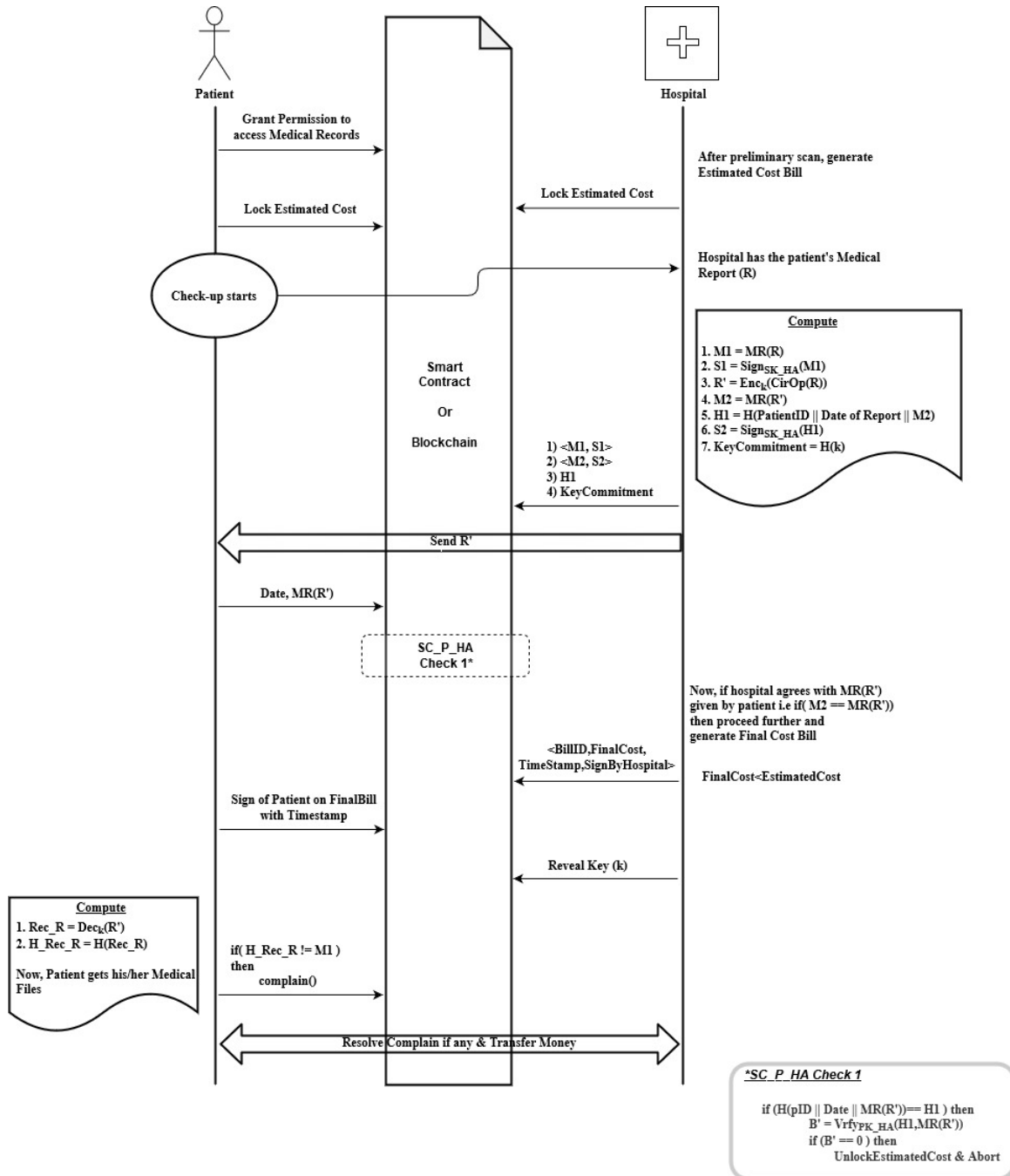
Patient   **Hospital**

Grant Permission to access Medical Records

After preliminary scan, generate Estimated Cost Bill

Lock Estimated Cost

Lock Estimated Cost

Check-up starts

Hospital has the patient's Medical Report (R)

Smart Contract Or Blockchain

Compute
1. M1 = MR(R)
2. S1 = $Sign_{SK\_HA}$(M1)
3. R' = Enc_k(CirOp(R))
4. M2 = MR(R')
5. H1 = H(PatientID || Date of Report || M2)
6. S2 = $Sign_{SK\_HA}$(H1)
7. KeyCommitment = H(k)

1) <M1, S1>
2) <M2, S2>
3) H1
4) KeyCommitment

Send R'

Date, MR(R')

SC_P_HA Check 1*

Now, if hospital agrees with MR(R') given by patient i.e if(M2 == MR(R')) then proceed further and generate Final Cost Bill

<BillID,FinalCost, TimeStamp,SignByHospital>

FinalCost<EstimatedCost

Sign of Patient on FinalBill with Timestamp

Reveal Key (k)

Compute
1. Rec_R = Dec_k(R')
2. H_Rec_R = H(Rec_R)

Now, Patient gets his/her Medical Files

if(H_Rec_R != M1) then
    complain()

Resolve Complain if any & Transfer Money

SC_P_HA Check 1
if (H(pID || Date || MR(R'))== H1) then
    B' = $Vrfy_{PK\_HA}$(H1,MR(R'))
    if (B' == 0) then
        UnlockEstimatedCost & Abort

Fig. 3: *Interaction between Patient and Hospital*

within the specific time window, then the user can withdraw his/her locked money from the system.

The next scenario is Insurance Claiming Procedure. We have proposed a solution with the aid of blockchain, by which an insurance company can verify the claims raised by the user (or patient) for reimbursement of medical bills are correct or not. When the user claims for his/her medical policy, the insurance company asks the user to grant read permission to the information related to the medical expenditure from the Medical Database (assuming the claimant has uploaded all the files related to the medical treatment). The patient's medical information is divided into three categories -

1) General Information (like name, address, age, etc.)
2) Medical Treatment Related Information (like test reports, prescriptions, etc.)
3) Medical Expenditure Related Information (like medical bills, invoices, etc.)

Dividing information into these granular levels helps to provide the information to a third party in a restricted manner. The user grants the necessary permission to the insurance company for reading his/her data from the medical database. Upon getting proper access permission from the user, the Insurance company asks the Database Owner (DBO) to provide the relevant files/records of the user. Here, Database Owner first verifies whether the insurance
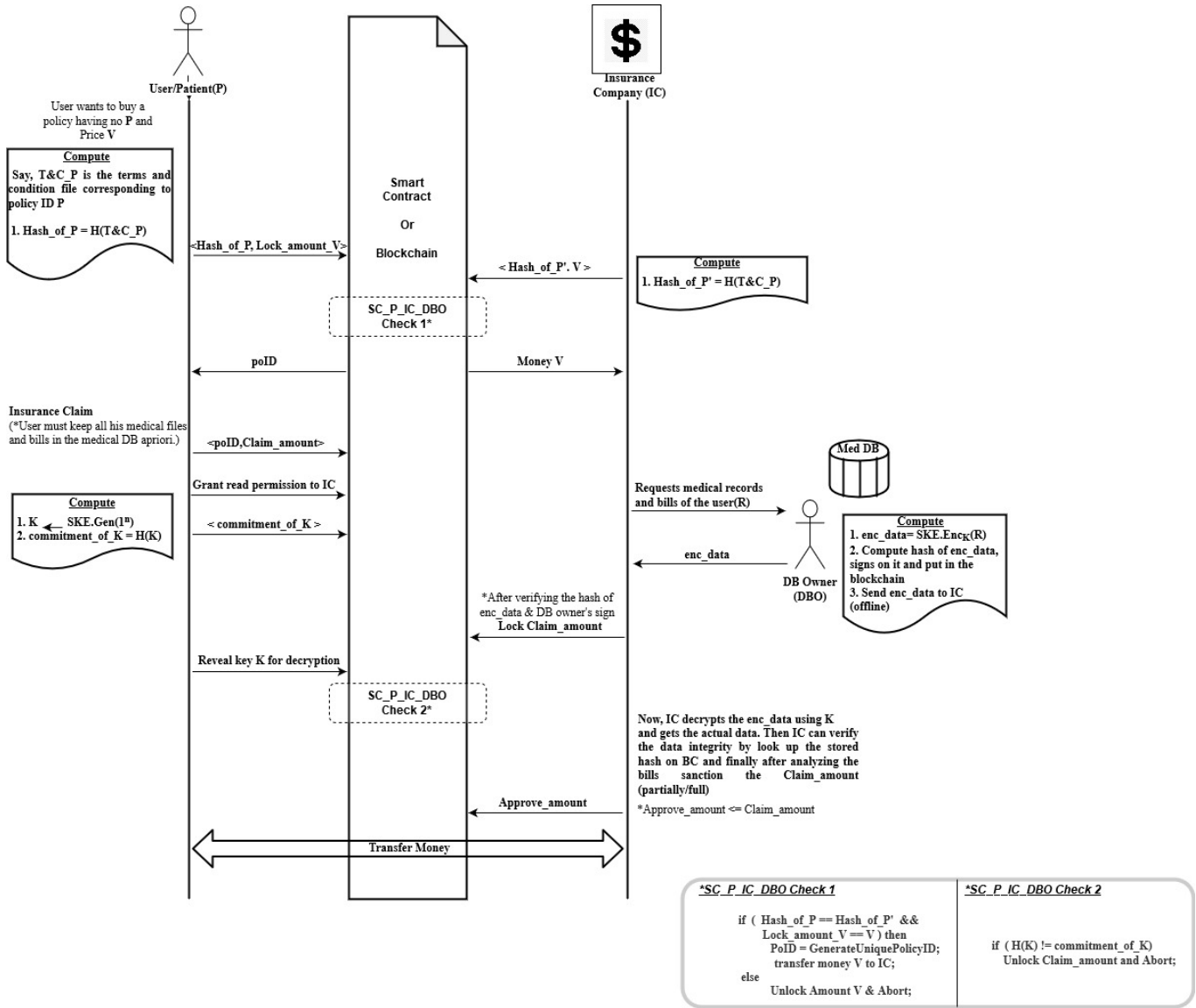
7

Fig. 4: *Interaction among Patient, Insurance Company and Database Owner*

company has proper permission to access this information. Upon receiving the necessary file(s), the insurance company verifies whether the terms and conditions of the policy are met, and accordingly approves the fund. At this stage, it will be a foolish decision to hand over the data in plaintext format to the Insurance Company. It might just steal the data without reimbursing any money to the patient. To prevent such an attack, we propose a solution. First, the user will generate a one-time secret key K and communicate it with the database owner and keep the commitment of the key in the blockchain. When the Insurance Company asks for the patient's data from DBO, DBO encrypts the data using the key K and sends the encrypted data to the insurance company (offline), and keeps the hash of the encrypted data along with the digital signature on the blockchain. Receiving the encrypted file, the Insurance Company asks the user to provide the key K for decryption. The user only reveals the key if he/she finds that the insurance company has already locked the claimed amount in the smart contract. Having received the key K from the user, the Insurance Company

can decrypt the encrypted information. Then the Insurance Company verifies the correctness of the file with the help of blockchain and approves the amount to be sanctioned within some fixed time window. By fetching the stored hash value from the blockchain, the Insurance Company can verify the integrity of the information. The Insurance Company can also check the authenticity of the medical bills by verifying the Digital Signatures (of Hospital Authority and Patient), which are also stored in the blockchain.

An Insurance Company may choose not to respond at all when a user raises a claim request, although the Insurance Company has actively participated in the policy selling process to gain the user's money. This is certainly a malicious behavior of the Insurance Company, which tries to cheat the buyer. We mitigate this sort of risk by allowing the Insurance Companies to lock a certain amount as the security deposit in the system. The security money should not be lesser than the price of the most expensive policy which the company offers to the user (i.e., Security Money >= Costliest Policy Price). An insurance company can deposit or withdraw its
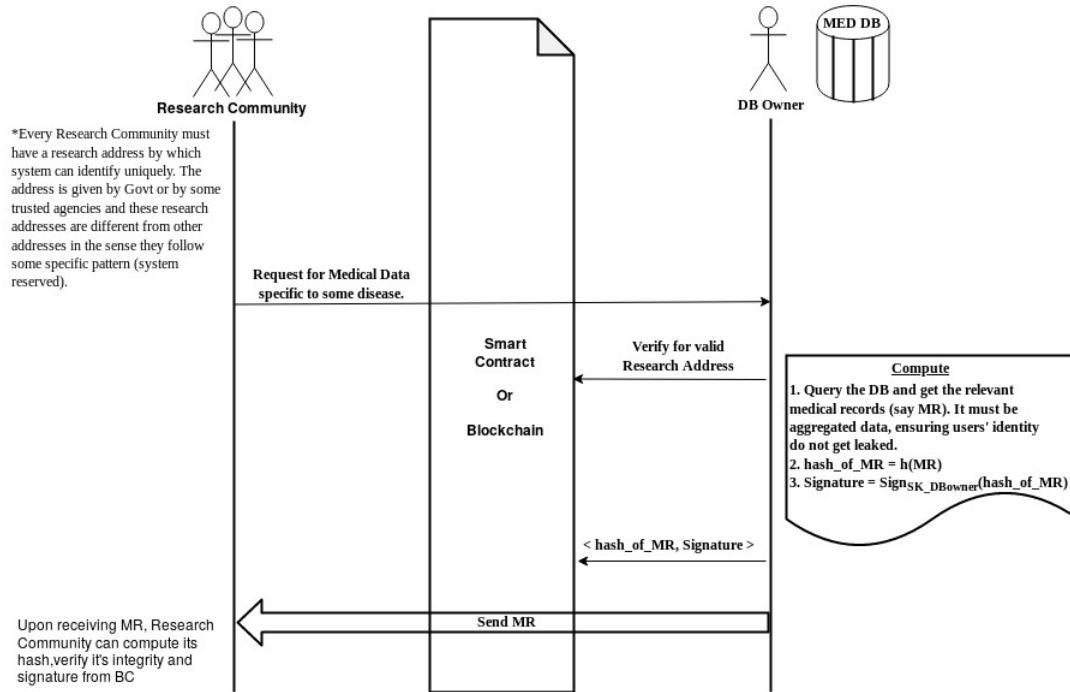
Fig. 5: *Interaction among Government, Research Community and Database Owner*

security money at any time, however, the company needs to maintain the minimum threshold amount locked all the time. The security deposit compensates the buyer to some extent. If it is not possible to provide the claimed amount due to insufficient security money, the system would try to refund the policy price at least, and even it is not possible to refund the policy price, the insurance company gets deregistered from the system. The activity sequence among patient, insurance company, and database owner is depicted in Figure 4.

3.6 Communication among Government, Research Community and Database Owner

If any Research Community requests access to medical data for research purposes, the system must ensure that the data provided must not be personally identifiable. It means that they will get aggregated data, but they do not have any clue about the patient's identity who is suffering from any health issue. It can be realized in the following way- Research Communities are provided some special addresses (say, addresses starting with some special prefix) by the trusted authority or the Government so that when the research community asks for some data from a database owner, the database owner can easily identify the research group. After this authentication, the database owner provides the aggregated data for specific research interests. The database owner also provides the hash of the aggregated data, digital signature over it, and puts them in the blockchain, so that the research community has some level of confidence about the given medical data. The activity sequence between government, research community, and database owner is depicted in Figure 5.

4 IMPLEMENTATION AND TECHNICAL DETAILS

In this section, we will depict the algorithms/procedures for the major functions of our smart contracts.

4.1 Terminology

Prior to describing the actual algorithms, let us discuss some important terminologies and notations that we have used in our implementation (Table 2).

TABLE 2: Terminology & Notation used in our Scheme

Abbreviation	Interpretation
BC	Blockchain
SC	Smart Contract
P	Patient
HA	Hospital Authority
IC	Insurance Company
DBO	Database Owner
RC	Research Community
pAddr	Patient Address
hAddr	Hospital Address
icAddr	Insurance Company Address
dboAddr	Database Owner Address
rcAddr	Research Community Address
pID	Patient ID
hID	Hospital ID
icID	Insurance Company ID
dboID	Database Owner ID
rcID	Research Community ID
ebID	Estimated Cost Bill ID
fbID	Final Cost Bill ID
msID	Multi-Signature ID
poID	Policy ID
cID	Claim ID
rdID	Research Data ID
asID	Application for Storing ID
T_{Event}	Timestamp, when the Event occurs
MR_File	Merkle Tree Root Hash of File
CST	Current System Time
TTL	Threshold Time Limit

Table 2 is not exhaustive. Other abbreviations used in the paper are self-explanatory. The readers can refer to Table 3 for the various structure definitions used in the Algorithms (for more details, the reader may refer to *Appendix*

B for various programming constructs - structures, arrays, mappings).

TABLE 3: Structure Definition

Structure Name	Member Variables
EstimatedCheckUpCost	ebID, pID, hID, estimatedCost, $T_{Estimate}$, $T_{LockingByHA}$, $T_{LockingByP}$, $T_{CheckUpStart}$, $T_{UnlockingByHA}$, $T_{UnlockingByP}$
FinalCheckUpCost	fbID, pID, hID, finalCost, $T_{FinalBilling}$, $T_{ComplaintByP}$, $T_{UnlockingByHA}$, $T_{UnlockingByP}$, $T_{FinalConsentByP}$
MultiSigOnMedicalData	msID, pID, hID, MR_MedData, MR_EncCircuitOperatedMedData, H_x, $Sign_{SK_{HA}} MR\_MedData$, $Sign_{SK_{HA}} x$, $T_{SigningByHA}$, $T_{VerificationByP}$, $T_{UnlockingByHA}$, $T_{UnlockingByP}$ *x=(pID||Date||MR_EncCircuitOperatedMedData)
FileProperties	fileSize, fileChunkSize, depth
KeyAndExchange	key, keyHash, $T_{KeyReveal}$
ApplicationForStoring	asID, pAddr, dboAddr, msID, key, MR_File, MR_EncFile, $T_{Application}$, $T_{VerificationMR}$, $T_{KeyReveal}$, $T_{Complain}$, $T_{Approval}$, $T_{UnlockingByP}$, $T_{UnlockingByDBO}$
PolicyDetails	poID, buyerID, icID, $T_{BuyingPolicy}$, terms&ConFileHash, claimIDs[ ]
ClaimDetails	cID, ebID, $T_{GeneratingClaimByP}$, claimedAmount, approvedAmount, $comm_K$, K, $T_{RevealKey}$, $T_{LockingByIC}$, $T_{UnlockingByIC}$, $T_{Approval}$

4.2 Algorithms

We have classified the algorithms into three categories -

1) **Algorithms for Entity Registration**
2) **Algorithm for Data Upload and Access Control**
3) **Algorithm for Data Access and Incentives**

A. Algorithms for Entity Registration: Every entity has a <SK, PK> pair, where PK and SK act as the address and authentication factor of the entity, respectively (Table 4).

TABLE 4: Key Generation

Entity	KeyGen: $(SK, PK) \leftarrow$ **ASKE.Gen**(1^n)	Comments
P	(SK_P, PK_P)	Key-pair used by Patient. PK_P can also be treated as pAddr.
HA	(SK_{HA}, PK_{HA})	Key-pair used by Hospital Authority. PK_{HA} can also be treated as hAddr.
IC	(SK_{IC}, PK_{IC})	Key-pair used by Insurance Company. PK_{IC} can also be treated as icAddr.
DBO	(SK_{DBO}, PK_{DBO})	Key-pair used by Database Owner. PK_{DBO} can also be treated as dboAddr.
RC	(SK_{RC}, PK_{RC})	Key-pair used by Research Community. PK_{RC} can also be treated as rcAddr.

The patient can register his or her name in the Healthcare System by calling the function **PatientRegistration** (Algorithm 1). The parameter $hash\_of\_Personal\_Info \leftarrow H(name\|\|age\|\|mob\|\|address)$ is a hash value. It helps to preserve the patient's privacy. Once the registration is done, the patient receives a unique identifier.

Similarly, other entities can also register in the system -
HA: $hID \leftarrow$ HospitalRegistration($PK_{HA}, name$)
IC: $icID \leftarrow$ InsuranceCompanyRegistration($PK_{IC}, name$)
DBO: $dboID \leftarrow$ DatabaseOwnerRegistration($PK_{DBO}, name$)
RC: $rcID \leftarrow$ ResearchCommunityRegistration($PK_{RC}, name$)
A **DBO** or an **RC** cannot register themselves directly into the system. A trusted third party or TTP (here, we consider Government as TTP) is responsible to introduce

---

Algorithm 1: Patient Registration Function

Function `PatientRegistration`(PK_P, $hash\_of\_Personal\_Info$)**:**
  if *(function invoker address != pAddr OR PK_P is already registered)* **then**
    | Exit
  end
  Generate a unique pID
  make a mapping or association between PK_P and corresponding pID
  Record the $hash\_of\_Personal\_Info$ corresponding to pID in **BC**
end

---

a DBO or RC into the system. They will call these two functions (i.e., DatabaseOwnerRegistration and ResearchCommunityRegistration).

B. Algorithms for Data Upload and Access Control: Digital medical data or Electronic Health Records (EHR) is generated by the hospital and then transferred to the patient (upon successful discharge and payment of final cost bill). We will discuss later how the hospital generates the EHRs and transfer the same to the patient securely. When the treatment gets completed successfully, either the patient can store the medical files in his/her local storage devices or can decide to store them in some medical repository/database/cloud server. If the patient raises a request for claiming insurance on the medical bill, then the EHRs must be kept in a public medical repository/database, owned by the semi-trusted third-party vendor (also known as DBO or Database Owner). The patient needs to store his/her medical records in the database, and DBO provides the storage space service for some charges. Any two-party fair exchange protocol can be used, however, in this protocol we implement the **Faiswap Protocol** [27], denoted as **SC_P_DBO**, for ensuring a fair exchange of information between patient and DBO. If the DBO finds that the file provided is not correct, it will not provide storage space and abort. The member variables of structure ApplicationForStoring are populated at various invocations of the protocol and these values help in instantiating the two-party exchange protocol. An ID (asID) for applicationForStoring is generated when the patient decides to store the file given by the hospital earlier and invokes **SC_P_DBO** between these two parties.

C. Algorithms for Data Access and Incentives:

1) **Generating Medical Reports of Patient by Hospital**: Here, we will discuss the algorithms related to generating the new medical reports of a patient, fairly communicate the same to the patient party preserving the integrity of the files, generating the final cost bill, and finally, discharging the patient. We break it into two phases - **Pre-Treatment** and **Generation of Medical File and Final Bill**, described in Algorithm 2 and Algorithm 3 respectively.

a) *The Algorithms for **Pre-Treatment** (Algorithm 2)*

–The patient, having ID pID, reports to the hospital the medical problem he or she is facing. Hospital, having ID hID, estimates the cost for the medical service (price V) and locks the amount with the invocation of the

---

Algorithm 2: Pre-Treatment Functions

Function `GenerateEstimatedCostBill`(pID, $estimatedCost$)**:**

  if *(function caller ID != hID)* **then**
    | Exit
  end

  if *(ether value passed with function call != estimatedCost)* **then**
    | Exit
  end

  Generate a unique $ebID$
  Instantiates a structure $EstimatedCheckUpCost\ ec$
  Lock the $estimatedCost$ in the **SC**
  Record the CST (say, $T1$) in **BC**

end

Function `LockEstimatedAmount`(hID, $ebID$)**:**

  if *(function caller ID != pID)* **then**
    | Exit
  end

  Retrieve the structure from **BC** : $EstimatedCheckUpCost\ ec$ (corresponding to $ebID$)

  if *((CST - T1) > TTL OR (ether value passed with function call != ec.estimatedCost))* **then**
    | Exit
  end

  Lock the $estimatedCost$ amount in **SC**
  Record the CST (say $T2$) in **BC**

end

GenerateEstimatedCostBill function (say, at time $T1$). The function generates a unique estimated cost bill ID $ebID$ and also instantiates $EstimatedCheckUpCost$ with this $ebID$. The $ebID$ is communicated with the patient offline.

–Upon receiving the $ebID$, the patient needs to invoke the function *LockEstimatedAmount* (say, at time $T2$) within a limited time period (TTL) to lock the $estimatedCost$ amount.

b) *The Algorithms for **Generating Medical file and Final Bill** (Algorithm 3)*

–Upon patient locking the estimated amount, the hospital needs to invoke the function *StartTreatment* (say, at time $T3$) within a limited time period (TTL) to start the treatment process. The medical files are generated after the function invocation. Keeping in mind the variable nature of the medical processes, there is no time limit set for the next immediate function (*KeepSignedHashToBlockchain*)'s invocation.

–The medical file(MedData) generated is broken into chunks of predetermined size. A Merkle tree is constructed with the chunks and the Merkle root $MR(MedData)$ is obtained. The chunks are operated according to a circuit. In our case, the Boolean circuit maps to the construction of the Merkle tree. Hash is the primary operation of the gates used. In any gate of this circuit, the child gate's output vectors are concatenated and used as inputs to the gate. The concatenated vector is hashed, and the resultant value is the output of the gate. The gate outputs are encrypted with a key, and all the encrypted chunks are used to construct a different Merkle tree and obtain

---

Algorithm 3: Functions for Generating Medical File and Final Bill

---

Function `StartTreatment`(pID, $ebID$)**:**

  if *(function caller ID != hID)* **then**
    | Exit
  end

  Generate the medical files
  Record the CST (say $T3$) in **BC**

end

Function `KeepSignedHashToBlockchain`(pID, $MR(MedData)$, $MR(Enc(CircuitOperatedMedData))$, $H(pID||Date||Enc(CircuitOperatedMedData))$, $Sign_{SK_{HA}}(H(pID||Date||Enc(CircuitOperatedMedData)))$, $Sign_{SK_{HA}}(MR(MedData))$, $H(key)$)**:**

  if *(function caller ID != hID)* **then**
    | Exit
  end

  Keep $MR(MedData)$ and $MR(Enc(CircuitOperatedMedData))$ in **BC**
  Keep $Sign_{SK_{HA}}(MR(MedData))$ and $Sign_{SK_{HA}}(H(pID||Date||Enc(CircuitOperatedMedData)))$ in **BC**
  Store $H(key)$ in **SC**
  Record the CST (say, $T4$) in **BC**

end

Function `VerifyAndGiveConsent`($msID$, $H(pID||Date||Enc(CircuitOperatedMedData))$)**:**

  if *(function caller ID != pID)* **then**
    | Exit
  end

  if *(CST - T4) > TTL)* **then**
    | Exit
  end

  Verify the two hashes and if their respective signatures were signed by the right HA
  Record the CST (say, $T5$) in **BC**

end

Function `DischargeAndGenerateFinalCostBill`($ebID$, pID, $finalCost$)**:**

  if *(function caller ID != hID)* **then**
    | Exit
  end

  if *(CST - T5) > TTL)* **then**
    | Exit
  end

  Get $estimatedCost$ corresponding to $ebID$
  if *(finalCost>estimatedCost)* **then**
    | Exit
  end

  Instantiate the $FinalCheckUpCost$ struct and update the price
  Record the CST (say, $T6$) in **BC**

end

Function `ConsentFinalBillPatient`($fbID$, hID)**:**

  if *(function caller ID != pID)* **then**
    | Exit
  end

  if *(CST - T6) > TTL)* **then**
    | Exit
  end

  Record the CST (say, $T7$) in **BC**

end

Algorithm 4: Functions for File Consent

Function `KeyReveal`(*pID, ebID, key*)**:**
  if *(function caller ID != hID)* **then**
    | Exit
  end
  if *(CST - T7) > TTL)* **then**
    | Exit
  end
  if *(Hash(key) in* **BC** *!= Hash(key))* **then**
    | Exit
  end
  Store the key in the **BC**
  Record the CST (say, $T8$) in **BC**
end

Function `PatientFinalConsent`(*ebID, hID*)**:**
  if *(function caller ID != pID)* **then**
    | Exit
  end
  if *(CST - T8) > TTL)* **then**
    | Exit
  end
  Record the CST (say, $T9$) in **BC** as $T_{FinalConsentByP}$
end

Function `PatientComplain`(*complaint, encodedVectorsThisGate, gateIndexes, ebID, hID*)**:**
  Get $fbID$, $msID$ from the $ebID$
  Get $finalCheckUpCost$, $multiSigOnMedicalData$, $MultiSignIdToEncFileProperties$ structs from the corresponding IDs
  if *(function caller ID != pID)* **then**
    | Exit
  end
  if *(CST - T9) > TTL)* **then**
    | Exit
  end
  if *($T_{FinalConsentByP}$!=0)* **then**
    | Exit
  end
  if *(length of complaint !=0 length of encodedVectorsThisGate)* **then**
    | Transfer the final amount to the hospital
    | Exit
  end
  if *(length of gateIndexes < maxLinesToGate+1)* **then**
    | Transfer the final amount to the hospital
    | Exit
  end
  if *(MerkleProof(encodedVectors,encFileRootHash)* **then**
    | Transfer the final amount to the hospital
    | Exit
  end
  Decrypt the encoded input vectors and the encoded output vector of the gate
  Calculate the resultant vector from the input vectors and the gate operation
  if *(Resultant gate output vector != Decrypted gate output vector)* **then**
    | Transfer the final amount to the hospital
    | Exit
  end
  Record the CST (say, $T10$) in **BC** as $T_{ComplaintByP}$
end

---

a Merkle root $MR(Enc(CircuitOperatedMedData))$. This Merkle root is concatenated with the patient ID, date of report, etc. It is hashed and signed $Sign_{SK_{HA}}(H(pID||Date||Enc(CircuitOperatedMedData)))$. Hospital also signs on the Merkle root of the file $Sign_{SK_{HA}}(MR(MedData))$. The two hashes, the corresponding signatures, and few file-related variables are the parameters to the function ***KeepSignedHashToBlockchain***. Let us say the function is invoked at the time $T4$.

–The patient by this time might have received the encrypted file chunks from the hospital. The patient constructs the Merkle tree with the received $Enc(CircuitOperatedMedData)$ and checks whether the constructed Merkle root and the root that is kept in the blockchain. Patient also verifies the matching of the concatenated hash $(H(pID||Date||Enc(CircuitOperatedMedData)))$ discussed in the last function. The two signatures - $Sign_{SK_{HA}}(MR(MedData))$ and $Sign_{SK_{HA}}(H(pID||Date||Enc(CircuitOperatedMed$ $Data)))$ are also verified by the patient. If these parameters are valid, then the patient invokes the function ***VerifyAndGiveConsent*** (say, at time $T5$) within a limited time period (TTL).

–If the estimated cost and the actual cost are the same, the hospital invokes the function ***DischargeAndGenerate-FinalCostBill*** (say, at time $T6$) within a limited time period (TTL) by keeping the final cost, $finalCost$, same as the estimated cost $estimatedCost$. Otherwise, the hospital puts up a new final cost by invoking the function. Here, the final cost is always less than or equal to the estimated cost because the estimated cost is an overestimation of the treatment cost.

–The patient calls the function ***consentFinalBillPatient*** (say, at time $T7$) within a limited time period (TTL) if the final cost provided by the hospital is satisfactory.

The Algorithm for **File Consent** is described in Algorithm 4. In this step, the hospital provides the key for decrypting the file. Any misbehavior detected leads to penalization of the party involved in providing wrong or no information.

a) *The Algorithms for **File Consent** (Algorithm 4)*

–The hospital reveals the key by invoking the function ***KeyReveal*** (say, at time $T8$) within a limited time period (TTL). Then the key is hashed and checked with the hash of the key $H(key)$ (key's commitment) kept in the contract.

–When the hospital discloses the key in the blockchain, the patient decrypts the file chunks and performs the circuit operation of the gates one by one. If all the checks are valid, then the patient gives consent by invoking the function ***PatientFinalConsent*** (say, at time $T9$) within a limited time period (TTL) and the locked amount is transferred to the hospital.

–If the above-mentioned check produces any mismatch, an appropriate complaint is generated, and it is produced by the patient through the function ***PatientComplain*** (say, at time $T10$) within a limited time period (TTL). If the complaint is a valid one, the hospital is penalized and vice-

versa.

2) **Accessing client's medical report by Insurance Company in order to approve policy claim.** We can further subcategorize these algorithms into four parts - Procedure for **Buying an Insurance Policy** fairly from an IC is stated in Algorithm 5. Procedure for **Claiming an Insurance** by a Policy Buyer is stated in Algorithm 6 and for **Processing a Claim** is stated in Algorithm 7. **Approving a Claim** has been described in Algorithm 8.

Algorithm 5: Functions for Buying an Insurance Policy fairly

Function `BuyPolicyPhaseOne`($icID$, $hashOfTermsAndCon$)**:**
    Store the $hashOfTermsAndCon$, $icID$ in **BC**
    Lock the Policy Price V in **SC**
    Record the CST (say, $t1$) in **BC**
end
Function `BuyPolicyPhaseTwo`(pID, $price$, $hashOfTermsAndCon$)**:**
    if *(function caller ID != $icID$)* **then**
        | Exit
    end
    if *($price > securityMoney_{icID}$)* **then**
        | De-register the IC from the System and Exit
    end
    Record the CST (say, $t2$)
    if *(($t2 - t1$) > TTL)* **then**
        | Exit
    end
    Retrieve the amount of money locked by buyer pID from **BC** and matches with $price$
    Retrieve the hash of terms and condition file mentioned by buyer pID from **BC** and matches with $hashOfTermsAndCon$
    if *(the above matches are found incorrect)* **then**
        | Exit
    end
    Generate a unique $poID$
    Instantiate a structure $PolicyDetails\ pd$ corresponding to the $poID$ and store in **BC**
    return *($poID$)*
end
Function `WithdrawLockedPolicyBuyingMoney`($icID$)**:**
    Retrieve the timestamp($t1$) of locking the money from **BC**
    if *(CST - $t1$) > TTL) AND the seller IC does not respond* **then**
        | Unlock the money from **SC** and Credit to the Buyer's Account
    end
end

---

a) *The Algorithms for **Buying an Insurance Policy** fairly from an IC (Algorithm 5)*

–Say, a user having ID pID wants to buy a policy from an IC (having ID $icID$). The price of the policy is V unit. The user reads the terms and conditions file ($T\&C\_P$ in fig 4), where all the policy-related details and benefits are mentioned. Then the user computes the hash of the terms and condition file locally as -
$hashOfTermsAndCon \leftarrow H(T\&C\_P)$

Algorithm 6: Function for Claiming Insurance

Function `ClaimMoney`($poID$, $ebID$, $asID$, $claimedAmount$, $comm\_K$)**:**
    Obtain $fbID$ and $msID$ from $ebID$
    Retrieve the structures from **BC** : $PolicyDetails\ pd$, $FinalCheckUpCost\ fc$ and $ApplicationForStoring\ as$ (corresponding to $poID$, $fbID$ and $asID$ respectively)
    Parse the structures' data and Verify the necessary field values for correctness
    Check(function caller ID == $pd.buyerID$ == $fc.pID$ == $as.pID$)
    Check($as.multiSigID$ == $msID$ AND $as.timestamp\_approval$!= 0)
    Check($claimedAmount \leq fc.finalCost$)
    Embed the business logic to check that the user has not claimed for the same bill before.
    if *(any of the above checks is FALSE)* **then**
        | Exit
    end
    Generate a unique cID
    Instantiate a structure $ClaimDetails\ cd$
    $cd.comm_K \leftarrow comm\_K$
    $cd.T_{GeneratingClaimByP} \leftarrow CST$
    Store cd into **BC**
    Make an entry in mapping: $cIDTopoID[cID] \leftarrow poID$
    $GrantReadAccessPermission(pd.icID)$
end

---

and invokes the function **BuyPolicyPhaseOne** (say, at time $t1$).

–In the first phase of buying, the user locks the policy price V in the SC and also keeps the hash of the terms & conditions file in the BC. In the second phase, the IC should acknowledge the buyer's interest to buy a policy within a fixed time window (TTL). So, the IC invokes the function **BuyPolicyPhaseTwo** (say, at time $t2$). IC needs to specify the hash of the terms & conditions file and the policy price. The smart contract verifies whether these values specified by both parties (specified by policy buyer in phase one and so by the insurance company in phase two) match or not.

–After the first phase of buying, if the IC does not respond within the fixed time period (i.e., $(t2 - t1) > TTL$), the policy buyer can withdraw his or her locked money (policy price) from the system by calling the function **WithdrawLockedPolicyBuyingMoney**.

b) *The Algorithms for **Claiming an Insurance** by a Policy Buyer(Algorithm 6)*

–The policy buyer or the user can claim to the IC for reimbursement of his or her medical expenditure by calling the function **ClaimMoney**. The user generates a temporary key K and computes the commitment of the key K offline.
$K \leftarrow SKE.Gen(1^n)$.
$comm\_K \leftarrow H(K)$.
The user sends the key K to the DBO (offline, one-to-one communication) and specifies the $comm\_K$ (as function parameters) while invoking the function. So, $comm\_K$ would go to the BC, once the transaction gets mined. This function essentially authenticates the identity of the user, validates the important data (mentioned as the member

Algorithm 7: Functions for Processing an Insurance Claim

Function `KeepSigOnHashOfEncFile`(cID, $sign\_DBO$):

    Store $<sign\_DBO$ and $CST>$ corresponding to cID in the **BC**

    Mark the key K as expired

end

Function `LockClaimedMoney`(cID, $amount$):

    Obtain $poID \leftarrow cIDTopoID[cID]$

    Retrieve the structures from **BC**: $PolicyDetails\ pd$, $ClaimDetails\ cd$ (corresponding to $poID$ and cID respectively)

    if (*function invoker ID != pd.icID*) **then**

        Exit

    end

    if (*amount == cd.claimedAmount*) **then**

        Lock the *amount* in the **SC**

        $cd.T_{LockingByIC} \leftarrow CST$

    end

end

Function `RevealSecretKey`(cID, K):

    Obtain $poID \leftarrow cIDTopoID[cID]$

    Retrieve the structures from **BC** : $PolicyDetails\ pd$, $ClaimDetails\ cd$ (corresponding to $poID$ and cID respectively)

    if (*function invoker ID != pd.buyerID*) **then**

        Exit

    end

    if (*H(K)!= cd.comm$_K$ OR (CST - cd.T$_{LockingByIC}$) > TTL*) **then**

        Exit

    end

    $cd.K \leftarrow K$

    $cd.T_{RevealKey} \leftarrow CST$

end

Function `WithdrawLockedClaimedMoney`(cID):

    Obtain $poID \leftarrow cIDTopoID[cID]$

    Retrieve the structures from **BC** : $PolicyDetails\ pd$, $ClaimDetails\ cd$ (corresponding to $poID$ and cID respectively)

    if (*function invoker ID != pd.icID*) **then**

        Exit

    end

    if (*cd.T$_{RevealKey}$ == 0 AND cd.T$_{LockingByIC}$!= 0 AND (CST - cd.T$_{LockingByIC}$) > TTL*) **then**

        Unlock the *claimedAmount* and Credit to the IC's Account.

    end

end

Algorithm 8: Functions for Approving an Insurance Claim

Function `ApproveClaim`(cID, $approvedAmount$):

    Obtain $poID \leftarrow cIDTopoID[cID]$

    Retrieve the structures from **BC** : $PolicyDetails\ pd$, $ClaimDetails\ cd$ (corresponding to $poID$ and cID respectively)

    if (*function invoker ID != pd.icID*) **then**

        Exit

    end

    if (*cd.T$_{RevealKey}$ == 0 OR cd.T$_{Approval}$!= 0 OR (CST - cd.T$_{RevealKey}$) > TTL*) **then**

        Exit

    end

    $cd.approvedAmount \leftarrow approvedAmount$

    $cd.T_{Approval} \leftarrow CST$

    Unlock the *approvedAmount* and Credit to the Buyer's Account

    $remainingAmount \leftarrow (cd.claimedAmount - cd.approvedAmount)$

    if (*remainingAmount > 0*) **then**

        Unlock the *remainingAmount* and Credit to the IC's Account

    end

end

Function `SelfApproveClaim`(cID, $icID$):

    Obtain $poID \leftarrow cIDTopoID[cID]$

    Retrieve the structures from **BC** : $PolicyDetails\ pd$, $ClaimDetails\ cd$ (corresponding to $poID$ and cID respectively)

    if (*function invoker ID != pd.buyerID*) **then**

        Exit

    end

    if (*cd.T$_{RevealKey}$!= 0 AND cd.T$_{Approval}$ == 0 AND (CST - cd.T$_{RevealKey}$) > TTL*) **then**

        $cd.approvedAmount \leftarrow cd.claimedAmount$

        $cd.T_{Approval} \leftarrow CST$

        $cd.isSelfApproved \leftarrow TRUE$

        Unlock the *claimedAmount* and Credit to the Buyer's Account.

    end

end

---

variables of various structures), verifies the signatures, and also checks for the timestamps of various key activities with the aid of BC. The algorithm also checks that the policy-holder can not claim for the same medical bill more than once. If everything is found right, then the function generates a unique cID and communicates the same to the user for future references. The function also instantiates a structure of type $ClaimDetails$ with necessary values and stores in the BC. The User also needs to grant read access permission to the IC for verifying the medical files/bills from the repository.

c) *The Algorithms for **Processing a Claim** (Algorithm 7)*

–Obtaining the Key K from the user, the DBO verifies whether $comm\_K$ matches with the received key. If it matches, then DBO encrypts the user's file (say, the file is R).

$enc\_data \leftarrow SKE.Enc_K(R)$

Once the encryption is done, the DBO must invalidate the key K. Invalidation of K is essential, else it leads to security risk. The DBO computes the hash of the encrypted data, signs over it, and puts it into the BC for accountability purpose (i.e., $Sign_{SK_{DBO}}(H(enc\_data))$). The DBO invokes the function ***KeepSigOnHashOfEncFile*** and sends the encrypted file in offline mode to the IC.

–Next, the IC needs to lock the claimed $amount$ in the **SC** and invokes the function ***LockClaimedMoney***.

–Once IC locks the claimed amount, the user (i.e., policy buyer) should reveal the key K within some fixed time window(TTL) so that IC can decrypt the encrypted files. The user calls the function ***RevealKey***.

–In case the policy claimer (i.e., user) has not provided the key K within the specified time window, the IC can withdraw the locked money by invoking the function ***WithdrawLockedClaimedMoney***.

d) *The Algorithms for **Approving a Claim** (Algorithm 8)*

–Getting the key K from the user, the IC decrypts the encrypted file and verifies all the necessary information. After verification, the IC approves the claim as per the norms specified in the *Terms and Conditions* of the policy (i.e., $T\&C\_P$). The claim is not necessarily approved all the time fully, it may also be approved partially. Here, the key point is that - the IC is confined to obey all the terms defined in the $T\&C\_P$ file. There is no way to bypass any rules or norms, as the hash of this file was already stored in the BC. The IC should invoke the function *ApproveClaim* within a fixed time window(TTL) after the key is revealed by the user.

–If the IC does not invoke the above function within the specified time window (even if the user has revealed the key in proper time), we give the user the provision to validate the claim amount entirely by himself (or herself). In such cases, the policy-holder can take away the locked money of the IC by calling the function *SelfApproveClaim*. It takes care of the user's fairness.

Algorithm 9: Functions for Accessing Medical Data by RC

Function `RequestDataForResearch`(*dboID*, *hash_Query*)**:**
  if *(function caller ID is not a $rcID$)* **then**
    | Exit
  end
  Generate a unique rdID
  Store <*dboID*, *rcID*, *hash_Query*> to the **BC** correspoding to $rdID$
  Record the CST(say,t_1) in **BC**
end
Function `ProvideDataForResearch`(*rdID*, *hash_Data*, *sign*)**:**
  if *(function caller ID != $dboID$)* **then**
    | Exit
  end
  Fetch the timestamp(t_1) related to $rdID$ from **BC**
  if *(CST - t_1 > TTL)* **then**
    | Exit
  end
  Store <*hash_Data*, *sign*> to the **BC** correspoding to $rdID$
  Record the CST(say,t_2) in **BC**
end

---

3) **Providing medical data to RC for research purpose**: When Research Community (RC) seeks medical data, it can directly approach the DBO to provide medical data for the sake of research. We have already depicted the model in section 3.6. Procedure for *Accessing Medical Data by RC* is stated in Algorithm 9.

–First, RC must obtain a license from the Government to carry on the Research Activity. The Government has the only authority to get the RC registered in the system. Therefore, the former introduces the RC in the system by calling the function *ResearchCommunityRegistration*. As a result, the RC will receive a unique rcID.

–After getting registered in the system, the RC can ask the DBO to provide medical data for the sake of research by calling the function *RequestDataForResearch*.

–In response to the RC's data request, the DBO provides relevant medical data to the RC. Here, the key point to be remembered is that DBO cannot send any information which can compromise the patient's privacy. DBO can only provide aggregated data but, at the same time, abide by the rules stated to preserve privacy (semi-trusted assumption). The DBO computes the hash of the medical data which is to be sent and also puts a signature on this. These are kept in the BC. The DBO invokes this function *ProvideDataForResearch*.

Reader may refer to *Appendix A*, where the function (or activity) sequence diagrams are given for the contracts SC_P_HA, SC_P_IC_DBO and SC_RC_DBO.

5 SECURITY ANALYSIS

Blockchain technology uses some cryptographic primitives (e.g., hash function, digital signature). As long as the underlying cryptographic primitives are secured, the blockchain is secure, and so is our system. Assuming that the blockchain is secure, the money locked in the blockchain is protected, and hence the payment involved in the system is also safe. We claim that our system takes care of essential security aspects and provides fairness to the parties involved in the system. We state the security properties achieved by our protocol.

5.1 Fairness

We discuss the fairness of each party i.e., Patient, Hospital Authority, Insurance Company, and Research Community. Even if one of the parties acts malicious or more than one party colludes and tries to cheat, the honest party will not lose money or become a victim. In any situation, the malicious parties get penalized or reprimanded.

Proposition 1. (Patient's Fairness) The honest patient must not lose money or gets treated unfairly, no matter if the other parties (Hospital/Insurance Company) are behaving maliciously (say, they try to cheat the patient in various possible ways), under the assumption that the owner of Medical Data Repository is semi-trusted and the underlying blockchain is secure.

Proof: We will prove the proposition in the cases where an honest patient's interest might get compromised. Specifically, we will be analyzing the security on the grounds of price, data, and responsiveness.

– **Hospital Authority (HA) is malicious**

   HA tries to overcharge after treatment: Initially, at the time of admission, the Hospital Authority examines the medical history of the patient and generates a bill with an estimated cost, and it gets recorded in the BC. If the patient agrees with it, then only he or she would proceed further to take the services from the hospital. The final cost of check-up must not be higher than this estimated cost. Therefore, in our system, a patient knows the upper limit of the total check-up cost beforehand. It might be possible that the hospital has overcharged, or the patient is not fully satisfied with the treatment, then

both the parties can negotiate on this bill and come to a joint agreement (however, it can never be higher than the estimated cost, that was given initially).

HA sends wrong medical files to patient: Medical Data or EHR (Electronic Health Record) is a crucial factor in determining the fairness of the protocol. The patients have to be ensured of receiving the correct file and if not, then there is a provision of raising a complaint and aborting from the protocol. Before actually sending the medical reports to the patient, the hospital first splits the file into multiple chunks of equal size. The Merkle root of the file chunks is obtained; the hospital keeps this Merkle root and the signature on it in the blockchain. Let this root be M_1. The hospital encrypts the entire Merkle tree's input, starting from the leaf level, which is the chunks of the file, to the intermediate level up to the root, using a key and then computes the Merkle root of the encrypted inputs. Let this root be M_2. This Merkle root is concatenated with the patient identifier and the date before being signed by the hospital. The encrypted file Merkle root, along with the digital signature, is put in the immutable blockchain (refer to $< M_2, S_2 >$ in Figure 3). Next, the hospital sends the encrypted inputs to the patient using some fair exchange protocol (in our case, we are using fairswap protocol [27]). Upon receiving the encrypted inputs, the patient re-computes the Merkle root, verifies the result with the stored root in the blockchain, and verifies the signatures to ensure that the file indeed came from the same hospital (proof of authenticity).

Date and patient identifier has been used along with M_2 for the signature to ensure that the hospital is held accountable if in case it sends encrypted inputs of a file belonging to a different person, or it sends a patient's previous medical file. If in case the hospital keeps the date and patient identifier correct with a mismatched file, the patient on deriving the file after the revelation of the key might come to that file is corrupted. The Merkle root of the file and the signature on it that was kept in the blockchain serves as a sufficient means to hold the hospital for its maliciousness.

The key closely coupled with the file process also needs to be rightly served to the patient to ensure a fair protocol. There are two ways in which the key can become an issue for the patient. In the first case, the hospital keeps the commitment of one key in the blockchain and reveals a different key to be used for decryption. In the second case, right after encrypting with a key, the hospital commits to a different key and reveals the same. In this case, the patient will decrypt the file with the second key and on finding a mismatch in the gate output computation, it will lodge a complaint which will be resolved on-chain by penalizing the malicious party.

HA repudiates: In case the hospital authority provides a wrong file and computes the digital fingerprint as per the wrongly generated file, then the contract cannot detect whether things are out of place. When the patient later detects it has received the wrong medical report, in such a case the transcript of the interaction between hospital and patient present in blockchain can be used for accusing the authorities and taking legal action if needed.

HA becomes unresponsive in the midst of the protocol: A timely response is necessary for a fair protocol. A party can stop responding, which can lead to an indefinite delay in the termination of the protocol. So, when the patient locks the estimated bill amount in the smart contract; the hospital needs to start the treatment within a fixed time window (a tunable parameter that can be set as per the system requirement). Hospital immediately invokes a function $startTreatment$ (defined in SC_P_HA) and updates the state. That essentially makes a transaction entry in the blockchain. Failure to start treatment within a limited period causes the patient to unlock his/her money. A patient can then seek treatment from some other hospitals. We can also penalize the hospital for this negligence (by deducting some amount of locked estimated cost of the hospital). That is the reason behind locking the estimated cost amount by the hospital. If the hospital becomes unresponsive at any point in the protocol, the patient can call the respective exit function to abort from the protocol, and we can also penalize a hospital for any such malpractice by deducting its locked money.

– **Insurance Company (IC) is malicious**

IC tries to cheat user/policy buyer by not approving claim: The hash of the terms & conditions file of the policy is stored in the blockchain, and the patient also verifies the same at the time of buying the policy. Therefore, later on, the insurer can not deny it. Another thing is that when a policy buyer claims his or her medical bill amount, a transaction is made to reflect the same, and hence, the details go to the blockchain, and it must be settled within some fixed time window after the claim. Otherwise, the insurer will be penalized, and in a distributed setup like blockchain, the insurer loses its reputation among the community, which they obviously try to avoid. If IC approves a claim partially and the user/claimer is not satisfied with the approval, he/she can take help of the law complaining against the IC. The judge can easily sort out the dispute with the aid of blockchain (as the hash of terms and condition file is kept in BC) In case the IC does not respond to the user's claim at all, then the system will be able to return the price paid upon purchase of the policy to the user. If it is not possible to return the money (due to insufficient security money of IC), the IC would be blacklisted from the system.

IC steals patient's medical data: If we observe the claim process in Figure 4, the policy buyer needs to grant permission to IC to access his or her data stored in the medical database, so that insurer can verify the authenticity of the claim raised by the policy buyer in the future. However, if DBO releases the medical report, then the Insurance Company can take the data and stop any sort of interaction with the buyer. To prevent this, the Database owner sends the encrypted data (enc_data in Figure 4) to the Insurance Company. Encryption is done using a temporary key K, generated by the user itself. This key is used for one time only. Upon receiving the encrypted data, the Insurance Company locks the claimed amount in the smart contract and asks for the key (K) from the user. The user would not reveal the key unless the insurer locks the money in the contract. Another thing

is that we have discussed about logical data partitioning - a patient/user data can be divided into three different categories - Personal Information, Medical Reports, and Medical Expenses. If privacy is needed, users can only grant access to the Medical Expenses section of the data and hide the other parts. Hence, it takes care of the data privacy aspects as well.

IC becomes unresponsive in the midst of the protocol: For every subsequent function call in the smart contract, there is a fixed time window. If the IC fails to respond on time, then the user can take suitable actions accordingly, e.g., a user can unlock the money for buying a policy if the IC does not reply in Phase two of policy buying. Similarly, at the time of claiming as well, the user can take appropriate measures so that he/she would not lose money, in case the IC becomes unresponsive.

– **Both Hospital Authority and Insurance Company are malicious** Irrelevant since both the entities (HA and IC) act independently of the patient or policy buyer. The activity of HA does not influence the IC in any way and vice-versa.

Proposition 2. (Hospital's Fairness) The Hospital Authority would be able to get its money for all the valid services provided to the patient, in spite of the patient's misbehavior (say, the patient tries to take services from the hospital without paying the bill amount and then leave), under the assumption that the underlying BC is secure.

Proof: If a patient is malicious, he or she may try to cheat in the following ways:

Patient does not pay for the medical service: A patient can not deny paying the hospital after taking medical services. Because at the time of taking admission to the hospital, the hospital authority would guess the expected cost of the treatment. The patient is supposed to lock this amount in the smart contract, and only then the treatment begins. Again, at the time of discharge, the hospital authority calculates the exact/final treatment cost. In our case, we have considered that the final cost should not be greater than the estimated cost. Once the patient gives consent over the final bill by putting a signature, then the patient receives all the medical files (i.e., Prescriptions, reports, etc.). Otherwise, the patient does not obtain these medical files (Patient can have the encrypted files, but the decryption key is still with the hospital. It is only revealed after the final bill has been locked in the contract by the patient).

Patient produces a wrong complaint: Receiving the correct file, the patient might try to raise a false complaint to exonerate money from the hospital without paying any money for the obtained services. The encrypted Merkle root ensures that the complaint should be part of the Merkle tree. This is used for checking the validity of the complaint and the fact that the patient is lying gets detected.

Patient becomes unresponsive in the middle of protocol: Different exit points prevent an honest party from waiting indefinitely and keep its money locked in the smart contract. If this happens at any time in the middle of the protocol, the hospital can exit and unlock the money locked in the contract.

Proposition 3. (Insurance Company's Fairness) When a user buys a policy from an Insurance Company, the Insurance Company would be able to get the money stated in the terms while purchasing the policy. There should not be any such instances where a malicious user/buyer enjoys the benefits of a policy without paying the insurance company's policy price. Another aspect is that a user or policy buyer can not make a false claim and cheat the Insurer and force them to pay money.

Proof: If User (Policy buyer) is malicious, he or she may try to cheat in the following ways:

User gets benefited from a policy without paying for it.: In the first phase of policy buying (Algorithm 5), the policy buyer expresses his or her interest in keeping the money (i.e., Policy Price) locked in the contract. The user also puts the hash of the terms and conditions files into the BC. The IC verifies the same in the second phase of buying. If everything is found right, then the smart contract transfers the policy price to the IC's account, and a unique policy ID is communicated to the buyer. So, it is not possible to have a policy without paying money for the same.

User tries to generate a false claim: When the user claims reimbursement for his/her medical expenses (Algorithm 6), the system checks whether the claim has already been sanctioned. Then the system checks whether all files related to medical expenses are correct (by matching their hash and by verifying the signature of Hospital Authority and patient, stored in the BC). Finally, the system checks whether the claim follows the terms and conditions of the policy. Once it is verified, the claim is sanctioned (partially or full). So, the Insurance Company's fairness is preserved. Users cannot cheat an Insurance Company in our system.

User becomes unresponsive in the middle of the protocol: There is no motivation as such for the user to become unresponsive (as it is the user and not the IC, whose money has been locked in the system). However, a user may be unresponsive in the claiming process. After making an insurance claim, the user might delay revealing the key when the IC needs the key for decryption of the encrypted files. If it happens, then IC can exit from the protocol without approving any money claimed by the user and unlocks the IC's money locked in the contract. Therefore, IC does not get cheated in case the user becomes unresponsive.

Proposition 4. (Research Community's Fairness) In our system, the Research Community(RC) should have sufficient confidence regarding the correctness of the medical data, which they received from a medical repository.

Proof: It is essential to provide correct medical data to the RC, as analysis of wrong data may lead to an incorrect outcome in research, which is not desirable for advanced medical science. At the same time, it is not good practice to reveal the patient's identity while sending the data to RC (i.e., preserving patient privacy). Keeping these facts in mind, when RC requests for medical data, the DBO (which is a semi-trusted party appointed by the Government) sends the aggregated medical data of the specific disease (without disclosing the patients' identity) to the RC. It is right that we are dependent on the trust assumption of the DBO here. But DBO is not trusted. Whatever DBO sends to the RC, the hash of it as well as the signature over the computed hash, is kept in the BC. Hence, the DBO is also accountable for the activities, which he or she does. Once the data is received,

the RC computes the hash of it and verifies the signature. If everything is found right, collecting the medical data, the RC can carry on its research work with sufficient confidence.

5.2 Privacy

A patient's medical data is sensitive information. If a person's health record is available publicly, he or she may face embarrassment and might be subjected to discrimination in daily life. Hence, it must be ensured that access to patient data is provided only with the patient's consent.

Proposition 5. (Patient's Privacy) In our proposed system, none of the entities can access a patient's data unless it has been granted permission. At the same time, personally identifiable attributes o the patient remains hidden from public view.

Proof: A patient does not reveal the attributes like name, age, address, contact number, etc while registering on the blockchain. Instead, the hash of these attributes is kept in the BC at the time of registration. Since our Healthcare model is patient-centric, no one can see the patient's medical data without his/her consent. The patient defines his/her access control matrix, mentioning the identities who have access to the patient's data. The patient has the provision to update the matrix from time to time - he or she can grant access permission to another party or revoke access permission from a party. The access control matrix is stored in the BC.

5.3 Data Security

Tampering a patient's medical data might have catastrophic consequences. If medicine doses are changed or if the ailment diagnosed is modified, then this could lead to life risk. Also, a patient may try to tamper with his or her medical history to claim reimbursement from the insurance company illegitimately. We discuss how our protocol mitigates such problems.

Proposition 6. Proposition 6 (Data Security) In our proposed system, no one, including the patient, can tamper with the medical data and misuse it for malicious purposes.

Proof: Since we have built our protocol using blockchain as the underlying framework, the security of medical data follows from the inherent property of blockchain's immutability.

6 PERFORMANCE EVALUATION AND DISCUSSION

6.1 Implementation Setup

We have implemented the Health Care Management System on Ethereum test networks in a system having Intel(R) Core(TM) i7-6700HQ running Linux Mint 18.04 19.1 (Tessa), 64-bit operating system using 16.00GiB of RAM. We have used the Ropsten test network and an infura endpoint.

6.2 Result

The main objective was to get the various results from the implementation. The two main obvious factors that determine the feasibility of any blockchain model are cost and time. Since the implementation is in the Ethereum blockchain, we have taken the transaction cost (Gas Cost converted to equivalent Dollar) of the protocols and the time taken to execute the same. Particularly, the entire implementation has been deployed and run in the Ropsten test network. The time as a metric when taken from the public network like Ropsten will give a much more pragmatic insight compared to the private network setup.

TABLE 5: Deployment Addresses of Smart Contracts

Smart Contract	Address
SC_Registration	0x5a818296705cC24Feec4CfEAF1DfdaE056fEf037
SC_P_HA_1	0x9528dA5753ae928Eb1e0284C7b1771e2FC17a766
SC_P_HA_2	0x7b88e153aC1b2BCA865CD58E1082f50Ed69f4c3c
SC_P_DBO	0xC062E1eF5EdB815bcF5B93C6BaD497ABCA407f31
SC_P_IC_DBO	0xD519535972d006DD72AbBd60453Ae78747065B5e
SC_RC_DBO	0xafb22029C41cA8a4c04D0a07284d2b2c257aE723

TABLE 6: Deployment Cost of Smart Contracts

Smart Contract	Deployment Cost(Ether)
SC_Registration	0.0353157
SC_P_HA_1	0.0994202
SC_P_HA_2	0.0583187
SC_P_DBO	0.04783231
SC_P_IC_DBO	0.0516384
SC_RC_DBO	0.0113837

Table 5 specifies the addresses of the deployed contracts. The contract deployment is a one-time occurrence. The transaction cost and the time taken for each contract deployment have been depicted in Figure 6 and Figure 7 respectively. The gas price was 18.9 Gwei and the ether cost was 2300.54 dollars at the time of deployment. Depending upon the size of the contracts, the deployment cost varies (Table 6). These are one-time costs. So, once deployed, we can get the benefits throughout the usage of this protocol.

The smart contracts for patients and hospitals (SC_P_HA_1 and SC_P_HA_2) have been split into two parts, citing the limited gas limit for blocks in Ethereum. The high gas for the collective patient and hospital contracts is a reflection of the slightly higher steps involved in the protocol.
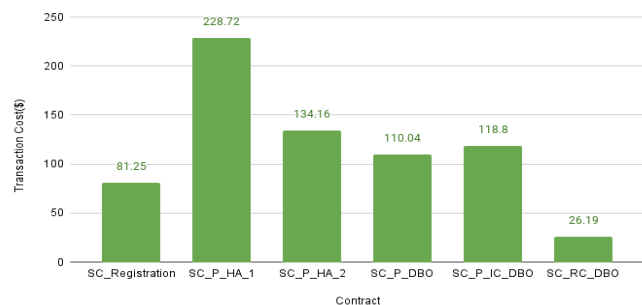

Fig. 6: *Transaction Cost for Contract Deployment*

The transaction costs and time taken associated with the entities' registration process, depicted in Figure 8 and
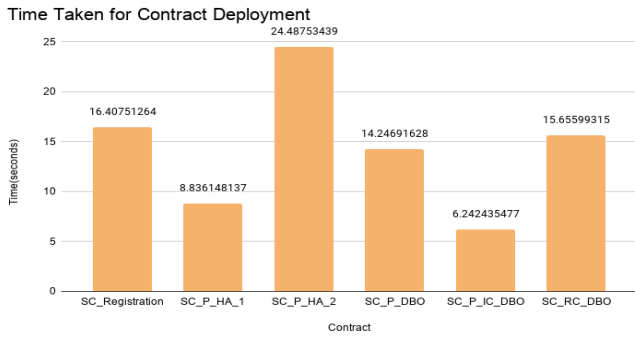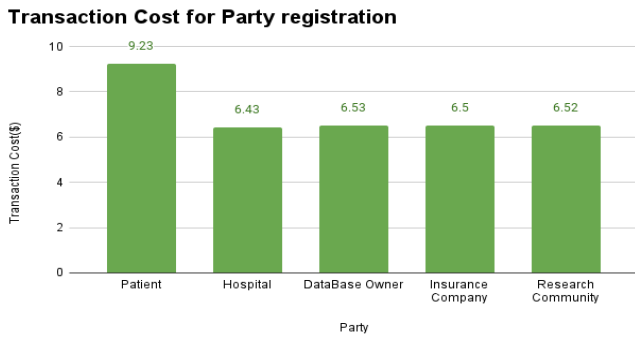
18

Fig. 7: *Time for Contract Deployment*

Fig. 11: *SC_P_HA Transaction Cost for 8 Input Gates*

Fig. 8: *Transaction Cost for Party Registration*

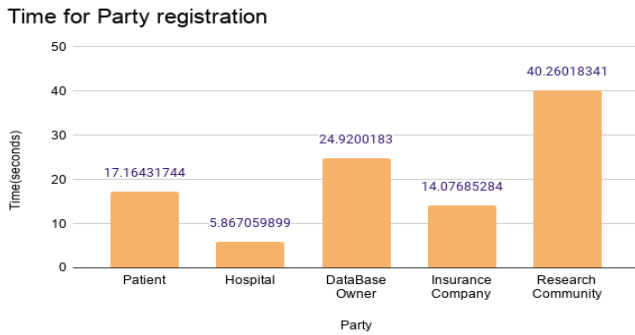Fig. 12: *SC_P_HA Transaction Cost for 16 Input Gates*
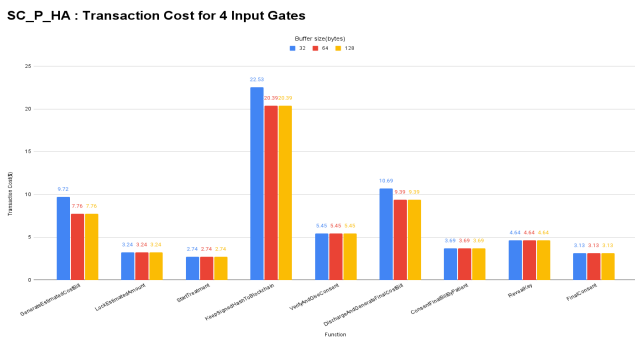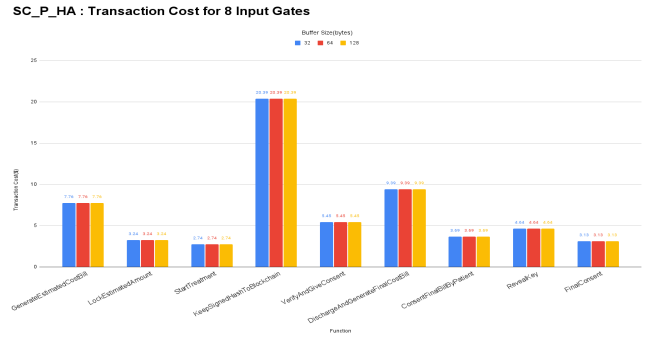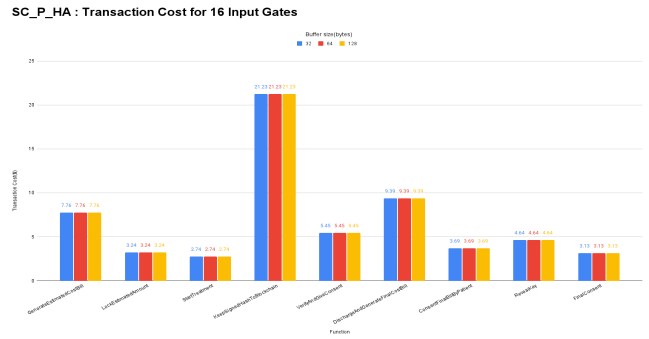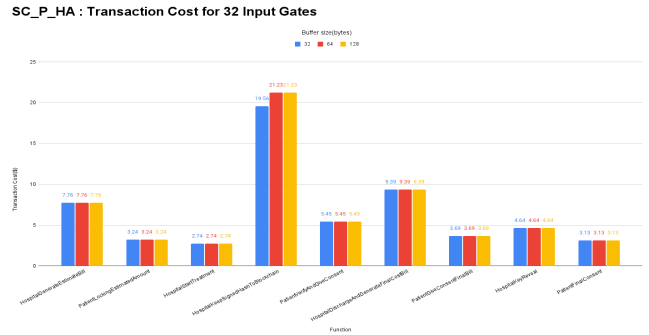

Fig. 9: *Time for Party Registration*

Fig. 13: *SC_P_HA Transaction Cost for 32 Input Gates*

Fig. 10: *SC_P_HA Transaction Cost for 4 Input Gates*

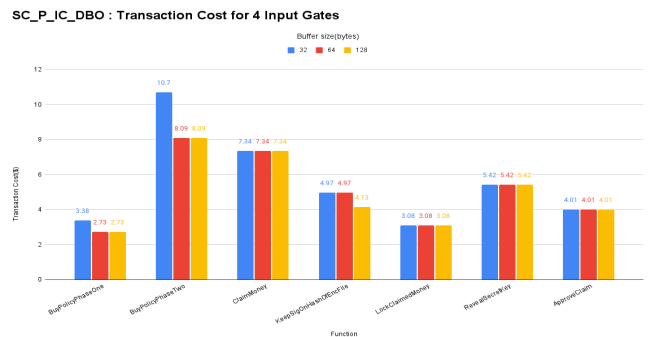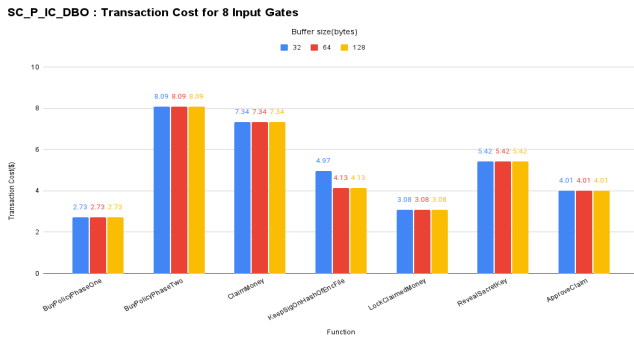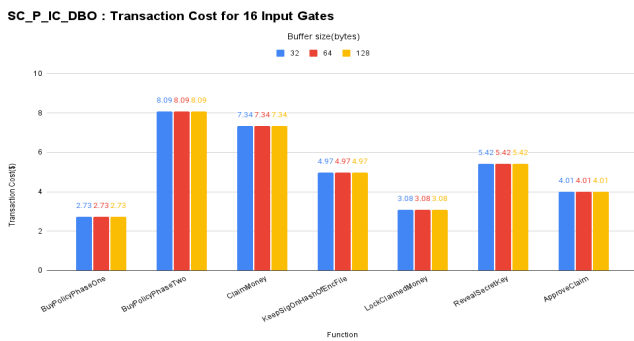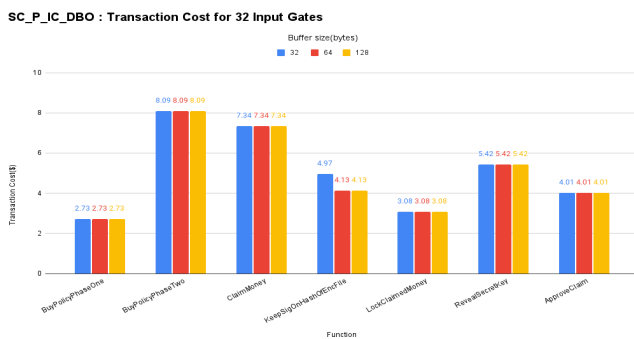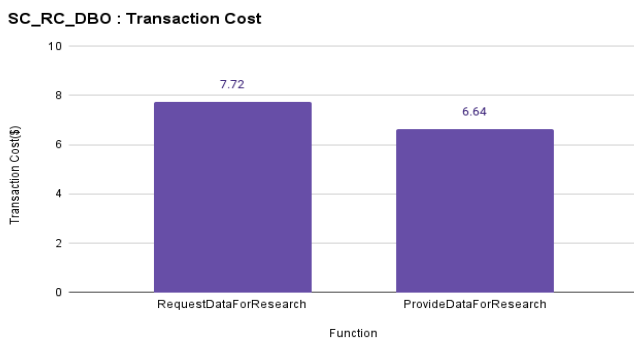Fig. 14: *SC_P_IC_DBO Transaction Cost for 4 Input Gates*

Fig. 15: *SC_P_IC_DBO Transaction Cost for 8 Input Gates*

Fig. 16: *SC_P_IC_DBO Transaction Cost for 16 Input Gates*

Fig. 17: *SC_P_IC_DBO Transaction Cost for 32 Input Gates*

Fig. 18: *SC_RC_DBO Transaction Cost*

Figure 9 respectively, are similar in scale for different parties except for the patients which are slightly higher due to the few more variables involved in the registration for patients.

The transaction cost for many of the functions involved in the protocol depends on the size of the files. The file size may vary depending upon the treatment and the corresponding result produced. The base file is constructed indifferently for subsequent usage in the protocol. The file is divided into numbers of chunks. The number and size of the chunks are the varying parameters, which are referred to as the number of input gates and the buffer size of the gate respectively in the paper.

We have shown the cost associated with various functions call for the contracts SC_P_HA (Figure 10, 11, 12, 13) and SC_P_IC_DBO (Figure 14, 15, 16, 17) with varying number of input gates and the buffer size. The number of input gates is varied in the range of 4, 8, 16, 32 and the buffer sizes used are 32, 64, and 128. The file size can be derived by multiplying the number of input gates and the buffer size. So, a file having 4 input gates and a buffer size of 32 would have a file size of 128 bytes.

We find from the graphs of SC_P_HA (Figure 10, 11, 13) that they follow a similar kind of trend with varying numbers of input gates. While keeping the number of input gates constant and varying the buffer size, the transaction cost for different functions hardly varies. Also, this holds true for the graphs SC_P_IC_DBO (Figure 14, 15, 16, 17) as well. The cost associated with the functions of SC_RC_DBO is shown in Figure 18. As shown in the graphs, some functions require a higher cost. This is due to their heavy functionalities. It is viable as the utilities of the functions outweigh the cost.

7 CONCLUSION

The use of blockchain in healthcare systems plays an important role in the healthcare domain. It can result in automated data collection and verification processes, enables the collection of correct and aggregated data from various sources that are immutable, tamper-resistant, and provide secured data, with a reduced probability of cybercrime [28]. It also supports distributed data, with redundancy and fault tolerance of the system. Being a distributed system, it also eliminates the role of a trusted third party. In this paper, we have depicted a novel *Secure and Smart Healthcare System* where every involved party's fairness is preserved without trusting each other. Electronic health records (EHR) are tamper-proof and free from any unauthorized access in our healthcare system enabled by Blockchain Technology. This shows the blockchain capability and importance in various areas and proves that it could be the next revolutionary technology for replacing current healthcare systems.

Medical data is of utmost importance for research purposes. Nowadays, it is challenging to get fresh unaltered medical data from the source, which necessarily leads to poor research results while applying various data mining techniques. Cleaning raw data is one of the hardest jobs. We have also tried to focus on this area so that Research Community can get the medical data with a certain level of confidence about data integrity. We have proposed, prototyped, and deployed our Healthcare system, which works

fine in the private network as well as in the Ropsten test network. We have built the system in the permissionless setting of the blockchain environment. We have ensured the patient's privacy does not get compromised. Experimental result shows the satisfactory outcome of various performance metrics. Further, we can improve the performance of this model by making further optimizations that are left as a part of our future work.

REFERENCES

[1] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with iot. challenges and opportunities," *Future generation computer systems*, vol. 88, pp. 173–190, 2018.

[2] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for iot security and privacy: The case study of a smart home," in *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*, pp. 618–623, IEEE, 2017.

[3] S. Nakamoto, "Re: Bitcoin p2p e-cash paper," *The Cryptography Mailing List*, 2008.

[4] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE international congress on big data (BigData congress)*, pp. 557–564, IEEE, 2017.

[5] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[6] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[7] Q. Xia, E. Sifah, K. Asamoah, J. Gao, X. Du, and M. Guizani, "Medshare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. PP, pp. 1–1, 07 2017.

[8] M. A. Uddin, A. Stranieri, I. Gondal, and V. Balasubramanian, "Continuous patient monitoring with a patient centric agent: A block architecture," *IEEE Access*, vol. 6, pp. 32700–32726, 2018.

[9] P. Zhang, J. White, D. Schmidt, G. Lenz, and S. Rosenbloom, "Fhirchain: Applying blockchain to securely and scalably share clinical data," *Computational and Structural Biotechnology Journal*, vol. 16, 07 2018.

[10] K. Griggs, O. Ossipova, C. Kohlios, A. Baccarini, E. Howson, and T. Hayajneh, "Healthcare blockchain system using smart contracts for secure automated remote patient monitoring," *Journal of Medical Systems*, vol. 42, 07 2018.

[11] H. Li, L. Zhu, M. Shen, F. Gao, X. Tao, and S. Liu, "Blockchain-based data preservation system for medical data," *Journal of Medical Systems*, vol. 42, 08 2018.

[12] J. Dias, L. Reis, H. Ferreira, and A. Martins, "Blockchain for access control in e-health scenarios," *arXiv preprint arXiv:1805.12267*, 05 2018.

[13] X. Liang, S. Shetty, J. Zhao, D. Bowden, D. Li, and J. Liu, "Towards decentralized accountability and self-sovereignty in healthcare systems," in *International conference on information and communications security*, pp. 387–398, Springer, 2017.

[14] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, "Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control," *Journal of medical systems*, vol. 40, no. 10, pp. 1–8, 2016.

[15] J. Zhang, N. Xue, and X. Huang, "A secure system for pervasive social network-based healthcare," *Ieee Access*, vol. 4, pp. 9239–9250, 2016.

[16] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li, "Integrating blockchain for data sharing and collaboration in mobile healthcare applications," in *2017 IEEE 28th annual international symposium on personal, indoor, and mobile radio communications (PIMRC)*, pp. 1–5, IEEE, 2017.

[17] S. Jiang, J. Cao, H. Wu, Y. Yang, M. Ma, and J. He, "Blochie: a blockchain-based platform for healthcare information exchange," in *2018 ieee international conference on smart computing (smartcomp)*, pp. 49–56, IEEE, 2018.

[18] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, "Medblock: Efficient and secure medical data sharing via blockchain," *Journal of medical systems*, vol. 42, no. 8, pp. 1–11, 2018.

[19] H. Wang and Y. Song, "Secure cloud-based ehr system using attribute-based cryptosystem and blockchain," *Journal of medical systems*, vol. 42, no. 8, pp. 1–9, 2018.

[20] R. Guo, H. Shi, Q. Zhao, and D. Zheng, "Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems," *IEEE access*, vol. 6, pp. 11676–11686, 2018.

[21] Y. Sun, R. Zhang, X. Wang, K. Gao, and L. Liu, "A decentralizing attribute-based signature for healthcare blockchain," in *2018 27th International conference on computer communication and networks (ICCCN)*, pp. 1–9, IEEE, 2018.

[22] X. Zhang and S. Poslad, "Blockchain support for flexible queries with granular access control to electronic medical records (emr)," in *2018 IEEE International conference on communications (ICC)*, pp. 1–6, IEEE, 2018.

[23] G. Yang and C. Li, "A design of blockchain-based architecture for the security of electronic health record (ehr) systems," in *2018 IEEE International conference on cloud computing technology and science (CloudCom)*, pp. 261–265, IEEE, 2018.

[24] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric)," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 253–255, IEEE, 2017.

[25] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20 000 transactions per second," *International Journal of Network Management*, vol. 30, no. 5, p. e2099, 2020.

[26] S. Tanwar, K. Parekh, and R. Evans, "Blockchain-based electronic healthcare record system for healthcare 4.0 applications," *Journal of Information Security and Applications*, vol. 50, p. 102407, 2020.

[27] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 967–984, 2018.

[28] S. Tanwar, K. Parekh, and R. Evans, "Blockchain-based electronic healthcare record system for healthcare 4.0 applications," *Journal of Information Security and Applications*, vol. 50, p. 102407, 2020.

APPENDIX A
ACTIVITY SEQUENCE DIAGRAM

let us take a close look at the activity/functional sequence diagrams that provide a transparent view of our system model. These schematic diagrams are the implementation view of the same system model, which was discussed in the earlier Section 3 as a high-level view. It specifies which entity will invoke which function of the smart contracts along the timeline.
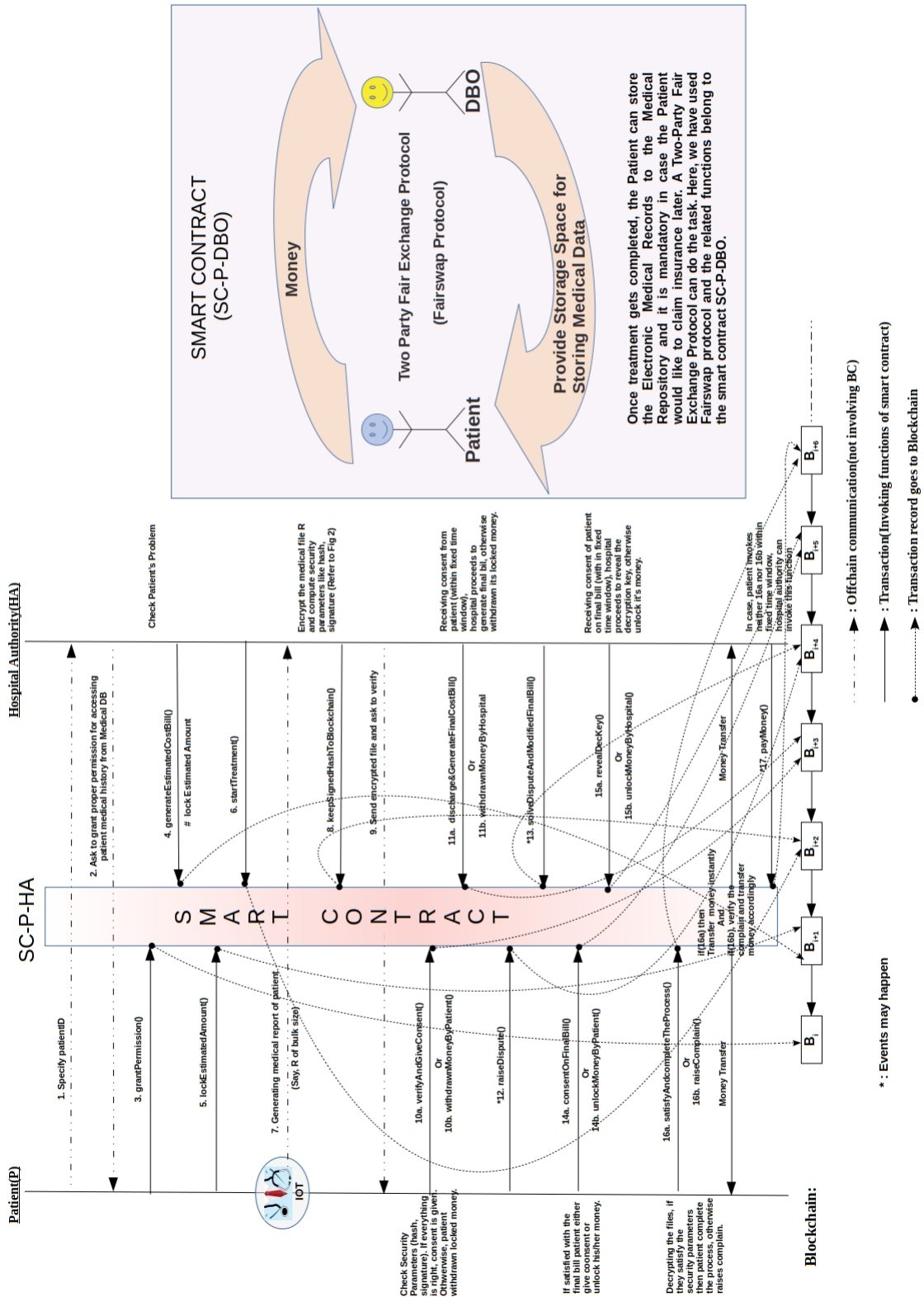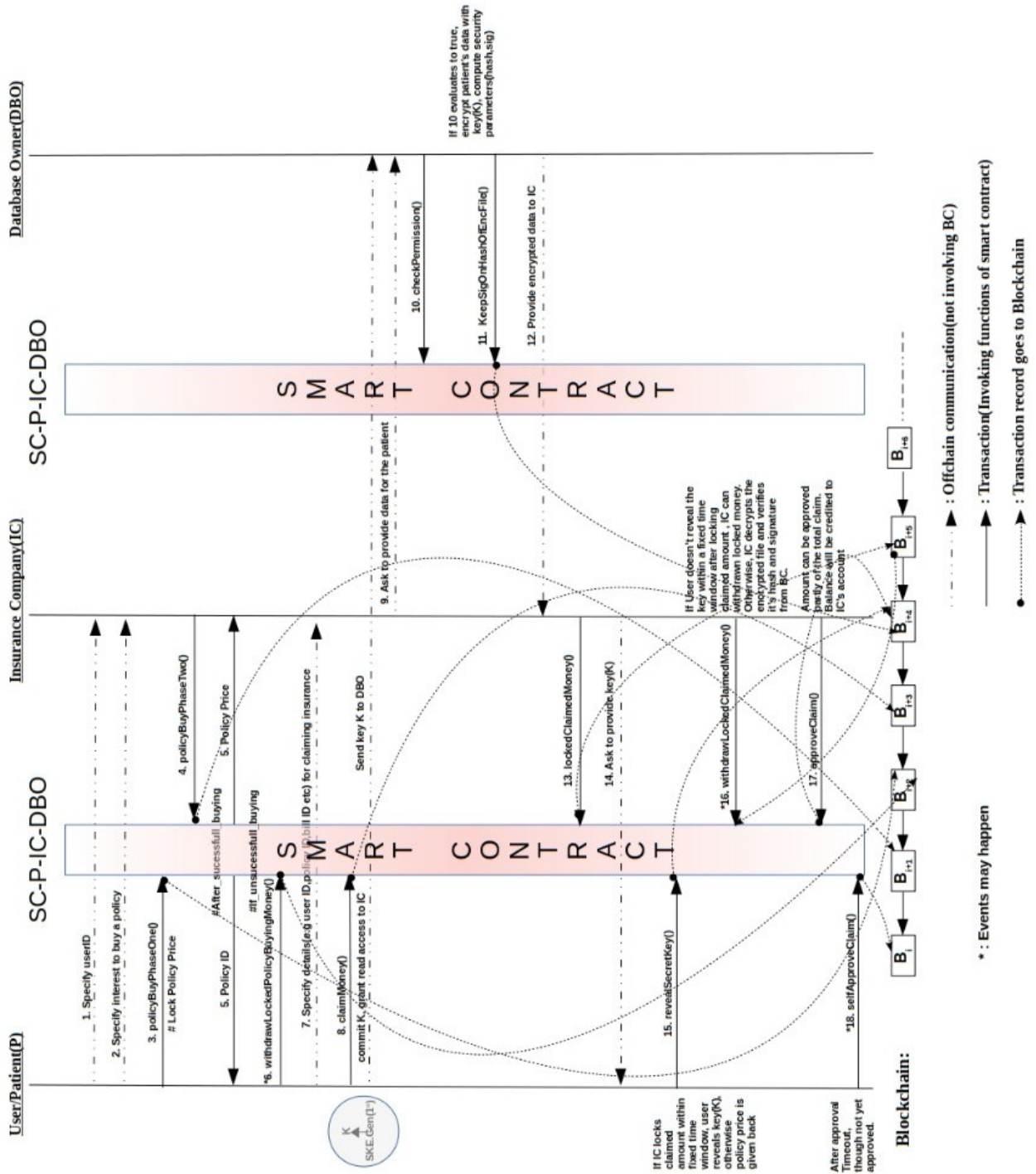

Fig. 19: Activity Sequence Diagram between Patient Hospital Authority and Database Owner

Fig. 20: Activity Sequence Diagram between User, Insurance Company and Database Owner
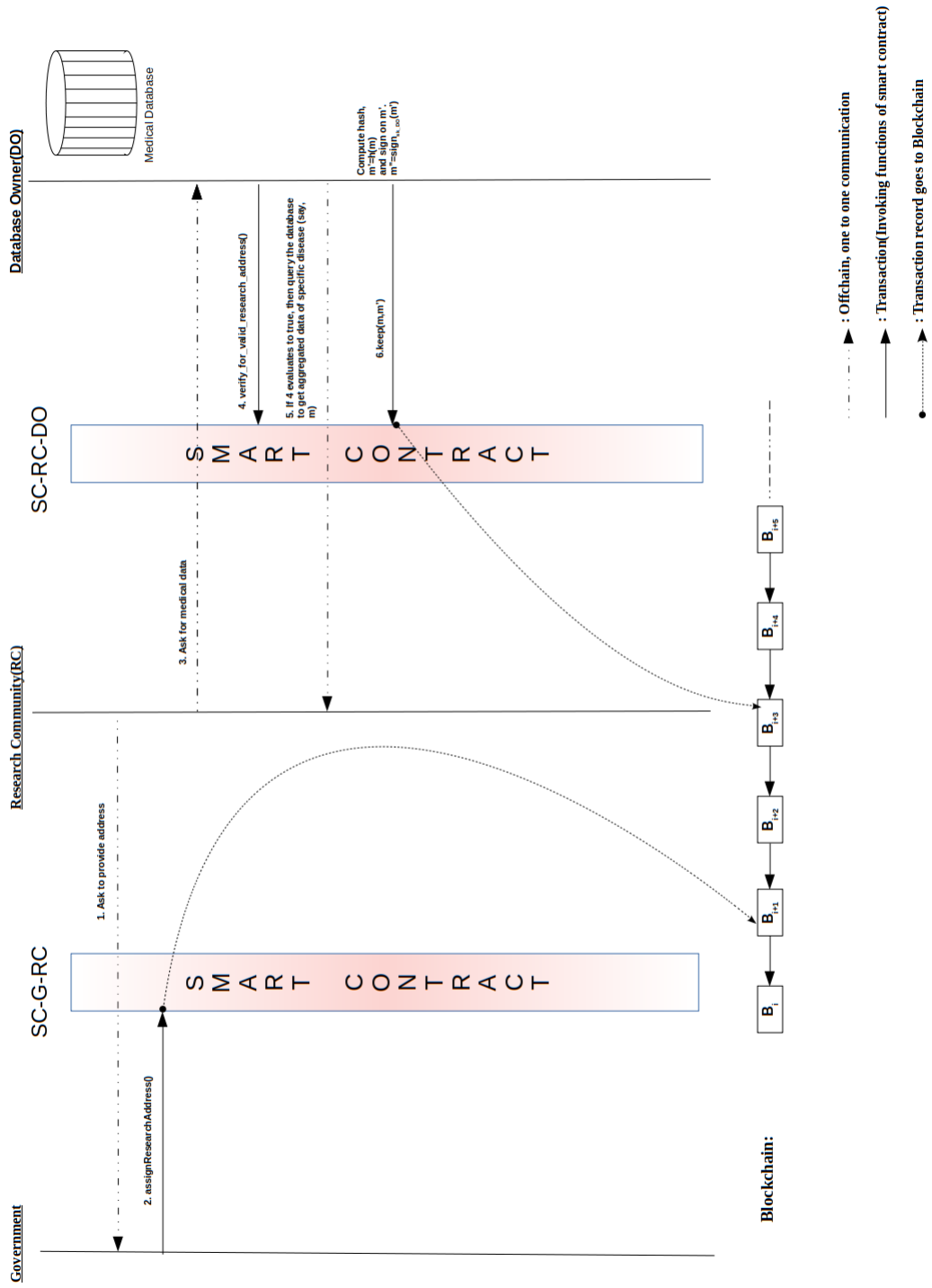
Fig. 21: Activity Sequence Diagram between Research Community and Database Owner

APPENDIX B
DATA STRUCTURES AND PROGRAMMING CONSTRUCTS

Besides the primitives data types available in Solidity language (e.g., uint, address, bytes, bytes32, string, bool, etc.), we have also defined user-defined data types. These are mentioned below.

　　1. **Structure:** To understand the algorithms clearly, we have to perceive the following major structure definitions that are used in our system implementation.

(I)

```
struct Patient
{
    address pAddr; //Patient's Address
    uint patientID; //Patient ID
    bytes32 hashOfPatientDetails
}
```

(II)

```
struct Hospital
{
    address hAddr; //Hospital's Address
    uint hospitalID; //Hospital ID
    string name;
}
```

(III)

```
struct InsuranceCo
{
    address icAddr; //Insurance Company's Address
    uint insuracnceID; //Insurance Company ID
    string name;
}
```

(IV)

```
struct ResearchCommunity
{
    address rcAddr; //Research Community's Address
    uint rcID; //Research Community's ID
    string name;
}
```

(V)

```
struct DataBaseOwner
{
    address dboAddr; //Database Owner's Address
    uint dboID; //Database Owner's ID
    string name;
}
```

Fig. 22: Structure of Different Actors in the System(defined in SC_Registration)

(I)

```
struct MultiSigOnMedicalData
{
    uint multiSigID;
    uint patientID;
    uint hospitalID;
    bytes32 hashOfData;
    bytes32 hashOfEncryptedData;
    bytes32 hashOfPatientIdDateHashEncFile;
    bytes signOnFileHashByHospital;
    bytes signOnHashOf_PatientId_DateOfReport_HashOfEncryptedFile;
    uint timestampOfSignByHospital;
    uint timestampOfVerificationByPatient;
    bool verified;
    uint timestampOfUnlockingByPatient_multi;
    uint timestampOfUnlockingByHospital_multi;
}
```

(II)

```
struct FileProperties
{
    uint fileSize; //size of the file..
    uint fileChunkSize; //size of each file chunk..
    uint depth; //Merkle tree constuction. Root at depth 0
}
```

(III)

```
struct KeyAndExchange
{
    bytes32 key;
    bytes32 keyHash;
    uint timestampOfKeyReveal;
}
```

(IV)

```
struct EstimatedCheckUpCost
{
    uint estimatedCostBillID;
    uint pID;
    uint hID;
    uint estimatedCost;
    uint timestampOfEstimate;
    uint timestampOfLockingByHospital;
    uint timestampOfLockingByPatient;
    uint timestampOfCheckUpStart;
    uint timestampOfUnlockingByPatient;
    uint timestampOfUnlockingByHospital;
}
```

(V)

```
struct FinalCheckUpCost
{
    uint finalCostBillID;
    uint pID;
    uint hID;
    uint finalCost;
    uint timestampOfFinalBilling;
    uint excessOrDeficitID;
    uint timestampOfPatientFinalBillConsent;
    uint timestampOfComplain;
    uint timestampOfUnlockingByHospital_final;
    uint timestampOfUnlockingByPatient_final;
}
```

(VI)

```
struct ExcessOrDeficit
{
    uint edID;
    uint pID;
    uint hID;
    uint deficitAmount;
    uint timestampOfLockingDeficitByHospital;
    uint timestampOfLockingDeficitByPatient;
    uint excessAmount;
    uint timestampOfRefundingExcessToPatient;
}
```

(VII)

```
struct Dispute
{
    uint disputeID;
    uint pID;
    uint hID;
    uint finalBillID;
    uint timestampOfRaisingDispute;
    uint timestampOfSolvingDispute;
    bool finalCostChanged;
}
```

Fig. 23: Important Structures used in the Smart Contract SC_P_HA

25

```
(I)

struct PolicyBuyPhaseOne
{
    uint p_price; //Price locked by the policy buyer
    uint time;
    bytes32 hash_of_file; //Hash of Terms & Condition File
}

(II)

struct PolicyDetails
{
    uint poID; //Policy ID
    address buyerAddr; //Policy Buyer's Address
    address icAddr; //Insuarance Company's Address
    uint timestamp_policyBuying;
    uint policyPrice;
    bytes32 hashOfTerms&Cons //Hash of Terms & Condition File
    uint[] claimIDs;
}
```

```
(III)

struct ClaimDetails
{
    uint cID; //claim ID
    uint estimatedBillID;
    uint timestamp_patientGenerateClaim;
    uint claimedAmount;
    uint approvedAmount;
    bytes publicKey;
    bytes32 commitmentOfSecretKey;
    bytes secretKey;
    uint timestamp_keyReveal;
    uint timestamp_icLocking;
    uint timestamp_icUnlocking;
    uint timestamp_approval;
    bool isSelfApproved;
    bool isICMalicious;
}
```

```
(IV)

struct DBODetails
{
    address dboAddress;
    bytes signature_DBO_hashOfEncFile;
    uint timeStamp_dboGivingEncFile;
}

(V)

struct ICReputation
{
    uint noOfCustomers;
    uint noOfMaliciousTagReceived;
}
```

Fig. 24: Important Structures used in the Smart Contract SC_P_IC_DBO

```
(I)

struct ApplicationForStoring
{
    uint appID; //application ID..
    address patientAddress;
    address dboAddress;
    uint multiSigID;
    bytes32 key;
    bytes32 merkleRootOfFile;
    bytes32 merkleRootOfEncFile;
    uint timestamp_application;
    uint timestamp_merkleRootVerification;
    uint timestamp_keyReveal;
    uint timestamp_complainOrApproval;
    uint timestamp_Unlocking_Patient;
    uint timeStamp_Unlocking_DBO;
}
```

Fig. 25: Important Structures used in the Smart Contract SC_P_DBO

```
(I)

struct ResearchDataDetails
{
    uint researchDataID;
    address rcAddr;
    address dataBaseOwner;
    uint timeStamp_rcRequestData;
    uint timeStamp_dboSendData;
    bytes32 hashOfMedData;
    bytes sigOnHashOfMedData;
}
```

Fig. 26: Important Structures used in the Smart Contract SC_RC_DBO

2. **Array:**

No	Array Name	Datatype	Description
1	patient[]	struct Patient	When a new patient registers in the system, we make an entry into this array. The variable **PatientIDGenerator** keeps the count of the total no of registered patients/users in the system (or alternatively it specifies the length of the array).
2	hospital[]	struct Hospital	When a new hospital authority wants to register its name in the system, we make an entry into this array. The variable **HospitalID-Generator** keeps the count of the total no of the registered hospitals in the system.
3	insuranceCo[]	struct InsuranceCo	When a new insurance company wants to register its name in the system, we make an entry into this array. The variable **insurance-CoIDGenerator** keeps the count of the total no of the registered insurance companies in the system.
4	researchCommunity[]	struct ResearchCommunity	When a new research Community gets registered in the system by the Government, we make an entry into this array. The variable **rcIDGenerator** keeps the count of the total no of registered research communities in the system.
5	dataBaseOwner[]	struct DataBaseOwner	When a new database owner gets registered in the system by the Government, we make an entry into this array. The variable **dboID-Generator** keeps the count of total no of registered DBO in the system.
6	estimatedCheckUpCost[]	struct EstimatedCheckUpCost	When a patient comes to a hospital for treatment, hospital generates the estimated cost bill. We make an entry of the same into this array. The variable **estimatedBillIDGenerator** keeps the count of total no of estimated bill generated in the system.
7	finalCheckUpCost[]	struct FinalCheckUpCost	When a patient is about to be discharged, the hospital generates the final cost bill. We make an entry of the same into this array. The variable **finalBillIDGenerator** keeps the count of total no of the final bill generated in the system.
8	excessOrDeficit[]	struct ExcessOrDeficit	When the final cost of treatment varies from the initial estimated cost, the hospital needs to instantiate a structure of ExcessOrDeficit. We make an entry of the same into this array. The variable **excessOrD-eficitIDGenerator** keeps the count of total no of such instances in the system.
9	multiSigOnMedicalData[]	struct MultiSigOnMedicalData	When a hospital is going to transfer the encrypted medical file to the patient, it instantiates a structure of MultiSigOnMedicalData. The security-related aspects (e.g., hash, signature, timestamp, etc.) are taken care of by this structure. We make an entry of the same into this array. The variable **multiSigIDGenerator** keeps the count of the total no of such instances created in the system.
10	applicationForStoring[]	struct ApplicationForStoring	When a patient wants to store his/her medical files in the medical repository,(s)he applies to the DBO for the same. An entry is made in this array. A unique appID is assigned to each of these entries. The variable **applicationIDGenerator** keeps the count of the total no of such applications created in the system.
11	policyDetails[]	struct PolicyDetails	When a user buys a policy from an insurance company, an entry is made in this array. The variable **policyIDGenerator** helps to assign a unique policy ID against each such entry.
12	claimDetails[]	struct ClaimDetails	When a user claims for reimbursement of his/her medical bills against his policy, an entry is made in this array. The variable **claimIDGenerator** helps to assign a unique claim ID against each such entry.

TABLE 7: Major Arrays used in the Smart Contracts

3. **Mapping:**

No	Mapping Name	Description
1	mapFromAddrToID_* (* : P/H/IC/DBO/RC)	Associating entity's address (Patient/ Hospital/ Insurance Company/ Database Owner/ Research community) to their corresponding ID.
2	read_permission_to_* (* : H/IC)	Associating read access permission to the third party address (Hospital/ Insurance Company).
3	patientID_hospitalID_estimatedBillID	Coupling (patientID, hospitalID) to estimatedBillID.
4	estimatedCostBillIDToMultiSigOnMedicalDataID	Mapping from Estimated Bill ID to corresponding Multi-Sig Data ID.
5	estimatedBillIDToFinalBillID	Mapping from Estimated Cost Bill ID to corresponding Final Cost Bill ID.
6	MultiSignIDToEncFileProperties	Refering to the Encrypted File properties(i.e struct FileProperty) from Multi-Sig Data ID.
7	MultiSignIDToKeyAndExchange	Refering to the Encryption key properties(i.e struct KeyAndExchange) from Multi-Sig Data ID.
8	moneyLocked	Coupling (buyer Address, IC Address) to struct policyBuyPhaseOne.
9	reputationOfIC	Associating Insurance Company address to its reputation(i.e struct ICReputation).
10	policyID_map	Coupling (Policy buyer Address, Insurance Company address) to the policyID.
11	securityMoney_IC	Given an address of an IC, it refers to the amount of security money being locked in the system by the IC.
12	deRegister_IC	Given an address of an IC, it specifies whether the IC has been de-registered or not.
13	cIDTopoID	Mapping from Claim ID to Policy ID.
14	claimIDToDBODetails	Refering to the struct DBODetails from Claim ID.

TABLE 8: Major Mappings used in the Smart Contracts