MyEasyHeathcare: An efficient and secure three-tier blockchain-based healthcare system

Kanika Agrawal[1], Mayank Aggarwal[1], and Sudeep Tanwar[2]

[1]Gurukul Kangri (Deemed to be University)
[2]Nirma University of Science and Technology Institute of Technology

January 30, 2024

Abstract

Blockchain is emerging as a solution to secure healthcare records but faces certain shortcomings like transaction time, execution time, gas consumption, etc. The current article designed an extensive blockchain-based healthcare system (MyEasyHeathcare) with reduced gas consumption and execution time, along with enhanced security at three levels. At the first level, the professionals and patients get registered, which provides identity access management. Secondly, authorization is required for each registered entity by the owners. Lastly, the third level includes a doctor-patient relationship where a specific patient is assigned to a particular doctor by the hospital's owner. The data is protected from the outer world and is preserved only between the doctor and the patient. Moreover, to include the majority of tasks for hospital management, the developed system incorporates a smart contract to record seven different parameters for patient diagnosis by physician and fifteen different parameters by a pathologist. The designed system is evaluated for the amount of gas consumed and execution cost to decide the usage in the real world; the results testify the proposed system is useful in the real world.

MyEasyHeathcare: An efficient and secure three-tier blockchain-based healthcare system

Kanika Agrawal, Mayank Aggarwal, Sudeep Tanwar

Department of Computer Science & Engineering, Faculty of Engineering & Technology, Gurukula Kangri (Deemed to be University), Haridwar, India

Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad, Gujarat-382481, India

Abstract

Blockchain systems have seen vast growth due to the immense potential in developing secure applications for education, healthcare, etc. The healthcare system is extensively researched to provide convenience to human life. With the exponential growth in healthcare systems and devices, patient data security and privacy issues are becoming primary concerns. Blockchain is emerging as a solution to secure healthcare records but faces certain shortcomings like transaction time, execution time, gas consumption, etc. The current article designed an extensive blockchain-based healthcare system (*MyEasyHeathcare*) with reduced gas consumption and execution time, along with enhanced security at three levels. At the first level, the professionals and patients get registered, which provides identity access management. Secondly, authorization is required for each registered entity by the owners. Lastly, the third level includes a doctor-patient relationship where a specific patient is assigned to a particular doctor by the hospital's owner. The data is protected from the outer world and is preserved only between the doctor and the patient. Moreover, to include the majority of tasks for hospital management, the developed system incorporates a smart contract to record seven different parameters for patient diagnosis by physician and fifteen different parameters by a pathologist. The designed system is evaluated for the amount of gas consumed and execution cost to decide the usage in the real world; the results testify the proposed system is useful in the real world.

Keywords: Blockchain, Ethereum, Healthcare, Solidity, Treatment.

1. Introduction

With increased internet use and technological advancements, online storage systems and communication are becoming the topmost priority for different users. As data flow is continuous, the threats to privacy and security have tremendously increased [1]. Hence, it is required to actively consider security concerns along with data heterogeneity, integrity, and redundancy. Security and privacy concerns are important to be taken care of in any organization [2] [3]. Unauthorized data breaching and information leakage lead to the wastage of resources and financial losses. Without a secure model, the system faces many attacks, such as Distributed Denial of Service (DDoS), data rate alteration, manipulation, network congestion, Address Resolution Protocol(ARP) spoofing attacks, config threats, phishing, and noise interference. To mitigate these threats, many centralized-based architectures were developed. But, the problem faced by these architectures was that the central server failed to survive cyber attacks and crashed down. To secure the systems, many security mechanisms such as Advanced Encryption Standard (AES) and Data Encryption Standard (DES) were also used, but they were facing high communication overhead. The revolution came up with the introduction of the blockchain (BC) concept and bitcoin [4]. Unlike the usual existing models, the BC technique helps in digital assets peer-to-peer (P2P) transference without any support of an intermediate party [5]. For the first time,

Email addresses: `ka841995@gmail.com` (Kanika Agrawal), `mayank@gkv.ac.in` (Mayank Aggarwal), `sudeep.tanwar@nirmauni.ac.in` (Sudeep Tanwar)

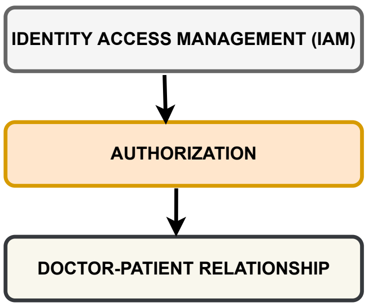

Figure 1: Conceptual view of three-level security of the proposed system

the BC concept was proposed in the year 2008 and was later implemented by Nakamoto in 2009. The decentralized BC is a technology that connects blocks chained together and stores all the committed transactions in a public ledger [6]. Some important BC key characteristics include auditability, transparency, immutability, trustworthiness, and decentralized nature[7]. Because of its safe and secure nature, BC is applied to many applications all around the world, such as the education system, aircraft system, agriculture system, etc.

The healthcare industry is one of the applications that use BC to help save the lives of patients [8]. All this is possible due to the advancement in technologies in the healthcare diagnosis processes opted for by healthcare professionals. The healthcare industry has faced transformations from Healthcare 1.0 to Healthcare 4.0. Initially, healthcare 1.0 was more doctor centric and allowed doctors to maintain manual records of the patient's medical history. But, with time, healthcare 2.0 was developed, replacing these manual records with electronic ones. However, with an improved version of Healthcare 3.0, real-time tracking of patient healthcare history was done using wearable devices (WDs) [9]. Later, an Electronic Health Record (EHR) system came into existence that helped store the patient records in the database repository electronically.

To maintain patient data security and privacy, healthcare 4.0 is developed to store the patient's record in the centralized EHR system and deliver uninterrupted services in real-time [10]. Patients' health is monitored through implantable medical devices (MDs) and WDs. Healthcare sensors are fixed in WDs [11] that help to measure the patient's blood pressure, heart rate, glucose level, and temperature remotely and help store them in a centralized EHR called telehealthcare. Internet of things(IoT) with telehealthcare can be used to coordinate and cooperate the disease management [12]. Healthcare IoT has a significant influence on the healthcare industry. It generates a huge amount of data at regular intervals that helps attackers launch various security attacks, such as data confidentiality, integrity attacks, and privacy. The traditional storage systems were incapable of handling such ever-evolving real-time data. So this leads to the development of cloud technology to store and process massive amounts of data safely and securely. Moreover, to improve the national healthcare system and to secure health information, the healthcare industries have defined the health information technology for economic and clinical health act (HITECH) and health insurance and portability and accountability act (HIPAA). Motivated by these advancements, this paper designs a secure three-level healthcare system involving various stakeholders to help maintain, secure and preserve healthcare data from attackers and intruders. Figure 1 shows the proposed three-level security healthcare system. The work improves efficiency by writing effective smart contracts which reduce gas consumption, and an additional level of security is provided by adding three-tier security to the designed system. The identity access management level helps in the secure registration of the patient and the doctor by the owner of the hospital. Authorization helps authorize a specific patient to a specific doctor at the hospital and helps secure the doctor-patient relationship. All three-level security is provided using different features of the smart contract.

1.1. Comparison of existing healthcare architecture with the proposed architecture

In Healthcare 4.0, the concept of Remote Patient Monitoring (RPM) has become a flexible and powerful tool for patient observation using wearable sensors. The use of a wireless communication system, the system allows doctors to get real-time information about their patients remotely. It provides quality care to the patients. Motivated by these facts, this article proposes a BC-based healthcare architecture that uses smart contracts to manage and secure patients' information and privacy. Table 1 compares existing healthcare architecture with the proposed architecture. In detail, the system is deployed on the Ethereum platform and Remix IDE, which includes healthcare owners and providers (such as a hospital), professional doctors (such as physicians and pathologists), and patients. Doctors measure various health parameters and conditions of patients, and such information is taken as input into the developed blockchain system manually. For a specific patient, a specific doctor is authorized for the treatment. The system allows specific conditions and parameters to be checked by specialized doctors for their patients. The data is protected from the outer world and is preserved only between the doctor and the patient. Moreover, several features for patient diagnoses are implemented by physicians and pathologists.

To enhance the privacy and security of the patient data, the authors in [13] and [14] presented BC-based permission healthcare architecture and smart contracts to manage patients' details and medical devices, respectively. They also highlighted the usage of Machine Learning (ML) with BC technology to enhance security in the healthcare industry. However, the authors lack security at the owner level and the details of algorithms. The evaluation of patients based on different types of medical features and gas cost evaluation is also not present. Thus this work eliminates all such problems and shows a three-tier secure healthcare system that helps in providing security features like IAM, Authorization, and a secure doctor-patient relationship. Also, the gas cost is reduced for each function in a smart contract which any researcher does not yet show to date.

The authors of [15] discussed a BC-based system that integrated testing and vaccination system to be transparent. An efficient vaccination system was implemented, termed the "Digital Vaccine Passport" (DVP) system, yet the article lacks security at the doctor-patient level, a reading map of the work, and an in-depth analysis of the patients based on medical features. Article [16] integrated IoT with BC and proposed a secure healthcare system called BlockMedCare. The system supported RPM for regular monitoring of chronic diseases. However, the paper lacks many medical features and applications for patient analysis. The authors in [17] proposed a reliable and intelligent veterinary information management system that analyzed veterinary clinic patient's appointment data and discovered the underlying patterns using ML algorithms. Still, the work does not evaluate the gas cost consumption, which is solved in this work. The authors explored several solutions for improving current limitations in the healthcare system and proposed an Access Control Policy Algorithm using the chaincode concept in [18]. This was done to enhance healthcare services and to improve the data accessibility between healthcare providers using BC.

The authors in [19] applied Quantum Computing (QC) to the traditional encryption system. They proposed a BC-based architecture for Healthcare that helped users to access the data from the database with a defined role. However, the work lacks certain features for a safe patient-doctor relationship and gas cost evaluation. The reading map and diagnoses of the patient based on other different medical treatment features were also lacking. This research thus provides a safe solution for healthcare systems. It ensures secure IAM, authorization, and doctor-patient relationships. It also evaluates the performance based on gas cost consumption. To improve the BC system's performance and higher query efficiency, the authors in [20] extended the Blockchain Cryptographic Service Provider (BSP) module in the Hyperledger Fabric. Moreover, the authors also designed a transaction process using Upgraded BCCSP (UBCCSP). However, the paper lacks various other medical features to be dealt with for patient checkups resolved in this paper.

1.2. Motivation

In 2008, different business firms faced an economic crisis due to stakeholders' distrust and negligence. This led to an urge to keep the businesses back on track. Thus, this change resulted in the evolution of BC and bitcoin. Since then, till 2018, BC was started being used for cryptocurrency. In 2018, BC was applied in different applications that included healthcare systems. Many kinds of research were carried out on BC and its applications in healthcare systems, but they have yet to propose a secure, efficient, and effective healthcare system. The current research focused only on the doctor and patient relationship along with fewer number features provided to patients. Thus there was a need to get the desired solution for patients and doctors where a particular patient could be assigned to a particular specialized doctor. Also, without consent, one doctor cannot access the other doctor's patient details. If required, a patient can also request the hospital for other doctor consultations. Inspired by these facts, this paper deploys a secure healthcare system on Remix IDE using solidity language. Here, the hospital assigns a particular patient to a particular doctor along with fifteen different features (seven for physician and eight for pathologist) provided to the doctor that can be checked on the patient.

Table 1: Comparison of existing healthcare architecture with the proposed architecture.

Ref.	Year	Objective	Key Contributions	1	2	3	4	5	6	7	Limitations and open issues
[13]	2019	To enhance the security and privacy of the patient data.	The authors presented a Permissioned blockchain-based healthcare architecture. They also gave the usage of Machine learning with BC technology in the healthcare industry.	X	X	X	X	X	X	X	Lacks the security at doctor-patient level and algorithm details along with performance evaluation.
[14]	2022	To improve the efficiency and management of the testing and vaccination process.	The authors have presented a BC-based system that seamlessly integrates testing and vaccination system to be transparent.	✓	✓	X	✓	X	✓	X	Lacks the reading map of the research and evaluation for more medical features.
[15]	2019	To protect the device-generated and personal information of the patient.	The authors proposed a Remote Healthcare System based on BC smart contracts. Also, a processing mechanism was proposed to store medical device information efficiently.	X	✓	X	✓	X	X	X	Lacks the security at third level of doctor-patient relationship and details of algorithm and reading map of the research.
[16]	2022	To improve the security and privacy of the healthcare systems that are more susceptible to attacks.	The author proposed a secure healthcare system, BlockMedCare, that integrates IoT with BC. The system supported remote patient monitoring, especially for chronic diseases regular monitoring.	✓	✓	X	✓	X	X	X	Lacks the gas cost evaluation along with medical applications and reading map of the paper.
[17]	2021	To increase the healthcare systems' efficiency and reduce the overall overhead of the entire blockchain network.	The author proposed a reliable and intelligent veterinary information management system. Also, the predictive and data modules were built to analyze veterinary clinic patient's appointments data to build a robust prediction model and discover underlying patterns using machine learning algorithms.	X	✓	X	✓	X	X	X	Only useful for veterinary data and lacks the algorithm details.
[18]	2020	To enhance the healthcare services and to improve the data accessibility between healthcare providers using BC.	The authors explored several solutions for improving current limitations in the healthcare system and proposed an Access Control Policy Algorithm using chaincode concept.	✓	✓	X	✓	X	X	X	Lacks speed and various medical applications diagnose for patients evaluation.
[19]	2020	To analyze the security architectures for (EHRs) and to Propose an architecture for the Healthcare system.	The authors applied Quantum Computing (QC) to the traditional encryption system. They proposed a BC-based architecture for Healthcare that helped users to access the data from the database with a defined role. Furthermore, the authors protected the traditional encryption system from quantum attacks using the Quantum blind signature.	✓	✓	X	✓	X	X	X	Lacks the security for safe doctor-patient relationship and diagnoses of the patient based on medical treatment features.
[20]	2022	To improve the BC system's performance and higher query efficiency and solve the Hyperledger Fabric of not supporting Chinese Commercial Cryptographic (CCC) algorithms.	The authors extended the Blockchain Cryptographic Service Provider (BSP) module in the Hyperledger Fabric. Moreover, the authors also designed a transaction process using Upgraded BCCSP (UBCCSP). All of this helped improve the BC system's performance and higher query efficiency.	✓	✓	X	✓	X	X	X	Lacks user's remote anonymous authentication research in depth.
Proposed Work	2022	To propose a secure and effective healthcare system with various medical features for patient treatment.	The authors proposed a secure healthcare system deployed on Remix IDE. The system has physicians and pathologists who check patients based on various medical features. Each patient is authorized to a specific doctor, and none can view patient records except their doctor. Finally, the performance is evaluated based on gas cost and execution gas cost in wei.	✓	✓	✓	✓	✓	✓	✓	-

1: Algorithms used, 2: Authorization, 3: More than twenty (20) Medical Treatment Features/Applications for patients, 4: Performance Analysis, 5: Reading map, 6: Gas cost evaluation, 7: Doctor-Patient relationship (Third level security)

1.3. Research Contribution

This research paper proposes a secure and effective healthcare system that helps patients to build trust in a doctor for their treatment. This research studies existing BC-based healthcare systems and builds a better plan for improving patient-doctor relationships in a hospital. Moreover, this paper also evaluates the performance of the proposed system based on gas cost. The major contributions of this paper are as follows:

- *MyEasyHeathcare* is built and deployed using solidity language on Remix IDE.
- Proposes a trustworthy model that improves the patient-doctor relationship. The designed system does not allow the third party to view the details of their respective patients. Only the assigned doctor can view and check patient details.
- Various features are used to test a patient. Overall, twenty-two different parameters and algorithms are shown

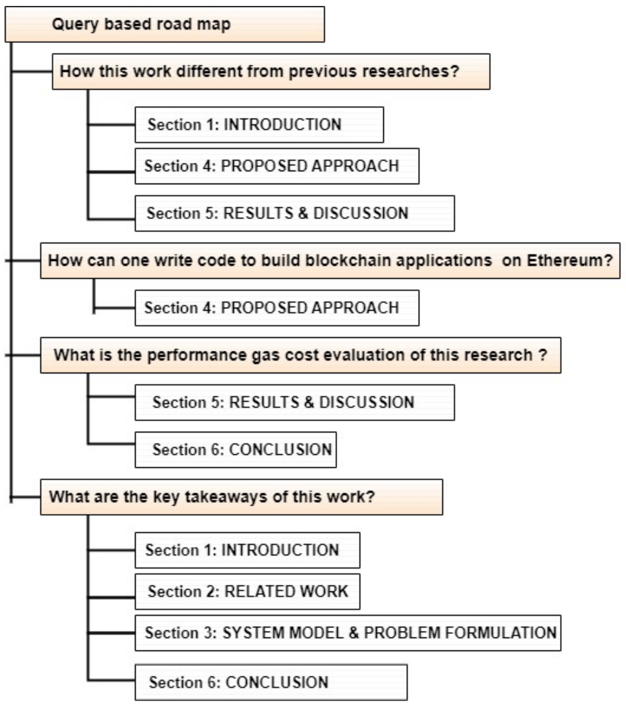
Query based road map
How this work different from previous researches?
Section 1: INTRODUCTION
Section 4: PROPOSED APPROACH
Section 5: RESULTS & DISCUSSION
How can one write code to build blockchain applications on Ethereum?
Section 4: PROPOSED APPROACH
What is the performance gas cost evaluation of this research ?
Section 5: RESULTS & DISCUSSION
Section 6: CONCLUSION
What are the key takeaways of this work?
Section 1: INTRODUCTION
Section 2: RELATED WORK
Section 3: SYSTEM MODEL & PROBLEM FORMULATION
Section 6: CONCLUSION

Figure 2: Reading map of the work

and used for the patient's treatment. Two doctors, such as physicians and pathologists, treat the patient in the deployed model.

- The performance of the proposed system is evaluated based on gas cost and execution cost.

1.4. Organization and Reading map

Table 2 depicts the abbreviations used in this paper. The article proposes a healthcare system deployed on Remix IDE. The paper has a definite structure and is further organized into different sections. Section II deals with the related work and details of existing healthcare systems and BC technology used. Section III gives an overview of BC. Section IV presents the flowchart and provides a detailed description of the proposed healthcare system deployed. Section V evaluates the system's performance and shows its piechart representation based on gas cost and execution gas cost in wei, followed by a conclusion. Figure 2 provides a reading map for the readers to better understand this research paper. Readers interested in understanding this proposed work and its novelty can focus on Sections I, IV, and V. The steps to write a code to build BC applications on Ethereum are detailed in Section IV. The performance evaluation based on gas cost consumption is discussed in Sections V and VI. Finally, Sections I, II, III, and VI show the paper's key takeaways.

2. Related work

This section summarizes the literature on BC technology used in the healthcare system. The summary contains the reputed journals and conferences from the last five years. For example, Abzug *et al.* [16] proposed a secure BC and IoT-integrated healthcare system termed BlockMedCare. Regularly, the system monitored patients and their chronic diseases remotely. The authors ensured processing time, scalability, and security of the system for better results. The security was ensured using the re-encryption proxy, and the system's scalability was established through the use of an InterPlanetary File System (IPFS), which used off-chain databases to store data. The data storage processing speed was increased using Ethereum BC-based Proof of Authority (PoA). The authors did the diabetes management using the system and showed better healthcare system results.

Table 2:
List of abbreviations.

Acronym	Explanation	Acronym	Explanation
ACP	Artificial systems + Computational experiments + Parallel execution	mg/dL	milligrams (mg) per decilitre (dL)
AES	Advanced Encryption Standard	mIU/L	milli-international units per liter
ALT	Alanine Transaminase	ML	Machine Learning
AST	Aspartate aminoTransferase	mmHg	millimetre(s) of mercury
ARP	Address Resolution Protocol	ng/dL	nanogram per deciliter
BC	Blockchain	NMI	Normalized Mutual Information
BCCSP	Blockchain Cryptographic Service Provider	P2P	Peer-to Peer
BMI	Body Mass index	PHSs	Parallel Healthcare Systems
BP	Blood Pressure	PoA	Proof of Authority
CTs	Clinical Trials	PoS	Proof of Stake
dApp	Decentralized Application	PoW	Proof of Work
DDoS	Distributed Denial of Service	QC	Quantum Computing
DES	Data Encryption Standard	R2	R-squared
DVP	Digital Vaccine Passport	RBC	Red Blood Cells
ECC	Elliptic Curve Cryptography	RIVIMS	Reliable and Intelligent Veterinary Information Management System
ECDLP	Elliptic Curve Discrete Logarithm Problem	RMSE	Root-Mean-Square Error
EHR	Electronic Health Record	RPM	Remote Patient Monitoring
FS-ANFIS	Feature Selection based Adaptive Neuro-Fuzzy Inference System	RTT	Round Trip Time
HIPAA	Health Insurance and Portability and Accountability Act	SSI	Self-Sovereign Identity
HITECH	Health Information Technology for Economic and Clinical Health	TSH	Thyroid Stimulating Hormone
IDE	Integrated Development Environment	TT3	Total T3
IoT	Internet of things	TT4	Total T4
IPFS	InterPlanetary File System	UBCCSP	Upgraded Blockchain Cryptographic Service Provider
MAE	Mean Absolute Error	WBC	White Blood Cell
MDs	Medical Devices	WDs	Wearable Devices

Yaqoob *et al.* [21] used BC for healthcare data management. The authors discussed BC features such as data provenance, immutability, decentralization, distributed ledger, transparency, consensus, and programmability, along with their benefits in the healthcare system. The significant opportunities offered, such as patient record management, improved drug traceability, consistent permissions, clinical trials, precision medicine, optimized health insurance coverage, medical billing systems, and protection of telehealth systems in the healthcare sector using BC, were also discussed. Several case studies, such as permissioned BC for medical device tracking in Swiss hospitals, Estonian e-health system, BC for healthcare and pharma data in UAE, and patiently Decentralized Application (DApp) solutions in healthcare systems, were also mentioned and discussed. Various open research challenges like scalability, integration of BC with existing healthcare systems, ensuring the accuracy of healthcare data, interoperability, navigating regulation uncertainty, tokenization, irreversibility and quantum computing, culture adoption, and BC developers were also elaborated. Future recommendations like converging BC and artificial intelligence, plugging BC technology into legacy healthcare systems, developing secure smart contracts, establishing BC policies, IoT-based healthcare systems, and latency and throughput bottlenecks were also discussed.

Shynu *et al.* [22] proposed secure and proficient medical care services using fog computing based on BC for disease prediction. Two important diseases, cardio and diabetic diseases, were considered for this research. The fog nodes helped to collect patient's data and then saved it on BC. The clustering algorithm based on the rule was used for patients to collect health records. Using Feature Selection based Adaptive Neuro-Fuzzy Inference System (FS-ANFIS), diabetics and cardio diseases were predicted. Experimentation and analysis were done to assess the work output of the healthcare data. The results were calculated using Normalized Mutual Information (NMI) metrics and Purity. Iqbal *et al.* [17] used smart contract and machine learning techniques to propose BC-based Reliable and Intelligent Veterinary Information Management System (RIMS). Initially, the authors used Hyperledger Fabric to frame a secure BC-based veterinary clinic system. Secondly, smart contracts and predictive analytics modules were developed using permission- BC technology. The prediction model and the data patterns of the veterinary clinic patient's appointments data were analyzed using machine learning algorithms. Hyperledger Caliper was also used for working, and the overall result of the model was determined by root-mean-square error (RMSE), Mean Absolute Error (MAE), and R-squared (R2) score. Ray *et al.* [23] discussed e-health and BC impact followed by developed consensus algorithms. The authors also reviewed different BC platforms used in IoT-based e-healthcare. The use cases also discussed IoT and BC features in healthcare services. The authors also proposed BC with IoT data-flow architecture called IoBHealth, which was used for managing e-healthcare data more efficiently.

Shuaib *et al.* [24] reviewed the BC-based Self-Sovereign Identity (SSI) solution in healthcare, along with its advantages and requirements. Moreover, a model use case was demonstrated for SSI applications in healthcare. Omar*et al.* [25] provided insights for clinical trials (CTs) and presented a survey to adopt BC technology. The authors also categorized the literature using the taxonomy. Furthermore, detailed knowledge of developments towards the deployment of BC in CTs and several challenges that hindered the implementation of BC in CTs were also discussed. Tanwar *et al.* [18] explored solutions in BC-based healthcare systems for improving their limitations that included various frameworks such as Hyperledger Fabric, Composer, Docker Container, Hyperledger Caliper, and the Wireshark. Furthermore, the paper proposed an Access Control Policy Algorithm using a chain code based on hyperledger to improve data accessibility and implement an EHRs sharing system. Finally, various performance metrics such as latency, throughput, and Round Trip Time (RTT) were optimized for enhanced results.

To secure EHRs, Bhavin *et al.* [19] studied various security architectures and applied Quantum Computing (QC) to previously developed encryption systems. They proposed BC-based architecture for a healthcare system that allowed users to conveniently access data from the database. Furthermore, the Quantum blind signature was used during the block creation for protection from quantum using Hyperledger Fabric BC. Finally, results showed efficacy compared to state-of-the-art schemes in terms of the proposed scheme's transaction throughput, resource consumption, and network traffic. Soltanisehat *et al.* [26] reviewed 64 articles on healthcare systems based on BC. The articles reviewed were published between the years 2016 and 2020. The authors tried to find the different BC challenges faced and applications in healthcare. Different aspects were also found, such as technical, temporal, and spatial healthcare systems for developed BC applications. Future research directions in designing and implementing BC healthcare systems were also found.

Saha *et al.* [27] addressed the need for security for IoT-based healthcare applications. The authors designed and implemented an access control mechanism using private BC technology to share confidential data within a group of trusted hospitals securely. They used the proposed mechanism's Elliptic-curve cryptography (ECC)- based signature. The security of the system solved the "collision resistant one-way hash function" and Elliptic Curve Discrete Logarithm Problem (ECDLP). The proposed scheme protected "user anonymity" and "untraceability" properties. The proposed scheme showed better security and required very few computational overheads. The decentralized nature of BC helped maintain the integrity but somehow, the feature needed to be improved to maintain the security and safety of patient-centric systems. Hence, Omar *et al.* [28] presented a patient-centric healthcare system that attained privacy using BC technology as storage. Various cryptographic functions were used to encrypt patient data and ensure pseudonymity. Authors studied data processing procedures and cost-effectiveness of the smart contracts used in the system. Zhang *et al.* [29] highlighted some consensus protocols of two different categories of the absolute-finality and the probabilistic-finality. Through analysis and comparison, the paper studied different parameters such as strengths, weaknesses, and applicability of consensus mechanisms. Finally, the authors concluded that a good protocol should be used appropriately rather than not only be fault-tolerant for a particular application. Agbo *et al.* [30] compared the popular general BC frameworks to select a viable platform for healthcare applications. The authors suggested that the Hyperledger Fabric BC framework had complete features for healthcare applications and suggested testing in terms of scalability, latency, and throughput.

Bhuiyan *et al.*. [31] proposed a solution to manage to share of cross-institutional as well as to maintain the security of individual health data. The solution included stakeholders such as hospitals, insurance companies, clinic providers, and patients and aimed to increase clinical efficiency and effectiveness. The system secured the ownership of the data and built trust among different participants. To increase the accuracy of diagnosis, Wang *et al.* [32] proposed a framework for Parallel Healthcare Systems (PHS). This system was based on the artificial systems + computational experiments + parallel execution (ACP) approach. The artificial healthcare system used to frame patients' treatment process. The computational experiments evaluated various therapeutic regimens and implemented parallel execution for decision-making support and real-time optimization in healthcare systems. In addition, the authors combined the BC technology with PHS to link patients, health bureaus, hospitals, and healthcare communities for medical records review, care auditability, and data sharing. Finally, a treatment system and the prototype parallel gout diagnosis were built to demonstrate and verify the effectiveness and efficiency of the PHS framework.

Sravan *et al.* [33] provided a fraud-free solution to validate insurance claims. The transparent transaction system interaction with BC was made possible by the usage of smart contracts. The authors designed a framework on Ethereum for processing health insurance transactions using BC. IPFS was used for interoperability concepts across different applications. The authors suggested implementing and extending the model on R3 Corda for future directions. Koshechkin *et al.* [34] highlighted the importance of closed BC technology to improve public service

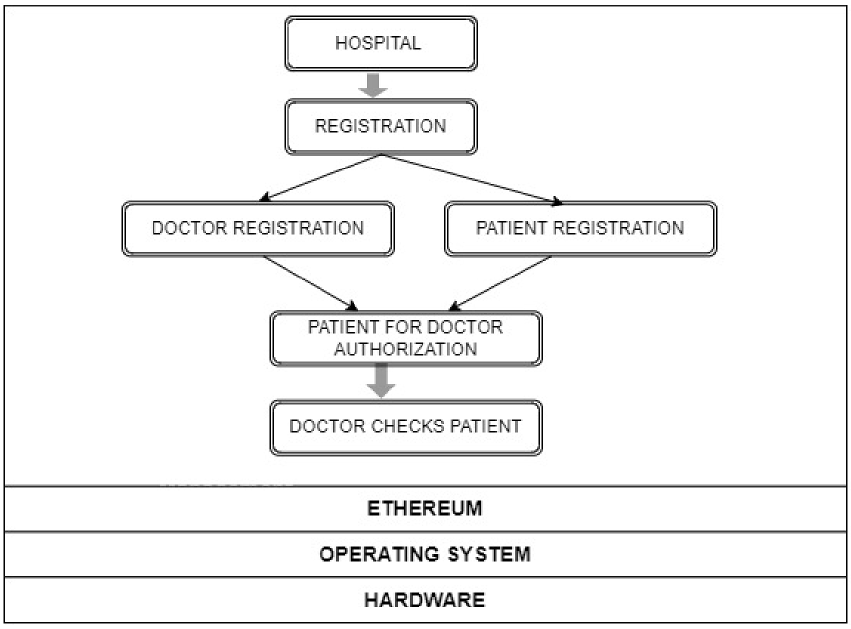

Figure 3: The proposed system model.

quality in electronic form. The authors suggested the importance of developing specific application solutions with their usage in the production and regulatory process. They seemed to get significant results in the maintenance of specialized registers, data processing by registration and licensing (in CTs), quality control of treatment, drug turnover, accounting for medical care provided, arrangement of mutual payments, support for decision making, remote monitoring of health status, distance consultations, etc.

Liang *et al.* [35] proposed a user-centric health data-sharing solution based on BC using a channel formation scheme to protect identity management privacy by utilizing the membership service. The health data was collected from personal wearable devices and manual inputs. The cloud was used to synchronize the data for sharing with healthcare insurance companies and providers. For all this work, the authors preserved the integrity of health data and deployed a mobile application. Moreover, a tree-based data processing and batching method was used to handle complex data. Raju *et al.* [36] presented a data bank, a data management approach that helped to track an individual's health and education in a personal specified data account. The right to data privacy and portability were equally important for bank design. The authors believed in developing a proof-of-principle prototype to study the role of smart contracts and various challenges in real-world implementation and to find a link between healthcare and education.

3. System Model and Problem Formulation

This research discusses the healthcare system that is developed on the BC network to help patients, doctors and hospitals to operate efficiently and effectively. The hospital system and its various features used by doctors such as physicians and pathologists are implemented in *MyEasyHeathcare* smart contract. Solidity language is used to write contracts for its smooth working.

3.1. System Model

Figure 3 shows a four-layered system, which includes the smart contracts and functions for the secure hospital management system. Different functions such as registration, modification, cancel of patients and doctors are built to help control the hospital. For the patient-doctor relationship, authorization is done using hashing algorithm along with twenty-two functions for the patient's treatment. All the code is done using solidity and above the second layer, Ethereum. Ethereum is an open-source and decentralized BC with a smart contract as its functionality. The code is executed on the Remix IDE platform for efficient working. The third layer is the operating system layer which is required to build the system and the last comes to the hardware layer, where the implementation is done on a real basis.

3.2. Problem Formulation

The existing healthcare systems are exposed to various issues due to inefficient storage system and the lack of trust. Some issues are discussed as follows:

- Lack of security and privacy in managing patient's information.
- Unsafe identity access management system in hospital.
- Unsafe doctor-patient relationship with lack of authorization.

Thus, to overcome the aforementioned issues with the existing healthcare systems, the BC-based "*MyEasyHeathcare*" system model is proposed to maintain trust and transparency among doctors and patients. The research helps the hospital to register doctors and patients in the hospital safely and securely. After registration, the patient is authorized to visit doctor for a check up. This authorization is done using hashing algorithm by the hospital to ensure safety. Finally, according to the disease and requirement, the patient is diagnosed and treated by only an authorized doctor and thus gets a trusted and safe environment for treatment.

4. Proposed Approach

Figure 4 shows the flowchart of the proposed approach and Each participant in the deployed hospital management system has a fixed role to play. The main authority to run the hospital is given to the owner who adds new patients by saving his account, id, name, age, and address. If required, the owner also helps to modify patient details, view details, and later on, removes him when not needed in the hospital. Similarly, the owner adds new doctors by saving their account address, id, name, age, address, and specialization details. The hospital can even modify, view, and remove doctor details as and when required. Figure 5 below shows the registration steps followed in the hospital system deployed using solidity language to register a doctor or a patient.

After the registration process, the smart contract deployed helps the hospital or owner to authorize a patient to a particular doctor as required by the patient. The novelty of our work is that the deployed system can assign a particular doctor of a particular specialization to a particular patient for checkups. But if requested for more doctors for personal satisfaction, a patient can be assigned to more doctors. The smart contract and functions built for this work use the keccak256 algorithm on the doctor's id and specialization for the assignment of a doctor to a particular patient. Meanwhile, the hospital checks authorized patient details for a particular doctor. After the patient's treatment and proper checkups hospital can even cancel the patient for a doctor.

The proposed system has specifically two different specialized doctors, such as physicians and pathologists. They treat patients and input their data in the form of Wei in the functions built in solidity to accept decimal values. The doctor checks and examines the patient's different parameters. The results are finally shown and prescribed to the patients by the doctors.

4.1. Features used for patient treatment

The proposed healthcare system has doctors and patients as its participants. This system specifically has two specialized doctors a physician and a pathologist. The smart contract written for features usually first checks the authorization of the patient for a specified doctor and then monitors. The doctor examines a set of parameters of an adult(18+) patient such as blood pressure, heart rate, body mass index, body temperature, pulse, respiration rate by physician and serum calcium, liver test, cholesterol, diabetes, hemoglobin, WBC count, RBC count, platelets count, thyroid test and kidney function test by a pathologist. The data for analysis is taken from various sources such as [37], [38], [39], [40], [41], [42] and [43]. Different features are discussed as follows:

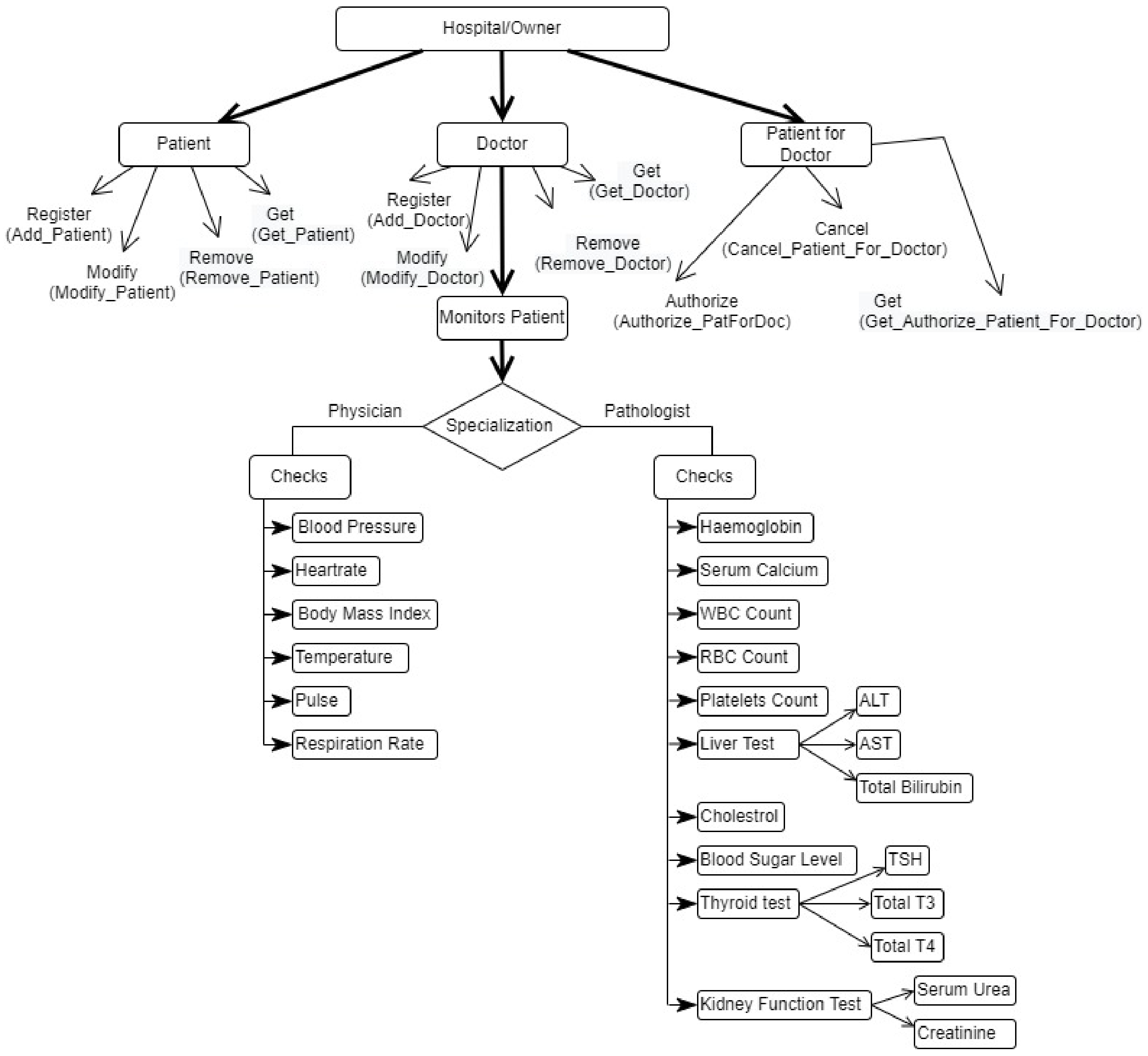

Figure 4: Flowchart of the proposed architecture

4.1.1. Blood Pressure

For blood pressure (BP) the doctor checks the upper and the lower reading of the bp. If the lower reading lies between 70mmHg-90 mmHg and the upper reading lie between 110mmHg-140mmHg then the patient has normal bp. Otherwise, if the upper reading is more than 140mmHg then the high bp, and if the lower reading is less than 70mmHg then the patient suffers from lower bp. Algorithm 1 shows the working of the checkup function. The function to check bp is deployed and executed in solidity on Remix IDE. The lower and upper bp values are given as input along with the patient's address and the output is returned as the doctor's advice for the patient. The function firsts check the successful registration of doctor and patient in the hospital. If both are registered, then check the authorization of the patient to a particular doctor. If all the conditions sets true, then the doctor finally checks the BP conditions of the patient and returns the advice. The complexity for the BP algorithm is O(1) i.e., it takes constant time to calculate BP of a particular person.

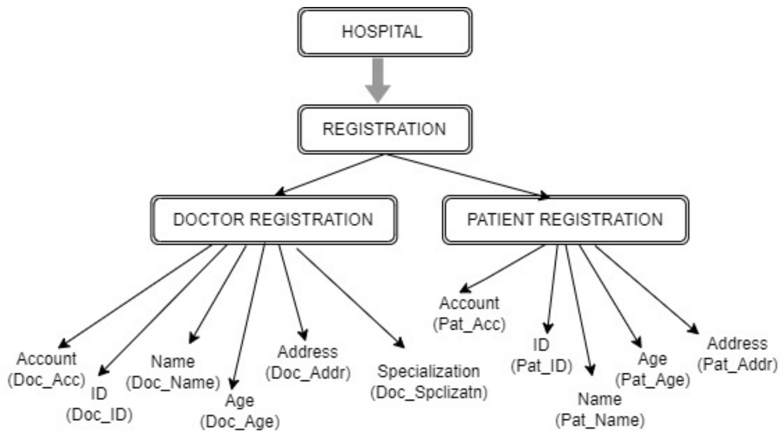

Figure 5: Registration process in a hospital

Algorithm 1 To check blood pressure of the patient

```
procedure CHECKBP(bplower, bpupper, _pataddress)
    Input: Two unsigned integer values of lower bp value and upper bp value along with patient address
    Output: Returns doctor advice
    doc_type ← physician
    s ← normal bp
    N ← n
    sh ← high bp..please be calm and meditate
    sl ← lower bp...please eat salty foods
    advice ← please take care
    require Doctor_Registration[msg.sender]==true    ▷ checking if doctor account is registered    ▷ Msg.sender is the doctor's address
    require Patient_Registration[_pataddress]==true    ▷ checking if patient account is registered
    doc ← docs[msg.sender]    ▷ docs maps to particular doctor
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if bplower ≥ 70 & bplower ≤ 90 then
                if bpupper ≥ 110 & bpupper ≤ 140 then
                    return s
                else if bpupper > 140 then
                    return sh
                else
                    return advice
                end if
            else if bplower < 70 then
                return sl
            else
                return advice
            end if
        end if
    end if
end procedure
```

4.1.2. Heartrate

To check the health of the heart, the doctor checks the heartbeat rate of the patient in beats per minute. If the value lies between the range of 60-100 heartbeats per minute, then the patient has a healthy heart otherwise high heart rate if the value is more than 100 or less heart rate if the value is less than 60 heartbeats per minute. Algorithm 2 shows the working of the heart rate function. The function to check heart rate is deployed and executed in solidity on Remix IDE. The heart beats per minute readings and the patient's address are given as input and the output is returned as the doctor's advice for the patient. The function firsts check the registration of the doctor and patient. If both are registered, then check the authorization of the patient to a particular physician. If all the conditions sets true, then the doctor finally checks the heart conditions of the patient and returns the advice. The complexity for the heartrate algorithm is O(1) i.e. it takes constant time to calculate the heartrate of a particular person.

Algorithm 2 To check heartrate of the patient

```
procedure HEARTRATE($beats\_permin$, $\_pataddress$)
    Input: An unsigned integer values of heart beats per minute along with patient address
    Output: Returns doctor advice
    doc_type ← physician
    s ← Congrats you have healthy heart
    sh ← high heart rate
    sl ← lower heart rate
    require $Doctor\_Registration[msg.sender]==true$
    require $Patient\_Registration[\_pataddress]==true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if beats_permin ≥ 60 & beats_permin ≤ 100 then
                return s
            else if beats_permin > 100 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

Algorithm 3 To check BMI of the patient

```
procedure BODYMASSINDEX($ht$, $wt$, $\_pataddress$)
    Input: Two unsigned integer values of patient's height and weight along with patient address
    Output: Returns doctor advice
    doc_type ← physician
    string[7] memory sr ← [Severely underweight, Underweight, Normal, Over Weight, Obese classI, Obese classII, Obese classIII]
    bm = (wt*1e19)/((ht/100)*(ht/100))
    require $Doctor\_Registration[msg.sender]==true$
    require $Patient\_Registration[\_pataddress]==true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if bm < 165 then
                return sr[0]
            else if bm ≥ 165 & bm≤ 184 then
                return sr[1]
            else if bm ≥ 185 & bm≤ 249 then
                return sr[2]
            else if bm ≥ 250 & bm≤ 299 then
                return sr[3]
            else if bm ≥ 300 & bm≤ 349 then
                return sr[4]
            else if bm ≥ 350 & bm≤ 399 then
                return sr[5]
            else
                return sr[6]
            end if
        end if
    end if
end procedure
```

4.1.3. Body Mass index (BMI)

BMI is derived from height and weight of a person.

$$BMI = kg/m^2 \tag{1}$$

where kg is the patient's weight in kilograms and m^2 is the patient's height in meters squared. The input provided to smart contracts is in Wei. This is because solidity does not accept decimals but rather supports integers. If the BMI value lies between:

less than 16.5, Severely underweight,

16.5-18.4, Underweight,

18.5 – 24.9, Normal weight,

Algorithm 4 To check body temperature of the patient

```
procedure TEMPERATURE($temp\_farenheit$, $\_pataddress$)
    Input: An unsigned integer values of body temperature in fahrenheit along with patient address
    Output: Returns doctor advice
    doc_type ← physician
    s ← Normal temperature
    sh ← Fever..Please take rest
    sl ← Lower temperature
    require $Doctor\_Registration[msg.sender]==true$
    require $Patient\_Registration[\_pataddress]==true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if temp_farenheit ≥ 97e18 & temp_farenheit ≤ 99e18 then
                return s
            else if temp_farenheit > 99e18 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

Algorithm 5 To check pulse rate of the patient

```
procedure PULSE_PERMIN($normal\_resting\_pulse$, $\_pataddress$)
    Input: An unsigned integer values of normal resting pulse along with patient address
    Output: Returns doctor advice
    doc_type ← physician
    s ← Normal resting pulse
    sh ← High pulse rate
    sl ← Low pulse rate
    require $Doctor\_Registration[msg.sender]==true$
    require $Patient\_Registration[\_pataddress]==true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if normal_resting_pulse ≥ 60 & normal_resting_pulse ≤ 100 then
                return s
            else if normal_resting_pulse > 100 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

25-29.9, Overweight,
30-34.9, Obese Class I,
35-39.9, Obese Class II,
40 or more, Obese Class III.

The algorithm 3 shows the working of the body mass index function. The function to check BMI is deployed and executed in solidity on Remix IDE. Two unsigned integer values of the patient's height and weight along with the address are given as input and the output is returned as the doctor's advice for the patient. The function firsts compute the BMI using the equation 1 and then checks the registration of the doctor and patient. If both are registered, then check the authorization of the patient to a particular physician. If all the conditions sets true, then the doctor finally checks the BMI conditions of the patient and returns the advice. The complexity of the BMI algorithm is O(1) i.e. it takes constant time to calculate the BMI of a particular person.

4.1.4. Body Temperature

The body temperature is taken in degrees Fahrenheit. For temperature values, the input is provided in Wei to accept decimals. If the temperature lies between 97-99 degrees Fahrenheit, then the patient has normal body temperature. If

Algorithm 6 To check respiration rate of the patient

```
 1: procedure RESPIRATIONRATE_PERMIN($respiration$, $\_pataddress$)
 2:     Input: An unsigned integer values of respiration rate along with patient address
 3:     Output: Returns doctor advice
 4:     doc_type ← physician
 5:     s ← Normal respiration
 6:     sh ← High respiration
 7:     sl ← Low respiration
 8:     require $Doctor\_Registration[msg.sender]{==}true$
 9:     require $Patient\_Registration[\_pataddress]{==}true$
10:     doc ← docs[msg.sender]
11:     patfordoc ← patfordocs[msg.sender]
12:     if patfordoc.Pat_Authorized[_pataddress] == true then
13:         if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
14:             if respiration $\geq$ 12 & respiration $\leq$ 20 then
15:                 return s
16:             else if respiration > 20 then
17:                 return sh
18:             else
19:                 return sl
20:             end if
21:         end if
22:     end if
23: end procedure
```

Algorithm 7 To check serum calcium of the patient

```
 1: procedure SERUMCALCIUM($sc$, $\_pataddress$)
 2:     Input: An unsigned integer values of serum calcium along with patient address
 3:     Output: Returns doctor advice
 4:     doc_type ← pathologist
 5:     s ← Normal Serum Calcium in milligram per decilitre
 6:     sh ← Higher Serum Calcium in milligram per decilitre
 7:     sl ← Lower Serum Calcium in milligram per decilitre
 8:     require $Doctor\_Registration[msg.sender]{==}true$
 9:     require $Patient\_Registration[\_pataddress]{==}true$
10:     doc ← docs[msg.sender]
11:     patfordoc ← patfordocs[msg.sender]
12:     if patfordoc.Pat_Authorized[_pataddress] == true then
13:         if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
14:             if sc $\geq$ 86e17 & sc $\leq$ 103e17 then
15:                 return s
16:             else if sc > 103e17 then
17:                 return sh
18:             else
19:                 return sl
20:             end if
21:         end if
22:     end if
23: end procedure
```

the temperature is more than 99 degrees Fahrenheit then the patient is suffering from a high fever otherwise patient has a low temperature if the temperature is less than 97 degrees Fahrenheit. The algorithm 4 shows the working of the temperature function. The function to check body temperature is deployed and executed in solidity on Remix IDE. An unsigned integer value of the patient's body temperature in Fahrenheit along with the address is given as input and the output is returned as the doctor's advice for the patient. The function firsts check the successful registration of doctor and patient in the hospital. If both are registered, then check the authorization of the patient to a particular physician. If all the conditions are set to true, then the doctor finally checks the temperature conditions of the patient and returns the advice. The complexity for the temperature algorithm is O(1) i.e. it takes constant time to calculate the body temperature of a particular patient.

4.1.5. Pulse

The doctor calculates and checks the normal resting pulse. If the pulse rate is more than and equal to 60 per minute and less than and equal to 100 per minute, then the patient has a normal resting pulse. If the pulse rate is more than 100 per minute then a high pulse rate otherwise low pulse rate if the rate is less than 60 per minute. The algorithm 5 shows the working of the pulse per minute function. The function to check pulse rate is deployed and executed in solidity

Algorithm 8 To check Liver ALT of the patient

```
procedure LiverALT(alt, _pataddress)
    Input: An unsigned integer values of alt along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal ALT in units per litre
    sh ← Higher ALT in units per litre
    sl ← Lower ALT in units per litre
    require Doctor_Registration[msg.sender]==true
    require Patient_Registration[_pataddress]==true
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if alt $\geq$ 7e18 & alt $\leq$ 56e18 then
                return s
            else if alt > 56e18 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

Algorithm 9 To check Liver AST of the patient

```
procedure LiverAST(ast, _pataddress)
    Input: An unsigned integer values of ast along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal AST in units per litre
    sh ← Higher AST in units per litre
    sl ← Lower AST in units per litre
    require Doctor_Registration[msg.sender]==true
    require Patient_Registration[_pataddress]==true
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if ast $\geq$ 8e18 & ast $\leq$ 48e18 then
                return s
            else if ast > 48e18 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

on Remix IDE. The normal resting pulse along with patient's address is given as input and the output is returned as the doctor's advice for the patient. The function firsts check the successful registration of doctor and patient on the hospital premises. If both are registered, then check the authorization of the patient to a particular physician. If all the conditions are set to true, then the doctor finally checks the pulse conditions of the patient and returns the advice. The complexity for the pulse algorithm is O(1) i.e. it takes constant time to calculate pulse of a particular person.

4.1.6. Respiration rate

The normal respiration rate lies between 12 and 20 breaths per minute. If the rate is more than 20 breaths per minute, then the patient has a high respiration rate otherwise if less than 12 breaths per minute, low respiration rate. The algorithm 6 shows the working of respiration rate per minute function. The function to check respiration rate is deployed and executed in solidity on Remix IDE. An unsigned integer value of respiration rate along with the patient's address is given as input and the output is returned as the doctor's advice for the patient. The function firsts check the successful registration of doctor and patient in the hospital. If both are registered, then check the authorization of the patient to a particular doctor. If all the conditions are set to true, then the doctor finally checks the rate of respiration conditions of the patient and returns the advice. The complexity of the respiration rate algorithm is O(1) i.e. it takes

Algorithm 10 To check Liver Total Bilirubin of the patient

```
procedure LiverTotalBilirubin($b$, $\_pataddress$)
    Input: An unsigned integer values of liver total bilirubin along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal Bilirubin in mg per dL
    sh ← Higher Bilirubin in mg per dL
    sl ← Lower Bilirubin in mg per dL
    require $Doctor\_Registration[msg.sender]{=}{=}true$
    require $Patient\_Registration[\_pataddress]{=}{=}true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if b == 12e17 then
                return s
            else if b > 12e17 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

constant time to calculate the respiration rate of a particular patient.

Algorithm 11 To check cholestrol of the patient

```
procedure Cholestrol($cl$, $\_pataddress$)
    Input: An unsigned integer values of cholestrol along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal cholestrol in mg/dL
    sh ← Higher cholestrol in mg/dL
    sl ← Lower cholestrol in mg/dL
    require $Doctor\_Registration[msg.sender]{=}{=}true$
    require $Patient\_Registration[\_pataddress]{=}{=}true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if cl ≥ 125e18 & cl ≤ 200e18 then
                return s
            else if cl > 200e18 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

4.1.7. Serum Calcium

It usually detects the amount of calcium in blood. The normal range lies between 8.6 milligrams (mg) per decilitre (dL)-10.3 mg/dL. The smart contracts deployed input the data in the form of Wei. If the value is greater than 10.3 mg/dL, then the patient has high serum calcium; otherwise, if it is less than 8.6 mg/dL has a low serum calcium amount. The algorithm 7 shows the working of serum calcium function. The function to check serum calcium is deployed and executed in solidity on Remix IDE. An unsigned integer value of an amount of serum calcium and the patient's address is given as input, and the output is returned as doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization with a pathologist. If all the conditions are true, the doctor finally checks the details of the serum calcium amount in the patient and returns the advice. The complexity of the serum calcium algorithm is O(1), i.e. it takes constant time to calculate the serum calcium of a particular person.

Algorithm 12 To check blood sugar level of the patient

```
procedure DIABETES_PREDICT($sugar, category\_pataddress$)
    Input: An unsigned integer values of sugar level and categories specifying fasting value, just after eating values or value after 2 hours of consuming glucose along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal body..Be happy
    sh ← Early diabetes
    sl ← Established Diabetes
    e ← Low glucose level in blood
    require $Doctor\_Registration[msg.sender]{==}true$
    require $Patient\_Registration[\_pataddress]{==}true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if category == 1 then    ▷ Press 1 for fasting    ▷ Press 2 for just after eating    ▷ Press 3 for Value for 2 hours after consuming glucose
                if sugar ≥ 70e18 & sugar ≤ 100e18 then
                    return s
                else if sugar ≥ 101e18 & sugar ≤ 126e18 then
                    return sh
                else if sugar > 126e18 then
                    return sl
                else
                    return e
                end if
            end if
            if category == 2 then
                if sugar ≥ 170e18 & sugar ≤ 200e18 then
                    return s
                else if sugar ≥ 200e18 & sugar ≤ 230e18 then
                    return sh
                else if sugar > 230e18 then
                    return sl
                else
                    return e
                end if
            end if
            if category == 3 then
                if sugar < 140e18 then
                    return s
                else if sugar ≥ 140e18 & sugar ≤ 200e18 then
                    return sh
                else
                    return sl
                end if
            end if
        end if
    end if
end procedure
```

Algorithm 13 To check hemoglobin of the patient

```
1:  procedure HEMOGLOBIN(hb, gender, _pataddress)
2:      Input:An unsigned integer values of hemoglobin level and gender specifying male or female along with patient address
3:      Output: Returns doctor advice
4:      doc_type ← pathologist
5:      s ← Normal Hemoglobin in grams per decilitre..Partyyy
6:      sh ← High hemoglobin in grams per decilitre
7:      sl ← Low hemoglobin in grams per decilitre
8:      require Doctor_Registration[msg.sender]==true
9:      require Patient_Registration[_pataddress]==true
10:     doc ← docs[msg.sender]
11:     patfordoc ← patfordocs[msg.sender]
12:     if patfordoc.Pat_Authorized[_pataddress] == true then
13:         if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
14:             if gender == 1 then                          ▷ Press 1 for male          ▷ Press 2 for female
15:                 if hb ≥ 132e17 & hb ≤ 166e17 then
16:                     return s
17:                 else if hb > 166e17 then
18:                     return sh
19:                 else
20:                     return sl
21:                 end if
22:             end if
23:             if gender == 2 then
24:                 if hb ≥ 116e17 & hb ≤ 150e17 then
25:                     return s
26:                 else if hb > 150e17 then
27:                     return sh
28:                 else
29:                     return sl
30:                 end if
31:             end if
32:         end if
33:     end if
34: end procedure
```

4.1.8. LiverTest

The proposed system tests liver test by measuring alanine transaminase (ALT), aspartate aminotransferase (AST) and total bilirubin amounts. The normal range for ALT lies between 7-56 units per litre, AST lies between 8-48 units per litre, and normal results for total bilirubin are 1.2 mg/dL for adults. The input in function is provided in Wei to accept decimals. The algorithms 8, 9 and 10 shows the working of liver test that included ALT,AST and total bilirubin functions. The functions to check liver ALT, AST and total bilirubin are deployed and executed in solidity on Remix IDE. Unsigned integer values of ALT, AST and total bilirubin, along with patient's address, are given as input to the respective functions. The output is returned as doctor's advice for the patient. The input to ALT and AST are provided in units per litre and total bilirubin in mg per dL. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization with a pathologist. If all the conditions are true, the doctor finally checks the patient's details and returns the advice accordingly. The complexity for all three algorithms is O(1), i.e. they take constant time to test the liver of a particular patient.

4.1.9. Cholesterol

The normal cholesterol level lies between 125-200 mg/dL. If the value lies below 125mg/dL, the patient has low cholesterol. Otherwise, if the value is above 200 mg/dL, the patient has high cholesterol and needs to take proper precautions. The input to the smart contract is provided in Wei. The algorithm 11 shows the working of the cholesterol function. The function to check cholesterol is deployed and executed in solidity on Remix IDE. An unsigned integer value of cholesterol, along with the patient's address, is given as input, and the output is returned as the doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, the doctor finally contains the details of the patient's cholesterol and the amount in mg per dL and returns the advice. The complexity of the cholesterol algorithm is O(1), i.e., it takes constant time to calculate the cholesterol of a particular person.

Algorithm 14 To check WBC count of the patient

```
procedure WhiteBloodCells_count(wbc, _pataddress)
    Input: An unsigned integer value of white blood cell count along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal wbc cell count in cells per microlitre
    sh ← Higher wbc cell count in cells per microlitre
    sl ← Lower wbc cell count in cells per microlitre
    require Doctor_Registration[msg.sender]==true
    require Patient_Registration[_pataddress]==true
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if wbc ≥ 4500e18 & wbc ≤ 11000e18 then
                return s
            else if wbc > 11000e18 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

Algorithm 15 To check RBC count of the patient

```
procedure RedBloodCells_count(rbc, gender, _pataddress)
    Input: An unsigned integer values of red blood cell count , gender as male or female along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal rbc cell count in million cells per microlitre
    sh ← Higher rbc cell count in million cells per microlitre
    sl ← Lower rbc cell count in million cells per microlitre
    require Doctor_Registration[msg.sender]==true
    require Patient_Registration[_pataddress]==true
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if gender == 1 then                ▷ Press 1 for male                ▷ Press 2 for female
                if rbc ≥ 45e17 & rbc ≤ 59e17 then
                    return s
                else if rbc > 59e17 then
                    return sh
                else
                    return sl
                end if
            end if
            if gender == 2 then
                if rbc ≥ 41e17 & rbc ≤ 51e17 then
                    return s
                else if rbc > 51e17 then
                    return sh
                else
                    return sl
                end if
            end if
        end if
    end if
end procedure
```

4.1.10. Diabetes

The categories for diabetes prediction are normal, early diabetes and established diabetes. The values taken are fasting values, just after eating, and values for 2 hours after consuming glucose. The input is provided in Wei to accept decimals. If the fasting value lies between 70mg/dL to 100 mg/dL, just after eating value lies between 170-200 mg/dL and the value after 2 hours is less than 140 mg/dL, then the patient has a normal blood sugar level. For Early diabetes, the range for fasting values is between 101-126 mg/dL, just after eating values between 200-230 mg/dL and the value after 2 hours of consuming glucose is between 140-200 mg/dL. For established diabetes, fasting values are more than

Algorithm 16 To check platelets count of the patient

```
procedure PLATELETS_COUNT($p$, $\_pataddress$)
    Input: An unsigned integer value of platelet count along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal cell count in platelets per microlitre
    sh ← Higher cell count in platelets per microlitre
    sl ← Lower cell count in plateletes per microlitre
    require $Doctor\_Registration[msg.sender]==true$
    require $Patient\_Registration[\_pataddress]==true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if p ≥ 150000e18 & p ≤ 450000e18 then
                return s
            else if p > 450000e18 then
                return sh
            else
                return sl
            end if
        end if
    end if
end procedure
```

Algorithm 17 To check TSH of the patient

```
procedure TSH($tsh$, $\_pataddress$)
    Input: An unsigned integer value of tsh along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    ntsh ← Normal tsh in miliInternational Units per litre of blood
    htsh ← Higher tsh in miliInternational Units per litre of blood
    ltsh ← Lower tsh in miliInternational Units per litre of blood
    require $Doctor\_Registration[msg.sender]==true$
    require $Patient\_Registration[\_pataddress]==true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if tsh ≥ 5e17 & tsh ≤ 45e17 then
                return ntsh
            else if p > 45e17 then
                return htsh
            else
                return ltsh
            end if
        end if
    end if
end procedure
```

126 mg/dL; just after eating, values lies between 230-300 mg/dL and value after 2 hours of consuming glucose are more than 200 mg/dL. The algorithm 12 shows the working of blood sugar level function for different categories. The function to check blood sugar level is deployed and executed in solidity on Remix IDE. An unsigned integer value of sugar level and category that specifies the sugar value being taken as fasting value, just after eating or after two hours of consuming glucose and the patient's address are given as input. The output is returned as per the doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, the doctor finally contains the details of blood sugar levels for specific categories and returns the advice accordingly. The complexity of to test blood sugar level algorithm is O(1), i.e., it takes constant time to calculate the diabetes of a particular person.

4.1.11. Hemoglobin

For males and females, the hemoglobin value ranges are different. For males, the normal range lies from 13.2 g/dL to 16.6 g/dL. For females, the range lies from 11.6 g/dL to 15 g/dL. The values in smart contracts are provided in Wei. The algorithm 13 shows the working of hemoglobin function. The function to check hemoglobin in males and females is deployed and executed in solidity on Remix IDE. An unsigned integer value of hemoglobin, gender, and patient's

Algorithm 18 To check TT4 of the patient

```
1: procedure TT4($tt4$, $\_pataddress$)
2:     Input: An unsigned integer value of tt4 along with patient address
3:     Output: Returns doctor advice
4:     doc_type ← pathologist
5:     ntt4 ← Normal tt4 in microgram per decilitre
6:     htt4 ← Higher tt4 in microgram per decilitre
7:     ltt4 ← Lower tt4 in microgram per decilitre
8:     require $Doctor\_Registration[msg.sender]==true$
9:     require $Patient\_Registration[\_pataddress]==true$
10:    doc ← docs[msg.sender]
11:    patfordoc ← patfordocs[msg.sender]
12:    if patfordoc.Pat_Authorized[_pataddress] == true then
13:        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
14:            if tt4 ≥ 54e17 & tt4 ≤ 115e17 then
15:                return ntt4
16:            else if tt4 > 115e17 then
17:                return htt4
18:            else
19:                return ltt4
20:            end if
21:        end if
22:    end if
23: end procedure
```

Algorithm 19 To check TT3 of the patient

```
1: procedure TT3($tt3$, $\_pataddress$)
2:     Input: An unsigned integer value of tt3 along with patient address
3:     Output: Returns doctor advice
4:     doc_type ← pathologist
5:     ntt3 ← Normal tt4 in microgram per decilitre
6:     htt3 ← Higher tt4 in microgram per decilitre
7:     ltt3 ← Lower tt4 in microgram per decilitre
8:     require $Doctor\_Registration[msg.sender]==true$
9:     require $Patient\_Registration[\_pataddress]==true$
10:    doc ← docs[msg.sender]
11:    patfordoc ← patfordocs[msg.sender]
12:    if patfordoc.Pat_Authorized[_pataddress] == true then
13:        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
14:            if tt3 ≥ 80e18 & tt3 ≤ 220e18 then
15:                return ntt3
16:            else if tt3 > 220e18 then
17:                return htt3
18:            else
19:                return ltt3
20:            end if
21:        end if
22:    end if
23: end procedure
```

address is given as input, and the output is returned as doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, the doctor finally checks the details of hemoglobin in grams per dL for male or female patients and returns the advice accordingly. The complexity of hemoglobin algorithm is O(1) i.e. it takes constant time to calculate hemoglobin of a particular patient.

4.1.12. White Blood Cell (WBC) count

The normal range of WBC lies from 4500-11000 cells per microlitre. The input provided is in Wei to the smart contracts. The algorithm 14 shows the working of WBC count function. The function to check WBC count is deployed and executed in solidity on Remix IDE. An unsigned integer value of WBC count in cells per microlitre, along with patient's address, is given as input and the output is returned as doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, the doctor finally checks the details of the WBC count of the patient and returns the advice. The complexity for WBC algorithm is O(1), i.e. it takes constant time to count WBC of a particular person.

Algorithm 20 To check thyroid of the patient

```
procedure THYROID($tt3, tt4, tsh\_pataddress$)
    Input:Three unsigned integer values of patient's tt3,tt4 and tsh along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    prhypo ← PRIMARY HYPOTHYROIDISM
    hypo ← HYPOTHYROIDISM
    prhyper ← PRIMARY HYPERTHYROIDISM
    hyper ← HYPERTHYROIDISM
    abnormal ← PITUITARY GLAND ABNORMAL
    require $Doctor\_Registration[msg.sender]$==$true$
    require $Patient\_Registration[\_pataddress]$==$true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if tt3 ≥ 80e18 & tt3 ≤ 220e18) & (tt4 ≥ 54e17 & tt4 ≤ 115e17) & (tsh > 45e17) then
                return prhypo
            else if (tt3 < 80e18) & (tt4 < 54e17) & (tsh > 45e17) then
                return hypo
            else if (tt3 ≥ 80e18 & tt3 ≤ 220e18) & (tt4 ≥ 54e17 & tt4 ≤ 115e17) & (tsh < 5e17) then
                return prhyper
            else if (tt3 > 220e18) & (tt4 > 115e17) & (tsh < 5e17) then
                return hyper
            else if (tt3 < 80e18) & (tt4 < 54e17) & (tsh < 5e17) then
                return abnormal
            end if
        end if
    end if
end procedure
```

Algorithm 21 To check serum urea of the patient

```
procedure SERUM_UREA($su, \_pataddress$)
    Input: An unsigned integer value of serum urea along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    n ← Normal serum urea in miligram per decilitre
    h ← Higher serum urea in miligram per decilitre
    l ← Lower serum urea in miligram per decilitre
    require $Doctor\_Registration[msg.sender]$==$true$
    require $Patient\_Registration[\_pataddress]$==$true$
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if su ≥ 6e18 & su ≤ 24e18 then
                return n
            else if su > 24e18 then
                return h
            else
                return l
            end if
        end if
    end if
end procedure
```

4.1.13. Red Blood Cells (RBC) count

The normal range differs in both males and females. For males, the range is 4.5-5.9 million cells per microlitre, and for females, the range is between 4.1-5.1 million cells per microlitre. The input provided is in Wei to the smart contracts. The algorithm 15 shows the working of RBC count function. The function to check RBC count in males and females is deployed and executed in solidity on Remix IDE. An unsigned integer value of RBC count, gender, and patient's address is given as input and the output is returned as doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization with a pathologist. If all the conditions are true, the doctor finally contains the details of RBC, the count in a million cells per microlitre for male or female patients individually and returns the advice accordingly. The complexity for RBC algorithm is O(1), i.e. it takes constant time to count RBC of a particular person.

Algorithm 22 To check creatinine of the patient

```
procedure CREATININE(cr, gender, _pataddress)
    Input: An unsigned integer values of creatinine , gender as male or female along with patient address
    Output: Returns doctor advice
    doc_type ← pathologist
    s ← Normal Creatinine in miligrams per decilitre..Partyyy
    sh ← High Creatinine in miligrams per decilitre
    sl ← low Creatinine in miligrams per decilitre
    require Doctor_Registration[msg.sender]==true
    require Patient_Registration[_pataddress]==true
    doc ← docs[msg.sender]
    patfordoc ← patfordocs[msg.sender]
    if patfordoc.Pat_Authorized[_pataddress] == true then
        if keccak256(abi.encodePacked((doc.Doc_Spclizatn))) == (keccak256(abi.encodePacked((doc_type)))) then
            if gender == 1 then                    ▷ Press 1 for male                    ▷ Press 2 for female
                if cr ≥ 74e16 & cr ≤ 135e16 then
                    return s
                else if cr > 135e16 then
                    return sh
                else
                    return sl
                end if
            end if
            if gender == 2 then
                if cr ≥ 59e16 & cr ≤ 104e16 then
                    return s
                else if cr > 104e16 then
                    return sh
                else
                    return sl
                end if
            end if
        end if
    end if
end procedure
```

4.1.14. Platelets count

The normal range for platelet count lies between 1,50,000-4,50,000 platelets per microlitre. The input provided is in Wei to the smart contracts. The algorithm 16 shows the working of the platelets count function. The function to check platelets count is deployed and executed in solidity on Remix IDE. An unsigned integer value of platelet count, along with patient's address, is given as input, and the output is returned as doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, the doctor finally checks the details of platelets per microlitre of the patient and returns the advice. The complexity of the platelet count algorithm is O(1), i.e., it takes constant time to count the platelets of a particular patient.

4.1.15. Thyroid test

The thyroid is predicted using the values of Total T3(TT3), Total T4(TT4) and thyroid stimulating hormone (TSH) values. The smart contracts built in the proposed system predict five situations corresponding to the different values of TT3, TT4 and TSH. The normal range for TT3 lies between 80-220 nanogram per deciliter (ng/dL); for TT4, the value lies between 5.4-11.5 microgram per deciliter, and for TSH, the value ranges from 0.5-4.5 milli-international units per litre (mIU/L). If T3 and T4 are normal, but the TSH level is elevated, then the patient has primary hypothyroidism. If T3 and T4 are low levels but TSH is high, then hypothyroidism is the case. If T3 and T4 are normal and TSH is low, the case is primary hyperthyroidism. If T3 and T4 levels are elevated and TSH is low, the patient suffers from hyperthyroidism. Finally, if all T3, T4 and TSH are at low levels, the patient has an abnormal pituitary gland. The input provided is in Wei to the smart contracts. The algorithms 17, 18 and 19 shows the working of TSH, TT4 and TT3. Based on these values, algorithm 20 helps predict the patient's thyroid. The functions to check TSH, TT4 and TT3 are deployed and executed in solidity on Remix IDE. An unsigned integer value of TSH, TT4 and TT3, and patient's address, is given input individually into their respective functions. The output is returned as doctor's advice for the patient accordingly. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, then doctor finally checks the details of TSH in miliInternational Units per litre of blood, TT4 in microgram per decilitre and TT3 amount in mg per dL of the patient, respectively, from their corresponding functions and returns the doctor's advice.

With the help of these values, the thyroid is predicted by the doctors in Remix IDE. Input for the function is taken as tt3, tt4 and TSH along with patient's address and output is returned as doctor's advice. The function helps the patient and the doctor know about the type of thyroid. Five categories are predicted for the patient: primary hypothyroidism, hypothyroidism, primary hyperthyroidism, and pituitary gland abnormality. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization with a pathologist's doctor. If all the conditions are true, the doctor finally checks the details of the patient's TSH, TT4 and TT3 amounts and returns the doctor's advice. The complexity of checking thyroid algorithms is O(1), i.e., they take constant time to check a particular person's TSH, TT4, TT3 and thyroid.

4.1.16. Kidney Function test

The kidney functionality is checked using serum urea and creatinine test. The normal range for serum urea lies from 6-24 mg/dL. For creatinine and males and females, the ranges are different. For males the value lies between 0.74-1.35 mg/dL and for females the range is between 0.59-1.04 mg/dL. The smart contracts built inputs the value in Wei. The algorithms 21 and 22 shows the working of serum urea and creatinine functions. The function to check serum urea content in milligrams per decilitre is deployed and executed in solidity on Remix IDE. An unsigned integer value of serum urea and the patient's address are given as input, and the output is returned as the doctor's advice for the patient. The function first checks the successful registration of doctors and patients in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, the doctor finally checks the details of the serum urea amount in the patient and returns the advice.

The function to check creatinine is deployed and executed in solidity on Remix IDE. An unsigned integer value of creatinine content in milligrams per decilitre, gender, and patient's address is given as input, and the output is returned as doctor's advice for the patient. The function first checks the successful doctor and patient registration in the hospital. If both are registered, check the patient's authorization to a particular doctor. If all the conditions are true, the doctor finally checks the creatinine amount for male and female patients and returns the advice. The complexity of kidney test algorithms is O(1), i.e. they take constant time to test the kidney of a particular person.

5. Results and Discussion

MyEasyHeathcare system has been deployed on the Ethereum test network (Rinkeby). It is made secure by adding an additional feature of the doctor-patient relationship which is not found in any previous work. The performance is evaluated based on gas and execution costs to decide whether the system can be used in the main Ethereum network. The gas cost incurred is shown concerning different functions built for execution.

5.1. Evaluation Methodology

The designed system is deployed on the Ethereum test network (Rinkeby) using Remix IDE. The proposed system is tested by using several use cases of different numbers of doctors and patients. For the evaluation of the patient, 22 parameters are taken, and 7 doctors and patients are added. More than 25 functions are being written and executed in the designed smart contract. To evaluate the performance of these different functions, two parameters: gas cost and execution gas cost, are considered. For more accurate analysis, different scenarios are taken.

- Seven iterations are executed for adding doctors and patients.
- For authorization of a particular patient to a doctor, different situations is taken into accounts, such as one doctor with one patient relationship, one doctor with two patients, two doctors for one patient (for example, if a patient is not satisfied with one doctor prescription and want to consult another doctor for the satisfaction of the same specialization), two different specialized doctors for one patient (if a patient wants to check his heart rate along with platelet count will consult physician and pathologist), one doctor for six patients and one doctor for all seven patients.
- To evaluate the performance of a specific parameter, four parameters are tested for seven different patients.
- Updation in a database for patients and doctors is evaluated using 5 iterations.

5.2. Simulation Setup

The simulation setup consists of the Ethereum test network(Rinkeby). The smart contract is built on Remix IDE, using solidity to secure doctor-patient relationships in the hospital.

Table 3: Functions with their respective gas cost and execution gas cost incurred from single iteration.

Functions	Gas cost (in wei)	Execution Gas cost (in wei)
Add_Patient	241532	210027
Modify_Patient	60378	52502
Remove_Patient	89380	48921
Get_Patient	-	39454
Add_Doctor	270137	234901
Modify_Doctor	68006	59135
Remove_Doctor	95348	49311
Get_Doctor	-	42144
Authorize_PatForDoc	67678	58850
Cancel_Patient_For_Doctor	39160	29252
Get_Authorize_Patient_For_Doctor	-	31830
Checkbp	38590	33556
Heartrate	38831	33766
Bodymassindex	40018	34798
Temperature	38788	33728
Pulse_permin	38249	33260
Respirationrate_permin	38189	33207
Serumcalcium	38582	33549
LiverALT	38283	33289
LiverAST	38293	33298
LiverTotalBilirubin	38797	33736
Cholestrol	38470	33452
Diabetes_Predict	39528	34372
Hemoglobin	38972	33888
WhiteBloodCells_count	38660	33617
RedBloodCells_count	39493	34341
Platelets_count	39130	34026
TSH	38835	33769
TT4	38875	33804
TT3	38494	33473
Thyroid	39742	34558
Serum_Urea	39334	34203
Creatinine	38921	33844

5.3. Evaluation Metrics

Two evaluation metrics, such as gas cost and execution gas cost, are used to evaluate the healthcare system. The gas cost represents the cost necessary to conduct or execute a transaction which depends on the complexity of the algorithm of the concerned transaction. The gas cost charged is tabulated once the system is initialized and deployed on Remix IDE. Figure 6 represents the gas cost in Wei of different functions executed on blockchain. Similarly, the execution cost in Remix IDE represents the computational operation cost executed in a transaction. Figure 7 shows the execution cost for every function built and developed in the healthcare system in solidity.

Table 3 tabulates the different functions along with their respective gas cost and execution gas cost for a single iteration. The developed system is built in Remix IDE using solidity language. The values are presented in terms of Wei. In Figure 6, the Add_Doctor function has the maximum gas cost of 270137 units followed by Add_Patient with 241532 units of gas. Other functions have values ranging from 0 to 96000 units. Get_Doctor, Get_Patient and Get_Authorize_Patient_for_Doctor show no gas units while deploying the code. This is because the above functions do not create a transaction and do not cause state change by clicking. Other functions deployed to create a transaction by clicking and thus cost gas. Figure 7 shows the execution costs for the functions with 234901 units of gas being maximum for Add_Doctor function. The Add_patient function shows the second-highest gas consumption of 210027 units during execution. Other functions have their gas consumption cost range between 29000 to 60000 units.

5.4. Evaluation Metric wise Results

Based on evaluation methodology and metrics, the healthcare system is evaluated for different cases, and these are as follows.

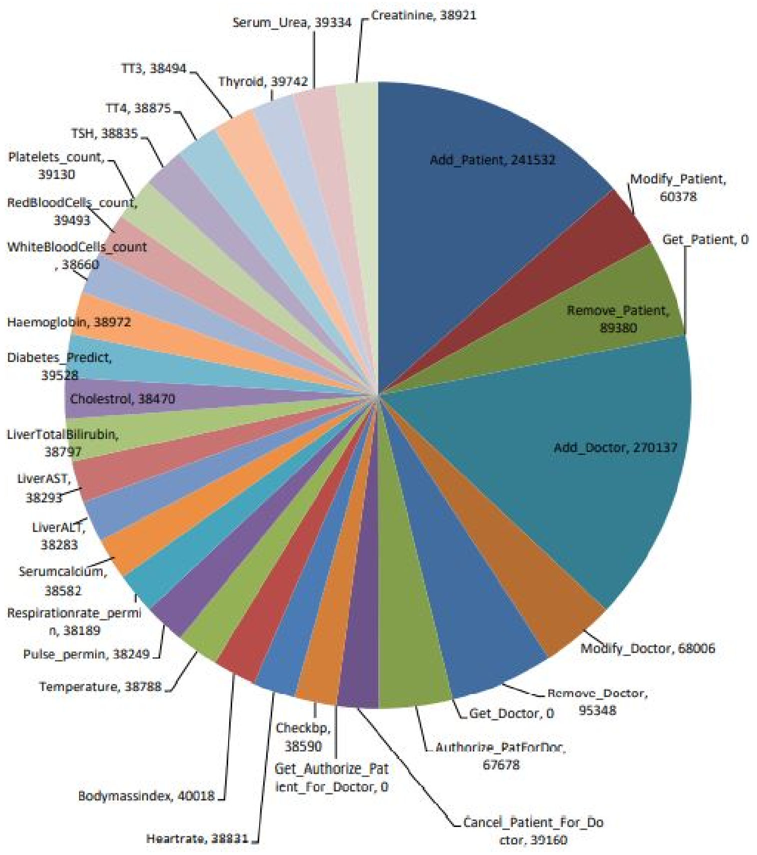

Figure 6: Piechart showing gascost incurred from single iteration for different functions.

5.4.1. Based on addition of doctor in hospital

The system deployed has seven doctors and seven patients that are taken into consideration. Each doctor detail is inserted into the function Add_Doctor for adding the doctors, and the transaction is created. Seven iterations are done for adding seven doctors into the hospital. Figure8 shows the line graph of the results noted for the gas cost and the execution cost spent in Wei for adding doctors. Figure clearly shows that while adding one doctor, the gas incurred is 270238 units, while the simultaneous iterations show the value as 230949 gas, 230908 gas, 230991 gas, 230853 gas, 230935 gas and 230880 gas units. The execution gas cost expenditure offers the value as 234989 units, 200825 units, 200789 units, 200861 units, 200741 units, 200813 units and 200765 units. Thus, this shows that as and when more doctors have added to the system, the gas cost and execution cost decrease and then remain stable and constant in a certain range. Hence our function built for adding doctors in a smart contract efficiently adds more users.

5.4.2. Based on addition of patient in hospital

Similarly, for adding patients, each patient detail is inserted into the function Add_Patient, and a transaction is created. Seven iterations are done for adding seven different patients into the hospital. Figure9 shows the line graph of the results noted for the gas cost and the execution cost spent in Wei for adding patients. It clearly shows that while adding one patient, the gas incurred is 241587 units, while the simultaneous iterations show the value as 202215 gas, 202229 gas, 2023401 gas, 202174 gas, 202243 gas and 202119 gas units. The execution gas cost expenditure is 210075 units, 175839 units, 175851 units, 175947 units, 175803 units, 175863 units and 175755 units. Thus, this also shows that as more patients are added to the system, the gas cost and execution cost decrease and remain stable and constant in a specific range. Hence our function built for adding patients in a smart contract efficiently adds more users.

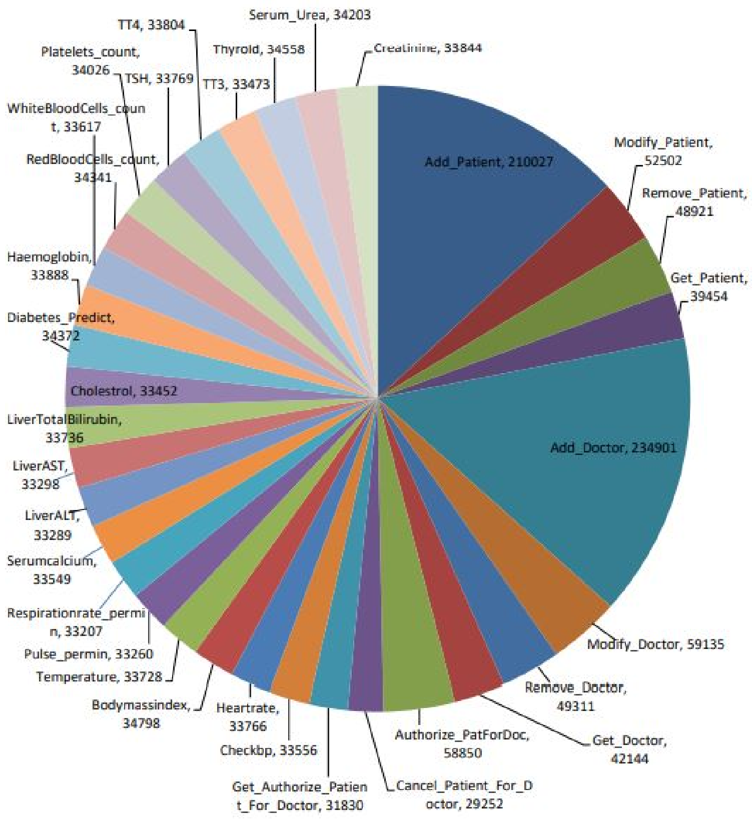

Figure 7: Piechart showing execution gascost incurred from single iteration for different functions.

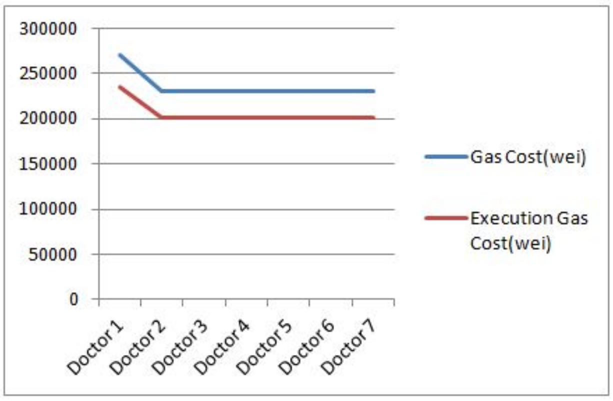

Figure 8: Line graph showing gas cost and execution gas cost spent in wei for adding doctors.

5.4.3. Based on authorization of patient with doctor in hospital

For authorization of patient with doctor, Figure10 clearly shows the gas expenditure incurred for different cases. It uses the acronym doc for doctor and pat. for the patient. For one doctor with one patient relationship, gas incurred is 67627 units, and the execution cost is 58806 units. For one doctor for two patients, the gas cost is 67627 units, and the execution cost is 58806 units. For two doctors of the same specialization for one patient relationship, the gas and execution costs are 67627 units and 58806 units. Similarly, for two doctors of different specializations for one patient relationship, the gas spent is 67655 and 58830 units. In the relationship for one doctor with six patients and one for

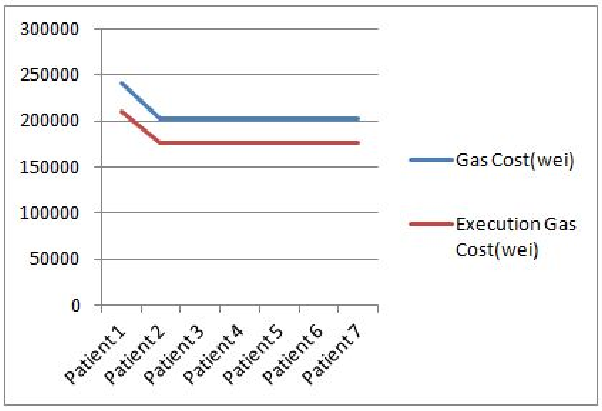

Figure 9: Line graph showing gas cost and execution gas cost spent in wei for adding patients

seven patients, the gas cost incurred is 67655 and 67627 units simultaneously, and execution gas costs spent are 58830 and 58806 units simultaneously. Thus, this shows that the relationships of any type have the almost same range of gas cost spent and hence are efficient and unaffected with any number of users.

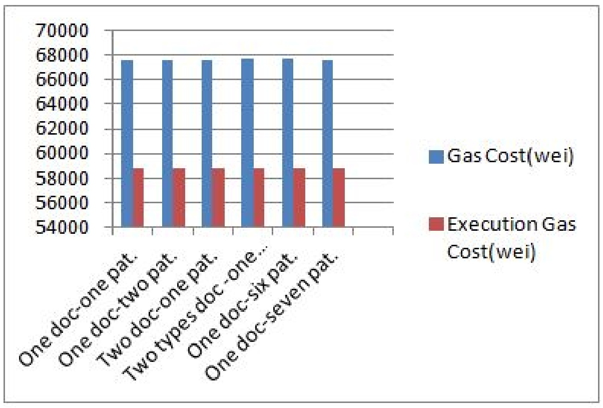

Figure 10: Bar graph showing gas cost and execution gas cost spent in wei for authorization of patient with doctor

5.4.4. Based on parameters checked by physician and pathologist

The system designed has physician and pathologist that checks the patient on the basis of different parameters. Out of which two parameters heart rate and respiration rate of physician and platelet count and serum urea of pathologist are shown here for their analysis. Figure11 and Figure12 shows the gas cost and the execution cost spent for seven different patients that are analyzed on the basis of heart rate and respiration rate respectively. Similarly, Figure13 and Figure14 shows the gas cost and the execution cost spent for seven different patients that are analyzed on the basis of platelet count and serum urea respectively. Thus from the given figures it is clear that with different input values for different patients, the gas spent and execution gas cost lies within the same range of values and also for different patients our function work efficiently without any effect on it's working. Hence, there is no increasing effect on the gas spent and we get the consistent results for different iterations with increasing number of patients.

5.4.5. Based on updation of details in hospital database

The system has a modification function, too, that helps to change any detail regarding the doctor and user. For analysis purposes, the Modify_Doctor function is evaluated. Table4 tabulates the five iterations used to modify doctor

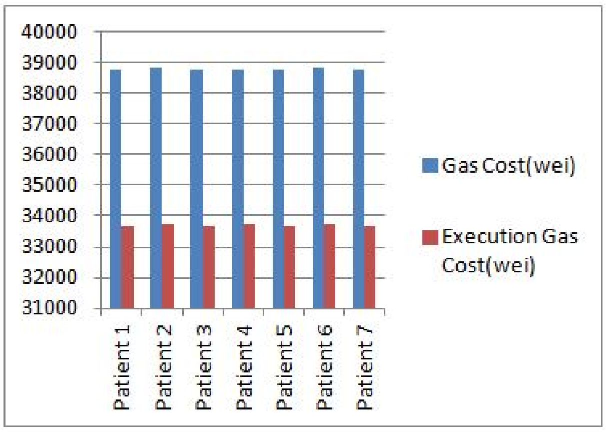

Figure 11: Bar graph showing gas cost and execution gas cost spent in wei for heartrate analysis

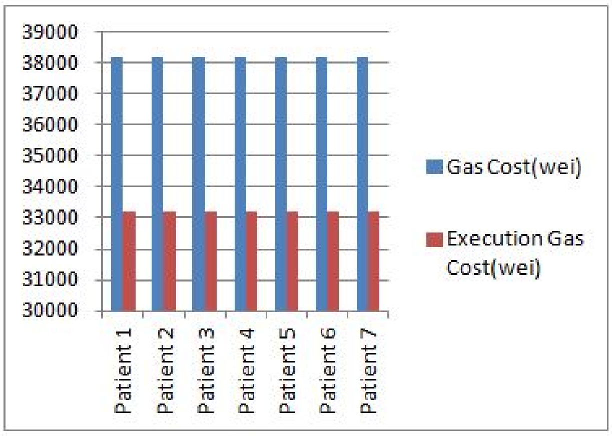

Figure 12: Bar graph showing gas cost and execution gas cost spent in wei for respiration rate analysis

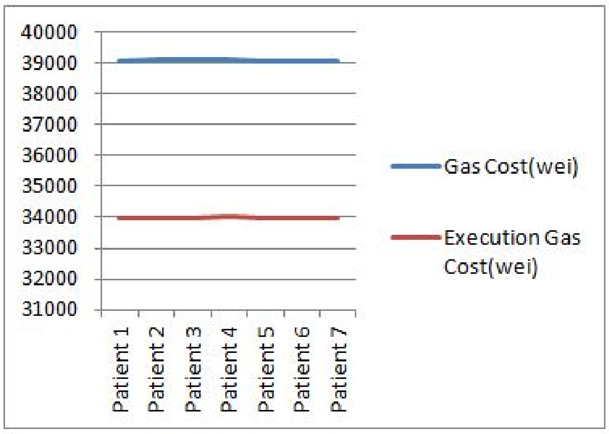

Figure 13: Line graph showing gas cost and execution gas cost spent in wei for platelet analysis

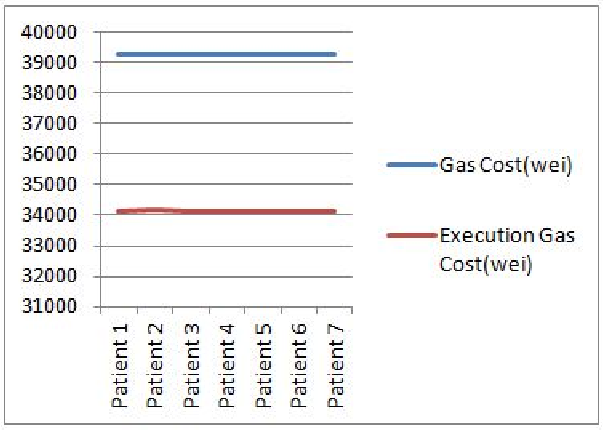

Figure 14: Line graph showing gas cost and execution gas cost spent in wei for serum urea analysis

Table 4: Gas cost and execution cost spent in wei to modify doctor's details.

	Gas Cost(wei)	Execution Gas Cost(wei)
Doctor1	64859	56399
Doctor1	61833	53563
Doctor1	61820	53767
Doctor1	61598	53666
Doctor1	61601	53701

details. There were situations when the name entered in the database was wrong, or one had shifted from one location to another. Hence this function helps update the doctor's address, name, and age. Also, the table clearly shows that with increasing modification iterations, the gas spent first decreases and then remains constant at a certain level. Thus the designed healthcare system is efficient and can be used by many users.

Thus, it is analyzed and shown that the designed smart contract with different functions as the parameters of the healthcare system is an efficient smart contract. With increasing users, our system decreases gas cost consumption and shows consistent performance for different functions. Hence the system modelled can be used with many users' access and thus can be extended in the near future.

6. Conclusion

In current work, a secure BC-based healthcare system is designed using the Ethereum test network. *MyEasyHeathcare* system incorporates twenty-two patient health parameters. BC has the inherent property of security, but the patient-doctor relationship added in the developed system enhances the security enabling it to be used in real applications. The Smart contracts are written in Solidity language, and their efficiency is evaluated regarding gas and execution costs. The obtained results justify the efficiency of the designed smart contracts. The BC-based healthcare system is often preferred to provide security, but the viability of the system is often questioned in terms of gas cost. There is always a trade-off between gas cost and security. The results obtained from different iterations for different scenarios testify that the developed system *MyEasyHeathcare* is sustainable. The developed system's performance justifies its usage in the main Ethereum network.

Data Availability Statement

There is no data available to carry out this research.

Conflict of Interest

The authors want to declare that at the time of submission of this article, there is no conflict of Interest.

References

[1] G. Gambhire, T. Gujar, and S. Pathak, “Business potential and impact of industry 4.0 in manufacturing organizations,” in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–6, IEEE, 2018.

[2] P. Fraga-Lamas and T. M. Fernández-Caramés, “A review on blockchain technologies for an advanced and cyber-resilient automotive industry,” *IEEE Access*, vol. 7, pp. 17578–17598, 2019.

[3] I. Makhdoom, M. Abolhasan, H. Abbas, and W. Ni, “Blockchain’s adoption in iot: The challenges, and a way forward,” *Journal of Network and Computer Applications*, vol. 125, pp. 251–279, 2019.

[4] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Decentralized Business Review*, p. 21260, 2008.

[5] T. Aste, P. Tasca, and T. Di Matteo, “Blockchain technologies: The foreseeable impact on society and industry,” *Computer*, vol. 50, pp. 18–28, 01 2017.

[6] K. Salah, M. H. U. Rehman, N. Nizamuddin, and A. Al-Fuqaha, “Blockchain for ai: Review and open research challenges,” *IEEE Access*, vol. 7, pp. 10127–10149, 2019.

[7] M. Kouhizadeh and J. Sarkis, “Blockchain practices, potentials, and perspectives in greening supply chains,” *Sustainability*, vol. 10, p. 3652, 10 2018.

[8] D. Aloini, E. Benevento, A. Stefanini, and P. Zerbino, “Transforming healthcare ecosystems through blockchain: Opportunities and capabilities for business process innovation,” *Technovation*, p. 102557, 2022.

[9] J. J. Hathaliya, S. Tanwar, S. Tyagi, and N. Kumar, “Securing electronics healthcare records in healthcare 4.0 : A biometric-based approach,” *Computers & Electrical Engineering*, vol. 76, pp. 398–410, 2019.

[10] L. Coventry and D. Branley, “Cybersecurity in healthcare: A narrative review of trends, threats and ways forward,” *Maturitas*, vol. 113, pp. 48–52, 2018.

[11] M. Al-rawashdeh, P. Keikhosrokiani, B. Belaton, M. Alawida, and A. Zwiri, “Iot adoption and application for smart healthcare: A systematic review,” *Sensors*, vol. 22, no. 14, 2022.

[12] H. Hamil, Z. Zidelmal, M. S. Azzaz, S. Sakhi, R. Kaibou, S. Djilali, and D. Ould Abdeslam, “Design of a secured telehealth system based on multiple biosignals diagnosis and classification for iot application,” *Expert Systems*, vol. 39, no. 4, p. e12765, 2022.

[13] J. Hathaliya, P. Sharma, S. Tanwar, and R. Gupta, “Blockchain-based remote patient monitoring in healthcare 4.0,” in *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, pp. 87–91, 2019.

[14] H. L. Pham, T. H. Tran, and Y. Nakashima, “A secure remote healthcare system for hospital using blockchain smart contract,” in *2018 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, 2018.

[15] S. S. Nabil, M. S. Alam Pran, A. A. Al Haque, N. R. Chakraborty, M. J. M. Chowdhury, and M. S. Ferdous, “Blockchain-based covid vaccination registration and monitoring,” *Blockchain: Research and Applications*, p. 100092, 2022.

[16] K. Azbeg, O. Ouchetto, and S. Jai Andaloussi, “Blockmedcare: A healthcare system based on iot, blockchain and ipfs for data management security,” *Egyptian Informatics Journal*, vol. 23, no. 2, pp. 329–343, 2022.

[17] N. Iqbal, F. Jamil, S. Ahmad, and D. Kim, “A novel blockchain-based integrity and reliable veterinary clinic information management system using predictive analytics for provisioning of quality health services,” *IEEE Access*, vol. 9, pp. 8069–8098, 2021.

[18] 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.

[19] M. Bhavin, S. Tanwar, N. Sharma, S. Tyagi, and N. Kumar, “Blockchain and quantum blind signature-based hybrid scheme for healthcare 5.0 applications,” *Journal of Information Security and Applications*, vol. 56, p. 102673, 2021.

[20] Y. Yang, T. Lin, P. Liu, P. Zeng, and S. Xiao, “Ucbis: An improved consortium blockchain information system based on ubccsp,” *Blockchain: Research and Applications*, vol. 3, no. 2, p. 100064, 2022.

[21] I. Yaqoob, K. Salah, R. Jayaraman, and Y. Al-Hammadi, “Blockchain for healthcare data management: Opportunities, challenges, and future recommendations,” *Neural Computing and Applications*, 01 2021.

[22] S. P G, V. Menon, L. Ramasamy, S. Kadry, and Y. Nam, “Blockchain-based secure healthcare application for diabetic-cardio disease prediction in fog computing,” *IEEE Access*, vol. PP, pp. 1–1, 03 2021.

[23] P. P. Ray, D. Dash, K. Salah, and N. Kumar, “Blockchain for iot-based healthcare: Background, consensus, platforms, and use cases,” *IEEE Systems Journal*, vol. 15, no. 1, pp. 85–94, 2021.

[24] M. Shuaib, S. Alam, M. Shabbir Alam, and M. Shahnawaz Nasir, “Self-sovereign identity for healthcare using blockchain,” *Materials Today: Proceedings*, 2021.

[25] I. Omar, R. Jayaraman, K. Salah, I. Yaqoob, and S. Ellahham, “Applications of blockchain technology in clinical trials: Review and open challenges,” *Arabian Journal for Science and Engineering*, vol. 46, 07 2020.

[26] L. Soltanisehat, R. Alizadeh, H. Hao, and K.-K. R. Choo, “Technical, temporal, and spatial research challenges and opportunities in blockchain-based healthcare: A systematic literature review,” *IEEE Transactions on Engineering Management*, pp. 1–16, 2020.

[27] S. Saha, A. K. Sutrala, A. K. Das, N. Kumar, and J. J. P. C. Rodrigues, “On the design of blockchain-based access control protocol for iot-enabled healthcare applications,” in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2020.

[28] A. A. Omar, M. Z. A. Bhuiyan, A. Basu, S. Kiyomoto, and M. S. Rahman, “Privacy-friendly platform for healthcare data in cloud based on blockchain environment,” *Future Generation Computer Systems*, vol. 95, pp. 511–521, 2019.

[29] S. Zhang and J.-H. Lee, “Analysis of the main consensus protocols of blockchain,” *ICT Express*, vol. 6, no. 2, pp. 93–97, 2020.

[30] C. C. Agbo and Q. H. Mahmoud, “Comparison of blockchain frameworks for healthcare applications,” *Internet Technology Letters*, vol. 2, no. 5, p. e122, 2019.

[31] M. Z. A. Bhuiyan, A. Zaman, T. Wang, G. Wang, H. Tao, and M. M. Hassan, “Blockchain and big data to transform the healthcare,” in *Proceedings of the International Conference on Data Processing and Applications*, ICDPA 2018, (New York, NY, USA), p. 62–68, Association for Computing Machinery, 2018.

[32] S. Wang, J. Wang, X. Wang, T. Qiu, Y. Yuan, L. Ouyang, Y. Guo, and F.-Y. Wang, “Blockchain-powered parallel healthcare systems based on the acp approach,” *IEEE Transactions on Computational Social Systems*, vol. 5, no. 4, pp. 942–950, 2018.

[33] N. P. V. Sravan, P. K. Baruah, S. S. Mudigonda, and P. V. Krihsna, "Use of blockchain technology in integrating heath insurance company and hospital," 2018.

[34] K. Koshechkin, G. Klimenko, I. Ryabkov, and P. Kozhin, "Scope for the application of blockchain in the public healthcare of the russian federation," *Procedia Computer Science*, vol. 126, pp. 1323–1328, 2018. Knowledge-Based and Intelligent Information and Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia.

[35] 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, 2017.

[36] S. Raju, V. Rajesh, and J. S. Deogun, "The case for a data bank: An institution to govern healthcare and education," in *Proceedings of the 10th International Conference on Theory and Practice of Electronic Governance*, ICEGOV '17, (New York, NY, USA), p. 538–539, Association for Computing Machinery, 2017.

[37] M. Clinic, "Why choose mayo clinic?." `https://www.mayoclinic.org/`, august 2022.

[38] J. H. MEDICINE, "Vital signs (body temperature, pulse rate, respiration rate, blood pressure)." `https://www.hopkinsmedicine.org/`, august 2022.

[39] C. Clinic, "Vital signs." `https://my.clevelandclinic.org/`, august 2022.

[40] U. H. System, "Center for high quality health care services." `https://www.uclahealth.org/`, august 2022.

[41] V. D'Rozario and P. K. G. Seshadri, "Blood sugar chart." `https://www.medindia.net/patients/calculators/bloodsugar_chart.asp`, august 2022.

[42] M. Sinai, "Mount sinai health system." `https://www.mountsinai.org/`, august 2022.

[43] U. D. of Health and H. Services, "National heart, lung, and blood institute." `https://www.nhlbi.nih.gov/`, august 2022.