SWASTHYA MITRA: SMART CONTRACT ENABLED PATIENT'S ELECTRONIC HEALTH RECORDS SHARING (EHR) SYSTEM

ABSTRACT

Electronic Health Records (EHRs) are very crucial and personal assets of a person and should be shared with medical practitioners for monitoring the health of a person based on the previous history for efficient diagnosis and proper treatment of the disease. Blockchain opens the new horizons sharing of EHRs among medical practitioners with the assurance of security and privacy. It is a decentralized, immutable, and secure architecture without the involvement of a third party in the transaction. The basic aim of the research paper is to propose a fast, temper poof, reliable and transparent EHR approach for sharing medical records on the Ethereum blockchain platform using smart contract without involving a third party. The encrypted medical records can be shared directly between the doctor and the patient. Moreover, medical laboratories can also share the patient test reports with the patient in secure fashion using blockchain. The proposed system will increase confidence in collaborating and sharing EHR with medical practitioners.

KEYWORDS: BLOCKCHAIN, SMART CONTRACT, E-HEALTH, ETHEREUM, ELECTRONIC HEALTH RECORDS, HEALTHCARE SECTOR INNOVATION, PROOF OF STAKE.

1. INTRODUCTION:

Blockchain [1] is an immutable [2] digital ledger to record the transactions across a peer-to-peer [3] network. Blockchain [4] was proposed in 2008 by Satoshi Nakamoto for a virtual currency, Bitcoin. It stores the transactions in blocks and each transaction is validated by the nodes present in the network. This technology ensures data transparency and prevents data forgery. Confidentiality of sensitive data is the prime concern today. In present scenario, people usually trust on mediators to store and secure our confidential information, which can trigger data breaching. Blockchain eliminates the mediators and protects sensitive data using the consensus process. In the blockchain, there is a unique concept of "Hashing" through which blockchain secures itself. Hash behaves like a fingerprint that is responsible for uniquely identifying the blocks in the blockchain. Each block contains its hash value and the hash of the prior block, the first block is known as Genesis Block [4]. Figure 1 shows the generalized structure of Blockchain.

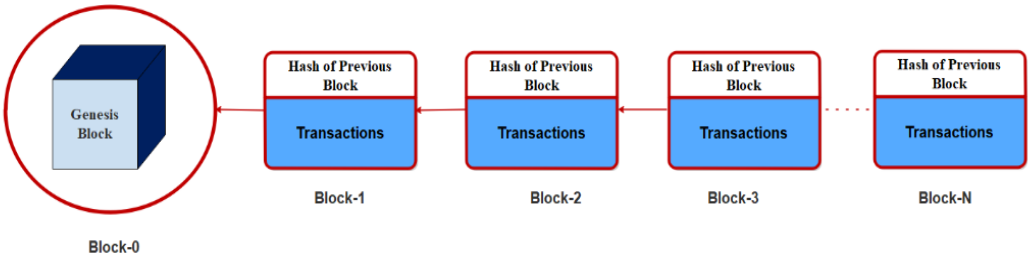

Figure 1: Generalized structure of blockchain network

Modifying a block will cause the hash value to change; this move will make all the present blocks invalid and cause rewriting of the blockchain.

Presently, blockchain is implemented in different industries like supply chain, forecasting, banking, cyber-security, healthcare, etc. Blockchain technology guarantees the secure accessing of patient record in the healthcare systems. The security of a patient's data is a matter of serious concern, according to a Canadian company faced data breaching, which impacted the data of 15 million patients. The increasing ransom attacks in the healthcare sector are also a reason for developing a secure system. Blockchain technology offers a secure platform for storing the medical records of a patient in the form of EHR [5]. Using our proposed system, patient health records can be shared with the doctors and patients can directly get their lab test results anytime and anywhere. In this paper, we use smart contracts on an Ethereum blockchain to create a secure EHR system. In the proposed system smart contract [6] is considered as a vending machine. It is an executable code that sets an agreement between two or more parties and eliminates the mediators as a vending machine does. In a smart contract, all the transactions become irreversible which makes it immutable. The proposed system is deployed on an Ethereum network using Proof of Stake (PoS) consensus algorithm.

Rest of paper organized as follows: section 2 describes consensus algorithms, section 3 briefly introduces Smart Contract, section 4 presents the smart Contract in EHR. section 5 contains the details about the deployment of smart contracts in the proposed system. Finally, section 6 concludes the paper along with work.

2. Consensus Algorithms

Transactions in blockchain are extremely secure despite the fact that there is no central authority present to verify and validate the transactions. This trust is built by using consensus algorithms for all the transactions in a blockchain network. A consensus algorithm is a procedure through which all the nodes of the Blockchain network come to a common agreement about the present state of the distributed ledger. Consensus algorithm is the high mathematical computation that validates all the data blocks on the blockchain network. It makes sure that every new block added to the Blockchain has one and only version of the truth that is agreed upon by most of the nodes in the Blockchain.

There are many types of consensus algorithms proposed by researchers that are used in blockchain. Some of the examples of blockchain consensus algorithms [4] are Proof of Stake (PoS), Proof of Work (PoW), Practical Byzantine Fault Tolerance (PBFT), and Proof of Burn (PoB)etc. Each consensus algorithm has a different use case. On the Ethereum platform the Proof of Stake (PoS) is used to completes all transactions and deploys the final smart contracts. A consensus algorithm is chosen according to the goal that needs to be achieved. The proposed work uses the Proof of Stake (PoS) consensus algorithm to achieve fast and secure sharing of EHRs through blockchain. In PoS, the coin owners offer their coins as collateral for the chance to validate a new block or a transaction by broadcasting the vote that encompasses cryptographic signatures signed. Validators [7] are selected randomly for validation of new block and receives revenue in return for their work. The validators are weighted in accordance with sum of stake they have. If a validator has higher stakes, then it has higher chance to be chosen as a validator. For instance, Ethereum will require 32 ETHs (Ethereum’s native cryptocurrency) to be staked before becoming a validator. Blocks or transactions are validated by more than one validator, and when a specific number of the validators verify that block or transaction, it is finalized and added to the blockchain

network. This makes the entire purpose even more secure. PoS consensus algorithm reduces the amount of computational work needed to validate new blocks of transactions which results in less energy consumption; thus, it is beneficial for the environment as well. Figure 2 shows working of Proof of Work (PoW) algorithm.

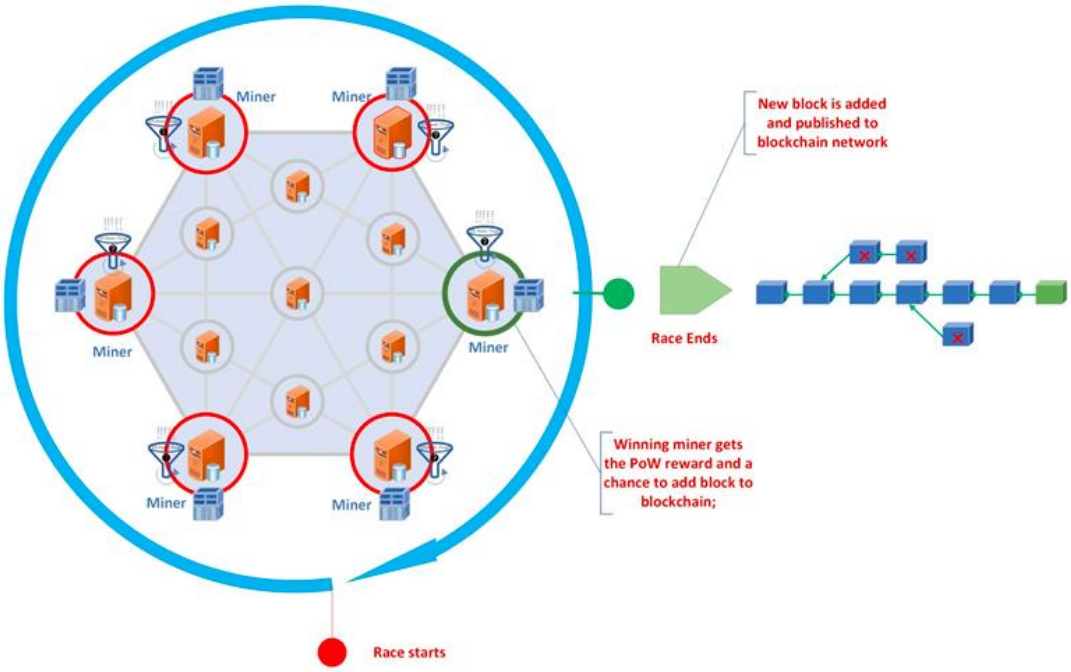

Figure 2: Working of PoW

3. Smart Contracts

Smart contracts are self-executing computer programs and performs the specified actions when predefined conditions are satisfied. It consists of contract storage, program code etc. Smart contract program code is fixed and cannot be updated once uploaded in the blockchain. It generally consists of the terms and conditions of an agreement between the end parties involved in a transaction. Smart contracts enable trusted transactions between anonymous parties without the intervention of a central authority. There is no paperwork or third-party involvement required due to the digital and automated nature of smart contracts [8-9]. Which in turn results in the reduction of costs and malicious activities. Thus, smart contract promotes immutability, security, rapidity, privacy, efficiency, and accuracy in transactions.

The shared records are encrypted and stored in a block. Every block is connected to the previous and subsequent blocks on a distributed ledger. This eliminates the chance of tempering in the records. Furthermore, it adds the security, trust, and transparency factors to the use of smart contracts with blockchain.

Smart contracts are categorized as deterministic smart contracts and non-deterministic smart contracts. A deterministic smart contract is a smart contract that does not require any information from an external source while it is being executed. On the other hand, a non-deterministic smart contract depends on information, known as data feeds, from an external source.

Due to the above-listed benefits of smart contracts, they can prove to be extremely useful in EHR sharing in the health sector [10-13]. In the proposed work, Solidity programming language is used to write deterministic smart contracts.

Sharing of medical records timely and securely is crucial activity. Sometimes, in critical situations patient's health records need to be shared with doctor in real-time fashion to start the treatment on time. Moreover, for experimental analysis, patient's health records are constantly targeted by cybercriminals. Smart contracts may have vital role in such situations. To stop the third parties from exploiting the data, smart contracts can be used to share and manage EHR securely. Smart Contracts leads to provide the high accessibility to EHR records which is securely share between the pathology labs, doctors, medical stores etc. over the distributed network.

4. Implementation of Smart contracts in Electronic Health Record (EHR) System

In this paper, the proposed work uses a multiple smart contract-based system for the sharing of EHR via blockchain. The paper includes three actors, patient, pathology lab, and doctor. Furthermore, four smart contracts are created for deployment in the proposed scenario. These smart contracts are name as patient to lab, lab to patient, patient to doctor and doctor to patient. Figure 3 shows working and entities of Swasthya Mitra.

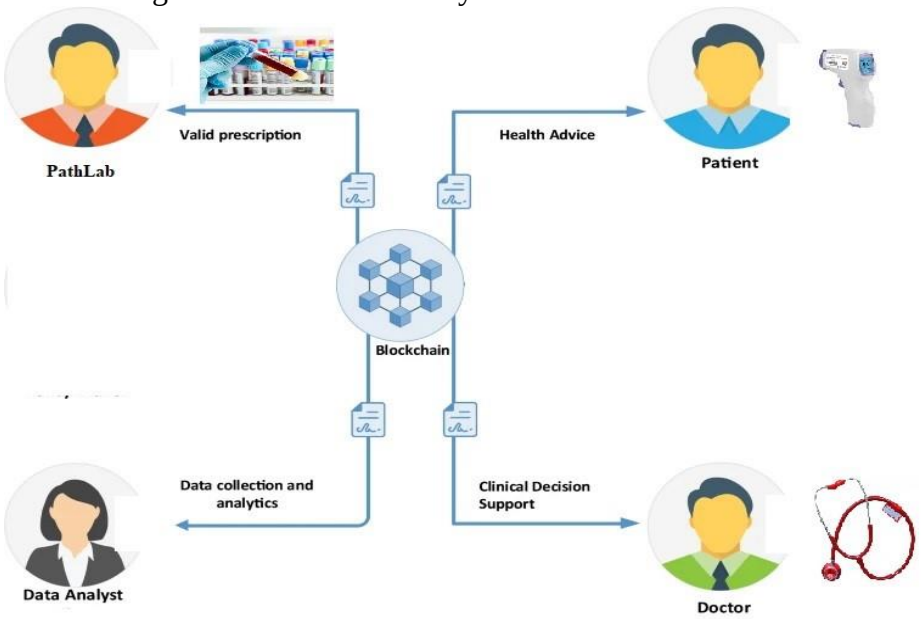

Figure 3: Working and entities of Swasthya Mitra

5. Deployment of Smart Contract in Electronic Health Record (EHR) System

This section deals with deployment of smart contract in the proposed Electronic Health Record (EHR) System. The proposed system includes total six smart contracts, which are discussed as follows:

5.1 Patient-to-Doctor Smart Contract

In this contract, the patient will share his/her information i.e, health issues with the doctor. The smart contract will include the fields such as DoctorName for the name of a doctor, problemDescription for describing the health issues of a patient, Doctor field for the 16-bit address of the doctor, Patient_Name for the name of the patient, Medicine_Data for the suggested medicines, Date_of_Prescription for the date of prescription and Consult_Fee for the consultation charges. The owner of this contract will be a Patient. Figure 4 shows code of patient-to-doctor smart contract and Figure 5 shows output/interface of patient-to-doctor smart contract.

```
pragma solidity ^0.4.25;

//Declare contract having the name 'File'.
contract File{
    //Declare the structure of the contract.
    struct FileInfo{
        string DoctorName;
        string problemDescription;
        address Doctor;
        string Patient_Name;
        string Medicine_Data;
        string Date_of_Prescription;
        uint Consult_Fee;
    }

    address public Patient; //Declare address of Patient in the contract.

    //Maping the address with structure of contract.
    mapping(address => FileInfo) public fileinfos;
    //Fileinfo[] public savingmap;

    //Call a constructor for sending the data.
    constructor() public{
        Patient = msg.sender;
        }

    //Set the permission
    modifier onlyPatient(){
        require(msg.sender == Patient,
    "Only Patient can set these parameteres."
    );
    _;

    }

    //Enable the event of the contract.
    event files(string DoctorName, string problemDescription, address Doctor,
string Patient_Name, string Medicine_Data, string Date_of_Prescription, uint
Consult_Fee);

    //Call a function to set the information in the contract.
    function setFileInfo(address _Patient, string DoctorName, string
problemDescription, address Doctor, string Patient_Name, string Medicine_Data,
string Date_of_Prescription, uint Consult_Fee)
        onlyPatient public{

        fileinfos[_Patient] = FileInfo( DoctorName, problemDescription, Doctor,
tient_Name, Medicine_Data, Date_of_Prescription, Consult_Fee );

        emit files(DoctorName, problemDescription, Doctor, Patient_Name,
dicine_Data, Date_of_Prescription, Consult_Fee);
    }
```

Figure 4: Code of the Patient-to-Doctor Smart Contract

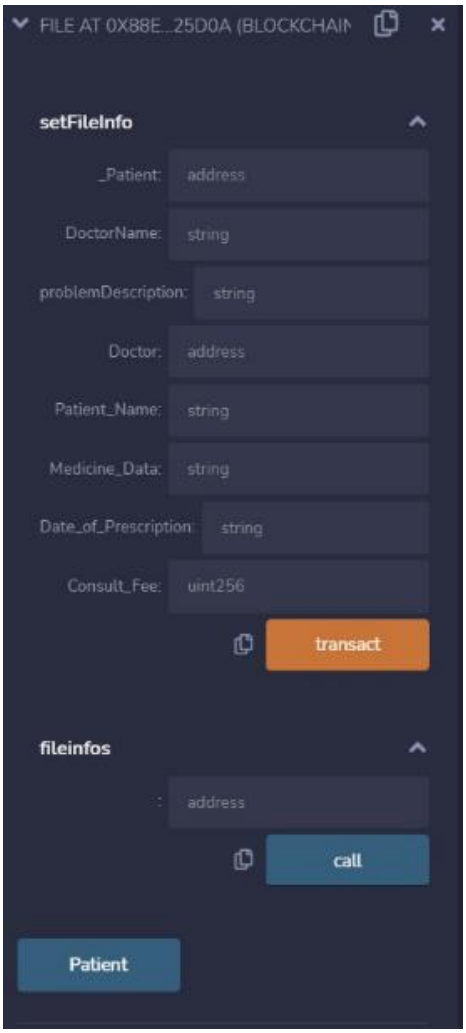

Figure 5: Interface of patient-to-doctor Smart Contract

After the execution of the code, the final smart contract will be generated. All the existing information in the smart contract will be encrypted. The original data can only be viewed by the 'Patient' and the 'Doctor', and the contract can be viewed by everyone. Figure 6 shows the final deployed Patient-to-Doctor smart contract.

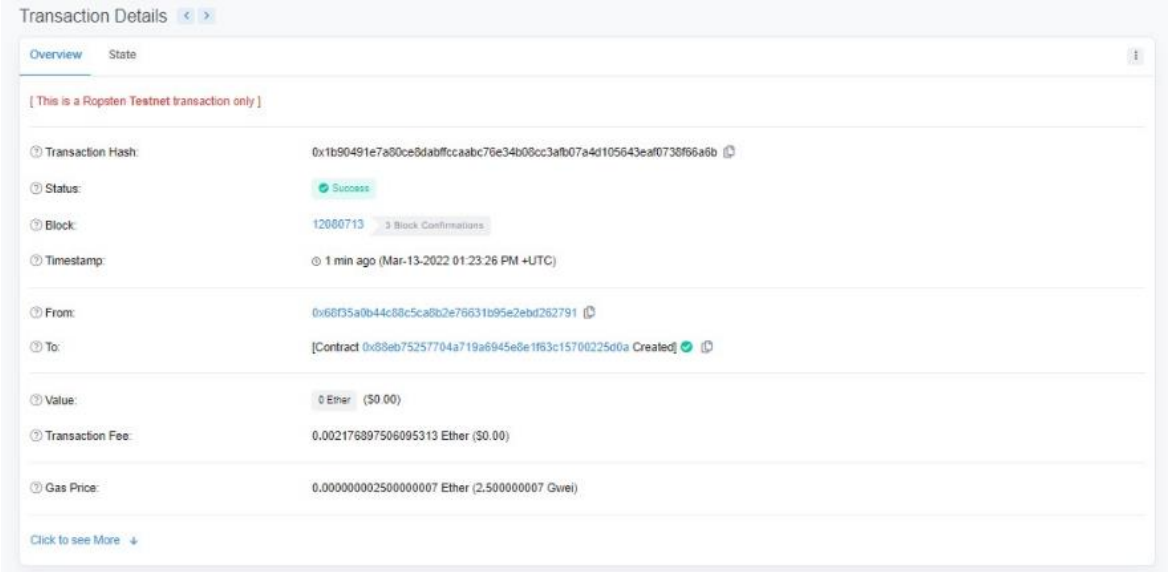

Figure 6: Deployment result of patient-to-doctor Smart Contract

5.2 Doctor-to-Patient Smart Contract

In this contract, the doctor will share the prescription with the patient. The smart contract will include the fields such as DoctorName for the name of a doctor, Patient for the 16-bit address of the patient, Patient_Name for the name of a patient, numberOfTestsToPerform for the number of medical tests, testNames for the name of the test and the Problem_disease for the name of the disease. The owner of this contract will be a doctor. Figure 7 shows code of doctor-to-patient smart contract and Figure 5 shows output/interface of doctor-to-patient smart contract.

```
pragma solidity ^0.4.25;

//Declare contract having the name 'File'.
contract File{
    //Declare the structure of the contract.
    struct FileInfo{
        string DoctorName;
        address Patient;
        string Patient_Name;
        string numberOfTestsToPerform;
        string testNames;
        string Problem_disease;
    }

    address public Doctor; //Declare address of Doctor in the contract.

    //Maping the address with structure of contract.
    mapping(address => FileInfo) public fileinfos;
    //Fileinfo[] public savingmap;

    //Call a constructor for sending the data.
    constructor() public{
        Doctor = msg.sender;
        }

    //Set the permission
    modifier onlyDoctor(){
        require(msg.sender == Doctor,
    "Only Doctor can set these parameteres."
    );
    _;

    }

    //Enable the event of the contract.
    event files(string DoctorName,
        address Patient,
        string Patient_Name,
        string numberOfTestsToPerform,
        string testNames,
        string Problem_disease);

    //Call a function to set the information in the contract.
    function setFileInfo(address _Doctor, string DoctorName,
        address Patient,
        string Patient_Name,
        string numberOfTestsToPerform,
        string testNames,
        string Problem_disease)
        onlyDoctor public{

        fileinfos[_Doctor] = FileInfo( DoctorName,
         Patient,
         Patient_Name,
         numberOfTestsToPerform,
         testNames,
         Problem_disease);

        emit files(DoctorName,
         Patient,
         Patient_Name,
         numberOfTestsToPerform,
         testNames,
         Problem_disease);
    }


}
```

Figure 7: Code of the Doctor-to-Patient Smart Contract

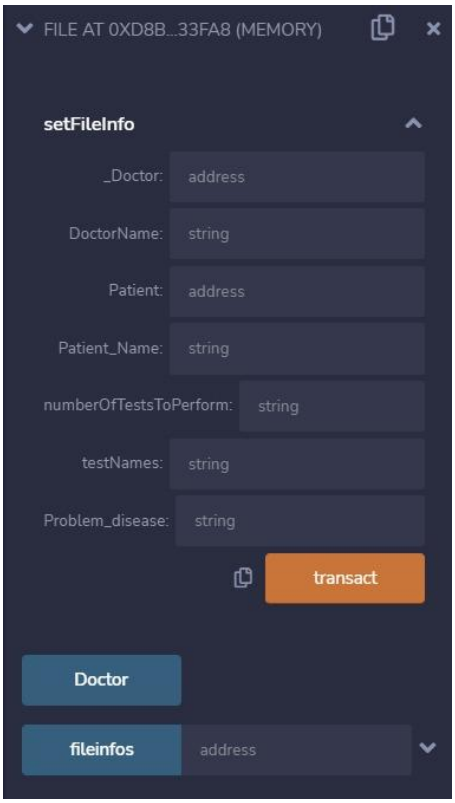

Figure 8: Interface of Doctor-to-Patient Smart Contract

After the execution of the code, the final smart contract will be generated. All the existing information in the smart contract will be encrypted. The original data can only be viewed by the 'Doctor' and the 'Patient' but the contract can be viewed by everyone.

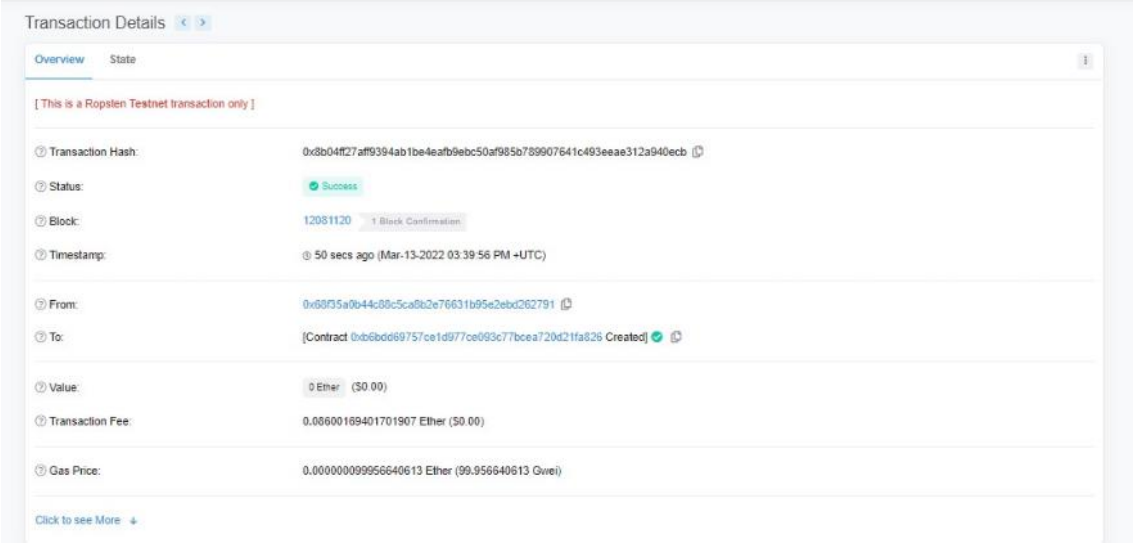

Figure 9: Deployment result of the Doctor-to-Patient contract

5.3 Patient-to-Lab Smart Contract

In this contract, the patient will share his/her medical test data with the lab. The smart contract will include the fields such as DoctorName for the name of a doctor, Lab for the 16-bit address of the lab, Lab_Name for the name of the lab, Patient_Name for the name of a patient, numberOfTestsToPerform for the number of medical tests, testNames for the name of the test and the Problem_disease for the name of the disease. The owner of this contract will be a Patient. Figure 10 shows code of patient-to-lab smart contract and Figure 11 shows output/interface of patient-to-lab smart contract.

```
pragma solidity ^0.4.25;

//Declare contract having the name 'File'.
contract File{
    //Declare the structure of the contract.
    struct FileInfo{
        string DoctorName;
        address Lab;
        string Lab_Name;
        string Patient_Name;
        string numberOfTestsToPerform;
        string testNames;
        string dateOfTest;
    }

    address public Patient; //Declare address of Patient in the contract.

    //Maping the address with structure of contract.
    mapping(address => FileInfo) public fileinfos;
    //Fileinfo[] public savingmap;

    //Call a constructor for sending the data.
    constructor() public{
        Patient = msg.sender;
        }

    //Set the permission
    modifier onlyDoctor(){
        require(msg.sender == Patient,
    "Only Patient can set these parameteres."
    );
    _;

    }

    //Enable the event of the contract.
    event files(string DoctorName,
        address Lab,
        string Lab_Name,
        string Patient_Name,
        string numberOfTestsToPerform,
        string testNames,
        string dateOfTest);

    //Call a function to set the information in the contract.
    function setFileInfo(address _Patient, string DoctorName,
        address Lab,
        string Lab_Name,
        string Patient_Name,
        string numberOfTestsToPerform,
        string testNames,
        string dateOfTest)

        onlyDoctor public{

        fileinfos[_Patient] = FileInfo( DoctorName, Lab, Lab_Name, Patient_Name,
numberOfTestsToPerform, testNames, dateOfTest );

        emit files(DoctorName, Lab, Lab_Name, Patient_Name,
numberOfTestsToPerform, testNames, dateOfTest);
    }


}
```

Figure 10: Code of Patient-to-Lab smart contract

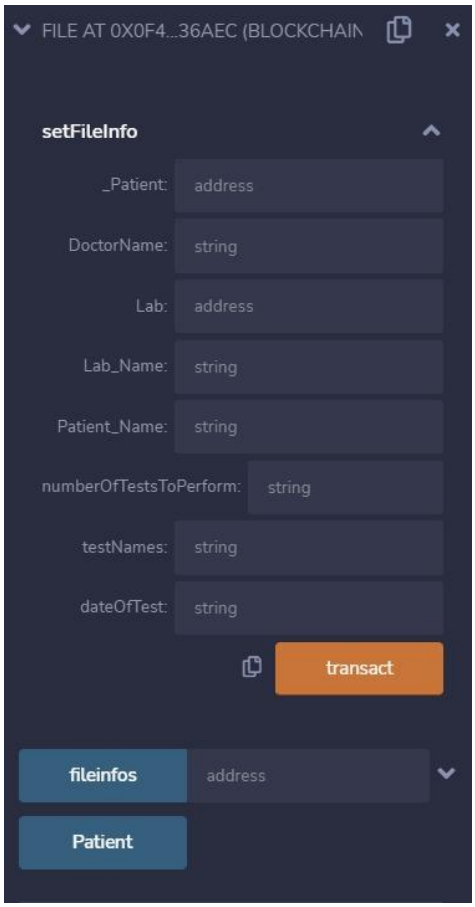

Figure 11: Interface of Patient-to-Lab smart contract

After the execution of the code, the final smart contract will be generated. All the existing information in the smart contract will be encrypted. The original data can only be viewed by the 'Patient' and the 'Lab' but the contract can be viewed by everyone. Figure 12 shows Deployment result of patient-to-lab contract.

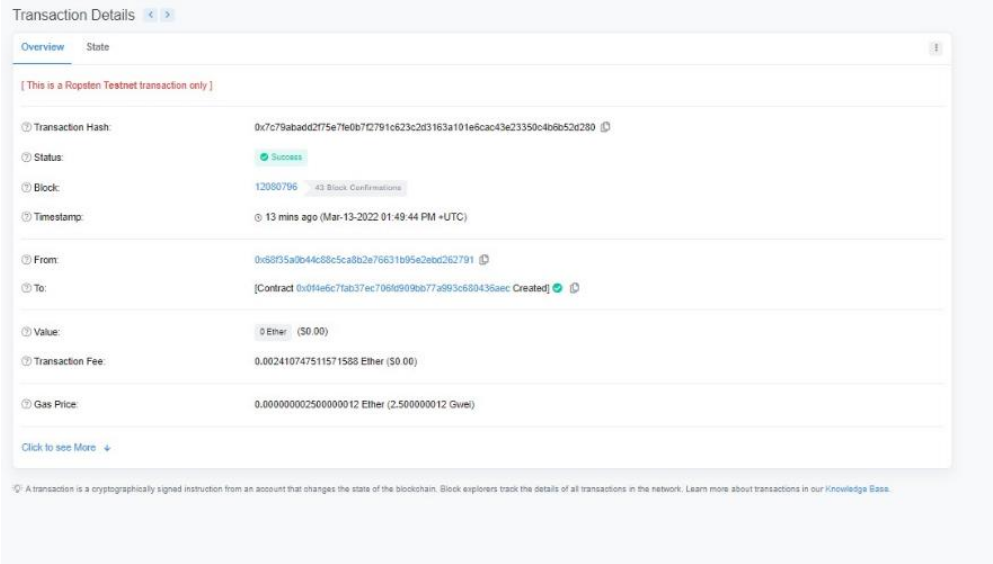

Figure 12 Deployment result of patient-to-lab contract

5.4. Lab-to-Patient Smart Contract

In this contract, the lab will share the results of the test with the patient. The smart contract will include the fields such as DoctorName for the name of a doctor, Lab_Name for the name of the lab, Patient_Name for the name of a patient, Patient for the 16-bit address of the patient, Date_of_Test for the date of the test, testReport_Data for the data of test report and Final_testResult for the final results of the test. The owner of this contract will be a Lab. Figure 13 shows code of Lab-to-Patient smart contract and Figure 14 shows output/interface of Lab-to-Patient smart contract.

```
pragma solidity ^0.4.25;

//Declare contract having the name 'File'.
contract File{
    //Declare the structure of the contract.
    struct FileInfo{
        string DoctorName;
        string Lab_Name;
        string Patient_Name;
        address Patient;
        string Date_of_Test;
        string testReport_Data;
        string Final_testResult;
    }

    address public Lab; //Declare address of Lab in the contract.

    //Maping the address with structure of contract.
    mapping(address => FileInfo) public fileinfos;
    //Fileinfo[] public savingmap;

    //Call a constructor for sending the data.
    constructor() public{
        Lab = msg.sender;
        }

    //Set the permission
    modifier onlyLab(){
        require(msg.sender == Lab,
    "Only Lab can set these parameteres."
    );
    _;

    }

    //Enable the event of the contract.
    event files(string DoctorName,
        string Lab_Name,
        string Patient_Name,
        address Patient,
        string Date_of_Test,
        string testReport_Data,
        string Final_testResult);

    //Call a function to set the information in the contract.
    function setFileInfo(address _Lab, string DoctorName,
        string Lab_Name,
        string Patient_Name,
        address Patient,
        string Date_of_Test,
        string testReport_Data,
        string Final_testResult)
        onlyLab public{

        fileinfos[_Lab] = FileInfo(  DoctorName,
         Lab_Name,
         Patient_Name,
         Patient,
         Date_of_Test,
         testReport_Data,
         Final_testResult);

        emit files( DoctorName,
         Lab_Name,
         Patient_Name,
         Patient,
         Date_of_Test,
         testReport_Data,
         Final_testResult);
    }


}
```

Figure 13: Code of the Lab-to-Patient smart contract

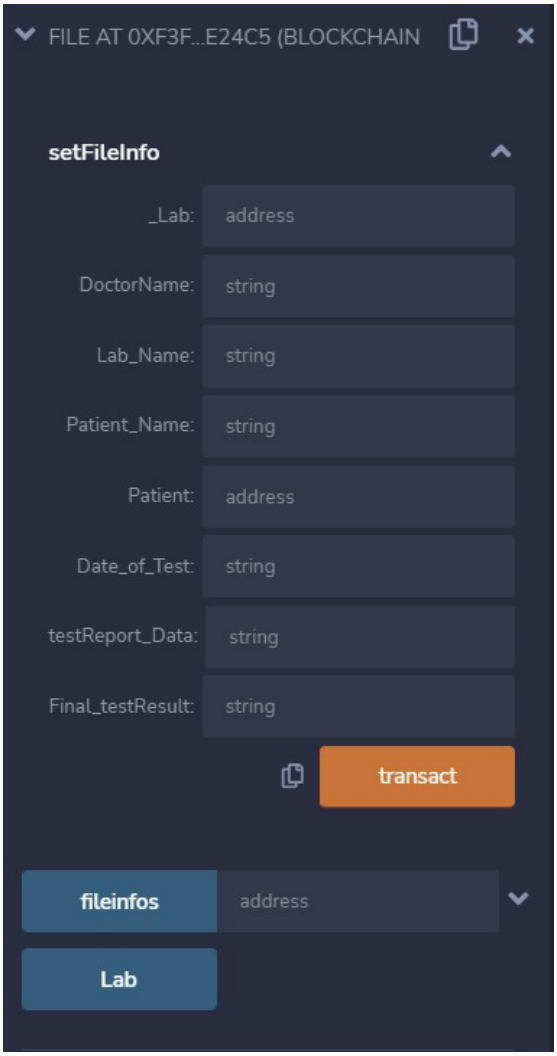

Figure 14: Interface of Lab-to-Patient smart contract

After the execution of the code, the final smart contract will be generated. All the existing information in the smart contract will be encrypted. The original data can only be viewed by the ‘Lab’ and the ‘Patient’ but the contract can be viewed by everyone. Figure 15 shows deployment result of patient-to-lab contract. Figure 15 shows deployment result of patient-to-lab contract.

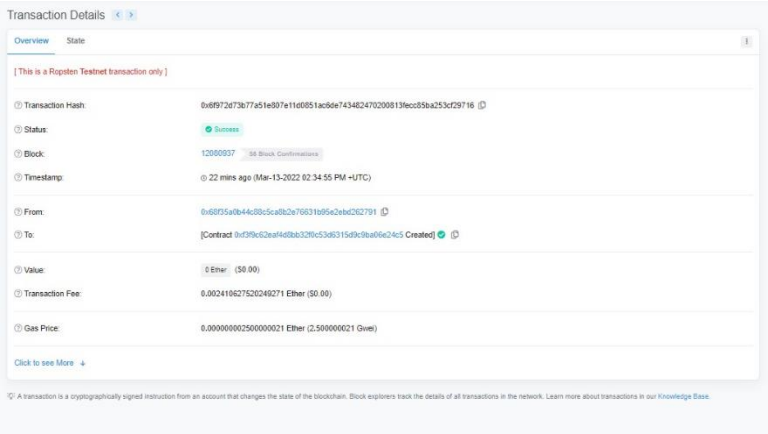

Figure 15: Deployment result of Lab-to-Patient contract

5.5. Patient-to-Doctor Smart Contract

In this contract, the patient will share the lab test data with the doctor. The smart contract will include the fields such as DoctorName for the name of the doctor, Patient_Name for the name of the patient, Doctor for the 16-bit address of the doctor, Lab_Name for the name of the lab, lab for the 16-bit address of the lab, Date_of_Test for the date of the test, testReport_Data for the data of test report and the Final_testResult for the final results of the test. The owner of this contract will be a Patient. Figure 16 shows code Patient-to-Doctor smart contract and Figure 17 shows output/interface of patient-to-Doctor smart contract.

```
pragma solidity ^0.4.25;

//Declare contract having the name 'File'.
contract File{
    //Declare the structure of the contract.
    struct FileInfo{
        string DoctorName;
        string Patient_Name;
        address Doctor;
        string Lab_Name;
        address lab;
        string Date_of_Test;
        string testReport_Data;
        string Final_testResult;
    }

    address public Patient; //Declare address of Patient in the contract.

    //Maping the address with structure of contract.
    mapping(address => FileInfo) public fileinfos;
    //Fileinfo[] public savingmap;

    //Call a constructor for sending the data.
    constructor() public{
        Patient = msg.sender;
        }

    //Set the permission
    modifier onlyPatient(){
        require(msg.sender == Patient,
    "Only Patient can set these parameteres."
    );
    _;

    }

    //Enable the event of the contract.
    event files(string DoctorName,
        string Patient_Name,
        address Doctor,
        string Lab_Name,
        address lab,
        string Date_of_Test,
        string testReport_Data,
        string Final_testResult);
```

```
    //Call a function to set the information in the contract.
    function setFileInfo(address _Patient, string DoctorName,
        string Patient_Name,
        address Doctor,
        string Lab_Name,
        address lab,
        string Date_of_Test,
        string testReport_Data,
        string Final_testResult)
        onlyPatient public{

        fileinfos[_Patient] = FileInfo( DoctorName,
         Patient_Name,
         Doctor,
         Lab_Name,
         lab,
         Date_of_Test,
         testReport_Data,
         Final_testResult);

        emit files(DoctorName,
         Patient_Name,
         Doctor,
         Lab_Name,
         lab,
         Date_of_Test,
         testReport_Data,
         Final_testResult);
    }
```

Figure 16: Code of the Patient-to-Doctor smart contract

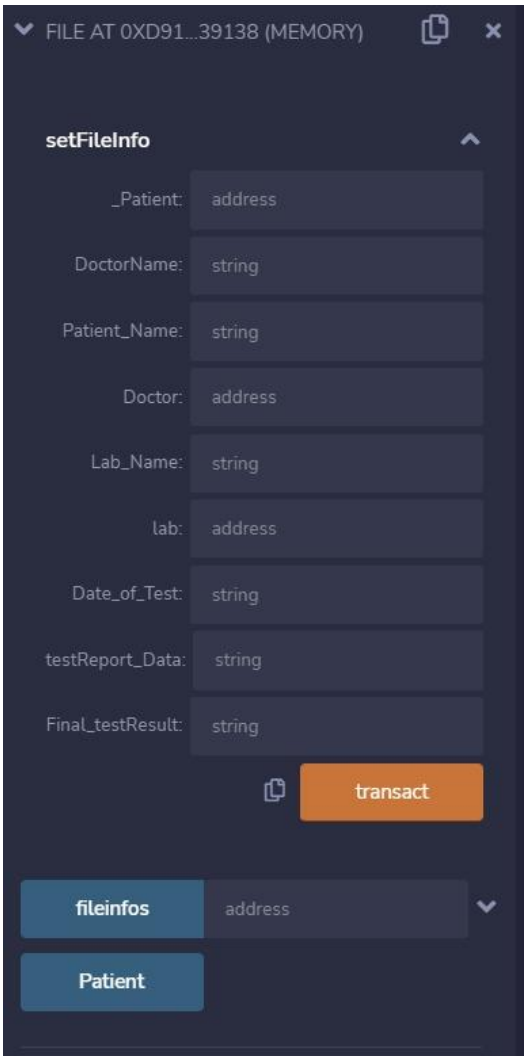

Figure 17: Interface of patient-to-doctor smart contract

After the execution of the code, the final smart contract will be generated. All the existing information in the smart contract will be encrypted. The original data can only be viewed by the ‘Patient’ and the ‘Doctor’ but the contract can be viewed by everyone. Figure 18 shows deployment result of patient-to-Doctor smart contract.

`

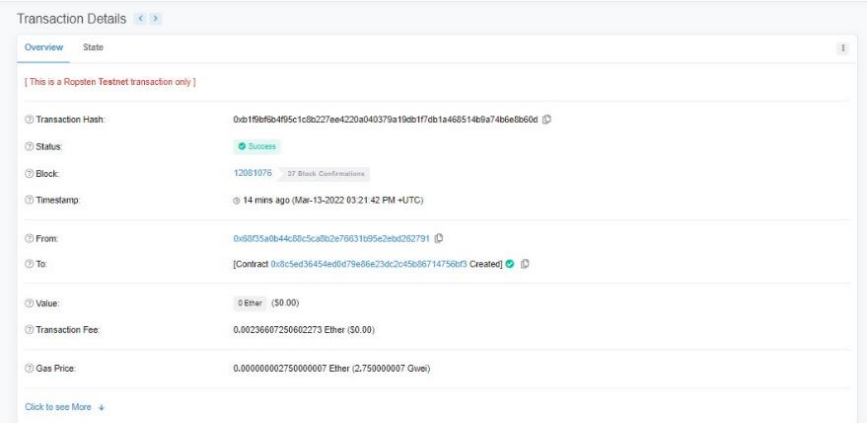

Figure 18: Deployment of the Patient-to-Doctor smart contract

5.6. Doctor-to-Patient Smart Contract

In this contract, the doctor will share the prescription with the patient. The smart contract will include the fields such as DoctorName for the name of a doctor, Patient for the 16-bit address of a patient, Patient_Name for the name of a patient, Medicine_Prescription for the prescribed medicines, testResult for the results of the test, Problem_disease for describing the health issues of a patient, Date_of_Prescription for the date of the prescription and the Consult_Fee for the consultation charges. The owner of this contract will be a doctor. Figure 19 shows code Patient-to-Doctor smart contract and Figure 20 shows output/interface of patient-to-Doctor smart contract.

```
pragma solidity ^0.4.25;

//Declare contract having the name 'File'.
contract File{
    //Declare the structure of the contract.
    struct FileInfo{
        string DoctorName;
        address Patient;
        string Patient_Name;
        string Medicine_Prescription;
        string testResult;
        string Problem_disease;
        string Date_of_Prescription;
        uint Consult_Fee;
    }

    address public Doctor; //Declare address of Doctor in the contract.

    //Maping the address with structure of contract.
    mapping(address => FileInfo) public fileinfos;
    //Fileinfo[] public savingmap;

    //Call a constructor for sending the data.
    constructor() public{
        Doctor = msg.sender;
        }

    //Set the permission
    modifier onlyDoctor(){
        require(msg.sender == Doctor,
    "Only Doctor can set these parameteres."
    );
    _;

    }

    //Enable the event of the contract.
    event files(string DoctorName, address Patient, string Patient_Name, string
Medicine_Prescription, string Problem_disease, string Date_of_Prescription,
string testResult, uint Consult_Fee);

    //Call a function to set the information in the contract.
    function setFileInfo(address _Doctor, string DoctorName, address Patient,
string Patient_Name, string Medicine_Prescription, string Problem_disease, string
Date_of_Prescription, string testResult, uint Consult_Fee)
        onlyDoctor public{

        fileinfos[_Doctor] = FileInfo(DoctorName, Patient,  Patient_Name,
Medicine_Prescription, Problem_disease, Date_of_Prescription,  testResult,
Consult_Fee );

        emit files(DoctorName, Patient, Patient_Name, Medicine_Prescription,
Problem_disease, Date_of_Prescription, testResult, Consult_Fee);
    }


}
```

Figure 19: Code for Doctor-to-Patient smart contract

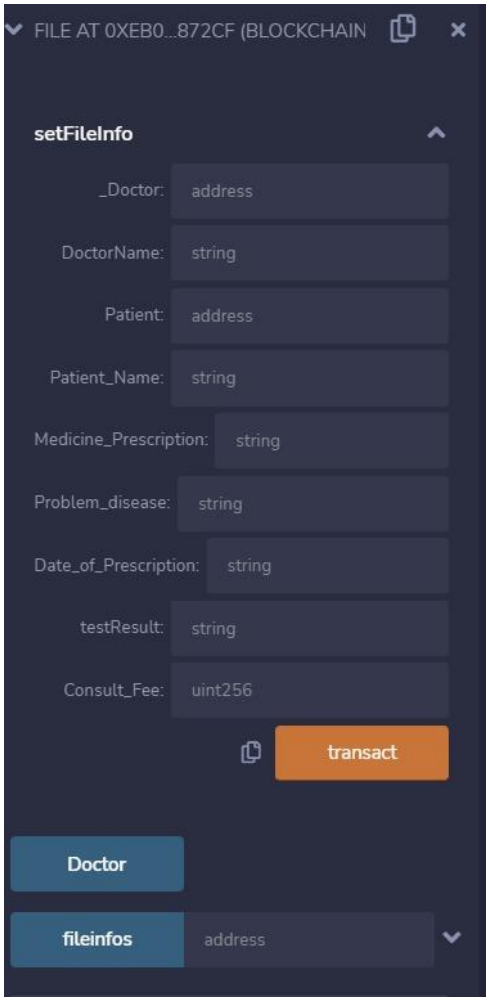

Figure 20: Interface of patient-to-doctor smart contract

After the execution of the code, the final smart contract will be generated. All the existing information in the smart contract will be encrypted. The original data can only be viewed by the 'Doctor' and the 'Patient' but the contract can be viewed by everyone. Figure 21 shows deployment result of doctor-to-patient contract smart contract.

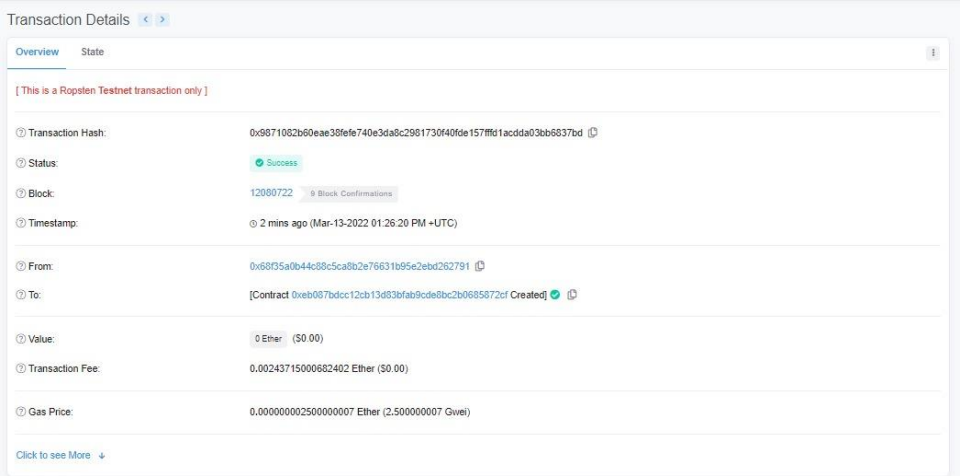

Figure 21: Deployment of the Doctor-to-Patient smart contract

6. Conclusion

The main aim of this research paper is to present an efficient and secure health record sharing system using blockchain technology to enhance the present systems and how blockchain solves the complex process of handling the sensitive data of a patient through smart contracts. Blockchain is a highly accepted and endorsed technology because of its decentralized infrastructure. This peer-to-peer technology is immutable and ensures data integrity. For the implementation of the smart contract enabled EHR sharing system, the deployment of smart contracts has been done on an Ethereum network by using the Proof-of-stake (PoS) consensus algorithm. EHRs are important assets of a patient. Smart contract based EHRs ensure the confidentiality of sensitive data and better treatment of a patient. This paper introduces an EHR system that helps patients to share their medical history with healthcare entities. The proposed system ensures the fast sharing of lab reports with the patients and doctors, it also eliminates data breaching. This study provides knowledge about blockchain and smart contracts in the health sector. The future work will be focused on the development of a proper GUI system to enhance the present system and the possibility of switching from Ethereum blockchain to Hyperledger.

References

[1] Bahar Houtan; Abdelhakim Senhaji Hafid; Dimitrios Makrakis, “A Survey on Blockchain-Based Self-Sovereign Patient Identity in Healthcare”, IEEE Access, vol. 8, pp. 90478–90494, 2020, DoI: 10.1109/ACCESS.2020.2994090.

[2] Griggs, K.N.; Ossipova, O.; Kohlios, C.P.; Baccarini, A.N.; Howson, E.A.; Hayajneh, T., “Healthcare Blockchain System Using Smart Contracts for Secure Automated Remote Patient Monitoring”. J. Med. Syst. vol. 42, article.130, 2018, DoI: 10.1007/s10916-018-0982-x

[3] X. Zhu and Y. Badr, ‘‘Identity management systems for the Internet of Things: A survey towards blockchain solutions,’’ Sensors, vol. 18, no. 12, pp. 1-18, 2018. DoI: https://doi.org/10.3390/s1812421

[4] Naresh Chandra, Vipin Kumar and Arun Kumar Tripathi, “A Deep Investigation on Blockchain Network based on Platforms and Consensus Algorithms”, International Journal of Advanced Science and Technology, vol. 29, issue 8s, pp. 3614 – 3629, 2020.

[5] M. Quinn, J. Forman, M. Harrod, S. Winter, K. E. Fowler, S. L. Krein, A. Gupta, S. Saint, H. Singh, and V. Chopra, ‘‘Electronic health records, communication, and data sharing: Challenges and opportunities for improving the diagnostic process,’’ Diagnosis, vol. 6, no. 3, pp. 241–248, Aug. 2019.
[6] Apoorv Jain, Arun Kumar Tripathi, Naresh Chandra and P. Chinnasamy, “Smart Contract enabled Online Examination System Based in Blockchain Network”, IEEE International Conference on Computer Communication and Informatics (ICCCI), 27-29 Jan. 2021
[7] T. McGhin, K.-K.-R. Choo, C. Z. Liu, and D. He, ‘‘Blockchain in healthcare applications: Research challenges and opportunities,’’ J. Netw. Comput. Appl., vol. 135, pp. 62–75, 2019.
[8] I. Berges, J. Bermudez, and A. Illarramendi, ‘‘Toward semantic interoperability of electronic health records,’’ IEEE Trans. Inf. Technol. Biomed., vol. 16, no. 3, pp. 424–431, May 2012.
[9] Ajay Kumar Shrivastava, Akash Rajak, Chetan and Arun Kumar Tripathi, "A Decentralized way to Store and Authenticate Educational Documents on Private Blockchain", IEEE International Conference ICICT-2019, 27-28 September 2019.
[10] De Aguiar EJ, Faic¸al BS, Krishnamachari B, Ueyama J (2020) A survey of blockchain-based strategies for healthcare. ACM Comput Surv (CSUR) 53(2):1–27.
[11] Guo, R.; Shi, H.; Zhao, Q.; Zheng, D. Secure attribute-based signature scheme with multiple authorities for Blockchain in electronic health records systems. IEEE Access 2018, 776, 1–12
[12] Azaria, A.; Ekblaw, A.; Vieira, T.; Lippman, A. Medrec: Using blockchain for medical data access and permission management. In Proceedings of the 2016 2nd International Conference on Open and Big Data (OBD), Vienna, Austria, pp. 25–30, 2016.
[13] Dwivedi, A.D.; Srivastava, G.; Dhar, S.; Singh, R. A decentralized privacy-preserving healthcare blockchain for IoT. Sensors, 2019, 19, 326, doi:10.3390/s19020326