--> 39 citations

LOTS OF GOOD EXAMPLES FOR PREVIOUS WORK SECTION !!!!!!!!!!!!!

A Smart-Contract-Based Access Control Framework for Cloud Smart Healthcare System

Akanksha Saini, *Graduate Student Member, IEEE*, Qingyi Zhu, *Member, IEEE*, Navneet Singh, Yong Xiang, *Senior Member, IEEE*, Longxiang Gao, *Senior Member, IEEE*, and Yushu Zhang, *Member, IEEE*

looks 2 good

***Abstract*—In current healthcare systems, electronic medical records (EMRs) are always located in different hospitals and controlled by a centralized cloud provider. However, it leads to single point of failure as patients being the real owner lose track of their private and sensitive EMRs. Hence, this article aims to build an access control framework based on smart contract, which is built on the top of distributed ledger (blockchain), to secure the sharing of EMRs among different entities involved in the smart healthcare system. For this, we propose four forms of smart contracts for user verification, access authorization, misbehavior detection, and access revocation, respectively. In this framework, considering the block size of ledger and huge amount of patient data, the EMRs are stored in cloud after being encrypted through the cryptographic functions of elliptic curve cryptography (ECC) and Edwards-curve digital signature algorithm (EdDSA), while their corresponding hashes are packed into blockchain. The performance evaluation based on a private Ethereum system is used to verify the efficiency of proposed access control framework in the real-time smart healthcare system.**

***Index Terms*—Access control, blockchain, cloud storage, Ethereum, smart contracts, smart healthcare.**

I. Introduction

Due to the rapid advancement of Internet-of-Things (IoT) technologies, an increasing number of smart healthcare gadgets provide greater access to electronic medical records (EMRs) than before for monitoring patient's regular health and recommending treatments. Although this IoT-enabled approach to e-Health is useful in the medical field, there are a lot of concerns about the confidentiality and integrity of accessing smart healthcare data [1], [2]. In addition, although the EMRs are conventionally recorded in centralized cloud health registries of a healthcare provider, they are managed by third-party authorities, which constitutes a high risk for a security breach to occur. Furthermore, there is a fundamental issue of interoperability among different healthcare providers such as the sharing of healthcare records [3]. The delegation, verification, and revocation of access rights to another healthcare provider is a critical concern while sharing confidential patient data. More challenging, the access should be restricted to authorized entities by keeping it highly confidential and patient-centric [4].

An analysis in [5] shows that the traditional IoT access control models which are predominantly based on established access control models, such as discretionary access control model (DAC) [6], role-based access control model (RBAC) [7], attribute-based access control model (ABAC) [8], and capability-based access control model (CaBAC) [9] validate the access rights through a centralized entity, which is prone to a single point of failure. Thus, decentralized access control has been a key research topic for years. However, current decentralized access control techniques, due to storage of access policies in the cloud, are unable to make the access control policies transparent and secure. To deal with this issue, the concept of distributed IoT has been proposed recently [10], where access authorization is performed at the end devices. However, it is very difficult to enforce as these IoT devices are computationally restrained.

Therefore, a crucial question is how to effectively achieve a decentralized, distributed, and patient-centric access control in a smart healthcare system. Blockchain is proposed as a solution to overcome these security issues encountered in the IoT systems [11]. With fault tolerance, transparency, and security, blockchain technology promotes distributed resource sharing across the whole network. A blockchain platform, when properly designed, can allow cross-domain data sharing and ensure the patients to have secure access to their medical records. Blockchain technology is hyped after the release of Bitcoin, the first digital cryptocurrency [12], and it is a distributed ledger technology (DLT) that can record and replicate transactions across all peers in the network. Each transaction is cryptographically signed and validated by a consensus mechanism before being added to the block and finally appended to the existing blockchain. It is a decentralized peer-to-peer mechanism that eliminates the need to trust a third party or

Manuscript received July 2, 2020; revised September 4, 2020 and October 4, 2020; accepted October 16, 2020. Date of publication October 22, 2020; date of current version March 24, 2021. *(Corresponding author: Yong Xiang.)*

Akanksha Saini, Yong Xiang, and Longxiang Gao are with the Deakin Blockchain Innovation Lab, School of Information Technology, Deakin University, Geelong, VIC 3220, Australia (e-mail: sainiakan@deakin.edu.au; yong.xiang@deakin.edu.au; longxiang.gao@deakin.edu.au).

Qingyi Zhu is with the School of Cyber Security and Information Law, Chongqing University of Posts and Telecommunications, Chongqing 400065, China (e-mail: zhuqy@cqupt.edu.cn).

Navneet Singh is with the Department of Information, Communication, and Electronics Engineering, Catholic University of Korea, Seoul 14662, South Korea (e-mail: navneetsingh@catholic.ac.kr).

Yushu Zhang is with the College of Computer Science, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China (e-mail: yushu@nuaa.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.3032997

an intermediary. In a survey conducted by IBM Institute for Business Value, 56% of healthcare executives interviewed are expected to have a marketable blockchain solution at scale sometime by the end of 2020 [13]. Not only businesses but also countries like Estonia take an interest in blockchain-based healthcare, which would allow its residents to access them in real time [14].

In this article, we use Ethereum [15]-based solution for smart healthcare. Unlike Bitcoin, Ethereum with its smart contract support provides a Turing-complete language that can support a wide range of computational instructions. Smart contracts [16] are self-executing with terms of agreement between two or more parties across blockchain network. They bring several additional advantages while accessing and sharing medical data by healthcare registries such as consent management, autonomy, fine-grain privacy control, transparency, reduced bureaucracy, and expenditure [17]. The smart contract-based access control for IoT systems has been adopted in [18]. Specifically, this work proposes multiple smart contracts to achieve distributed and trustworthy access control. It also provides a case study of smart contract-based access control in an IoT system, where the smart contracts are implemented on the Ethereum smart contract platform. In this article, we strive to design a smart contract-based, distributed, and dynamic access control for EMRs, with patients being epicentre of the system. To facilitate the secure access, EMR can only be accessed and retrieved across the system by authorized users. Besides, our scheme is also compliant with resource-constrained smart healthcare devices.

Our Contribution: The main contributions of this article are as follows.

1) We propose a smart contract-based distributed and dynamic access control mechanism that enables patients to be the real owners of their sensitive EMRs and securely share them. Furthermore, our proposed methodology covers the entire medical system scenario, which mainly encompasses three categories of entities: a) patients; b) hospitals; and c) computationally restrained IoT-enabled smart healthcare devices.
2) To avoid congestion and reduce overhead in the blockchain network, we have utilized cloud as storage. In other words, EMRs are encrypted and stored in the cloud at the time of creation, while the index number and hash of the corresponding EMRs are stored in the blockchain.

nice

3) We implement our scheme on a private Ethereum blockchain. The performance and security analysis clearly prove that our access control is efficient for real-time IoT-enabled smart healthcare system.

The remainder of this article is organized as follows. Section II gives an overview of the related work. Section III explains our proposed system architecture in detail. Section IV introduces our proposed smart control framework to manage access control policies. Section V explains the process of securely uploading and retrieving medical records from our proposed scheme. In Section VI, the implementation and performance analysis are given. Finally, this article is concluded and future work is given in Section VII.

II. Related Work

In this section, we discuss the generalized blockchain-based access control and blockchain-based access control for EMR system.

A. Blockchain-Based Access Control

Access control is an essential component of the information systems and maintains its security by verifying whether a user has the requisite rights to access the services that he/she requests. The current access control schemes face a lack of privacy issues and the presence of a third party. The characteristics of blockchain technology can alleviate these issues and access permissions can be enforced and controlled through the use of smart contracts.

A blockchain-based decentralized and secure personal data management system that can be incorporated into a mobile software development kit (SDK) has been proposed [19]. However, the system's encrypted response has been stored in a trusted third party, which can easily bring the risk of personal data disclosure. Ouaddah *et al.* [20] explored blockchain-based access control manager for IoT that allows users to control access privilege through various transactions, but suffers from a limited computing capability to get them processed. A blockchain-based decentralized run time access monitoring system (DRAMS) has been proposed for distributed access control in a federated cloud environment [21]. It can guarantee data security but does not facilitate efficient and secure sharing of digital records. A scalable access management architecture has been proposed for specific IoT plots in [22]. However, current blockchain technology by its protocol is limited to use as a database and remains an open research challenge. Another such blockchain-based storage system that combines the interplanetary file system (IPFS) and attribute-based encryption (ABE) is proposed to provide fine-grained access control [23]. However, it is not able to revoke the user's attributes or access policy update. A decentralized, federated capability-based delegation model (FCDM) is introduced in [24] to effectively protect large scale IoT devices. However, it has a higher latency of access request response due to storage overhead, resulting from the use of blockchain as a storage platform.

B. Blockchain-Based EMR System

Blockchain technology provides tamper-proof storage solution for the EMR and smart contracts provide the self-executing functionality for the EMR system.

In [25], an approach has been proposed for exchanging and managing the medical records, which utilizes the miners to comprehend and share data in order to obtain rewards. However, this is not considered ethical in healthcare due to financial conflicts of interest from token ownership to ethics guidelines [26]. Another scheme for the sharing of medical data is proposed to access and retrieve digital medical information [27]. However, it does not provide dynamic validation of entities. A privacy-preserving distributed mechanism named Medibchain has been proposed for regulating the healthcare data where blockchain is used as a storage [28]. Medibchain has been deployed and it ensures pseudonymity

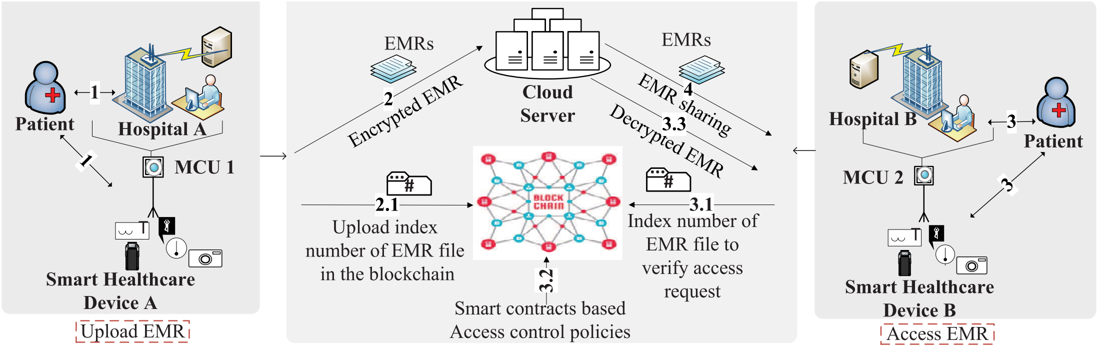

Patient
1
Hospital A
1
MCU 1
Smart Healthcare Device A
Upload EMR
EMRs
2
Encrypted EMR
Cloud Server
EMRs
4
EMR sharing
3.3
Decrypted EMR
2.1
Upload index number of EMR file in the blockchain
BLOCK CHAIN
3.2
Smart contracts based Access control policies
3.1
Index number of EMR file to verify access request
Hospital B
3
Patient
MCU 2
3
Smart Healthcare Device B
Access EMR

Fig. 1. Proposed architecture: Access model for IoT-enabled smart healthcare devices.

through cryptographic functions [29]. This approach needs to transfer the actual medical data to the blockchain servers, which in turn brings out an extra overhead for data transmission. A blockchain-based medical insurance system has been presented in [30], where the medical data is stored in the blockchain and protected by (n) servers. In this scheme, a hospital can make any node to act as a server where threshold (t) nodes are involved in obtaining data about the patient's expenditure. Nevertheless, the system is unreliable if these threshold servers are dishonest. A data sharing approach has been proposed for EMRs through consortium blockchain [31]. It utilizes a keyword-based search to match the digital record contained in the encrypted files. It has limitations of increased overhead communication with an increase in data chunks and can not retrieve storage space after sharing them.

Blockchain has been proposed as a remedy for cloud-based data security and privacy [32] as it detailed the limitations of current centralized technology to store sensitive medical data. It also highlights some challenges of using blockchain for medical data management; however, does not propose a technical approach or method to address them. A blockchain-based EMR authentication scheme has been proposed using the identity-based signature [33]. But the security of medical data generated by smart healthcare devices is not taken into account. Healthchain [34] proposes a distributed blockchain-based system constituting two chains for users and doctors to enable easy revocation. However, it is more complex than our proposed scheme because of two subchain models. Unlike [34], our proposed scheme enables all the entities to share EMR with the permission of the patient. Furthermore, our scheme allows the new entity to be added at any time making it more practical and dynamic.

To address the limitations of the above mentioned works, our scheme facilitates solutions for IoT-enabled smart health registries by using the emerging blockchain paradigm. From the data security, authenticity, and nonrepudiation standpoint, blockchain is a perfect fit for sharing EMRs since it provides an easily accessible, immutable, and transparent history of all contract-related data, and is adequate for building applications with trust and accountability. Also, our scheme only uploads the hash and index number of EMR to the blockchain network by leveraging the cloud as a storage. Since the hash and index number of EMR are very small in size, they reduce the communication overhead, resulting in faster EMR transmission.

III. Proposed System Architecture

As shown in Fig. 1, the system architecture considered in our proposed scheme mainly consists of entities, such as hospitals, patients, smart healthcare devices, medical control units (MCUs), and cloud. The roles of these peers in our scheme are explained as follows.

1) *Hospitals:* Hospitals in our scheme mainly include government or private hospitals that generate and share medical data among themselves. In a real scenario, there is a massive problem of securely sharing healthcare records among healthcare providers. Our scheme is designed to address this issue and facilitates the sharing of EMR among hospitals with the consent of patients on the basis of their mutual access agreement policies.
2) *Patients:* Patients are a vital part of our scheme. The access control mechanism is designed to make them the real owners of their data. They not only have access to their medical records but also manage access attempts through smart contracts that prevent the security issues of illegal data sharing and data theft among healthcare providers.
3) *Smart Healthcare Devices:* These devices in the scheme include wearable and other IoT devices in the homes or hospitals for real-time monitoring of patients' health. They are registered and monitored by the respective hospitals on the basis of their unique IDs, which are obtained by the hash values of the MAC addresses of these devices.
4) *MCUs:* These MCUs are the devices with certain computational power such as smart phones, computers or laptops. They act as a bridge between smart healthcare devices and blockchain nodes that are installed in hospitals. Each smart healthcare device is connected with MCU through WiFi or Bluetooth for exchanging the medical data. MCU encrypt the EMR generated from these devices and transmit them to the respective hospital through the secure communication channel.
5) *Cloud Server:* Cloud server is responsible for authenticating an entity's credentials. Given the limited block size and large size of EMR, it also acts as a storage

while the hash of the EMR file goes to the blockchain to increase the efficiency of our scheme. It can be managed by the government or any health organization.

In a typical medical system, each entity has valuable medical records that other entities might need to access. Therefore, the access control method must be implemented by the data owners to prevent illegal and unauthorized use of their private medical records. The data owner must be able to recognize the adversary and restrict those requests for illegal access. The requesters have access to the data only after satisfying smart contract-based access control policies explained in detail in Section IV.

IV. Proposed Smart Contract-Based Access Control Framework

In this section, we introduce the proposed smart contract-based distributed and patient centric access control framework.

In our proposed framework, we assume that each participant, such as patient, hospital has registered in the National Healthcare or identification system and has a unique identification number, i.e., ID. During the registration process, each entity needs to provide their personal data (D_p). Our scheme uses SHA-256 to hash the D_p and generates a unique ID based on it. P_{uid} and H_{uid} are the unique IDs of the patient and hospital, respectively. The registration information (D_p) is stored in cloud in the encrypted form, the same way as the EMR is stored. After completing the registration process, each entity has a respective unique address in the blockchain corresponding to its unique ID. It is only accessible by the cloud server (administrator or manager) with authorized permission from respective entity. Once registered, the cloud server will authenticate an entity's claim of credentials (unique ID, password, and private key associated with the blockchain address) and pass on the verification to the proposed system via DAuth protocol [35].

As shown in Fig. 2, our proposed framework consists primarily of four smart contracts, namely, validation contract (VC), GetAccess contract (GAC), grant contract (GC), and revoke contract (RC), each of which implements the access control policies for a pair of entities. VC validates and verifies the registration of each participant in our proposed framework. GAC decides whether an entity gets access or not based on the agreement policies between the EMR requester and EMR owner. GC checks if there is a fair request or any misconduct by the EMR requester and grants the access permission to the requested entity for a limited span to keep the EMR secure and pseudonym. Lastly, RC revokes the access control in case of validation failure, misconduct, or bombardment of multiple requests in a short period. These smart contracts are deployed across the blockchain network to provide access to the EMR generated by IoT-enabled healthcare devices. Each of these contracts are introduced in detail as follows.

1) *VC:* During the process of validating an access request, VC verifies a specific blockchain address and ID of EMR owner and requester. It maintains an activity list to store the time whenever a new entity becomes a part of our scheme and requests access to the EMR.

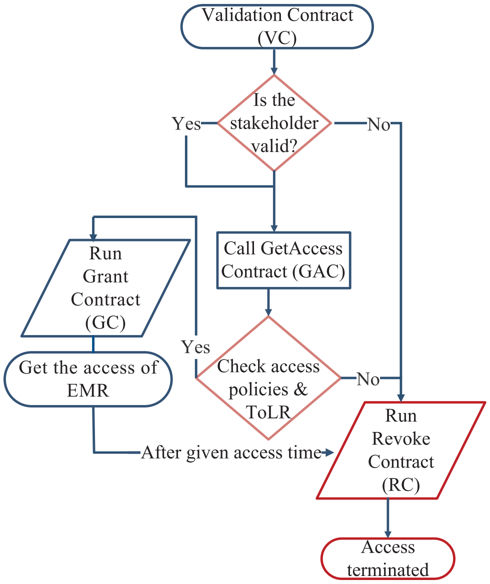


Fig. 2. Access control flowchart.

2) *GAC:* It includes access control agreement policies between the two entities. The main principle of GAC is to determine whether or not an individual is eligible to access a requested EMR. For that, the access control policies are verified by checking the time of the last request and the type of EMR requested. If these terms and conditions are manipulated, the RC instance will be called, and the access will be subsequently terminated.

   a) *SetGuidelines():* This application binary interface (ABI) is used to set access guidelines between the EMR owner and EMR requester. These guidelines are set by the EMR owner by providing various inputs such as minGap, noFR, and threshold (explained in Section VI).

   b) *SetGrant():* It takes the contract address of GC as an input to link it to GAC. GC sends misconduct reports, if any, via *setGrant*(). GAC then acts accordingly.

   c) *SetAccessTime():* It is the ABI to set time period during which access is provided by the EMR owner. It takes values in seconds.

   d) *DeleteGetAccess():* This ABI is invoked by the patient when the access control agreement and permission between the patient and other entities need to be terminated or revoked. It takes the EMR file ID, the EMR requester ID, and the access agreements as inputs to call the RC.

3) *GC:* After satisfying all the access conditions by the GAC, GC finally provides access of an EMR to the requester for a limited period as defined in *setAccessTime*()f. After that period is over, GC calls the instance of RC, and the access is terminated. The GC also contains the following main ABI to determine the misconduct, impose fine on the faulty requester, and update the RC.

   a) *MisconductCheck():* This ABI runs in real-time in case of any misconduct, which is checked based on the policies and conditions defined in the contract. It judges the misconduct committed by the EMR requesters based on the misconduct history and penalizes them. Further, it returns the fine decision to the GAC and calls the RC to block the access for that EMR requester. The requester is only able to make another access request after the imposed fine (blocked time) has expired.

4) *RC:* RC is called when the EMR requester fails to justify its validity or does not fulfill the terms and conditions defined in GAC or when *deleteGetAccess*() is called. It checks the misconduct in real-time if the requester is sending too many requests frequently or having a successive failure in gaining access. RC also functions after the access period is over as per coded in GAC. To keep track of faulty EMR requesters, RC maintains a record of the misconduct history of all the requesters in the fine list. It contains the following main fields.
   a) *EMR requester ID:* The entity that requests invalid access. It can be a hospital, clinic, or a patient.
   b) *EMR file ID:* The ID of the requested medical file for the invalid access.
   c) *Misconduct:* The misconduct by an EMR requester asking access for an EMR file is categorized as unable to validate himself, sends frequent requests in a short period, failure of terms and conditions of access control agreements, or experiences successive failure in gaining access.
   d) *Time of misconduct:* The time at which the requester performs the misconduct.
   e) *Fine:* The fine is imposed on the EMR requester for exhibiting misconduct to restrain his right of access. In an event of validation failure, the requester will be blocked permanently and will not be able to send any access request in the future, while in other cases he will be blocked for a certain period.

To facilitate the access control method in our proposed mechanism, the VC, GAC, GC, and RC provide various functionalities to manage the access control policies for the EMR. All the smart contracts manage an activity list and update the fields of this list from time to time. The fields of the activity list contain the following information.

1) *EMR Owner ID:* The entity that generates the EMR.
2) *EMR Requester ID:* The requester that asks the right of access to an EMR file from its owner.
3) *EMR file ID:* The medical record that a requester wants to access.
4) *Type of EMR:* The type of EMR is the various types of medical data such as blood pressure, glucose level.
5) *Response:* The access request made by the requester is responded as allowed to read, write, or update the EMR file by the owner of the EMR.
6) *Permission:* This field defines the access authority (static/predefined or dynamic) that is defined on the response by the EMR owner like allow, deny, full, etc.
7) *ToLR:* The last time at which the EMR requester makes the access request.

V. Secure EMR Upload and EMR Retrieval

This section describes how EMRs are securely stored and accessed through our proposed scheme.

When an IoT-enabled smart healthcare device connected with the respective MCU or hospital generates a new EMR, it gets encrypted and signed. The corresponding hash as a new transaction is appended to the shared ledger. We first introduce the cryptographic scheme utilized to securely upload and access EMR in the proposed access control scheme followed by its work flow. In order to delegate access to a requester by the EMR owner, transport layer security (TLS) has been used to provide a secure connection between them. After sharing EMR, an EMR requester accesses it in read-only mode and hence cannot be modified.

A. Cryptographic Scheme

1) *Elliptic Curve Cryptography (ECC)-Based Encryption and Decryption Using ECDH:* We have used an asymmetric encryption scheme based on ECC [36]. ECC uses the trapdoor function with one-way property in which it is easier to compute $M \rightarrow N$ but infeasible to reverse $N \rightarrow M$. Our proposed scheme utilizes the elliptic curve Diffie–Hellman (ECDH) key exchange to derive a shared secret key for EMR encryption and decryption. It generates static (permanent) and ephemeral (temporary) keys. An EMR owner uses static keys to encrypt and decrypt the EMR while storing and accessing it in the cloud. Assume that we have a cryptographic elliptic curve over a finite field with its generator point G. And denote SK_{pub} and SK_{priv} as the static public and private keys of patient, respectively. The generation of static keys of the EMR owner is as follows.
   a) Randomly generate $\leftarrow SK_{\text{priv}}$.
   b) Generate $SK_{\text{pub}} = SK_{\text{priv}} * G$.

   Ephemeral keys are used for re-encryption and decryption of EMR while sharing it to the other entity. The life of these keys is equal to the *setAccessTime*() defined in the GAC during the agreement between the EMR owner and EMR requester. The EMR owner generates its new ephemeral public and private keys, which are EK_{pub} and EK_{priv}, respectively. Similarly, the EMR requester generates an ephemeral public and private key pair, ER_{pub} and ER_{priv}. Both entities exchange their public keys and calculate the ephemeral shared secret key, K_{ES}, to be used for re-encrypting and decrypting the EMR while sharing. The generation of ephemeral keys for both entities is as follows.
   a) EMR owner randomly generates $\leftarrow EK_{\text{priv}}$.
   b) Generate $EK_{\text{pub}} = EK_{\text{priv}} * G$.
   c) EMR requester randomly generates $\leftarrow ER_{\text{priv}}$.
   d) Generate $ER_{\text{pub}} = ER_{\text{priv}} * G$.
   e) Both exchange their respective ER_{pub} and EK_{pub} with each other.
   f) Calculate $K_{ES} = ER_{\text{pub}} * EK_{\text{priv}} = EK_{\text{pub}} * ER_{\text{priv}}$.

   In the proposed scheme, Edwards-curve digital signature algorithm (EdDSA) [37] is used as a signature tool to generate the signature of hospital, H_{sig} for securing the authenticity of the EMR. Similarly, the EMR owner and EMR requester also generate their signatures, K_{sig} and R_{sig}, respectively. These signatures are exclusive for providing authentication and the forward-secure functionality. Assume the elliptic curve for the EdDSA algorithm comes with a subgroup order q, generated from G. The generation of the hospital's keys, signing of the EMR and verification of the signature will take place as follows.

a) Generation of the hospital's keys:
   - i) hospital randomly generates $\leftarrow H_{\text{priv}}$;
   - ii) generate $H_{\text{pub}} = H_{\text{priv}} * G$, where H_{pub} and H_{priv} are the public and private key pair of hospital.

b) Signing of the EMR:
   - i) $S_i = \text{hash}(\text{hash}(H_{\text{priv}}) + \text{EMR}) \text{mod } q$, where S_i is a secret integer;
   - ii) calculate the public key point, P_{pubk} behind S_i as $P_{pubk} = S_i * G$;
   - iii) calculate $X = \text{hash}(P_{pubk} + H_{\text{pub}} + \text{EMR}) \text{mod} q$;
   - iv) calculate $S = (S_i + X * H_{\text{priv}}) \text{mod} q$;

   $(\text{EMR}, H_{\text{priv}}) \rightarrow \{P_{pubk,} S\} = H_{\text{sig}}$.

c) Verification of the signature:
   - i) calculate $X = \text{hash}(P_{pubk} + H_{\text{pub}} + \text{EMR}) \text{mod} q$;
   - ii) calculate $V_{p1} = S * G$, where V_{p1} is the first verification point;
   - iii) calculate $V_{p2} = P_{pubk} + X * H_{\text{pub}}$, where V_{p2} is second verification point;
   - iv) return $V_{p1} = V_{p2}$;

   $(\text{EMR}, H_{\text{pub,}} H_{\text{sig}}) \rightarrow \text{valid/invalid}$.

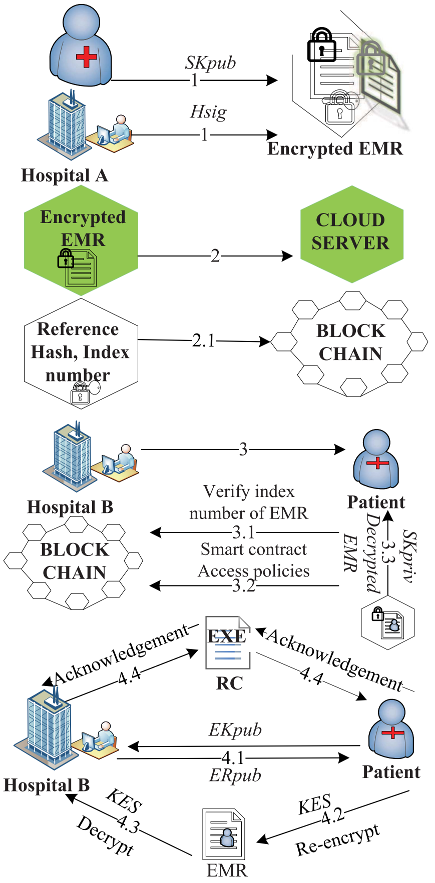


Fig. 3. Illustration of upload and retrieval of EMR in the proposed scheme.

B. Workflow of Proposed Scheme

The workflow of our proposed scheme depicted in Fig. 3 is explained in the steps as below.

Step 1: When a new EMR file, [EMR $= P_{uid}||H_{uid}||$ health data] is generated, it is signed with the hospital's signature H_{sig}, encrypted with the patient's public key SK_{pub}, [CipherEMR $\leftarrow E_{SK_{\text{pub}}}(\text{EMR}||H_{\text{sig}})$] and then uploaded to the cloud by the hospital A. By using asymmetric key encryption, our method ensures that the patient and other entities maintain precise control over the accessibility of medical records.

Step 2: The encrypted EMR file gets uploaded in the cloud while the corresponding reference hash and the index number is stored in the blockchain.

Step 3: When the patient goes to another hospital B (or any other entity), it will request the access to patient's historical EMRs, then, a corresponding smart contract will be called. The smart contract will verify the identity of the hospital, index number of EMR and then, grant (or deny) the access to the hospital according to the patient's permission defined in access policies. If succeed, the patient decrypts the *CipherEMR* using his/her private key SK_{priv}, [EMR $\leftarrow D_{SK_{\text{priv}}}$(CipherEMR)].

Step 4: Both the entities exchange their respective ephemeral keys to calculate the shared secret key, K_{ES}. To share the EMR file, the patient re-encrypts the EMR using K_{ES}, signs with his K_{sig}, [ReCipherEMR $\leftarrow E_{K_{ES}}(\text{EMR}||K_{\text{sig}})$] and sends it to the hospital B. To decrypt it, hospital B uses his K_{ES}, [EMR $\leftarrow D_{K_{ES}}$(ReCipherEMR)]. Finally, it will verify the integrity of the EMR with the public key H_{pub} of hospital A, and ephemeral public key of patient EK_{pub}, to make sure that the EMR has not been modified. Upon completion of access time, RC sends an acknowledgment to the hospital and patient for notifying the same, and keys will get expire simultaneously and the activity list will be subsequently updated by the smart contracts. Thenceforth, the hospital does not have access to EMR instead it needs to request the patient to access the file again.

This access control scheme ensures that only the known entities have access to the EMR while third parties are unable to access it. Our proposed access control method facilitates identity protection and data integrity and maintains patient-centricity through this encryption scheme.

VI. Implementation and Performance Analysis

This section explains the implementation setup and analysis of our proposed smart contract-based access control scheme. It also describes the hardware and software used to perform the experiments and finally, a performance analysis is made.

A. Hardware and Software Specifications

We simulate our model using a laptop and desktop computer to create multiple virtual Ethereum nodes on each device. The Table I shows the device specifications, which we have used to perform experiments.

TABLE I
DEVICE SPECIFICATIONS

Device Model	Processor	Operating System	Installed Memory
Lenovo ThinkCentre	Intel(R)Core™ i5-7500 CPU @3.40GHz	Windows 10 Enterprise, 64-bit	16 GB
HP Pavilion X360	Intel(R)Core™ i7-8550 CPU @3.40GHz	Windows 10 Home, 64-bit	8 GB

```
Algorithm 1 methodRegister and Validation ABI
Input: credential of entities, methodRegister
Output: status of validation, confirmation status of registered method.
Required: name, unique ID and blockchain address of entity, methodName, contractName, requesterAddress, ownerAddress, deployerOfcontract, contractAddress, abi.
1) Select(entity←patient, hospital) then
2) Info← entity[name, unique ID, blockchain address]
3) If (info == True ) then
       validation successful← true
4) Else
       validation← false
5) End
6) If (validation and GAC deployed)← true then
       Info←methodRegister([methodName][contractName] [requesterAddress] [ownerAddress] [deployerOfcontract] [contractAddress] [abi]) then
       Info← true then
       confirmation← true
7) Else
       confirmation← false
8) End
```

To create an Ethereum node on each device, a geth client [38] is installed on both devices. With geth client, we configure an Ethereum account for each node and form a private Ethereum blockchain. The laptop and desktop act as miners and deploy the contracts by sending transactions to the blockchain network. Remix [39] has been used to compile the smart contract code and Ethereum wallet has been used to deploy the contracts. We also use web3.js [40], an Ethereum JavaScript API at the EMR requester side to communicate with the corresponding geth client though HTTP connections for deploying the contracts and maintaining their various events. On the receiving side, web3.js communicates with the geth client for receiving the access messages and results from the smart contracts. The smart contracts are executed at every participating node. Each node updates the blockchain data every time when a new block is appended to the blockchain network. Since each node has the same copy of data, which ensures that the contracts are executed correctly.

1) Implementation: The process of implementation of the VC, GAC, GC, and RC is explained as follows.

1) *VC:* In VC implementation, we define a simple structure *methodRegister* to record the details of a participant in the scheme as explained in Algorithm 1. To keep the activity list updated, we have used mapping. Based on the registration, an appropriate contract is called. For example, if the registration is valid, the access request is forwarded to GAC and in event of an invalid access request, RC is called.
2) *GAC:* In the implementation of the GAC, we have introduced a misconduct structure to send multiple frequent access requests over a short period of time. The following fields have been added to the policies to help differentiate the misconduct.
   a) *MinGap:* The minimum permissible time interval between two successive requests. If the time interval between two consecutive requests is less than or equal to *minGap*, the later request is perceived as a frequent request.
   b) *NoFR:* The number of frequent requests made in a short period.
   c) *Threshold:* The threshold on the *noFR*. If the *noFR* is larger than or equal to the*threshold*, the GAC concludes that misconduct has occurred. As a fine for the misconduct, the EMR requester is blocked for a certain period.

   We also introduce a variable*timeOfmisconduct* for each resource to denote the time until the requests are blocked, which is set to 0 when the requests are unblocked. Then, to store the fields of a policy, we have used a *struct* and applied a 2-D mapping from the fields of resource (primary key) and action (secondary key) to this *struct* in order to construct the policy list. In case of any misconduct, the GAC also encloses a GC instance, through which the*misconductCheck* ABI of the GC is executed. Based on the above fields and variables, we design the *GetAccess* ABI as explained in Algorithm 2, which receives the inputs of*emrfile*, *emrType*, *response, time of request sent*, *setAccessTime* and returns the *requestId*, *access result*, and *fine*. The static validation is performed where the loop of*info.guideline* is executed and the dynamic validation starts from where *info.minGap* is calculated. The event *grantAccess*(*requestId, access result, fine*) is used to return the access result and fine to both the EMR owner and EMR requester.
3) *GC:* During the implementation of GC, we have used a dynamic array to store the misconduct records of an EMR requester, as shown in Table II. We have considered an artless misconduct judging method, which treats all potential misconduct received from the GAC as misconduct. The *misconductCheck* ABI pushes the misconduct into the misconduct record array on receiving a misconduct report of a requester from the GAC and then uses the following function to determine the corresponding fine:

$$\text{fine} = (\text{base})^n$$

Algorithm 2 GetAccess ABI

```
Input: emrfile, emrType, response, time of request sent, setAccessTime
Output: requestId, access result, fine
Required: validation of emrRequester ← true, checkconduct ← false, Guidelines, timeofUnblock of emrFile, RC instance Revoke, GC instance grnt.
 a) requestId ← requestId ++.
 b) If (request ← emrRequester) then
    Info ← Guidelines[emrfile][response][setAccessTime]
 c) If (timeofUnBlock) $\leq$ time) then
 d) If (timeofUnblock > 0) then
    Info.NoFR ← 0,
    Info.ToLR ← 0,
    timeofUnblock ← 0.
 e) End
 f) If info.guideline = "access permitted" then
    checkguidelines ← true.
 g) Else
    checkguidelines ← false.
    Revoke Contract ← invoke.
 h) End
 i) If Time − info.ToLR $\leq$ Info.minGap then
    Info.NoFR ← info.NoFR ++.
 j) If info.NoFR $\geq$ info.limitOfmisconduct then
    Detect a misconduct miscon.
    checkconduct ← false.
    fine ← grnt.misconductCheck(emrRequester, miscon).
    timeofUnBlock ← time + fine.
    Push miscon into misConduct list of emrfile.
 k) End
 l) Else
    Info.NoFR ← 0.
 m) End
 n) End
    Info.ToLR ← time.
 o) End
 p) result ← checkguidelines and checkconduct.
 q) Trigger event grantAccess(requestId, access result, fine).
```

TABLE II
SYSTEM VARIABLES

Variable	Value	Definition
minGap	100	minimum gap between two consecutive requests (seconds)
threshold	3	threshold on number of frequent requests
base	2	Fine-related variable
totalGap	10	Fine-related variable
n	$(length/totalGap)^2$	Fine-related variable

where n is the number of misconducts exhibited by the EMR requester (i.e., the length of the misconduct record array), which is equal to

$$n = (\text{length}/\text{totalGap})^2.$$

Note that the *base* and *totalGap* are initialized during the deployment of GC.

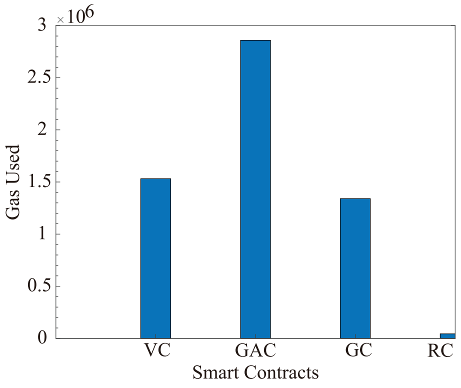


Fig. 4. Gas used to deploy smart contracts.

4) *RC:* The instance of RC is coded in all the three above contracts. The action of RC is to terminate the access of the EMR file according to the conditions mentioned in those contracts, if any misconduct is detected by GC, or after the access time is over.

5) *JavaScript at EMR Owner and JavaScript at EMR Requester:* To implement our access control framework, we need to create two JavaScript (one at the EMR owner side and another at the EMR requester side) using web3.js to interact with the contracts. Through HTTP, using JSON_RPC, we communicate with the geth node over JavaScript run-time environment in script mode.

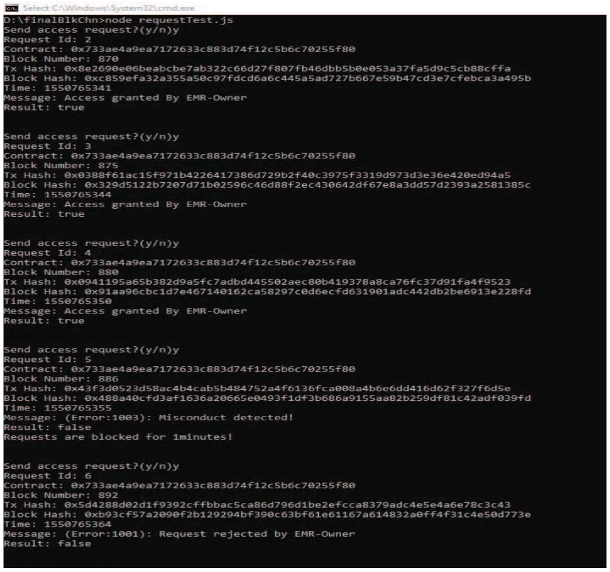
2) Results and Experimental Analysis: Ethereum blockchain platform denotes the amount of work done in the form of a unit called gas. In our case study, the gas required to deploy VC, GAC, GC, and RC are 1 531 432, 2 859 097, 1 340 488, and 43 714, respectively, as depicted in Fig. 4. Notice that we have used the test Ethereum network, these gas values are just test values, not a real cryptocurrency. For a real system, these values can be reduced further by using low cost consensus, such as PoS, DPoS, or PBFT.

The access results in Fig. 5 show the outcomes obtained while deploying our framework. If EMR owner allows access of the EMR file, then the message of "Access granted by EMR-Owner" appears. However, if any misconduct is detected, the access request is blocked for some time, depending on the misconduct record in the past. The last request shows the rejection by an EMR owner when the terms and conditions are not satisfied between the entities.

The results in Fig. 6 represent the access denial in the case of validation failure. In this case, the message of "Validation failure! Invalid EMR-Owner or EMR-Requester" is displayed and the access results are shown as "false." The "Request Id" is used in our scheme to identify and keep track of the access events per request, and allows the backtracking of requests in the event of system attack. This makes our scheme more feasible to use in real scenarios. Table III shows the parameters appeared in Figs. 5 and 6.

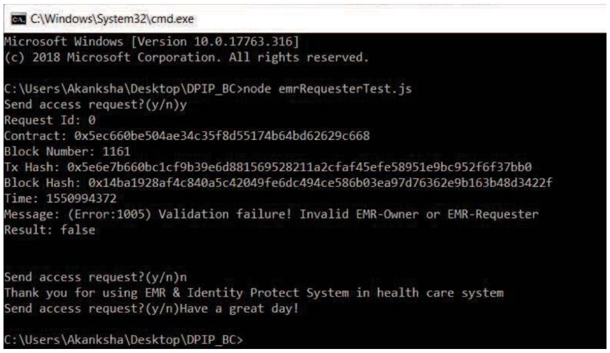
B. Performance Evaluation and Efficiency Analysis

In this section, we first evaluate the performance of our access mechanism, which is primarily reflected by smart contracts. The performance of our scheme mainly depends on the



```
D:\finalBlkChn>node requestTest.js
Send access request?(y/n)y
Request Id: 2
Contract: 0x733ae4a9ea7172633c883d74f12c5b6c70255f80
Block Number: 870
Tx Hash: 0x8e2690e06beabcbe7ab322c66d27f807fb46dbb5b0e053a37fa5d9c5cb88cffa
Block Hash: 0xc859efa32a355a50c97fdcd6a6c445a5ad727b667e59b47cd3e7cfebca3a495b
Time: 1550765341
Message: Access granted By EMR-Owner
Result: true


Send access request?(y/n)y
Request Id: 3
Contract: 0x733ae4a9ea7172633c883d74f12c5b6c70255f80
Block Number: 875
Tx Hash: 0x0388f61ac15f971b4226417386d729b2f40c3975f3319d973d3e36e420ed94a5
Block Hash: 0x329d5122b7207d71b02596c46d88f2ec430642df67e8a3dd57d2393a2581385c
Time: 1550765344
Message: Access granted By EMR-Owner
Result: true


Send access request?(y/n)y
Request Id: 4
Contract: 0x733ae4a9ea7172633c883d74f12c5b6c70255f80
Block Number: 880
Tx Hash: 0x0941195a65b382d9a5fc7adbd445502aec80b419378a8ca76fc37d91fa4f9523
Block Hash: 0x91aa96cbc1d7e467140162ca58297c0d6ecfd631901adc442db2be6913e228fd
Time: 1550765350
Message: Access granted By EMR-Owner
Result: true


Send access request?(y/n)y
Request Id: 5
Contract: 0x733ae4a9ea7172633c883d74f12c5b6c70255f80
Block Number: 886
Tx Hash: 0x43f3d0523d58ac4b4cab5b484752a4f6136fca008a4b6e6dd416d62f327f6d5e
Block Hash: 0x488a40cfd3af1636a20665e0493f1df3b686a9155aa82b259df81c42adf039fd
Time: 1550765355
Message: (Error:1003): Misconduct detected!
Result: false
Requests are blocked for 1minutes!


Send access request?(y/n)y
Request Id: 6
Contract: 0x733ae4a9ea7172633c883d74f12c5b6c70255f80
Block Number: 892
Tx Hash: 0x5d4288d02d1f9392cffbbac5ca86d796d1be2efcca8379adc4e5e4a6e78c3c43
Block Hash: 0xb93cf57a2090f2b129294bf390c63bf61e61167a614832a0ff4f31c4e50d773e
Time: 1550765364
Message: (Error:1001): Request rejected by EMR-Owner
Result: false
```

Fig. 5. Access granted and rejected by the EMR owner.



```
Microsoft Windows [Version 10.0.17763.316]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Akanksha\Desktop\DPIP_BC>node emrRequesterTest.js
Send access request?(y/n)y
Request Id: 0
Contract: 0x5ec660be504ae34c35f8d55174b64bd62629c668
Block Number: 1161
Tx Hash: 0x5e6e7b660bc1cf9b39e6d881569528211a2cfaf45efe58951e9bc952f6f37bb0
Block Hash: 0x14ba1928af4c840a5c42049fe6dc494ce586b03ea97d76362e9b163b48d3422f
Time: 1550994372
Message: (Error:1005) Validation failure! Invalid EMR-Owner or EMR-Requester
Result: false


Send access request?(y/n)n
Thank you for using EMR & Identity Protect System in health care system
Send access request?(y/n)Have a great day!

C:\Users\Akanksha\Desktop\DPIP_BC>
```

Fig. 6. Validation failure.

TABLE III
KEY PARAMETERS APPEARED IN FIGS. 5 AND 6

Parameters	Meaning
Request ID	Request ID to identify access events
Contract	Contract address (EMR owner address)
Block Number	Number of block where Tx inhabits
Tx Hash	Hash of transaction containing access requests
Block Hash	Hash of block where Tx inhabits
Time	Timestamp (in seconds) at the access request is sent
Message	Message displaying the result of access request
Result	True if authorization succeeds, otherwise false

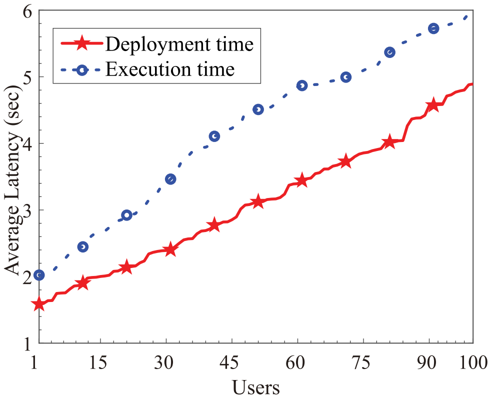
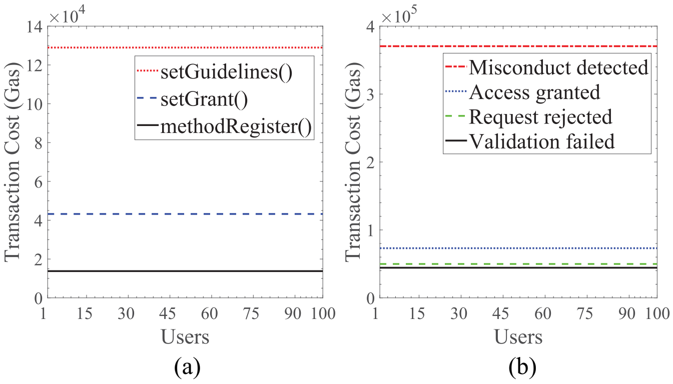


Fig. 7. Latency during the deployment and execution of smart contracts.



(a)

(b)

Fig. 8. Transaction cost of (a) execution of smart contract functions and (b) different responses.

blockchain platform, smart contracts, cloud storage, and cryptographic primitives (encryption schemes). The efficiency of the proposed mechanism is then studied and compared with some of the existing methods.

1) Latency for Smart Contracts Deployment and Execution: Firstly, we analyze the latency of smart contracts deployment, which is the average time to deploy all the contracts during the initialization state, depicted in solid line in Fig. 7. The latency of the execution of smart contracts, which is the average time to execute the input functions coded within the contracts, such as *methodRegister*, *setGuidelines*, and *setGrant*, is shown with dotted line in Fig. 7. These results indicate that the deployment and execution time of smart contracts increases monotonically with the number of users. In addition, the transaction cost associated with the execution of various functions within the smart contracts is depicted in Fig. 8(a) and (b) shows the transaction cost of responses received by the EMR requester. In Figs. 7 and 8, each value is calculated by taking an average of 100 measures of latency occurrences per number of users.

2) Computational Latency Analysis: The effects of encryption and decryption on an end-to-end delay need to be investigated especially for real-time smart healthcare services. We analyze the latency during encryption and the signing of EMR against the size of EMR depicted by the dashed line in Fig. 9, while the solid line shows the decryption process or retrieval of EMR from the scheme. We can see that the computational latency linearly increases with the increase in size of EMR. Note that the calculated values are the result of two-step encryption employed in our scheme to achieve patient-centricity and data integrity. It enables access revocation by preventing EMR requester to achieve life time access.

3) Latency of Access-request Response: The response cycle between the submission of an access request and access response of our scheme needs to be investigated primarily from the viewpoint of real-time smart healthcare services. By using our testbed, we measure the latency for the response time of access request submitted to our smart contract-based scheme. The final value is calculated by taking an average of 100 measurements per user. It is assumed that each user has a

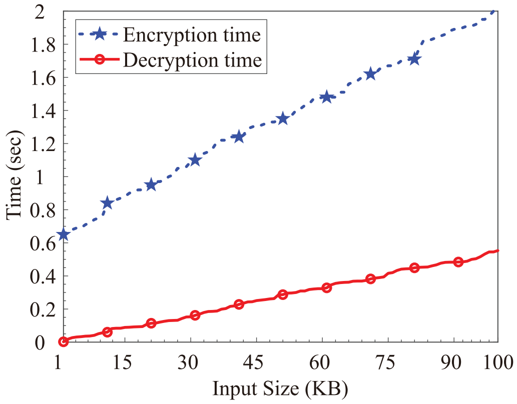


Fig. 9. Computational latency during the registration and retrieval of EMR.

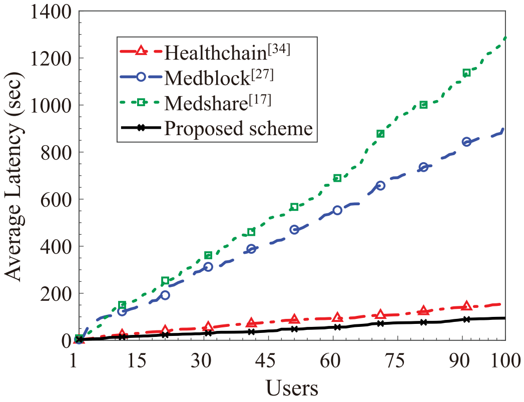


Fig. 10. Latency for the access-request response.

1 KB EMR file with 10 new users joining the proposed scheme every month. We observe that the latency for response time gradually increases with an increase in the number of users as depicted in Fig. 10.

4) Security Against Various Attacks: In this section, we provide the security analysis of our scheme in the conventional way as in [28] and [33], focusing on attack resistance, privacy protection, and nonrepudiation.

1) *Pseudonymity:* The patient and hospital will not be identified by an intruder in our scheme. When an EMR is encrypted by using the public key (SK_{pub}) and hospital signature (H_{sig}), its hash is the new identification of that encrypted EMR, which is a long integer and it does not reveal any information about user.
2) *Privacy and Data Integrity:* The personal data D_p of an entity is not visible to anyone in the system after the validation process is completed. Each entity has its unique ID generated by the hash of D_p. P_{uid}, H_{uid}, and IP_{uid} cannot be reverse engineered to get D_p. Hence, an adversary cannot find the healthcare data or patient identity without the encryption keys SK_{pub}, SK_{priv}, and reference hash of EMR and will not be able to associate a patient with any medical record. In addition, a slight modification in the EMR changes its hash, making it no longer match with the hash of the file stored in the cloud. Therefore, our system preserves the integrity of the data.
3) *Accountability:* Accountability refers to the responsibility and trustworthiness of healthcare providers while generating and sharing EMRs with other entities. In the proposed scheme, each EMR is signed by the respective hospital and the corresponding hospital is therefore liable for immorality of any kind.
4) *Resistance to Masquerade Attack:* A masquerade attack is performed by an adversary to gain an unauthorized access to a system, usually by stealing passwords, logins, figuring loopholes in programs, etc. Our scheme is resistant to masquerade attacks since the adversary is unable to pretend as a registered entity due to the reason that each entity owns a unique ID (P_{uid}, H_{uid}, and IP_{uid}). Our scheme validates all the entities before they join the system. Also, the adversary is unable to access the EMR stored in the ledger as it is encrypted by using the patient's key.
5) *Resistance to Replay Attack:* An adversary may intercept access requests sent by the users and replay the access requests in order to gain the invalid access of EMR. Since the requests are sequenced with the *Request ID*, the adversary could not send multiple requests with the same request ID and eventually gets blocked permanently as per the conditions built in GC and GAC.
6) *Resistance to Hash Collision:* The proposed scheme employs SHA-256 to hash the data, where the hash length is set to be 256 b. Although hash collision cannot be ruled out in theory due to the birthday paradox, the collision probability is extremely small in our scheme. This can be explained as follows. Since the hash length is of 256 b, the total number of different hash values is $N = 2^{256} \approx 1.2 \times 10^{77}$. Let M be the maximum number of EMRs that could be stored in the proposed smart healthcare system, which could reach billions (i.e., 10^9) in a practical system. Obviously, $N \gg M^2 \gg M \gg 1$. It is shown in [41] that the collision probability is

$$p(M, N) \approx 1 - e^{\frac{-M(M-1)}{2N}}$$

which, due to $M \gg 1$, can be further reduced to

$$p(M, N) \approx 1 - e^{\frac{-M^2}{2N}}.$$

It is known that the Maclaurin series of e^{-x} is $e^{-x} \approx 1 - x$ for $|x| \ll 1$, where $|x|$ denotes the absolute value of x. Since $0 < (M^2/2N) \ll 1$, the above expression of $p(M, N)$ can be written as

$$p(M, N) \approx 1 - \left(1 - \frac{M^2}{2N}\right) = \frac{M^2}{2N}.$$

Considering an application case with 50 billion (or 5×10^{10}) EMRs, the corresponding collision probability is $p(M, N) \approx ((5 \times 10^{10})^2/[2 \times (1.2 \times 10^{77})]) \approx 1.04 \times 10^{-56}$, which is extremely small. Therefore, hash collision is extremely unlikely to occur in our proposed smart healthcare system.

The comprehensive comparison of the proposed work with the related works [17], [25], [27]–[29], and [34] is summarized in Table IV. The comparison is conducted in terms of various metrics that are listed in the first column of the table. Like the previous works, the proposed work has the features of privacy preserving, attack resistance, and tamper proof. The privacy protection feature comes from the applied cryptographic

TABLE IV
COMPARISON BETWEEN OUR PROPOSED SOLUTION AND RELATED WORK

Metrics	Medshare[17]	Medrec[25]	Medblock[27]	Medibchain[28]	Medibchain[29]	Healthchain[34]	Proposed method
Privacy-preserving	✓	✓	✓	✓	✓	✓	✓
Attack resistant	✓	✓	✓	✓	✓	✓	✓
Tamper-proof	✓	✓	✓	✓	✓	✓	✓
Cloud storage	×	×	×	✓	✓	×	✓
Distributed access control	×	✓	×	×	×	×	✓
Smart contracts-based access control policies	×	×	×	×	×	×	✓
Access revocation	✓	×	×	×	×	✓	✓
Dynamic validation of entities	×	×	×	×	×	×	✓
Smart device enabled	×	×	×	×	×	✓	✓
Access-response time	Slow	N/A	Medium	N/A	N/A	Fast	Faster

scheme, while the other two features result from the use of blockchain technology. To address the storage problem in the blockchain system due to its limited block size, the cloud-storage is adopted in the proposed scheme like [28] and [29]. The distributed access control is only found in [25], and the proposed scheme. In addition, only the proposed scheme use the smart contract-based access control policies for sharing of EMR. While [17], [34] and the proposed scheme are equipped with access revocation, only the proposed solution performs the dynamic validation of the entities. Besides, only [34] and our scheme are enabled for smart devices. Furthermore, our scheme has a faster access-response time than [17], [27], and [34]. Fig. 10 shows the average access latency of the compared methods versus the number of users. It is clear that the proposed scheme has the smallest average access latency or in other words the fastest access-response time.

VII. CONCLUSION

This article proposed and developed an access control model for IoT-enabled smart healthcare devices and the medical system through blockchain-based smart contract. We have achieved our goal of patient-centricity and accessibility of medical records across the system. Cloud is used as a storage to avoid the congestion in the blockchain network. EMR is securely registered and retrieved through the cryptographic functions of ECC and EdDSA. The proposed scheme has been implemented on the Ethereum private blockchain network. The performance evaluation and efficiency analysis demonstrate the feasibility of the proposed scheme in the real-time smart healthcare system for a secure, decentralized, distributed, and patient-centric access control.

While the proposed scheme has demonstrated attractive features, the integration of blockchain and cloud to provide decentralized access control incurs challenges of scalability and performance. By using edge computing, the latency that occurs in processing and fetching the EMR can be greatly reduced. Therefore, further research is required to address this issue. Furthermore, in order to enhance the liveliness and fairness of the system, another future research work is to develop an incentive mechanism for EMR owners in the proposed scheme.

REFERENCES

[1] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, "Access control in the Internet of Things: Big challenges and new opportunities," *Comput. Netw.*, vol. 112, pp. 237–262, Jan. 2017.

[2] N. Fatema and R. Brad, "Security requirements, counterattacks and projects in healthcare applications using WSNs—A review," 2014. [Online]. Available: arXiv:1406.1795.

[3] J. Sun and Y. Fang, "Cross-domain data sharing in distributed electronic health record systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 6, pp. 754–764, Jun. 2010.

[4] L. J. Kish and E. J. Topol, "Unpatients-why patients should own their medical data," *Nat. Biotechnol.*, vol. 33, no. 9, p. 921, 2015.

[5] S. Ravidas, A. Lekidis, F. Paci, and N. Zannone, "Access control in Internet-of-Things: A survey," *J. Netw. Comput. Appl.*, vol. 144, pp. 79–101, Oct. 2019.

[6] S. Osborn, R. Sandhu, and Q. Munawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies," *ACM Trans. Inf. Syst. Security*, vol. 3, no. 2, pp. 85–106, 2000.

[7] R. S. Sandhu, "Role-based access control," in *Advances in Computers*, vol. 46. London, U.K.: Elsevier, 1998, pp. 237–286.

[8] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[9] R. S. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, Sep. 1994.

[10] E. B. D. Hussein and V. Frey, "A community-driven access control approach in distributed IoT environments," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 146–153, Mar. 2017.

[11] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Gener. Comput. Syst.*, vol. 82, pp. 395–411, Nov. 2018.

[12] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[13] *Blockchain Momentum Rallies Healthcare*. Accessed: Jan. 6, 2017. [Online]. Available: http://www.ibm.com/blogs/blockchain/2017/01/blockchain-momentum-rallies-healthcare/

[14] J. Priisalu and R. Ottis, "Personal control of privacy and data: Estonian experience," *Health Technol.*, vol. 7, no. 4, pp. 441–451, 2017.

[15] *Introduction to Ethereum*. [Online]. Available: https://ethereum.org/en/developers/docs/intro-to-ethereum/

[16] *Introduction to Smart Contracts*. [Online]. Available: https://solidity.readthedocs.io/en/v0.7.0/introduction-to-smart-contracts.html

[17] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "MedShare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14757–14767, 2017.

[18] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1594–1605, Feb. 2018.

[19] G. Zyskind *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Security Privacy Workshops*, 2015, pp. 180–184.

[20] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "FairAccess: A new blockchain-based access control framework for the Internet of Things," *Security Commun. Netw.*, vol. 9, no. 18, pp. 5943–5964, 2016.

[21] M. S. Ferdous, A. Margheri, F. Paci, M. Yang, and V. Sassone, "Decentralised runtime monitoring for access control systems in cloud federations," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2017, pp. 2632–2633.

[22] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Mar. 2018.

[23] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[24] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A smart contract enabled decentralized capability-based access control mechanism for the IoT," *Computers*, vol. 7, no. 3, p. 39, 2018.

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

[26] Q. DuPont. (2019). *Guiding Principles for Ethical Cryptocurrency, Blockchain, and DLT Research*. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3431738

[27] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, "Medblock: Efficient and secure medical data sharing via blockchain," *J. Med. Syst.*, vol. 42, no. 8, p. 136, 2018.

[28] A. Al Omar, M. S. Rahman, A. Basu, and S. Kiyomoto, "Medibchain: A blockchain based privacy preserving platform for healthcare data," in *Proc. Int. Conf. Security Privacy Anonymity Comput. Commun. Storage*, 2017, pp. 534–543.

[29] A. Al 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 Gener. Comput. Syst.*, vol. 95, pp. 511–521, Sep. 2019.

[30] L. Zhou, L. Wang, and Y. Sun, "MISTORE: A blockchain-based medical insurance storage system," *J. Med. Syst.*, vol. 42, no. 8, p. 149, 2018.

[31] A. Zhang and X. Lin, "Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain," *J. Med. Syst.*, vol. 42, no. 8, p. 140, 2018.

[32] C. Esposito, A. De Santis, G. Tortora, H. Chang, and K.-K. R. Choo, "Blockchain: A panacea for healthcare cloud-based data security and privacy?" *IEEE Cloud Comput.*, vol. 5, no. 1, pp. 31–37, Jan./Feb. 2018.

[33] F. Tang, S. Ma, Y. Xiang, and C. Lin, "An efficient authentication scheme for blockchain-based electronic health records," *IEEE Access*, vol. 7, pp. 41678–41689, 2019.

[34] J. Xu *et al.*, "HealthChain: A blockchain-based privacy preserving scheme for large-scale health data," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8770–8781, Feb. 2019.

[35] *Dauth—A Distributed Authentication Mechanism for Blockchains*. [Online]. Available: https://github.com/madhavanmalolan/dauth/blob/master/dauth.pdf

[36] J. Lopez and R. Dahab. (2000). *An Overview of Elliptic Curve Cryptography*. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.2771&rep=rep1&type=pdf

[37] D. J. Bernstein, S. Josefsson, T. Lange, P. Schwabe, and B.-Y. Yang, "EddSA for more curves," in *Proc. Cryptol. ePrint Archive*, 2015, p. 677.

[38] *Geth-Go Implementaion of Ethereum Protocol*. [Online]. Available: https://github.com/ethereum/go-ethereum

[39] *REMIX-IDE for Smart Contract Deployment Provided by Ethereum*. [Online]. Available: https://remix.ethereum.org/

[40] *Web3 Javascript API to Interact With Ethreum Nodes*. [Online]. Available: https://github.com/ethereum/wiki/wiki/JavaScript-API

[41] G. Gupta. (Feb. 2015). *What Is Birthday Attack?*. [Online]. Available: https://www.researchgate.net/publication/271704029_What_is_Birthday_attack

Akanksha Saini (Graduate Student Member, IEEE) received the B.Tech. degree in electronics and communication engineering from Punjab Technical University, Jalandhar, India, in 2015, and the M.Tech. degree in information, communication, and electronics engineering from Catholic University of Korea, Seoul, South Korea, in 2018. She is currently pursuing the Ph.D. degree in information technology with Deakin University, Geelong, VIC, Australia.

Her research interests include identity and privacy protection, network security, cryptography, Internet of Things, and blockchain technology.

Qingyi Zhu (Member, IEEE) received the Ph.D. degree in computer science and technology from the College of Computer Science, Chongqing University, Chongqing, China, in 2014.

He is currently an Associate Professor with the Chongqing University of Posts and Telecommunications, Chongqing. He has published more than 30 academic papers in peer-reviewed international journals. His current research interests include cybersecurity dynamics, complex systems, and blockchain.

Dr. Zhu has also served as an Invited Reviewer for various international journals and conferences.

Navneet Singh received the B.Tech. degree in computer engineering from Punjab Technical University, Jalandhar, India, in 2018. He is currently pursuing the M.Tech. degree in information, communication, and electronics engineering with Catholic University of Korea, Seoul, South Korea.

His research interests include AI, ML, IoT, indoor localization, and blockchain technology.

Yong Xiang (Senior Member, IEEE) received the Ph.D. degree in electrical and electronic engineering from the University of Melbourne, Melbourne, VIC, Australia, in 2003.

He is a Professor with the School of Information Technology, Deakin University, Geelong, VIC, Australia. He has published five monographs, over 160 refereed journal articles, and numerous conference papers in these areas. He was invited to give keynote speeches at a number of international conferences. His research interests include information security and privacy, signal and image processing, data analytics and machine intelligence, Internet of Things, and blockchain.

Prof. Xiang is the Senior Area Editor of IEEE SIGNAL PROCESSING LETTERS and an Associate Editor of *IEEE Communications Surveys and Tutorials*. He was an Associate Editor and a Guest Editor of several reputed international journals. He served as the honorary chair, the general chair, the program chair, the TPC chair, the symposium chair, and the track chair for many international conferences.

Longxiang Gao (Senior Member, IEEE) received the Ph.D. degree in computer science from Deakin University, Geelong, VIC, Australia, in 2014.

He was a Postdoctoral Research Fellow with IBM Research and Development Australia, Southbank, VIC, Australia. He is currently a Senior Lecturer with the School of Information Technology, Deakin University. He has over 40 publications, including patents, monographs, book chapters, and journal and conference papers. Some of his publications have been published in the top venues, such as the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. His research interests include data processing, mobile social networks, fog computing, and network security.

Dr. Gao received the 2012 Chinese Government Award for Outstanding Students Abroad (ranked No.1 in Victoria and Tasmania consular districts). He is active in the IEEE Communications Society. He has served as the TPC co-chair, the publicity co-chair, the organization chair, and a TPC member for many international conferences.

Yushu Zhang (Member, IEEE) received the Ph.D. degree in computer science and technology from the College of Computer Science, Chongqing University, Chongqing, China, in December 2014.

He held various research positions with Southwest University, Chongqing; the City University of Hong Kong, Hong Kong; the University of Macau, Macau, China; and Deakin University, Geelong, VIC, Australia. He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. His current research interests include multimedia security, blockchain, cloud computing, and big data security. He has published more than 100 refereed journal articles and conference papers in the above areas.

Prof. Zhang is currently an Associate Editor of *Information Sciences and Signal Processing*.