Journal of
*Sensor and Actuator Networks*

MDPI

*Article*

# Implementation of a Biometric-Based Blockchain System for Preserving Privacy, Security, and Access Control in Healthcare Records

**Ezedin Barka [1], Mohammed Al Baqari [1], Chaker Abdelaziz Kerrache [2] and Jorge Herrera-Tapia [3,*]**

[1] College of Information Technology, United Arab Emirates University, Al Ain P.O. Box 15551, United Arab Emirates
[2] Laboratoire d'Informatique et de Mathématiques, Université Amar Telidji, Laghouat 03000, Algeria
[3] Faculty of Computer Science, Universidad Laica Eloy Alfaro de Manabí, Manta 130214, Ecuador
* Correspondence: jorge.herrera@uleam.edu.ec

**Abstract:** The use of Electronic Health Record (EHR) systems has emerged with the continuous advancement of the Internet of Things (IoT) and smart devices. This is driven by the various advantages for both patients and healthcare providers, including timely and distant alerts, continuous control, and reduced cost, to name a few. However, while providing these advantages, various challenges involving heterogeneity, scalability, and network complexity are still open. Patient security, data privacy, and trust are also among the main challenges that need more research effort. To this end, this paper presents an implementation of a biometric-based blockchain EHR system (BBEHR), a prototype that uniquely identifies patients, enables them to control access to their EHRs, and ensures recoverable access to their EHRs. This approach overcomes the dependency on the private/public key approach used by most blockchain technologies to identify patients, which becomes more crucial in situations where a loss of the private key permanently hinders the ability to access patients' EHRs. Our solution covers component selection, high-level implementation, and integration of subsystems, was well as the coding of a prototype to validate the mitigation of the risk of permanent loss of access to EHRs by using patients' fingerprints. A performance analysis of BBEHR showed our system's robustness and effectiveness in identifying patients and ensuring access control for their EHRs by using blockchain smart contracts with no additional overhead.

**Keywords:** blockchain; healthcare; EHR; fingerprint; biometric; access control



**Citation:** Barka, E.; Al Baqari, M.; Kerrache, C.A.; Herrera-Tapia, J. Implementation of a Biometric-Based Blockchain System for Preserving Privacy, Security, and Access Control in Healthcare Records. *J. Sens. Actuator Netw.* **2022**, *11*, 85. https://doi.org/10.3390/jsan11040085

Academic Editors: Mohamed Benbouzid, Leandros Maglaras and Mohamed Amine Ferrag

Received: 1 November 2022
Accepted: 8 December 2022
Published: 13 December 2022



## 1. Introduction

In the healthcare sector, the move towards electronic health record (EHR) systems has been speeding up in parallel with the increased adoption of IoT and smart devices. This is driven by the expected advantages for patients and healthcare providers. The Office of the National Coordinator (ONC) for Health Information Technology within the U.S. Department of Health and Human Services [1] defined EHR as "a digital version of a patient's paper chart. EHRs are real-time, patient-centered records that make information available instantly and securely to allow users. While an EHR contains the medical and treatment histories of patients, they build an EHR system to go beyond standard clinical data collected in a provider's office and can be inclusive for a broader view of a patient's care. They are built to share information with other healthcare providers, such as laboratories and specialists, so they contain information from all the clinicians involved in the patient's care". The ONC identified some advantages of EHR systems, such as:

- They maintain and synchronize patients' medical history, diagnoses, medications, treatment plans, immunization dates, allergies, radiology images, and laboratory and test results.
- They allow access to evidence-based tools that providers can use to decide about patients' care.

- They automate and streamline provider workflow.

In parallel with the move towards EHR healthcare systems, Satoshi Nakamoto introduced the blockchain technology in 2009 [2]. Since then, blockchain technology has received significant attention from both the academic and industrial research communities. Blockchain's decentralized nature, along with its cryptographic services, has increased its potential to be the future platform for many distributed systems. The initial phase of blockchain technology was limited to the financial sector and mainly focused on cryptocurrency, such as Bitcoin, which has evolved to become the most popular cryptocurrency application.

Over time, the blockchain technology has been strengthened with the emergence of Ethereum and its smart contract capability, Buterin [3,4], which provides the programmability component of the Ethereum blockchain. This step gave blockchain technology huge potential and expanded its scope from the financial sector to other sectors, including those of healthcare, education, government, and manufacturing. In particular, for the healthcare sector and EHR systems [5], blockchain technology offers many capabilities that can fulfill several EHR requirements, as described by McGhin, Choo, Liu, and He [6] and summarized below.

***EHR Requirements***

1. System security, including authentication, integrity, access control, and non-repudiation for multiparity-integrated EHR systems.
2. Interoperability between different EHR standards implemented by various healthcare providers, research entities, insurance providers, and pharmacies.
3. Data sharing of health records.
4. Mobility of healthcare systems with the introduction of IoT and smart devices that allow patients to share and access their health records.
5. Availability of the healthcare system.

***Blockchain Opportunities***

1. Security, assurance, and immutability are provided by using cryptography, namely, private and public keys combined with hash-chaining between blocks of data.
2. The smart contract capability provides an abstraction layer to enable communication among miners in distributed healthcare providers running different EHR standards.
3. The decentralized architecture allows multiple entities to share health records.
4. Shared data across distributed ledgers enable near-real-time updates across the network for all parties.
5. The technology provides high availability and resilience through its decentralized model of operation.

The close convergence between the requirements of EHRs and blockchain's capabilities is a key driver that has led many proposals to include blockchain-based EHR applications, including EHR monitoring and auditing, mobility applications, and exchange of information [7].

Despite the promising convergence between blockchain and EHR systems, some limitations have been identified regarding the integration between blockchain and EHR systems, specifically:

- Identity management in the current implementations of blockchain-based EHR systems is based on private/public key pairs. Patients that use their private keys as their identities to control access to and to sign their EHRs are subject to permanent loss of access to their records with lost private keys, as discussed by McGhin [6]. This is because, in asymmetric cryptography, a private key is not recoverable from the public key (computationally infeasible).
- The lack of standardization for various deployments of blockchain in healthcare systems generates a challenge regarding the interoperability and exchange of EHRs, which limits the success of deployments [6].
- There is the potential for privacy leakage due to the unencrypted nature of blocks that hold information related to patients' health. Even with encrypted blocks, the ability to

access the blocks publicly in public ledgers makes them subject to cryptanalysis attacks, which can exploit patients' privacy if the encryption algorithms are compromised [6].
- Scalability and IoT overhead can occur due to the increased number of medical IoT devices and medical sensors joining the blockchain network. The more IoT devices join the blockchain network, the greater the computational complexity of the ledger will be, which leads to the need for more computational power on these IoT devices. However, these IoT devices have very limited computational capabilities and are not designed to support the complex operations required by blockchain hashing algorithms [6].

The majority of the existing solutions rely on the implementation of Ethereum and prioritize access control services. However, none of the current solutions have discussed the adversary models that their solutions might run into. Additionally, because they all use the public/private key strategy, the entire system is vulnerable to authorized inside invaders.

In a previous work [8], we proposed an architecture that combined biometric-based blockchain technology with an EHR system. The integration ensured the integrity and availability of access control data for a patient's electronic healthcare records (EHRs), which were synchronized and exchanged by using blockchain technology between distributed healthcare providers [9,10]. In this paper, we built and implemented a prototype for BBEHR according to the design specifications proposed in [8]. We will evaluate this prototype against functional and security requirements of BBEHR. In addition, the performance of the BBEHE prototype will be evaluated with respect to the time delay introduced by blockchain smart contracts and their impact on the operation of BBEHR.

The rest of this paper is organized as follows. Section 2 discusses some blockchain-related preliminaries. We examine the existing EHR implementations in Section 3. Section 4 provides an overview of the proposed BBEHR architecture, and then, we present the implementation of the BBEHR system. Section 5 describes the functional and security testing, in addition to a performance evaluation and a discussion of the results. Finally, Section 6 concludes the paper.

## 2. Blockchain in Healthcare: Background and Related Work

The literature reviewed in this section covers the challenges, capabilities, and some proposed implementations of blockchain in EHRs, specifically:

- The challenges in integrating existing legacy centralized EHR systems.
- The required capabilities in any blockchain-based EHR systems in order to be accepted.
- Proposals for blockchain-based EHR systems, including MedRec, BBDS, OmniPHR, and MedShare.

For instance, the authors of [11] conducted a thorough analysis of the literature on blockchain strategies created for EHR systems, concentrating primarily on privacy and security concerns. They also identified a number of research possibilities and problems.

Shahnaz et al. [12] suggested a framework that would establish a decentralized platform for the storage of patient medical records and grant providers or concerned parties, such as patients, access to such records. As it is not in the architecture of blockchain to store enormous volumes of data on it, they also sought to address the scalability issue. In order to tackle the scalability issue, they adopted an off-chain scaling approach that stored the data on the underlying medium.

In [13], various blockchain-based solutions for addressing current healthcare systems' limitations were examined. These solutions included frameworks and tools for assessing the performance of such systems, such as the Composer, Docker Container, Hyperledger Caliper, and Wireshark capture engine. The authors also suggested an access control policy algorithm to increase data accessibility across healthcare professionals.

By examining several publications, Fatima et al. [14] offered the current state of the art for a blockchain-based medical healthcare system. The blockchain-based healthcare industry is also facing certain difficulties. Among them, the acceptance of blockchain-based techniques, which differ significantly from the conventional method, their adaptability,

and the requirement for additional study in blockchain applications for assuring data security and privacy are notable.

### 2.1. Centralized-Based EHR

The move towards blockchain-based EHR systems to integrate distributed healthcare providers raises questions about the challenges of integrating current centralized EHR systems that are distributed among healthcare providers with existing legacy technology. In [15], Magyar listed four challenges in integrating existing centralized EHR systems.

In centralized systems, EHRs are maintained in different formats that suit each provider's business model. This requires various interfaces and protocols for integration, and there is no single protocol accepted across all providers. Because of this complexity, there is a high potential for compromising the security of EHRs and the privacy of patients across the different middleware technologies and protocols that are used.

The current model of centralized systems provides high central authority to the dominant health provider of the patient, which complicates the exchange of health information in the case of unplanned treatment in an emergency situation and can cause serious results, such as fatality, because of the lack of timely access to EHRs.

Auditing patients' history and traceability is a significant concern in centralized EHR systems, as the information passes by multiple healthcare providers. This is especially a concern when institutional incentives influence the history of a patient's data. The availability of patient data in integrated, centralized EHR systems is inconsistent, and the related regulations are unclear. It is subject to the resilience of each centralized healthcare provider.

An example of integrating centralized EHR systems was a five-year agreement in 2016 between Google DeepMind and the Royal Free London NHS Foundation Trust. This integration encountered significant problems, which were summarized by Pilkington [16]:

- Lack of transparency and privacy.
- Mismanagement of patients' data and identities.
- Delayed treatment due to malicious software infections, which caused delayed service recovery.

### 2.2. Required Capabilities of Blockchain-Based EHR

Zhang, Walker, White, Schmidt, and Lenz [17] conducted research on the required metrics for any blockchain-based EHR system to be accepted. The researchers identified seven metrics:

1. *The entire workflow of the system is HIPAA-compliant:* For a healthcare solution to be accepted and adopted, it must fulfill the regulatory requirements of a country's national health authority (NHA). Considering the HIPAA Act as an example of a health regulation act in the US, Dagher, Mohler, Milojkovic, and Marella [18] analyzed its requirements and concluded that of the five HIPAA titles, Title II is relevant to blockchain-based EHR. This title comprised the standards for the privacy of individually identifiable health information (privacy rule) and the security standards for the protection of protected electronic health information (security rule). Magyar [15] further analyzed the HIPAA requirements and concluded that blockchain technology can fulfill the HIPAA requirements of secured access, privacy, lack of centralized government, and cost reduction. Zhang, Walker, White, Schmidt, and Lenz [17] highlighted precautions that should be considered when implementing HIPAA-complaint blockchain-based solutions. Peng et al. stated, "A core tenet of HIPAA compliance is that Personally Identifiable Information (PII) must be protected against a confidentiality breach. In particular, the end-to-end workflow of a healthcare app from entering to processing then delivering the data must be HIPAA compliant". This can be achieved in centralized systems by using encryption techniques; however, in blockchain, encryption may not be useful because any data stored in the blockchain are replicated across all of the miners and are accessible by any party. Therefore, any breach of the currently used encryption algorithms makes EHR information vul-

nerable, especially data in the Blockchain that are immutable and cannot be deleted. Accordingly, the authors of [17] recommended storing the encrypted metadata of EHRs in the blockchain (with a minimum level of information), which would ensure that EHR data are securely stored.

2. *The framework supports Turing-complete operations:* Zhang, Walker, White, Schmidt, and Lenz [17] stated that any blockchain-based EHR system should be Turing-complete and have programming capabilities that enable simple integration and interoperability with legacy systems. In addition, it should have the capability for simple upgrades and feature enhancements. Blockchain networks built specifically for healthcare applications are not scalable and cannot fulfill these requirements.
3. *Support for user identification and authentication:* In EHR systems, users are classified as patients or healthcare professionals. The authors of [17] stated that any blockchain-based EHR system should be able to "uniquely" identify and distinguish each user while maintaining their anonymity on the blockchain, securely authenticate users, and be capable of recovering a user's authentication information if it is lost or stolen.
4. *Support for structural interoperability at a minimum:* The system should enable the exchange of medical data and interpretation of the received data in its current standards [17], i.e., the system should be able to communicate with known industry standards, such as FHIR and HL7.
5. *Scalability across large populations of healthcare participants:* This was described in [17] as follows: "A successful health app should leverage the Blockchain to enhance interoperability, while maintaining its quality when users or components of the app scale up and out".
6. *Cost-effectiveness:* Any blockchain-based EHR system should be cost-effective compared to the existing legacy systems without affecting its capabilities [17]. This factor has a significant impact on the selection of the blockchain's parameters, including its type, consensus algorithm, and incentive model.
7. *Support for a patient-centered care model:* According to [17], any blockchain-based EHR system should provide patients with the ability to control or monitor their information without compromising other functionalities. These features may include self-reporting health information, access to personal medical records and prescription histories from different providers, auditing existing access to patient health records, and the ability to share or revoke access to patients' own medical data.

### 2.3. MedRec

MedRec was proposed by Azaria, Ekblaw, Vieira, and Lippman [19] for the utilization of blockchain technology to integrate existing centralized EHR systems among distributed healthcare providers. This solution uses the Ethereum blockchain's smart contract capability to facilitate this integration. Each healthcare provider should contribute an Ethereum mining node (usually a dedicated server) to participate in MedRec. In addition, patients should also contribute an Ethereum mining node (on a PC or mobile device) to participate in MedRec. The main functions of MedRec are to:

- Enable inter-provider access to patients' EHRs by using API interfaces. The API information of the providers is stored in the blockchain.
- Provide patients with the capability for managing access control for their EHRs. Access control lists for patients and providers are stored in the blockchain.
- Detect and notify patients about new access requests for their EHRs. Access can be granted or rejected only by patients.
- Notify patients about changes to their EHRs and log the changes in the blockchain.
- Provide a copy of the EHRs on patients' nodes and dominant providers' nodes.

Although MedRec accelerates the deployment of EHR systems by integrating with existing systems using blockchain technology to overcome the major limitations of centralized EHR systems, MedRec has shortcomings that limit its feasible production and implementation:

- A mandatory component of MedRec is the presence of patients' nodes, which are used to communicate with patients for access control management. This limits the scope of MedRec solutions to blockchain-enabled patients (i.e., patients should have an Ethereum account). This is a major limitation of the solution from the patients' perspective (but not the providers' perspective). Any proposed solution should be capable of supporting all patients without restrictions.
- The Ethereum blockchain uses POW as its consensus algorithm, which is known to have significant computing power requirements. While healthcare providers can contribute powerful mining nodes to use MedRec, this cannot be (practically) achieved for patients, whose nodes are on PCs or mobile devices, making MedRec practically infeasible.
- If a patient loses the private key to their Ethereum account (which is possible when using a mobile device or PC), MedRec does not provide a mechanism for a patient to recover control of their EHRs.
- The use of current centralized EHR systems raises an interoperability problem regarding inter-provider access. The solution must assume that all providers utilize the same EHR format standard, such as HL7 or FHIR [19], which is not the case with the current centralized systems.
- MedRec does not provide a mechanism for emergency access to EHRs if a patient is admitted to a non-authorized hospital for emergency treatment.

### *2.4. BBDS*

Xia, Sifah, Smahi, Amofa, and Zhang [20] proposed a blockchain-based data-sharing (BBDS) system to provide access control management for EHRs stored in the cloud based on the blockchain technology. The proposed BBDS system utilizes permissioned blockchain consisting of an issuer that grants users or organizations access to the system, a verifier that validates requests from system members and grants corresponding access rights, and consensus nodes that facilitate the interface between members and the verifier, in addition to logging requests in the blockchain for auditing and forensic purposes. The BBDS system provides the following functionalities:

- A proof-of-verification (POV) algorithm between the issuer and users/organizations to enroll them in the BBDS system. The POV algorithm is based on a proposed lightweight Diffie–Helman key exchange to generate a session key for encryption and an electronic registration form to be validated by the issuer.
- Controlled access to EHRs, stored in the cloud, by a verifier node for members of the BBDS system. This verification process is based on a per-member private key that is generated during the registration phase by the issuer and communicated to members and the verifier. After successful verification of a member's identity, the verifier validates the request against member rights and, if access is granted, the verifier retrieves data from the cloud and passes them to the member or reads data from the member and posts them in the cloud.
- Audit logging in the blockchain ledger by using consensus nodes, where each member's request to read or post an EHR is stored in a separate block. The information recorded in the block includes the user identity, purpose of the request, processing consensus node, verification result, and timestamps, including request creation, request retrieval from an unprocessed request pool, verification time, block broadcast time, and data sending time.

Although the BBDS system is not limited by cryptographic key recovery and emergency access restrictions (because the actual EHRs are not stored in the blockchain and their access is controlled by the issuer/verifier), it has other limitations in the current implementation:

- The use of a permissioned blockchain eliminates decentralized authority, which is a core advantage of blockchain technology. The model provides central authority to the issuer (not the patient) for verifying and accepting members in the system.

- Because of the use of a non-Turing-complete blockchain (i.e., there are no smart contracts), the BBDS system records each event in a single block to be able to uniquely identify the events by the block reference. This limits the scalability of the system because of numerous blocks are recorded in a very short time.
- The proposed model provides the data to the requester before recording the request details in the blockchain, which introduces vulnerability in the system, as data are provided without a recorded request.
- The proposed BBDS system does not have a mechanism for detecting modifications/tampering in EHRs in the cloud caused by system-independent reasons, such as malicious activity in the cloud.

### *2.5. MedShare*

MedShare was proposed by Yang et al. [21] to connect centralized healthcare entities and exchange EHRs by using a hybrid cloud infrastructure. The proposal was prototyped with three healthcare entities: Hospital Conde S. Januário (HC), Kiang Wu Hospital (KW), and Macau University of Science and Technology Hospital (UH). Medshare functions as follows:

- Each healthcare entity has a private cloud that converts EHRs from an entity's specific format into a standard EHR format and stores them locally in a private cloud. In other words, each entity has two copies of an EHR—in a standard format and a non-standard format.
- Standard-format EHRs are indexed by using hash maps, and the index values are stored in a public cloud that is connected with private clouds. The public cloud has a synchronizer component that is used to replicate per-patient EHRs across all private clouds (scheduled replication).
- Doctors locally authenticate with the healthcare entity and query the EHRs of patients. If a healthcare entity cannot find a patient ID locally (assuming that replication has not yet been done), it queries the public cloud to locate the patient's EHR and, after successful validation, obtains the EHR.

The Medshare model is very practical and overcomes major limitations in legacy EHR systems, but it lacks the following:

- Neither private nor public clouds guarantee immutable access control rules, privacy isolation between patients, or immutable integrity verification, which are provided by blockchain technology.
- The replication of EHRs among healthcare entities is not scalable when a large number of healthcare entities are involved in the system. This requires $n \times (n-1)$ connections to achieve full replication of EHRs.
- Medshare uses patients' ID cards as a mechanism for uniquely identifying patients and obtaining their consent to grant healthcare providers access rights for their EHRs, which is known to be an insecure technique compared to biometric identity verification.

### *2.6. OmniPHR*

Roehrs, da Costa, and da Rosa Righi [22] proposed a blockchain-based application, OmniPHR, to address the following problems:

- Providing patients with a unified view of their healthcare records from anywhere at any time.
- Providing up-to-date information to healthcare providers about patients regardless of whether the data are local to the provider or are from an external provider.
- Providing a single standard for healthcare records.

Each member of OmniPHR joins the blockchain through a miner, which is called a leaf node. OmniPHR uses a 'routing overlay' node (called a super node), which is responsible for managing leaf nodes and inter-communication with other routing overlays. Some other roles of the routing overlay node are:

- *EHR handling*: Accepting input medical records from IoT devices or healthcare organizations, converting them into an open EHR format (the standard EHR format used by OmniPHR), dividing the EHRs into chunks of blocks, and distributing the blocks across blockchain miners by using load-balancing algorithms.
- *Security*: Encrypting blocks, signing blocks, validating blocks, and providing access to authentication and access control to the blocks.

OmniPHR has limitations in its capability for providing a unified EHR system:

- Similarly to BBDS, the use of a non-Turing-complete blockchain adds significant complexity for additional features or enhancements in the system compared to a Turing-complete blockchain, which can have added features through software coding.
- Storing large amounts of data in the blockchain (e.g., X-rays and MRI scans) is not practical due to the size requirements on the nodes, which entail significant overhead aside from the encryption and decryption processing overhead.
- The proposed model does not uniquely identify the author of data because all of the blocks are signed by the leaf nodes or super nodes.
- Access to EHRs should be authorized by patients, which does not address the limitation of unplanned treatment, such as emergency admission by unauthorized healthcare providers.
- OmniPHR does not overcome the limitation of duplicate data, such as duplicate patient registration information, which occurs when a patient registers with two healthcare providers with different identities. Ideally, OmniPHR should have a mechanism to "uniquely" identify each patient without duplication, such as biometric identity.

### 2.7. MedBlock

Fan, Wang, Ren, Li, and Yang [23] proposed the MedBlock system to share medical data efficiently using blockchain. The MedBlock system generates a private/public key pair for each patient, which is used to encrypt and sign medical records. The actual records are stored in the health provider's local database, while the blockchain holds the hash values of the records. The core functions provided by MedBlock are:

- A dedicated certificate authority server is used to generate keypairs for patients, community hospitals, and national hospitals.
- Patients submit their records through community hospitals or national hospitals signed with their private key and encrypted by their public key.
- Health records are not stored in community hospitals. Instead, they are stored directly in national hospitals' databases.
- The department that accepted records from a patient signs them using its local private key to ensure integrity and non-repudiation.
- Each geo-group of national hospitals has the same group of endorsers that will build the blocks and submit them to the consensus nodes (called orderers). The hospitals submit the hash values of the medical records to the endorsers, and the records are stored in the local database.
- Once the orderers reach a consensus, they post the block to the ledger.
- Access control is implemented by MedBlock through private key signatures. The client application scans the blocks until a valid signature that corresponds to the patient's data is found.

While MedBlock provides an efficient and scalable mechanism that uses role-based nodes to perform specific functions and guarantee security through double-signing, it has the following drawbacks:

- The use of private/public keys for patients to sign and encrypt records creates an issue in the case of unplanned treatment, such as emergency admission. In such a case, medical records will not be accessible, which can cause complications with treatment and even fatality.
- If a private key is lost, patients cannot recover their medical records.

- The access control mechanism used is inefficient, especially when the ledger grows to a very large number of blocks. In this case, examining all of the blocks until the records are found is not scalable.
- The lack of programmability is a major drawback of MedBlock if new features are required, as this would require the addition of new nodes.
- MedBlock does not provide a mechanism for exchanging medical records between hospitals, as the records are stored in the local databases of national hospitals.

Table 1 summarizes the main existing blockchain-based EHR implementation solutions. As we can notice, most of the existing solutions rely on the implementation of Ethereum and focus mainly on access control services. However, none of the existing solutions have discussed the adversary models that their solutions can face. In addition, all of them rely on the public/private key approach, which means that the whole system is unsecured against authorized internal intruders.

**Table 1.** Existing blockchain implementations for EHRs.

| Reference | Blockchain Trype | Main Feature | Targeted Security Service | Adversary Model |
|---|---|---|---|---|
| Tanwar et al. [13] | Hyperledger Fabric | Fast data accessibility | Access control | Not specified |
| Shahnaz et al. [12] | Ethereum | Off-chain data storage | Access control | Not specified |
| Azaria et al. [19] | Ethereum | Easy system integration with collaborative mining | Access control and availability | Not specified |
| Xia et al. [20] | Any permissioned blockchain | Secure data sharing the blockchain | Access control and availability | Not specified |
| Yang et al. [21] | Hybrid cloud infrastructure | Real-time testbed in three hospitals | Privacy and access control | Not specified |
| Roehrs et al. [22] | Not specified | Efficient datablock distribution | Authentication, privacy, and digital signature management | Not specified |
| Fan et al. [23] | Not specified | Scalable and secure interactions using double signing | Confidentiality and access control | Not specified |
| Proposed solution | Ethereum | Independent from private/public key standard approach | Privacy, access control, and confidentiality | Unauthorized access, read/write, and anonymity attacks |

To face the existing approaches' shortcomings, in the following section, we present our proposal—the BBEHR architecture—and its implementation.

## 3. System Architecture

In this section, we will provide an overview of the BBEHR architecture according to the design proposed in [8]. The approach followed in BBEHR the division of the system into layers and the provision of distributed functions in a modular structure. Accordingly, the system is divided into four layers, which are the User Interface (UI), Middleware, Blockchain, and Cloud Store. Figure 1 shows the layers that comprise BBEHR and the modules associated with each layer.

The UI layer is the presentation of the system to the users, through which they can interact with BBEHR. These users can be doctors, pharmacists, receptionists, officers, researchers, or insurance companies, among others. This layer is local to each healthcare provider that is a member of BBEHR. It has mandatory and customized components according to a provider's needs.

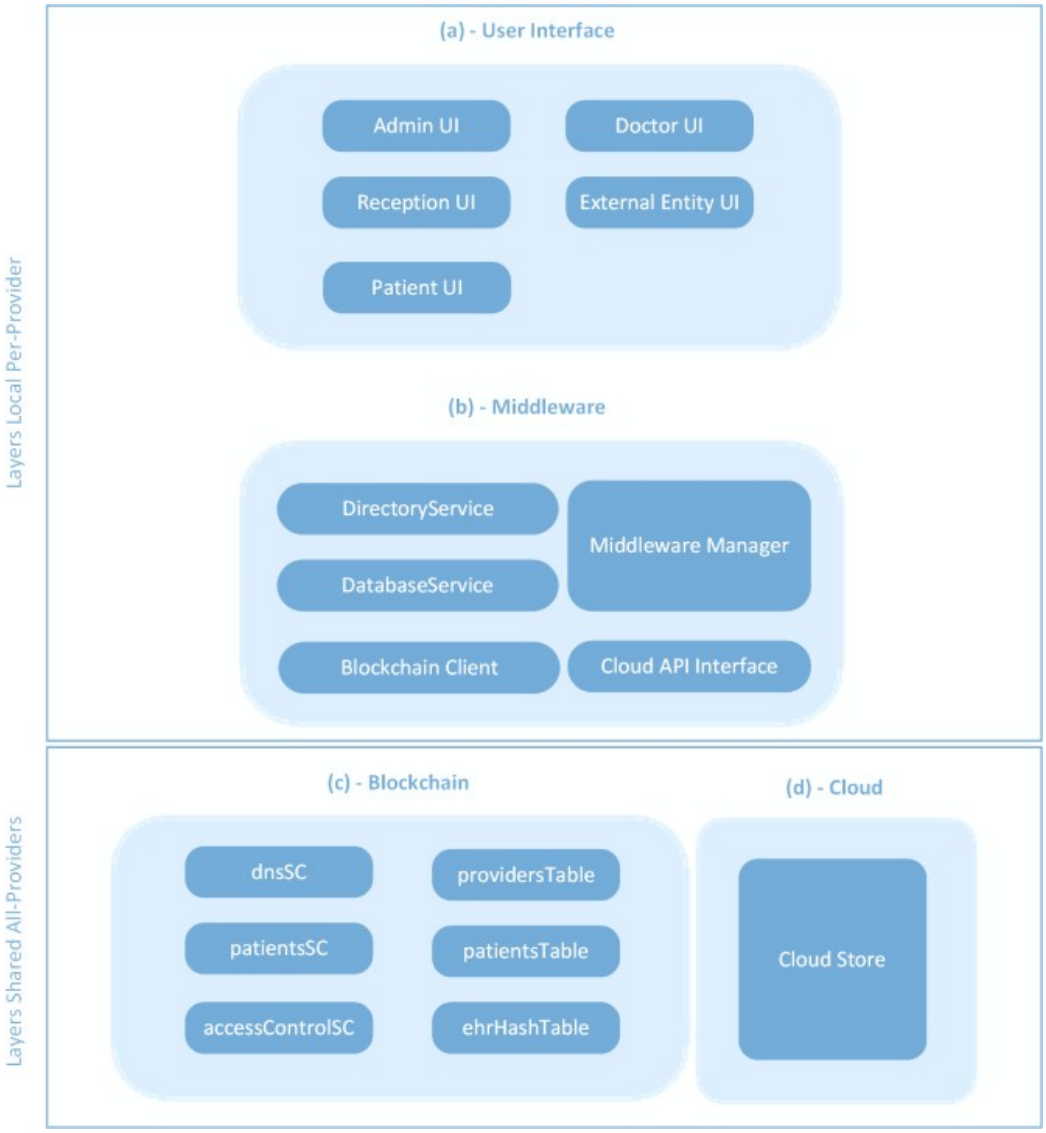


**Figure 1.** Overview of the BBEHR architecture.

The Middleware layer is the core of the BBEHR solution. It is used to interlink all of the other BBEHR layers, in addition to providing major BBEHR services, including directory services and database services.

The Blockchain layer is a shared layer across all healthcare providers participating in BBEHR. This layer provides two core functions in the BBEHR system, namely:

- Immutable smart contracts (SCs) for performing programmed logic functions.
- Immutable tables in the form of chained blocks to store different types of data that need to be protected against unauthorized tampering.

The Cloud Store layer is a shared layer across all healthcare providers participating in the BBEHR system. It holds patients' raw EHRs. The indexing keys for raw EHRs are stored in the Blockchain's immutable tables. Details of the individual components and their inter-operation, including functional use cases, are included in [8].

## 4. System Implementation

This section covers the implementation of a BBEHR prototype, which includes the selection process for the technologies used for the implementation, followed by an overview thereof.

### 4.1. Functional Requirements

The components used in this implementation were selected based on the following criteria:

- Simplicity of the component setup and configuration for implementing the required functions in BBEHR;
- Interoperability capabilities of the component with other technologies;
- Feature richness and built-in security capabilities of the component that are in line with the BBEHR requirements;
- Stability, reliability, and operational consistency of the component.

Based on the above requirements, the following technologies were selected:

- *Django Web Development Platform:* This was used to build the required web interfaces for the Doctor UI, Admin UI, and Reception UI. Django is a Python-based platform that has built-in directory services. This feature allows Django to integrate seamlessly with customized Python modules to provide additional functions, such as integration of the API with Ethereum.
- *Python*: The Python interpreter is at the core of the BBEHR prototype and provides the following functions: the Django web development coding language; interaction with Django's built-in directory services; API integration with the blockchain and cloud store; SQL interface with the healthcare provider database; integrity validation and hashing of EHRs.
- *SQLite*: SQLite was used to implement the database service module locally to the healthcare provider, which was mainly for storing information, including departments' IDs, appointments, and the provider's blockchain details. SQLite has a native database connector with Django through Python.
- *Ethereum blockchain:* This is a public blockchain technology that is used to provide immutable smart contracts and immutable databases. Ethereum was chosen to extend the accessibility to EHRs at a large scale, including cases in which patients relocated to different geographic areas. In such cases, a new geo-healthcare provider can connect to Ethereum and request access to a patient's EHRs.
- *Microsoft (MS) Azure Files:* This service is hosted on the public cloud to store EHRs. The selection of MS Azure Files was due to its simple accessibility and usability, in addition to its independence from the format of EHRs. MS Azure Files uses an SMB protocol to provide secure communication between an on-premise infrastructure and the cloud [24].

### 4.2. Django Architecture

Django consists of front-end and back-end layers. The front-end layer comprises HTML templates with which clients interact. The back-end layer, on the other hand, gets inputs from templates and performs the programmed functions accordingly. Figure 2 shows a block diagram of the Django architecture.

Django simplifies HTML coding by using Forms, which are Python functions defined in the forms.py file. These functions specify HTML input fields and their labels, types, maximum lengths, etc. (Forms do not include HTML styles and javascripts). The actual HTML pages presented to clients are combined versions of style sheets, javascripts, bootstraps, and other elements defined in HTML template files and with inputs returned from forms.py.

The input values returned from the clients can be written into the database through Models (models.py). Models act as an abstraction layer, and they are provided by Django and programmed using Python. They can translate Python instructions into database queries depending on the type of integrated database. Models can receive input values from HTML templates through Views (views.py). Additionally, Models can poll data from the database and pre-fill Forms' inputs to be presented to clients through HTML templates (such as through dropdown selection).

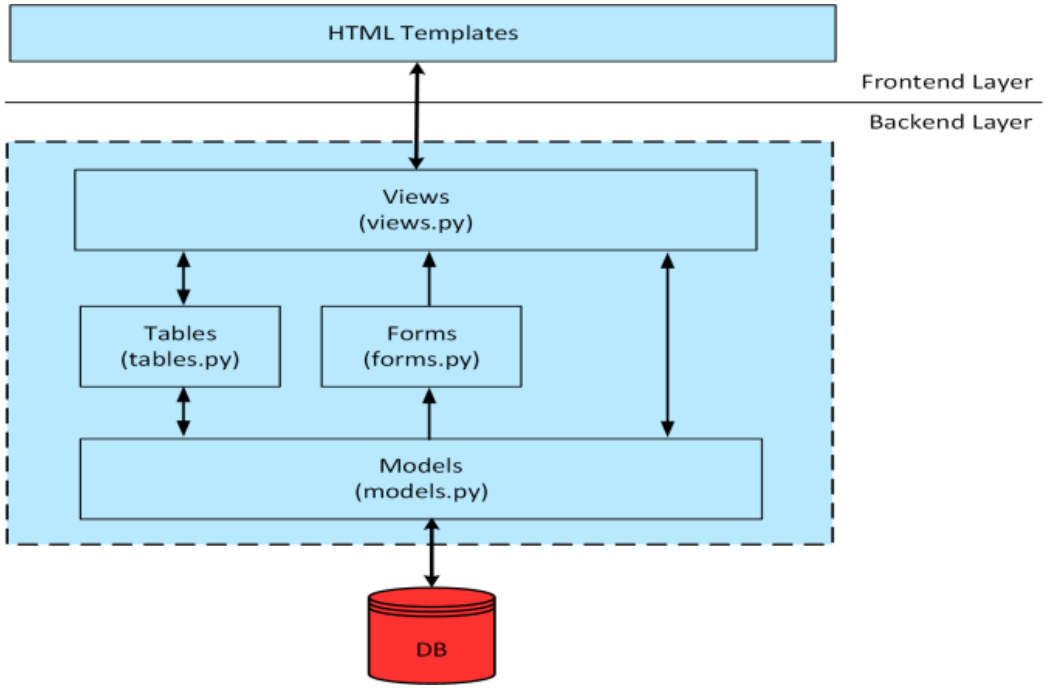


**Figure 2.** Django Web Development architecture.

Tables (tables.py) in Django are used to populate information from the database in table format and present it to clients. This simplifies the process of creating HTML tables compared to traditional HTML methods. Tables do not handle styles, as this is controlled through HTML templates.

Django uses Views to link Forms, Tables, and Models with Templates. Views control the logic of the Django web flow and how requests/responses are handled between clients and the web application. It is at this point that the core functions of BBEHR are implemented. Additionally, Views allow the import of custom Python models for extended functionalities that are not present in Django.

### *4.3. Implementation Steps*

The implementation steps of the BBEHR prototype are summarized as follows:

- Building the runtime environment.
- Initializing the web and database components of BBEHR.
- Building the BBEHR Django code.
- Building Ethereum smart contracts.
- Integrating Django with Ethereum.

#### 4.3.1. Building the Runtime Environment

The first step in building the BBEHR prototype was setting up the runtime environment that hosted the prototype components (described in Section A). These components are independent of the operating system. MacOS Catalina was selected as the operating system hosting the BBEHR system in the healthcare provider in order to run Django, Python, and SQLite and to communicate with Ethereum and MS Azure.

The next step was to download and install Python 3.8 and PyCharm 2019.3.2 IDE. From PyCharm IDE, a new virtual environment was created to use Python 3.8 as an interpreter for Django and the custom Python modules. This was followed by creating a new project in PyCharm called BBEHR to run on the created virtual environment, i.e., it was interpreted by using Python 3.8.

Following the creation of the BBEHR project in PyCharm, a list of required Python libraries was installed. Table 2 summarizes this list of libraries. Python used the PIP3 utility to install external libraries from the internet.

**Table 2.** Python libraries required for BBEHR.

| Library Name | Purpose |
|---|---|
| django | Web development framework; this installation includes SQLite 3 |
| django-Tables2 | For formatting and styling tables in Django |
| crispy | For formatting and styling HTML templates in Django |
| web3 | For API communication with Ethereum |
| azure-storage-file | For API communication with MS Azure |

#### 4.3.2. Initializing the BBEHR Web and Database Components

After preparing the runtime environment for hosting BBEHR, the next step was initializing the Django web framework and database. The initialization was done in the following order to ensure the successful running of Django:

1. Initializing SQLite3 DB to be ready for storing the provider's Ethereum information and clinics' information. This was done by running the following commands from the PyCharm BBEHR project terminal: *python manage makemigrations*; *python manage migrate*.
2. Creating a Django admin user to administrate the Django management console, including account creation in the Django built-in directory service. These accounts represented doctors, nurses, officers, and receptionists. Additionally, the admin user populated the SQLite3 database with information about the healthcare provider's clinics and Ethereum information. The command for creating an admin user was: *python manage createsuperuser*
3. Creating BBEHR user accounts and groups by navigating to http://localhost:8000/admin, accessed on 31 October 2022, signing in using the admin account, adding groups for different permissions, adding new users, and assigning users to their respective groups.
4. Creating local database tables for the BBEHR healthcare providers. These tables were stored in the SQLite3 database with the following structures: (i) Appointments Table: Name Column (patient name), NID, Date Column, Time Column, and Department Column (clinic to be visited). (ii) Departments Table: Code Column (clinic code) and Name Column (clinic name). (iii) Provider Table: OHP Column (Ethereum public key) and Secret Column (Ethereum private key). The Python code for creating these tables was written in models.py. Below is a sample of the code (see Figure 3).
5. From the Django admin console, populating departments and provider database tables with providers' clinics' details and Ethereum details, respectively (the appointment database is populated by the receptionist, as described later).

#### 4.3.3. Building the BBEHR Django Code

After the successful initialization of Django, the next step was writing the Django Python code to perform the required functions of the BBEHR prototype.

The first step in the Django coding was developing the HTML templates for the front-end layer. The approach for HTML coding was based on developing a base template (base.html) containing all shared components across all pages, such as the header, footer, title, and styles. Any child HTML template had page-specific components combined with the base template presented to the user. Figure 4 summarizes the HTML coding approach along with all of the HTML templates.

```
class department(models.Model):
  code = models.IntegerField(unique=True, primary_key=True)
  name = models.CharField(max_length=100)

  def str (self): return self.name


class provider(models.Model):
  ohp = models.CharField(max_length=100, primary_key=True)
  secret = models.CharField(max_length=100)

  def str (self): return self.ohp


class appointment(models.Model):
  name = models.CharField(max_length=100) nid = models.IntegerField()
  date = models.DateField(default=timezone.now) time = models.TimeField(default=timezone.now)
  department_code = models.ForeignKey('department', on_delete=models.CASCADE)

  def str (self): return self.name
```

**Figure 3.** Creation of the BBEHR tables.

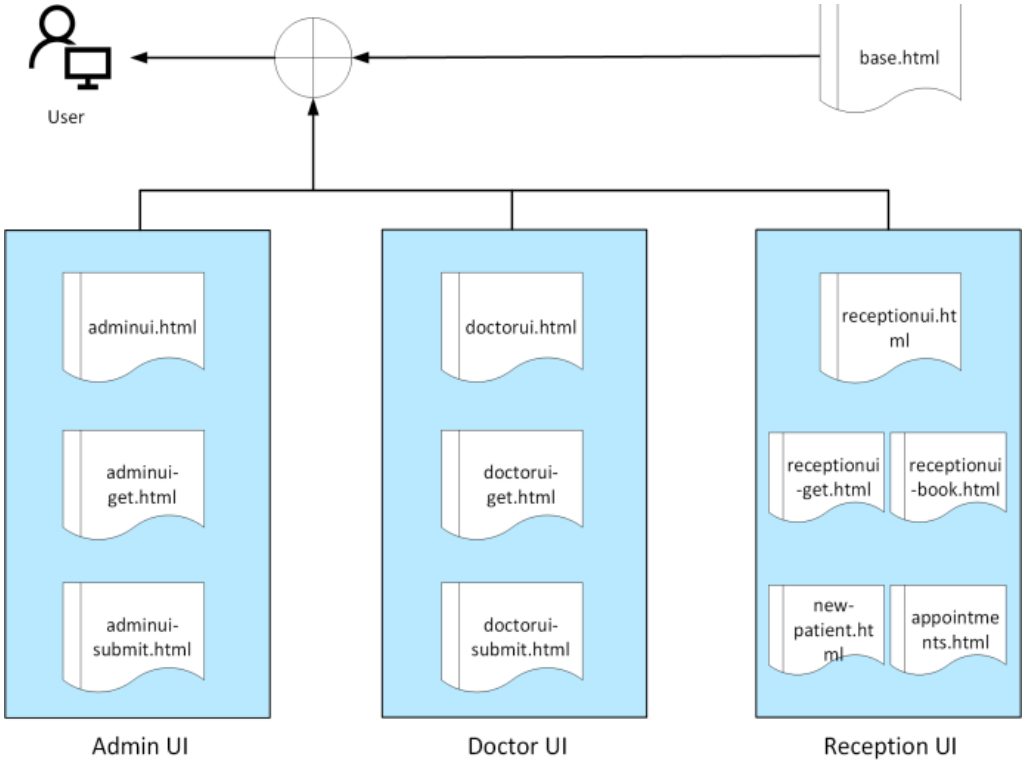


**Figure 4.** Summary of the HTML BBEHR template structure.

After creating the HTML templates, the next step was building the functions in forms.py (for the input fields to be presented with each template) and associating each form's function with its corresponding HTML template through functions in views.py. Figures 5–7 provide summaries of forms.py, views.py, and the corresponding UI.

Below is a sample of code in views.py, which links adminui-get.html with the getProviderInfo function from forms.py. The page should display one field for the user to enter 'Hospital Ethereum Address' (see Figure 8).

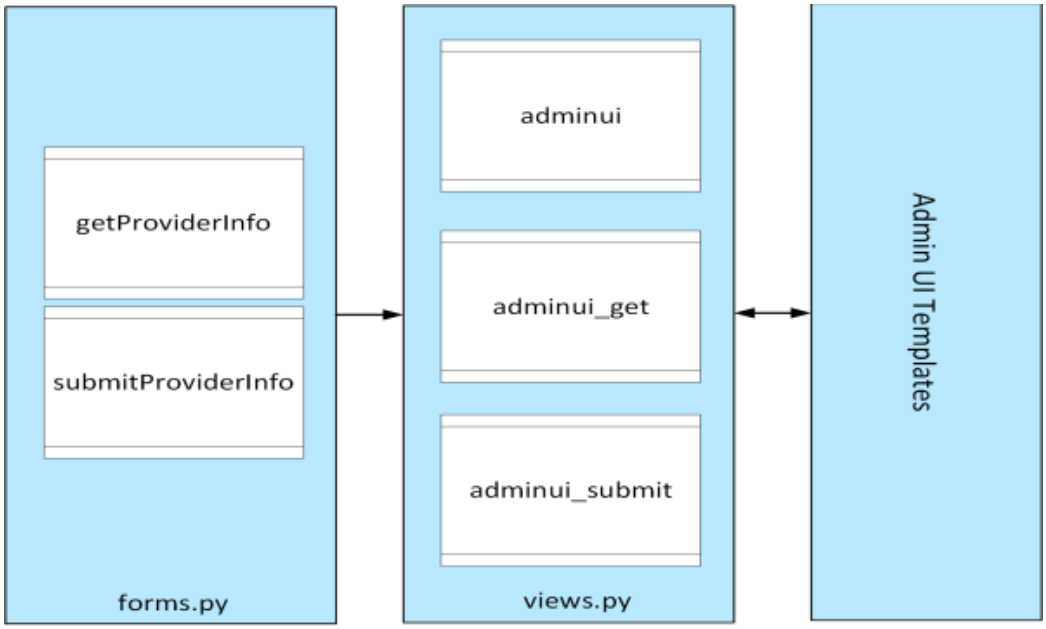


**Figure 5.** Admin UI with forms and views.

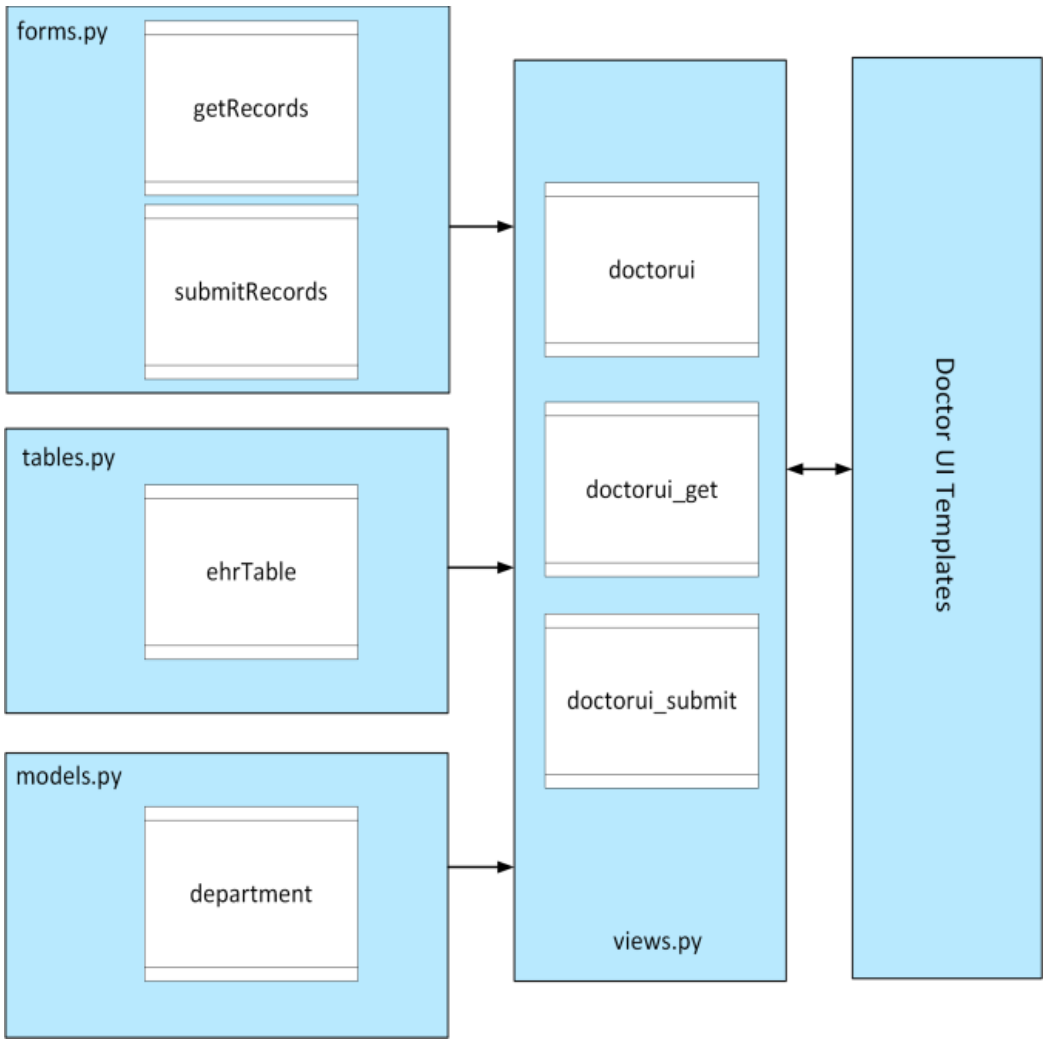


**Figure 6.** Doctor UI with forms and views.

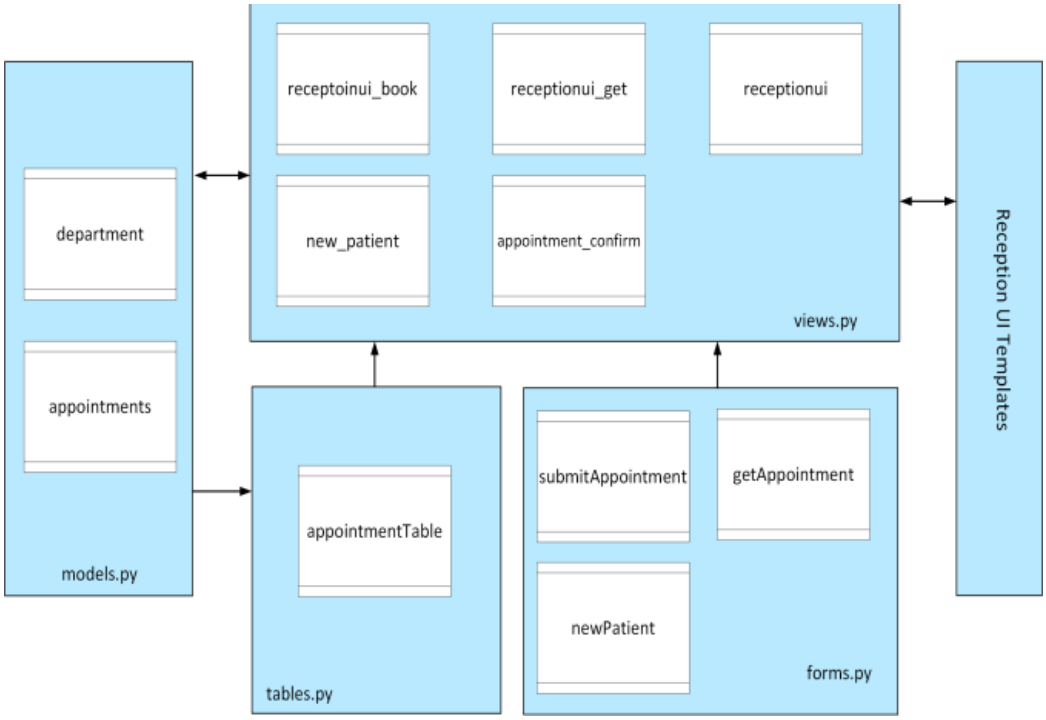


**Figure 7.** Reception UI with forms and views.

```
views.py
*****************************************

def adminui_get(request):
  # if this is a POST request we need to process the form data
  if request.method == 'POST':
    # create a form instance and populate it with data from the request:
    form = getProviderInfo(request.POST) # check whether it is valid:
    if form.is_valid(): account_address =
form.cleaned_data.get('OHP_Eth')
      provider = dnsSC_get(account_address) context = {
        'provider': provider
      }

      return render(request, 'SDHCARE/adminui-get.html', context)

  # if a GET (or any other method) we'll create a blank form else:
    form = getProviderInfo()


forms.py
*****************************************

class getProviderInfo(forms.Form):
  OHP_Eth = forms.CharField(label='Hospital Ethereum Address', max_length=100)
```

**Figure 8.** Linking adminui-get.html with the getProviderInfo function.

Access to each set of UIs was controlled using role-based access control (RBAC) implemented by Python decorators. The decorator obtained the session username, verified its group membership (which was configured during the Django initialization), and allowed access only if the user was a member of the required group. Below is a sample code of using decorators to limit doctors' access to the Doctor UI only. These decorators were implemented in views.py (see Figure 9).

```
@custom_user_passes_test(lambda u: Group.objects.get(name='SDHCARE-Admins') in u.groups.all())
def adminui(request):

@custom_user_passes_test(lambda u: Group.objects.get(name='SDHCARE-Admins') in u.groups.all())
def adminui_get(request):

@custom_user_passes_test(lambda u: Group.objects.get(name='SDHCARE-Admins') in u.groups.all())
def adminui_submit(request):
```

**Figure 9.** Role-based access control (RBAC) implementation.

Another access control mechanism is applied to grant permissions to access patients' EHRs by using decorators and smart contracts. Clinics are granted permission to access patients' EHRs only if the session user is a member of the BBEHR Reception group and the patient submits a valid fingerprint. A third form of access control is applied for doctors' access to EHRs, which is covered in the next section. The last step in the Django coding was linking the models to UIs through views. In Figures 5–7, the sets of functions in models.py were mapped to their respective UIs to have information read from the database and displayed to clients, such as a patient's appointments or a provider's OHP address, or to write information such as the booking of a new appointment into the database. Some database information, such as appointments, is formatted in tables; hence, the information from models.py is passed to views.py through tables.py.

In the following, we summarize the functions configured in the views, forms, and tables. These functions describe how the BBEHR design requirements are implemented in Django. The functions in models were described in the previous section that outlined the initialization of the Django SQLite database.

#### 4.3.4. Summary of View Functions

1. *home*: To render the home page template for users.
2. *about*: To render the about page template for users.
3. *adminui*: (i) Verifies that the session user is a member of the BBEHR Admin group. (ii) Renders the base and adminui templates for admins to select 'Enter Hospital Information' or 'Get Hospital Information'.
4. *adminui_get*: (i) Verifies that a session user is a member of the BBEHR Admin group. (ii) Reads the admin OHP input and obtains the hospital information stored in Ethereum providersTable for that OHP; this information is returned to the admin.
5. *adminui_submit*: (i) Verifies that the session user is a member of the BBEHR Admin group. (ii) Reads admin input (hospital name and web address) and stores the information in Ethereum providersTable using OHP as the indexing key; OHP is obtained from the address of the transaction sender.
6. *receptionui*: (i) Verifies that the session user is a member of the BBEHR Reception group. (ii) Renders the base and receptionui templates for reception to select 'New Patient', 'Get Patient Appointments', or 'Book New Appointment'.
7. *receptionui_get*: (i) Verifies that the session user is a member of the BBEHR Reception group. (ii) Accepts a patient's fingerprint and uses it as an index key to obtain all active appointments associated with that patient.
8. *appointment_confirm*: Performs two-factor validation by verifying a patient's fingerprint and reception group membership; if both are valid, the clinic is granted access to the patient's EHRs; this access is stored in the Ethereum patientsTable and covers all doctors in that clinic.
9. *receptionui_book*: (i) Verifies that the session user is a member of the BBEHR Reception group. (ii) Books new appointments for patients and stores them in the appointment database, which is indexed using the patient's NID.
10. *new_patient*: (i) Verifies that the session user is a member of the BBEHR Reception group. (ii) Creates a new patient record in the Ethereum patientsTable.

11. *doctorui*: (i) Verifies that the session user is a member of the BBEHR Doctor group. (ii) Renders the base and doctorui templates for the doctor to select 'Get Patient EHRs' or 'Submit Patient EHRs'.
12. *doctorui_get*: (i) Verifies that the session user is a member of the BBEHR Doctor group. (ii) Verifies that the doctor's clinic is granted access to the patient's EHRs using the patient's fingerprint. (iii) Displays a list of EHRs and hashes to the doctor, which are indexed using patients' fingerprints (from the Ethereum ehrHashTable); these records are formatted in a table before being passed to doctors. (iv) Once the doctor selects an EHR, this verifies that the cloud EHR's hash is the same as the hash in ehrHashTable. (v) Retrieves the EHR from Cloudstore.
13. *doctorui_submit*: (i) Verifies that the session user is a member of the BBEHR Doctor group. (ii) Verifies that the doctor's clinic is granted access to a patient's EHRs using the patient's fingerprint. (iii) Creates an EHR based on information submitted by a doctor; the EHR is stored in the Ethereum ehrHashTable. A Merkle root hash is calculated for the EHR and uploaded with the record.

#### 4.3.5. Summary of Forms' Functions

1. *getProviderInfo*: Presents one input field to the provider's admin: the provider's Ethereum address (OHP).
2. *submitProviderInfo*: Presents two input fields to the provider's admin: provider name and provider web address.
3. *bookAppointment*: Presents the following fields to the receptionist: name, fingerprint, date, time, and dropdown for the clinics.
4. *getAppointments*: Presents a single field to the receptionist: the patient's fingerprint.
5. *newPatient*: Presents the following fields to the receptionist: name, date of birth, and fingerprint.
6. *submitRecords*: Presents the following fields to the doctor: patient name, patient fingerprint, record name, record date, and record description.
7. *getRecords*: Presents one field to the doctor: fingerprint.

#### 4.3.6. Summary of Tables' Functions

1. *appointmentTable*: Formats the appointments retrieved from the local database in a table before posting them to the receptionist.
2. *ehrTable*: Formats the EHR list retrieved from the Ethereum ehrHashTable in a table before posting them for the doctor.

#### 4.3.7. Building Ethereum Smart Contracts

The prototype for BBEHR has three smart contracts, which are known as dnsSC, patientsSC, and accessControlSC, and they provide the required design functions. The main reason for creating three smart contracts instead of a combined one is to provide flexibility in extending the functionality of the BBEHR prototype by inheriting and importing smart contract functions. The dnsSC has two functions:

- *createProvider*: This function takes two string inputs for the healthcare provider's name and web address. These values are stored in the immutable database of the providersTable, which is indexed by the provider's Ethereum address (OHP). In the case of an existing provider, the function returns an exception error.
- *getProvider*: This function takes an address input (OHP) and returns two string variables that represent the provider's name and web address, and these are stored in the providersTable.

Similarly to dnsSC, accessControlSC has two functions:

- *addClinic*: This function takes two string inputs representing the patient's fingerprint hash and the clinic ID. The clinic ID is polled from the SQLite3 department database. Both strings are stored in an array indexed by the patient's fingerprint hash that

represents all of the clinics that can access the patient's EHRs. This array is part of the patientsTable.

- *grantClinicAccess*: This function takes two string inputs, namely, the patient's fingerprint hash and the clinic ID. It performs a lookup in the patientsTable by using the patient's fingerprint hash array to determine whether the clinicID is listed. If the clinicID is listed in the array, it returns 'True', which allows the doctor to access the patient's EHR. Otherwise, it returns 'False', which denies the doctor's access.

The patientsSC includes the following four functions:

- *createPatient*: This function accepts string inputs for the patient's fingerprint hash, name, date of birth, provider's Ethereum address, and reception ID. It stores this information in the patientsTable indexed by the fingerprint hash.
- *getPatient*: This function accepts a string input of the patient's fingerprint hash and returns the patient's stored values in the patientsTable (i.e., name, date of birth, provider's address, and reception ID).
- *createEHR*: This function accepts the patient's fingerprint hash, EHR name, date, status, and Merkle root hash. It stores the values in the ehrHashTable indexed by the fingerprint hash.
- *getEHR*: This function accepts a fingerprint hash input string and returns the patient's EHR list. For each EHR, the returned values are the name, date, Merkle root hash, and EHR status.

The smart contracts were deployed in Ethereum by using the MetaMask soft wallet, and each contract was allocated a unique address for communication. The provider was used to deploy smart contracts for demo purposes.

#### 4.3.8. Integrating Django with Ethereum

The integration between the Django web application and Ethereum was implemented by using Web3-customized Python modules on the Django side and the Infura mining pool on the Ethereum side. Figure 10 is a summary of the integration between Django and Ethereum.

To communicate with the Ethereum blockchain, a healthcare provider should contribute with a dedicated mining node running an Ethereum mining software (e.g., Get Ethereum, or GETH). A dedicated node is used to ensure patients' privacy. For the prototype, we leveraged the Infura mining pool, which offers mining nodes as a service to interact with Ethereum. The free version of Infura offers 100,000 Ethereum transactions within 24 h. An Infura account was created, which provided a unique URL for communicating with Infura nodes for posting and reading blocks to and from Ethereum.

For Django views to interact with Ethereum, custom Python modules were built by utilizing Web3 APIs. Each Python module has functions for interacting with the respective smart contract functions. Figure 11 summarizes the operation of the Python modules.

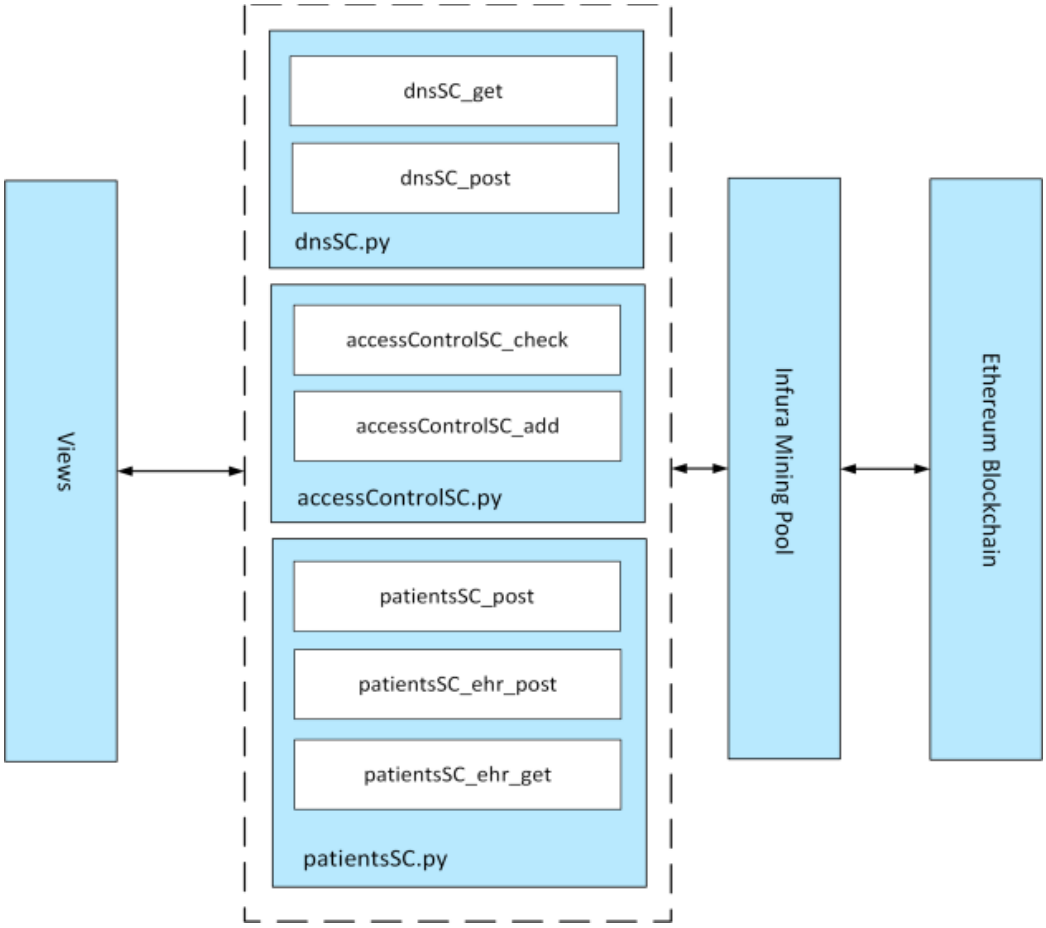


**Figure 10.** Integrating Django with Ethereum.

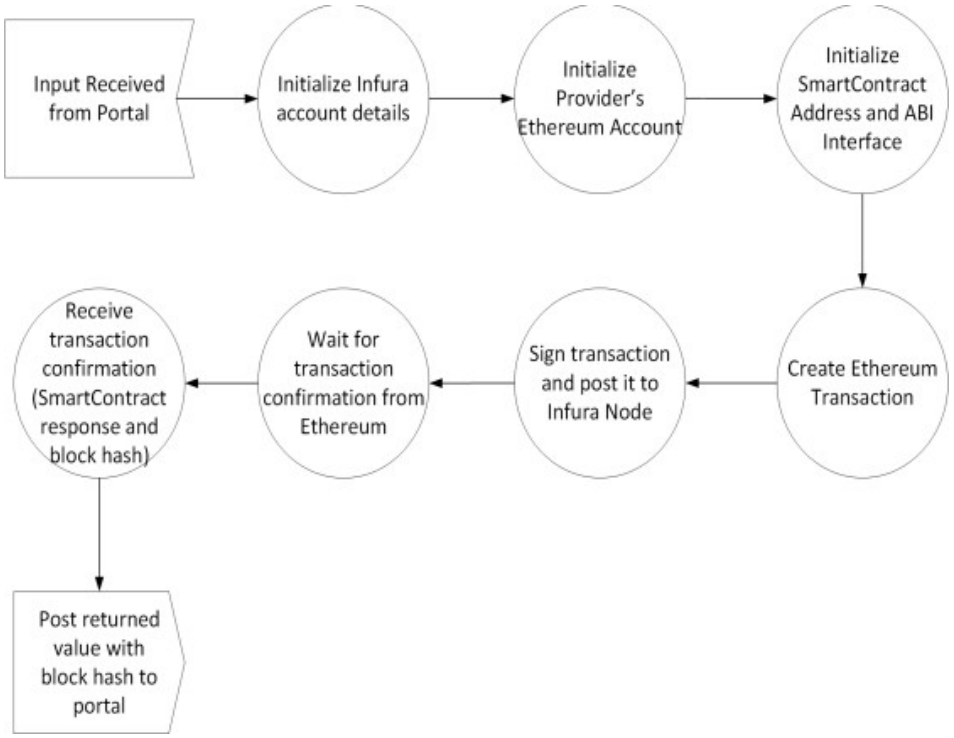


**Figure 11.** Custom Python module operation.

## 5. Testing and Performance Evaluation

The prototype implementation of BBEHR was tested from the functional, security, and performance perspectives. Functional testing ensured that the prototype is operating as expected by design. Security testing validated the security measures implemented in the prototype to protect it against unauthorized access and modifications of EHRs. Performance evaluation measured the time required to gain read or write access to EHRs.

### *5.1. Functional Testing*

The functional testing evaluated the BBEHR prototype against the design requirements and according to the following metrics:

- The system should provide a mechanism for exchanging and synchronizing EHRs between distributed providers by using the blockchain and the cloud store.
- The system should provide patients with a secure mechanism for recovering access to their EHRs.
- The system should ensure unique mapping between patients' identities and their respective EHRs.

The exchange and synchronization of EHRs between distributed providers were achieved by using the Ethereum public blockchain and MS Azure Files. This was validated by accessing the EHRs of the same patient from two BBEHR providers by using the patient's fingerprint (after granting access to each provider). Figure 12 shows the access results from the two BBEHR providers.

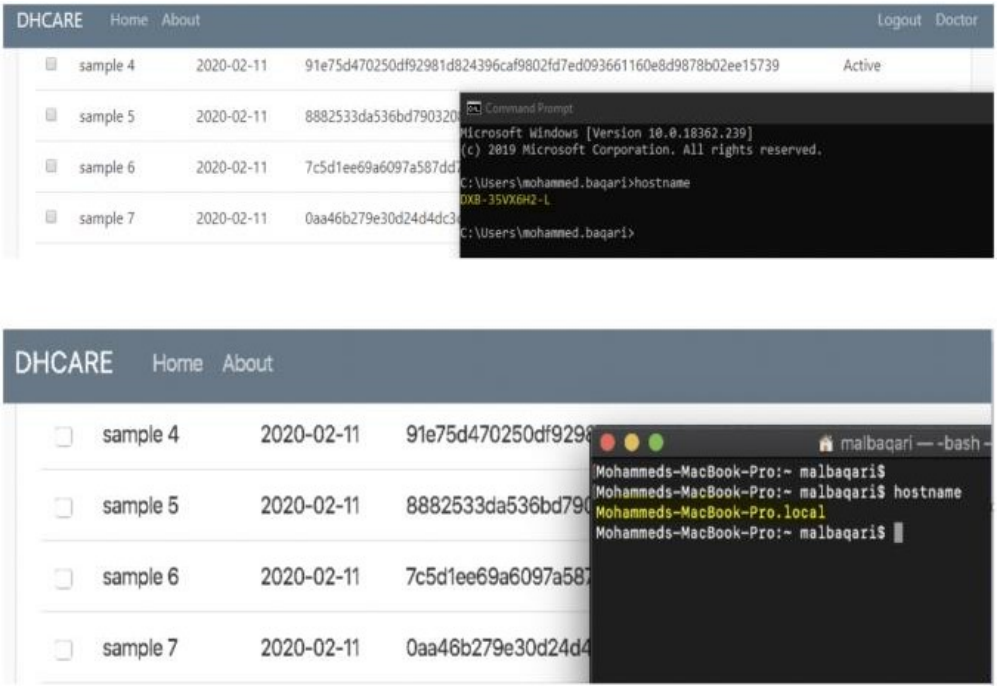


**Figure 12.** Access results from the two BBEHR providers.

To validate the access recovery mechanism for patients' EHRs, fingerprint hashes were used as index keys to retrieve the EHR list from ehrHashTable.

The unique mapping between patients' fingerprints and their EHRs was validated by comparing the Merkle root hash values in the ehrHashTable of the patient against the files' names that were stored in the MS Azure Files.

### *5.2. Security Testing*

The security testing of the BBEHR prototype covered the following aspects:

- The system should provide an access control mechanism to ensure authorized access to EHRs.
- The system should log all read/write activities on EHRs.
- The system should provide anonymity of EHRs in the cloud store.
- The system should validate the integrity of EHRs for read requests.

The access control system in BBEHR was implemented at multiple levels. The first level of authorization was implemented in Django directory services to validate the group membership of the users. This ensured that only authorized users can access their role-specific UIs. The next level of authorization was implemented by using patients' fingerprints to grant doctors read or write access to EHRs. Unless a valid fingerprint is submitted by the patient, clinic doctors cannot access EHRs.

All EHR read/write activities are logged in the Ethereum blockchain for audit trace purposes. The log messages include a unique hash identifier, the Ethereum address of the healthcare provider, a timestamp, and activity details.

The data stored in Azure Files were anonymized by using Merkle hash values as EHR names and suppressing patients' PII. This ensured that patients' identities were not traceable from the EHR raw data. Figure 13 shows an example of the anonymized data stored in Azure from the BBEHR prototype.

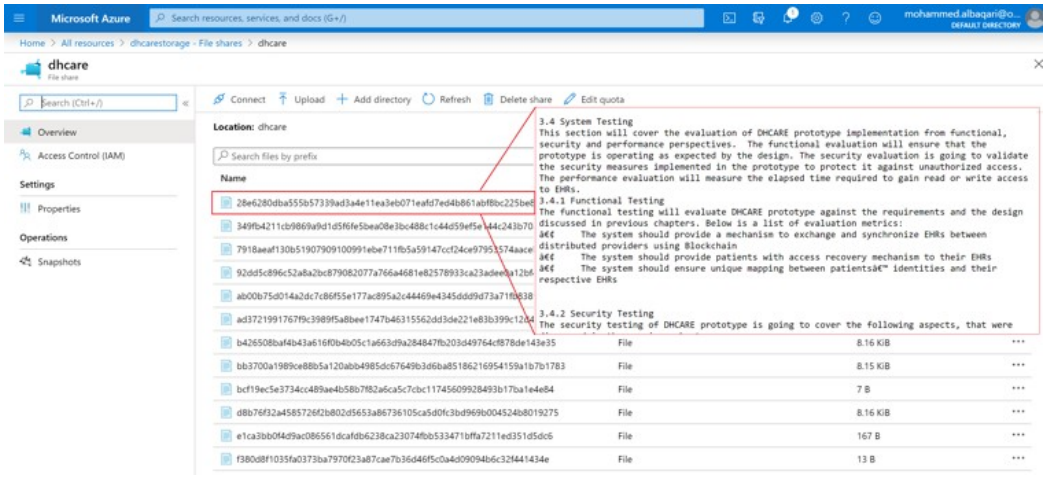


**Figure 13.** Sample of anonymized data stored in Azure Files.

### *5.3. Performance Evaluation*

Among all of the modules in the BBEHR design, the blockchain is considered the slowest component compared to the processing speeds of the other modules. This slowness is caused by the PoW consensus algorithm used in Ethereum. Hence, it was the focus for the performance evaluation.

The time delay introduced by the blockchain layer was evaluated by validating access requests and the granting of access to EHRs. These two components are controlled by the accessControlSC smart contract and patientsSC smart contract.

To isolate the impact of the speed of copying EHRs to the cloud and obtaining accurate performance measures for the blockchain, the EHR test samples used small text files (<20 KB). For the writing test, a test patient was created, and sample EHRs were written into the patient's respective ehrHashTable. For the reading test, each EHR sample stored in the ehrHashTable was read. In total, 15 samples were collected for reading and writing without/with the accessControlSC smart contract.

The time delay between requests and responses was measured by using Google Chrome Developer Tools. Figure 14 shows a sample time delay measurement.

All of the evaluation test cases were executed by using the same internet line to connect to the Ethereum blockchain and Azure Files. Figures 15 and 16 summarize the performance results.

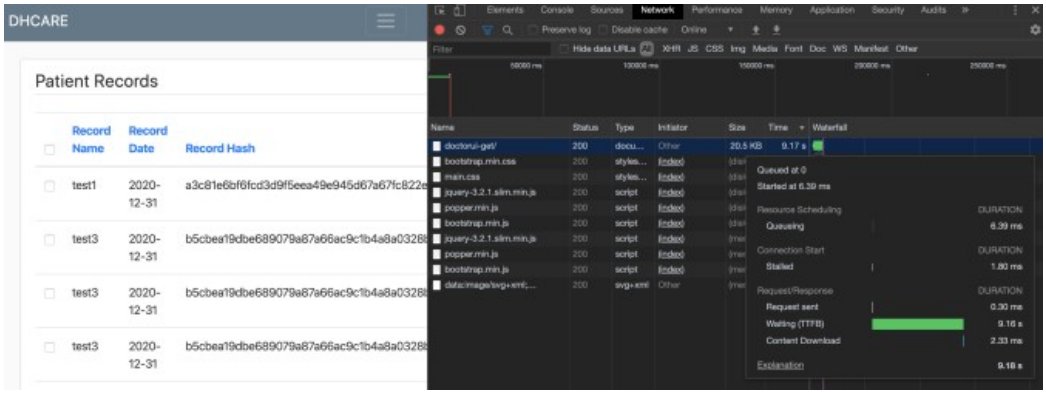


**Figure 14.** Performance measurement using Google Chrome.

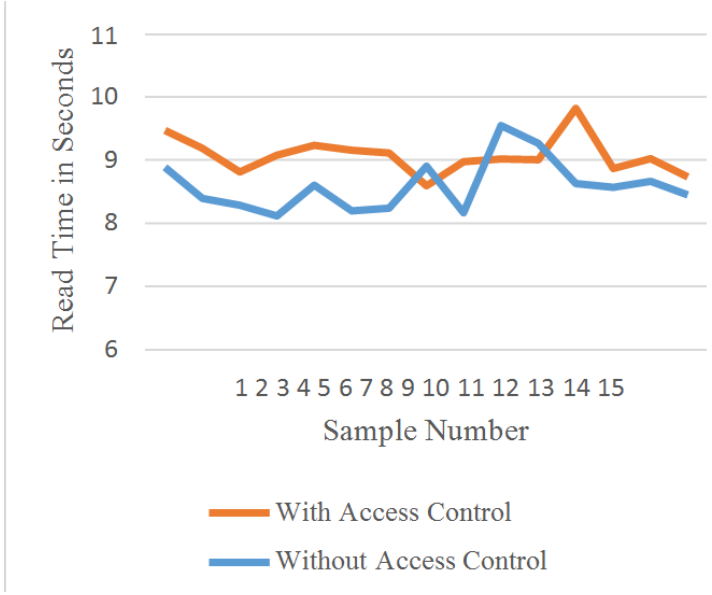


**Figure 15.** Reading performance testing.

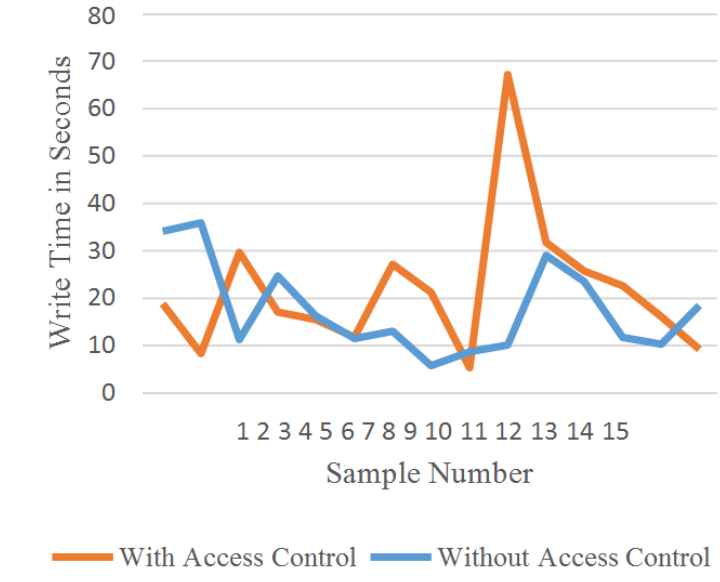


**Figure 16.** Writing performance testing.

The reading performance was superior to the writing performance. The average reading time was 8.81 s, while the average writing time was 16 s, i.e., approximately

twice the reading time. This observation held with and without access control. In terms of today's internet speeds, the reading and writing times are considered to have low performance. However, in the real-life circumstances of most healthcare environments, such time delays are acceptable. The main reason for these delays is the consensus algorithm used by the Ethereum blockchain to validate and accept blocks. Another important factor is the load of the mining pool and its incentive to mine the block.

From the results, it can be concluded that no significant overhead delay is added by implementing access control in the blockchain by using the accessControlSC smart contract. This is because a single block includes thousands of transactions, and transactions from both accessContractSC and paitentsSC are usually mined in a single block (the decision to group the transactions in blocks is subject to the miner). Hence, there is no difference between sending two transactions or one transaction, as they are mined in the same block. In one EHR sample (sample 6 in the writing test with access control), the transactions were mined in two separate blocks. Hence, the time delay for writing the EHR metadata in ehrHashTable was 67.33 s.

Another important observation from Figure 17 is that the writing time was consistent and independent of the number of records in the ehrHashTable, while the reading time was dependent on the number of records in the ehrHashTable. In this test, five more EHR samples were added to the test patient's ehrHashTable (total increased to 20 samples). After re-running the reading evaluation, the average reading time increased from 8.81 to 13.67 s. This was due to additional iterations executed in the code that were needed to list all EHR metadata associated with the patient in the ehrHashTable. Code and DB optimizations would be required in a real-life implementation to improve the reading time.

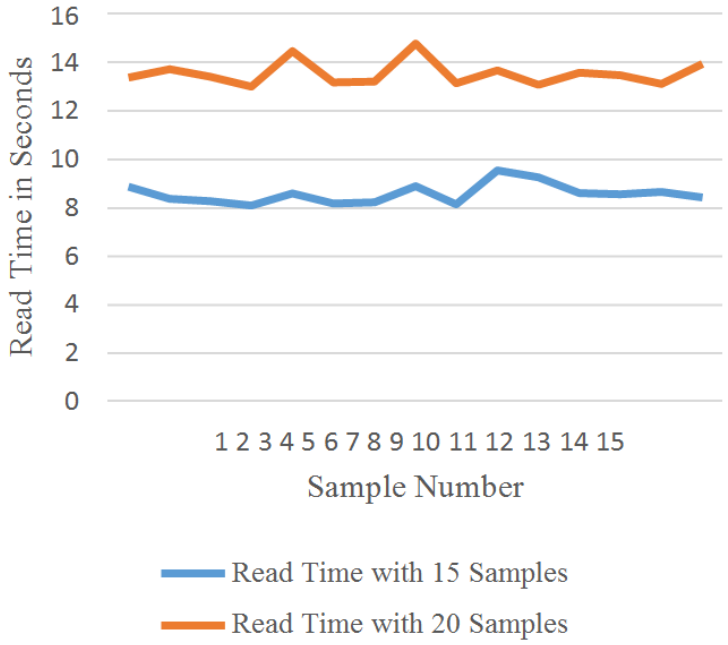


**Figure 17.** Reading time analysis with a larger ehrHashTable.

## 6. Conclusions and Future Work

This research investigated access control recovery mechanisms for EHRs that are synchronized and exchanged among distributed healthcare providers using blockchain. We first reviewed the current state of research on blockchain in healthcare to gain an understanding of the active areas. This was followed by narrowing the focus to research targeting blockchain in EHR systems.

An analysis of current challenges in blockchain-based EHR systems and the requirements for achieving a successful access control recovery mechanism for EHRs was undertaken. Accordingly, we proposed BBEHR, a multilayer system that splits the roles between

healthcare providers, the blockchain, and a cloud store. This model system allows for recovery access for EHRs from any provider within the blockchain network. Additionally, the model may accelerate the migration of healthcare providers to blockchain-based systems through the availability of external UI integration with existing legacy healthcare environments.

A prototype was built to validate the proposed approach by using Django, Python, Ethereum, and MS Azure. The prototype was coded to simulate all functional requirements and integrate the distributed layers of the design. This was followed by system validation and testing for functional requirements, security requirements, and performance. Our results indicated the successful operation of the proposed design from a functional and security perspective. The performance of the prototype was slow due to the functional operation of the Ethereum blockchain; however, this latency may be tolerable in healthcare environments. In addition, the main feature of BBEHR is its independence from the public/private key strategy, which makes it robust against both outside attackers and inside intruders.

For future work, we will attempt to evaluate the BBEHR design against hybrid blockchain ledgers that use faster consensus algorithms. This will aim to enhance the performance of BBEHR for the reading/writing of EHRs while maintaining the extended accessibility to the solution. Additionally, we will upgrade the prototype to include advanced biometrics that combine multiple fingerprints for more accuracy and privacy, and the results should be evaluated against the performance overhead. Another planned enhancement in BBEHR will be the introduction of additional roles in access control smart contracts, including access delegation, access revocation, and record deletion. Finally, we will evaluate the use of mobile-based biometric scanning to extend patients' manageability of access rights for EHRs.

**Author Contributions:** The authors contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** All implementation details, sources, and data will be delivered upon requesting the corresponding author Jorge Herrera-Tapia.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Office of the National Coordinator for Health Information Technology (ONC). Available online: https://www.healthit.gov/faq/what-electronic-health-record-ehr (accessed on 13 November 2020).
2. Nakamoto, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*; Technical Report; Decentralized Business Review; Manubot: 2019. Available online: https://manubot.org/ (accessed on 13 November 2020).
3. Buterin, V.; et al. A next-generation smart contract and decentralized application platform. *White Pap.* **2014**, *3*.
4. Rathee, G.; Kerrache, C.A.; Ferrag, M.A. A Blockchain-Based Intrusion Detection System Using Viterbi Algorithm and Indirect Trust for IIoT Systems. *J. Sens. Actuator Netw.* **2022**, *11*, 71. [CrossRef]
5. Rathee, G.; Ahmad, F.; Sandhu, R.; Kerrache, C.A.; Azad, M.A. On the design and implementation of a secure blockchain-based hybrid framework for Industrial Internet-of-Things. *Inf. Process. Manag.* **2021**, *58*, 102526. [CrossRef]
6. McGhin, T.; Choo, K.K.R.; Liu, C.Z.; He, D. Blockchain in healthcare applications: Research challenges and opportunities. *J. Netw. Comput. Appl.* **2019**, *135*, 62–75. [CrossRef]
7. Barka, E.; Dahmane, S.; Kerrache, C.A.; Khayat, M.; Sallabi, F. STHM: A Secured and Trusted Healthcare Monitoring Architecture Using SDN and Blockchain. *Electronics* **2021**, *10*, 1787. [CrossRef]
8. Al Baqari, M.; Barka, E. Biometric-Based Blockchain EHR System (BBEHR). In Proceedings of the 2020 International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus, 15–19 June 2020; pp. 2228–2234.
9. Ahmad, F.; Ahmad, Z.; Kerrache, C.A.; Kurugollu, F.; Adnane, A.; Barka, E. Blockchain in Internet-of-Things: Architecture, Applications and Research Directions. In Proceedings of the 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 3–4 April 2019; pp. 1–6. [CrossRef]
10. Antwi, M.; Adnane, A.; Ahmad, F.; Hussain, R.; Habib ur Rehman, M.; Kerrache, C.A. The case of HyperLedger Fabric as a blockchain solution for healthcare applications. *Blockchain Res. Appl.* **2021**, *2*, 100012. [CrossRef]

11. Shi, S.; He, D.; Li, L.; Kumar, N.; Khan, M.K.; Choo, K.K.R. Applications of blockchain in ensuring the security and privacy of electronic health record systems: A survey. *Comput. Secur.* **2020**, *97*, 101966. [CrossRef] [PubMed]
12. Shahnaz, A.; Qamar, U.; Khalid, A. Using blockchain for electronic health records. *IEEE Access* **2019**, *7*, 147782–147795. [CrossRef]
13. Tanwar, S.; Parekh, K.; Evans, R. Blockchain-based electronic healthcare record system for healthcare 4.0 applications. *J. Inf. Secur. Appl.* **2020**, *50*, 102407. [CrossRef]
14. Fatima, N.; Agarwal, P.; Sohail, S.S. Security and Privacy Issues of Blockchain Technology in Health Care—A Review. In *ICT Analysis and Applications*; Springer: Singapore, 2022; pp. 193–201.
15. Magyar, G. Blockchain: Solving the privacy and research availability tradeoff for EHR data: A new disruptive technology in health data management. In Proceedings of the 2017 IEEE 30th Neumann Colloquium (NC), Budapest, Hungary, 24–25 November 2017; pp. 000135–000140.
16. Pilkington, M. Can blockchain improve healthcare management? Consumer medical electronics and the IoMT. *Consum. Med. Electron. IoMT J.* **2017**. [CrossRef]
17. Zhang, P.; Walker, M.A.; White, J.; Schmidt, D.C.; Lenz, G. Metrics for assessing blockchain-based healthcare decentralized apps. In Proceedings of the 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom), Dalian, China, 12–15 October 2017; pp. 1–4.
18. Dagher, G.G.; Mohler, J.; Milojkovic, M.; Marella, P.B. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustain. Cities Soc.* **2018**, *39*, 283–297. [CrossRef]
19. 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, 22–24 August 2016; pp. 25–30.
20. Xia, Q.; Sifah, E.B.; Smahi, A.; Amofa, S.; Zhang, X. BBDS: Blockchain-based data sharing for electronic medical records in cloud environments. *Information* **2017**, *8*, 44. [CrossRef]
21. Yang, Y.; Li, X.; Qamar, N.; Liu, P.; Ke, W.; Shen, B.; Liu, Z. Medshare: A novel hybrid cloud for medical resource sharing among autonomous healthcare providers. *IEEE Access* **2018**, *6*, 46949–46961. [CrossRef]
22. Roehrs, A.; da Costa, C.A.; da Rosa Righi, R. OmniPHR: A distributed architecture model to integrate personal health records. *J. Biomed. Inform.* **2017**, *71*, 70–81. [CrossRef] [PubMed]
23. Fan, K.; Wang, S.; Ren, Y.; Li, H.; Yang, Y. Medblock: Efficient and secure medical data sharing via blockchain. *J. Med. Syst.* **2018**, *42*, 136. [CrossRef] [PubMed]
24. Microsoft Azure. What is Azure Files? Available online: https://azure.microsoft.com/en-in/products/storage/files/ (accessed on 31 January 2020).