computers

MDPI

*Article*

# A Blockchain-Based Electronic Health Record (EHR) System for Edge Computing Enhancing Security and Cost Efficiency

**Valerio Mandarino, Giuseppe Pappalardo and Emiliano Tramontana ***

Dipartimento di Matematica e Informatica, University of Catania, 95125 Catania, Italy; valerio.mandarino@phd.unict.it (V.M.); pappalardo@dmi.unict.it (G.P.)
* Correspondence: tramontana@dmi.unict.it; Tel.: +39-095-7383008

**Abstract:** Blockchain technology offers unique features, such as transparency, the immutability of data, and the capacity to establish trust without a central authority. Such characteristics can be leveraged to support the collaboration among several different software systems operating within the healthcare ecosystem, while ensuring data integrity and make electronic health records (EHRs) more easily accessible. To provide a solution based on blockchain technology, this paper has evaluated the main issues that arise when large amounts of data are expected, i.e., mainly cost and performance. A balanced approach that maximizes the benefits and mitigates the constraints of the blockchain has been designed. The proposed decentralized application (dApp) architecture employs a hybrid storage strategy that involves storing medical records locally, on users' devices, while utilizing blockchain to manage an index of these data. The dApp clients facilitate interactions among participants, leveraging a smart contract to enable patients to set authorization policies, thereby ensuring that only designated healthcare providers and authorized entities have access to specific medical records. The blockchain data-immutability property is used to validate data stored externally. This solution significantly reduces the costs related to the utilization of the blockchain, while retaining its advantages, and improves performance, since the majority of data are available off-chain.

**Keywords:** blockchain; healthcare; EHR; smart contracts; design patterns; big data



**Citation:** Mandarino, V.; Pappalardo, G.; Tramontana, E. A Blockchain-Based Electronic Health Record (EHR) System for Edge Computing Enhancing Security and Cost Efficiency. *Computers* **2024**, *13*, 132. https://doi.org/10.3390/computers13060132

Academic Editor: Paolo Bellavista

Received: 20 April 2024
Revised: 20 May 2024
Accepted: 21 May 2024
Published: 24 May 2024



## 1. Introduction

The digital transformation in healthcare systems has been increasing its pace, hence enabling healthcare providers to digitally store and share medical information, helping to build a continuum of care for patients. The adoption of electronic health records (EHRs) has recently spread globally across many countries. EHRs are a digital collection of patients and population health information [1] that can enhance healthcare services by making these data available to professionals and integrate the various stakeholders that patients meet along their continuum of care. Additionally, EHRs can also feed a broader database that can be useful for scientific research and epidemiological considerations.

EHR adoption is assisted by a multitude of technological frameworks employed for storing and accessing such records [2]. The heterogeneity of EHR-based software systems, however, can be an obstacle to the seamless exchange of information among healthcare institutions, professionals, and patients. The effort to unify such different, and often siloed, systems can be complex and articulated. As a result, the majority of digital health systems are unable to interoperate, sometimes even within the same medical facility, and any shortcomings in information exchange can negatively impact the entire healthcare delivery process. Blockchain technology can offer opportunities for improvement, since its inherent characteristics make it suitable for storing and monitoring activities on medical data, and can tackle several existing challenges faced by current healthcare systems [3,4]. Unlike traditional systems, blockchain's decentralized architecture ensures robustness against single points of failure and enhances security against attacks and data loss [4,5]. While

blockchain cannot be used to store all data in EHR records, due to size constraints and storage cost, it can be employed to index them using hashing functions. This capability along with its inherent transparency, allows for the traceable and secure tamper-evident mapping of patient health data. The adoption of blockchain technology can establish a secure layer for certifying the interactions among various actors that perform activities on EHRs, thanks to its robust user authentication mechanisms. This level of security effectively addresses trust issues and can empower patients with full control over their health records; that is, blockchain can enable patients to manage and authorize access to their records, making it unnecessary to resort to a trusted authority. To employ the blockchain in this context, however, some limitations have to be addressed, such as the cost of on-chain storage or issues related to privacy preservation, as data saved in its blocks can be viewed by anyone who has access to the blockchain. According to privacy regulations such as the Health Insurance Portability and Accountability Act (HIPAA) [6] in the United States or the General Data Protection Regulation (GDPR) [7] in Europe, regardless of how they are stored, sensitive information, including health, demographic, and genetic data, is subject to stringent protection requirements.

Addressing these concerns requires innovative solutions that leverage the blockchain's strengths such as decentralization (for security and resilience), interoperability across different systems, integrity of data, and a strong authorization system (to restrict the management of data). Other features need to be provided, such as ensuring adherence to privacy regulations, cost efficiency, ease of access for all relevant parties, scalable storage solutions for large data volumes, and the optimization of performance when using smart contracts. In this paper, we identify the key factors that are desirable for the development of a blockchain-based EHR and propose a novel model, called Edge-enhanced Decentralized Governance and EHRs (EdgeHR), as described in Sections 3, 5 and 6, which leverages the strengths of public blockchains, such as Ethereum [3,4], mitigating its inherent limitations, by integrating smart contract design patterns [8–12] and the edge computing paradigm [13,14]. Detailed discussions on the specific design patterns utilized are provided in Appendix A.

The remaining sections of the paper are as follows. Section 2 gives a synoptic overview of some related works whose main contribution is the use of blockchains in EHR use cases. Section 3 points out the unresolved issues of other blockchain-based EHRs and proposes a solution to address these issues. Section 4 shows two use cases to further illustrate the proposed solution. Section 5 provides a discussion of the main characteristics achieved by our proposed system. Section 6 delves into the EdgeHR smart, also providing various code snippets that illustrate how the main functions operate. Section 7 draws the conclusions. Appendix A describes useful design patterns used in our solution.

## 2. Related Works

Several data management solutions have been proposed for the healthcare sector. Centralized systems, commonly using cloud computing, as detailed in [15–18], offer high throughput and simplicity; however, they rely on a central entity. In contrast, other approaches [19–22] employ blockchain technology, which provides a decentralized solution, freed from a central authority, and gives several potential advantages in the medical domain by leveraging its capacity for providing an incorruptible, decentralized, and transparent record of patient data. Estonia, a pioneer in this domain, started leveraging blockchain's capabilities in healthcare and other sectors in 2012 [23], even before Bitcoin [24] became widely recognized and achieved significant popularity. In this section, we focus on several blockchain-based EHR use cases. Many proposed approaches used the Ethereum network [19–21,25,26], while others adopted Hyperledger Fabric technology [27–33]. Additionally, there are solutions that are compatible with any blockchain [25,34,35].

### 2.1. Approaches Using Ethereum Blockchain for EHRs

UniRec [20] consists of a private peer-to-peer network shared between different healthcare organizations, which was proposed as an EHR solution. In this model, each institution maintains medical data, sharing them through the InterPlanetary File System (IPFS https://ipfs.tech, accessed on 17 April 2024)—a protocol and peer-to-peer network designed for storing and sharing data in a distributed file system. To track the history of each EHR, a reference to data is anchored to a private Ethereum blockchain using an IPFS URI. Each healthcare entity independently manages access permissions for the EHRs they add to the network. Moreover, patients, through a mobile app, can alter access authorizations for their individual records at any given time.

MedRec [19] was designed to integrate with existing EHR infrastructures, addressing four major issues: fragmented and slow access to medical records; system interoperability; patient agency; and enhancement of data quality and quantity for medical research purposes. The proposed system assembled references to medical data, encoding them as hashed pointers stored in a private Ethereum network in order to build an accessible trail for medical history and ensure data integrity without storing raw medical data in the blockchain.

A cloud-assisted EHR storage and sharing system, which provides privacy preservation and data security, was based on consortium blockchain [26]. Patients' data are stored in the cloud, while a consortium blockchain is used to monitor keywords essential for data searching and sharing. To guarantee efficiency, reliability, and security, the authors designed an appropriate data structure and consensus mechanism.

Ancile [21], a blockchain-based framework, was designed to meet the requirements of HIPAA, focusing on safeguarding patients' privacy. The medical data are stored in databases of healthcare providers, while the blockchain is used to facilitate an interoperable exchange of medical records among patients, healthcare providers, and authorized entities, leveraging smart contracts.

### 2.2. Approaches Using Hyperledger Blockchain for EHRs

ACTION-EHR was proposed as a patient-centric system, based on the Hyperledger blockchain, for EHR data sharing and integration, with a specific focus on cancer care [29]. In such a system, hospitals provide the nodes, while patients and doctors engage in direct interactions through a web application. Metadata related to data sharing are stored on-chain, while the actual medical data are encrypted and stored off-chain using cloud services.

MedBlock [27] was introduced as a system designed to assist patients in managing their profiles, enabling them to keep all their medical records with minimal complexity. Each patient's record includes a reference to the address of the corresponding blockchain's blocks where their data are stored. Medical records are stored in cloud or hospital databases. National hospitals are tasked with managing all node roles within the blockchain network, whereas community hospitals serve as endorsers or optional orderers. MedBlock employs a breadcrumb mechanism to enhance the efficiency of record searching within the blockchain.

A medical blockchain platform, in which personal health data are stored in databases and used alongside each blockchain peer, while encrypted metadata are kept on-chain, was proposed [28]. Subnetworks are used to enable private and confidential communication among specific departments, allowing them to establish their own private networks within the broader network. Users are registered through a trusted user manager, which oversees cryptographic mechanisms and provides certificate services that include user enrollment, identity validation, signature generation and verification, and securing connections between users or components of the blockchain. Furthermore, an independent off-chain data repository (data lake) is employed for a variety of analyses.

Another system, adhering to the HIPAA technical safeguard, was designed to scale from single-peer hospitals to client-only small clinics [30]. Multiple hospitals gathered to form a consortium, creating a private peer-to-peer network and adopting a Hyperledger Fabric blockchain to store metadata about health records stored off-chain.

The Medicalchain model was proposed as a blockchain-based system that leverages Hyperledger Fabric, designed to manage EHRs [31]. The authors proposed solutions for the specific context of the Italian National Health System and the adoption of centralized governance and control to enhance regulatory compliance and operational efficiency.

A protocol for secure cloud-assisted EHRs using blockchain technology was proposed to enhance the security and efficiency of traditional EHR systems, and employing Hyperledger Fabric to ensure data integrity and access control, while leveraging cloud computing to provide scalable and secure data storage [32]. Issues related to security breaches such as data leakage or counterfeiting during the storage and sharing process of EHR data were addressed and aimed to prevent common cybersecurity threats such as man-in-the-middle and replay attacks. The protocol's security was analyzed through various means, including an AVISPA simulation (Automated Validation of Internet Security Protocols and Applications, https://www.avispa-project.org/, accessed on 17 April 2024).

HealthBlock was introduced as a modular framework for collaborative sharing of EHRs that uses Hyperledger Indy to provide a self-sovereign identity allowing patients full control over their EHRs; it uses Hyperledger Fabric to manage access control and IPFS to store and distribute patient EHRs [33].

### *2.3. Approaches Using Other Blockchains for EHRs*

BlocHIE [25] was proposed as a blockchain-based platform designed to facilitate the sharing of healthcare data between medical institutions and patients. Collected data are stored off-chain in external distributed databases belonging to hospitals and using two distinct loosely-coupled blockchains for on-chain verification purposes. Such blockchains handle different kinds of healthcare data (institutional medical data and personal health data from users' devices like smartwatches and thermometers). Additionally, two fairness-based transaction packing algorithms were proposed to manage how transactions are grouped and processed with the goal of improving fairness for users and throughput.

FHIRChain [34] was presented to fulfill technical requirements defined in the 'Shared Nationwide Interoperability Roadmap' by the US Office of the National Coordinator for Health Information Technology (ONC [36]). Its design is independent of any specific blockchain framework, but the authors developed the dApp leveraging a private testnet of the Ethereum blockchain and three Solidity smart contracts. The architecture integrated HL7's Fast Healthcare Interoperability Resources (FHIR) standards to facilitate data interchange among disparate healthcare systems.

EHRChain was proposed to leverage two Hyperledger Sawtooth blockchains (https://www.hyperledger.org/hyperledger-sawtooth-1-0, accessed on 17 April 2024) with IPFS to address some EHR challenges identified [35].

### *2.4. Limitations of Approaches Using Blockchains for EHRs*

Integrating blockchain technology into EHR systems is not free from challenges and requires a range of desirable features, that are critical for EHR efficacy and widespread adoption. We have analyzed the various proposals according to the following features.

Table 1 shows a summary of the comparison among several approaches. Interoperability was omitted because it is not influenced by the way blockchain is used; instead, it primarily depends on the EHR data format.

**Table 1.** Comparison of key features in selected EHR solutions.

| EHR Solution | Blockchain Type | Patient Control | Privacy Support | On-Chain Data | Off-Chain Data |
|---|---|---|---|---|---|
| MedRec [19] | private | full | AAA server + PKI authentication | hashed data | EHR centralized DBs |
| Ancile [21] | consortium | full | ACL smart contracts + proxy re-encryption | management data | EHR DBs |
| UniRec [20] | private | full | ACL policy + PGP EHR encryption | hashed data | IPFS-based EHRs |

**Table 1.** *Cont.*

| EHR Solution | Blockchain Type | Patient Control | Privacy Support | On-Chain Data | Off-Chain Data |
|---|---|---|---|---|---|
| BlocHIE [25] | public | full + IoT | Digital signatures + hash functions | EHR verification | EHR DBs |
| Wang et al. [26] | consortium | full | Proxy re-encryption + EHR encryption | keywords | EHR cloud DBs |
| MedBlock [27] | consortium | partial | CA + ACL protocol + PKI EHR encryption | EHR summaries | EHR DBs |
| Hang et al. [28] | consortium | full | Subnetworks + trusted user manager | encrypted metadata | EHR DBs data lake |
| ACTION-EHR [29] | consortium | full | 3rd party CA. | management data | EHR clustered DB |
| Tith et al. [30] | consortium | full | Membership auth + proxy re-encryption | management data | EHR DBs |
| Medicalchain [31] | consortium | full | ACL smart contracts | management + hashed data | EHR DBs |
| Kim et al. [32] | consortium | full | Network administrator + pseudo-identities | log transactions | EHR cloud DBs |
| HealthBlock [33] | private + consortium | full | Trust Anchor + Decentralized Identifiers + ACL + EHR encryption | management + hashed data | IPFS-based EHR |
| FHIRChain [34] | consortium | providers only | PKI DIDs + smart tokens + ciphered EHR | encrypted metadata | EHR DBs |
| EHRChain [35] | public/consortium | full | Fine-grained AC + EHR encryption | management data metadata | IPFS-based EHR |

- Decentralization: all the analyzed approaches preferred private or permissioned blockchains (with the exceptions of BlocHIE [25] and EHRChain [35]) rather than a public blockchain like Ethereum. Hyperledger Fabric is designed as a permissioned system by default, whereas Ethereum can operate as a private or consortium network, thus limiting network access by means of trusted validators, enhancing privacy, and adapting to the specific desired governance and confidentiality requirements. Private or Consortium blockchains prevent the indiscriminate exposure of sensitive data and mitigate scalability issues related to network congestion and the computational costs of executing smart contracts, which typically result in high transaction fees. However, this choice compromises the benefits of decentralization and the permanent availability of data, and leads to reliance on specific validators that control the whole system.
  BlocHIE does not indicate any specific platform, and the authors assert that the blockchain is not closed to the public. However, potential issues related to the use of a public network, such as congestion, the possibility of unprocessed transactions, or high fee costs, have not been addressed.
  EHRChain employs Hyperledger Sawtooth, which allows for both public and private blockchain configurations. Nonetheless, Hyperledger Sawtooth was moved to end-of-life status at the request of the maintainers on 1 February 2024.
- Accessibility: the EHR system should promote ease of access for all relevant participants, including patients, healthcare providers, and other authorized entities. Additionally, it is crucial that patients have control over their health data and are able to set rules and restrictions on who can access them. In most of the examined EHR use cases [19–21,26,27,29–32], user accessibility is well-addressed, as these systems provide access through various devices or a web portal. The majority of the EHR systems examined in this work are patient-centric. In order to contribute to a patient's EHR, access to eligible actors is granted or denied using private keys, tokens, and smart contracts. However, some solutions often involve a higher-level trusted authority, such as healthcare organizations [20] or entities managing network identities [27,29–31,33], to facilitate patient registration, access, or identity verification.
- Privacy: the EHR system should ensure that stored data visible to everyone are anonymized and only authorized users can add medical records, maintaining a high level of privacy and security. All the reviewed EHR systems adhere to privacy standards, and both on-chain and off-chain data are encrypted. In all examined cases, only the data digest is stored on-chain, ensuring the authenticity and integrity of external data.
- Cost-effective storage solutions: given the high volume and arbitrary size of the data involved, the EHR system must manage storage and its associated costs without com-

promising performance. Since the common solution is to store medical data off-chain, retaining on-chain only their constant-sized hash and URIs that point to the external storage locations, storage costs and blockchain overhead were reduced. Ancile [21] and MedRec [19] integrate their EHRs with existing databases; MedBlock [27], the systems in [28,31], and BlocHIE [25] utilize database servers maintained by institutions or hospitals, or deploy databases on nodes; [26], ACTION-EHR [29] and [32] leverage cloud services; and UniRec [20], HealthBlock [33], and EHRChain [35] utilize IPFS. These solutions often subject users to reliance on centralized services and introduce challenges related to latency, when the EHR systems access off-chain external data.

- Interoperability: the ability to seamlessly exchange data across various healthcare systems is crucial. Moreover, adherence to international standards and protocols is necessary to facilitate smooth interoperability. From this perspective, Ancile [21], FHIRChain [34], and omniPHR [37] stand out as the most mature technologies since they were designed to align with international standards for structured data formats and clinical coding.

## 3. Proposed Approach: EdgeHR

Our proposed approach for a blockchain-based EHR system leverages a public blockchain such as Ethereum. Our solution mitigates its inherent limitations by implementing the strategies described in this section and in Section 5, while coping with the shortcomings of other approaches reported in Section 2.

Figure 1 shows the key interactions among the proposed solution's components. The client of the decentralized blockchain-based application (dApp) initiates interactions according to the user's role and operates in conjunction with storage, some devices that are accessible locally, and the Edge smart contract that runs on the blockchain.

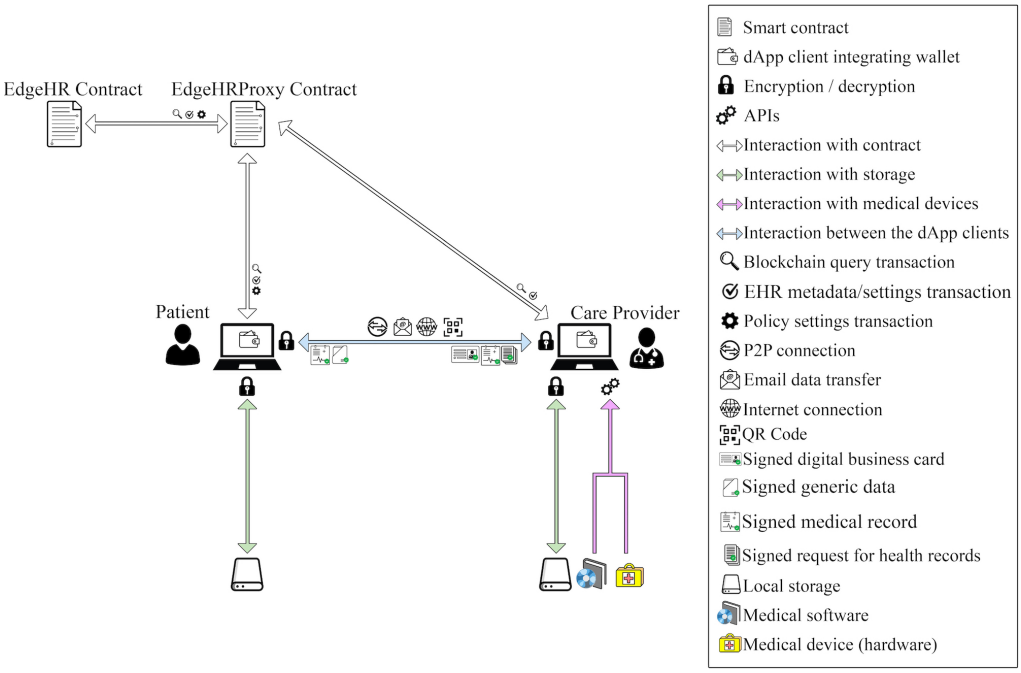


**Figure 1.** Interactions between the dApp clients, the smart contract, the storage, and medical devices.

The clients are executed on devices, such as personal computers, laptops, smartphones, or tablets with storage capabilities, that can connect to a LAN network and have access to the Internet. Clients incorporate cryptographic libraries, notably secp256k1 [38], to implement functionalities like the Elliptic Curve Digital Signature Algorithm (ECDSA) used by Ethereum. Clients can also access the blockchain and execute the wallet functionalities through the use of web3.js, a JavaScript library that provides a set of APIs for interacting with a local or remote Ethereum node. This enables seamless integration of blockchain

functionalities into the user interface, such as sending transactions, interacting with smart contracts, and managing accounts. As shown in Figure 1, the exchange of information between the patient and care providers, both acting as dApp clients, occurs through various communication methods such as a P2P connection on the same LAN, email, or other online channels, depending on the situation, as detailed below.

The type of interaction between these actors can be described by means of three phases: (i) first contact of patient and care provider; (ii) exchange of medical data and updating of metadata records; (iii) medical data consultation. Such phases are detailed in the following subsections.

### *3.1. Phase 1: Patient and Care Provider First Contact*

Typically, a patient may request multiple care providers to issue a specialist report on the state of their health. Therefore, the patient's EHR has to be accessed by several care providers at different times. To safeguard this process, access to the EHR has to be properly authorized by the patient. To ensure this, proper mechanisms are put into place in our solution. When a user (patient or care provider) joins the system, they log into the dApp client, which allows them to either create a new wallet address or import an existing one. The client's response varies depending on the user type. Specifically, for patients accessing the service for the first time, the client triggers the deployment of two distinct smart contracts, the EdgeHR and the EdgeHRProxy smart contracts, ensuring that every patient has their unique personal instances. Figure 2 shows this process as it occurs over time. EdgeHR serves as a patient-dedicated repository for the EHR index and metadata, while EdgeHRProxy is an intermediary smart contract that stores the address of the EdgeHR instance, as detailed in in Section 6. This mechanism implements the Proxy design pattern (see Appendix A), a solution for making the logic of smart contracts "upgradable" (i.e., masking the challenges associated with the immutability of smart contracts.

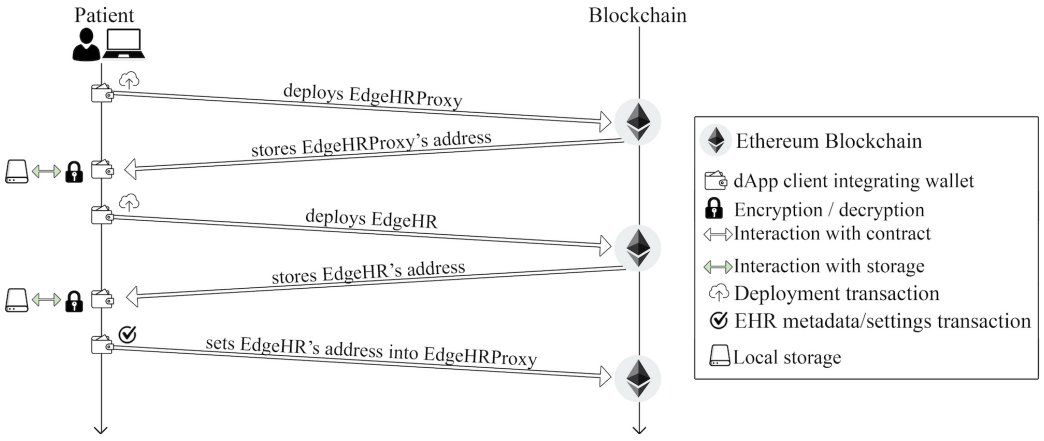


**Figure 2.** Deployment and configuration of the patient's smart contracts.

By using the Owner design pattern (see Appendix A), the patient becomes the owner of both the smart contracts, meaning that only the patient's wallet address can trigger specific functions. For subsequent logins, the deployed smart contracts will already be available.

When care providers interact with a patient for the first time, the dApp client generates a message $M$, which acts as a digital "business card" that is used to initiate secure communication with the patient's client and to facilitate authorization policies management. This message includes the care provider's wallet address and the related public key $P$, in addition to other pertinent information such as the professional's name, and contacts or clinic address. The authenticity of this message is ensured by its digital signature. The initial interaction between medical professionals and patients involves performing a "handshake", which can be executed via a direct P2P connection within the same LAN, using a QR code, or remotely via email. The care provider's dApp client sends the message $M$ to the patient's dApp client. Upon verifying the message's authenticity and ensuring

that it originates from the intended care provider, the patient's dApp client authorizes the healthcare provider's address. This authorization is carried out through the EdgeHR smart contract's authorizeCareProvider function, which grants access to the patient EHR, setting permissions and their duration. In response to concluding the handshake, the patient's dApp client sends back specific data, including the patient's wallet address, the address of the EdgeHRProxy smart contract instance owned by the patient, and the existing textual descriptions of the types of records, already mapped in the dApp client, such as visits, exams, analyses, treatments, and medical histories.

### 3.2. Phase 2: Exchange of Medical Data and EHRs Update

As a result of a medical visit or examination, the care provider generates data meant to update the patient's EHR record. These data can be any type of media (text files, formatted documents, images, videos, 3D models, etc.). At this point, several possible scenarios arise.

1. Digital data only: the provider only generates digital records.
2. Physical media or hard copy documents: the provider produces data in some physical formats.
3. Hybrid documentation: the provider creates a combination of both digital records and physical media or hard copy documents.

In the first case, which represents the best-case scenario, the provider supplies such data to their dApp client, which, in turn, formats them into a file $f$ and signs them. Subsequently, the care provider's client computes the hash of $f$, encrypts $f$ using the private key of the care provider, and stores the encrypted file (denoted as) $f'$ locally. If its size is smaller than a threshold $t$ (determined by the patient's dApp client, based on the device type), the provider transfers $f'$ to the patient's client, via a P2P connection. If the file size exceeds the threshold $t$, or if the care provider and the patient are not on the same LAN, and thus cannot connect their devices running the dApp clients, the care provider will resort to alternative methods such as digital media, email, online services, or the transfer capabilities of the software they use for patient services, to deliver $f'$ to the patient. Independently of how it is received, the patient's client decrypts the file with the care provider's public key and saves it locally by encrypting it with their own private key.

Figure 1 shows the interaction between the care provider and the patient's smart contract, which is performed by means of the patient's proxy smart contract. The care provider can update the indexing of the patient's EHR or, alternatively, could opt to transfer to the patient a signed gasless transaction (following the EIP-2771 protocol https://eips.ethereum.org/EIPS/eip-2771, accessed on 17 April 2024), which invokes the functions that update the smart contract's EHR indexing. The EIP-2771 standard allows a third party (or relayer), in this case the patient, to submit the transaction to the blockchain on behalf of the original sender, in this case the care provider. This process retains the transaction's integrity, as the original signature from the provider is preserved, while the gas fee is handled separately, ensuring that the original sender does not incur gas costs since they are covered by the relayer. This approach can be particularly useful during network congestion, when gas fees might be excessively high. Considering the successful secure storage of data within both dApp clients, it is strategically viable to defer updating the EHR indexing within the smart contract.

An additional alternative involves allowing the patient to directly update the EHR indexing in the smart contract. Using functions exclusively accessible to the owner, the patient can update the smart contract's state with the records produced by the care provider, including the provider's wallet address. In case the provider produces physical media or hard copy documents, namely the worst-case scenario, it will be the user's responsibility to digitize and store them through the dApp client, which, subsequently, can create and send a transaction carrying the file's hash along with other required information to update the EHR. In cases where the provider produces hybrid documentation, the same principles from the previously mentioned points can be applied.

The dApp clients offer the option to transfer or copy the medical records to additional local storage solutions, such as external hard drives used for backup purposes, to enhance data security. While the care provider might delete these data on their end, each patient is responsible for maintaining their EHRs. From the patients' perspective, maintaining EHRs is no more complex than managing traditional paper-based data, which are, however, prone to physical deterioration or loss. This approach empowers patients with control over their own personal health data, but it places the responsibility for their maintenance on them.

### *3.3. Phase 3: Medical Data Consultation*

To consult a patient's medical records, a care provider, who has previously engaged with the patient, can utilize their dApp client to retrieve them (when already present in their local storage) with no need to interact with the smart contract. However, in situations where the medical records are inaccessible, such as when data are not present in the local storage of the care provider's dApp, or particularly when the patient engages with a new medical facility, with which they have had no prior interaction that requires the examination of their records, (new) care providers can conduct a preliminary search. The smart contract's function getSummary (see Section 6) enables authorized entities to download the record summaries from the EHR's on-chain data. This process assumes that the patient has previously granted the necessary access permissions to the (new) care provider in the smart contract, after having completed the "handshake" (see Section 3.1), exchanging their initial digital "business card" message and the Proxy smart contract address along with the kind descriptions. Subsequently, (new) care providers can identify and select relevant data types based on their search criteria. Upon reviewing this summary, whether planning to meet the patient in person or to provide remote consultations, they can generate and sign a specific request, RecordAccessRequest, for detailed records through their dApp client. This request, including the progressive numbers of the required records and the corresponding data hashes, is sent to the patient via email or instant messaging services.

Upon receiving and opening this message, the patient's dApp client first verifies its authenticity. Following successful verification, the application decrypts the requested records and re-encrypts them using the care provider's public key. Finally, these re-encrypted records are sent back to the requester, as an email attachment, through a P2P connection if the patient and the care provider are on the same LAN, or via alternative communication channels.

## 4. Use Cases Involving the Patient

This section illustrates two use cases. In the first one, a patient, Bob, engages with the dApp for the first time and authorizes his care provider, Alice, to update his health records. Subsequently, due to some symptoms, Bob goes to Alice in person for a medical consultation. The second use case presents a scenario where Bob, already using the EdgeHR dApp, consults with a medical specialist, Charlie. This case explores the remote interaction and data management activities facilitated by the system.

### *4.1. First Interaction and Visit*

Throughout the use case described in this section, it is assumed that all transactions are performed at a given average gas price. Network conditions at the time of writing result in an average gas price of 5 Gwei (1 wei = $1 \times 10^{-18}$ ETH), as estimated by Etherscan (https://etherscan.io/gastracker, accessed on 9 May 2024), with an expected transaction execution time of approximately 30 s. The ETH exchange rate during the same period is reported to be $2983.35 by Coingecko (https://www.coingecko.com/en/coins/ethereum, accessed on 9 May 2024).

When Bob accesses the system for the first time, the dApp client triggers the deployment of the EdgeHRProxy smart contract that will be used by the care providers to interact with his EdgeHR smart contract instance. Deploying the Proxy smart contract consumes 1,244,072 gas units. According to the economic conditions assumed, this leads to a total deployment cost of approximately USD 18.56. Once the EdgeHRProxy has been success-

fully deployed, as shown in Figure 2, the dApp client deploys the EdgeHR smart contract and stores its address in the EdgeHRProxy by invoking the function setServiceAddress. The gas consumption for deploying the EdgeHR smart contract amounts to 3,082,230 units, resulting in a final cost of approximately USD 23.86.

During the medical consultation, the digital handshake is initiated between Bob's and Alice's clients via a P2P connection, as detailed in the Phase 1 (see Section 3.1). Alice's client generates the digital business card, which includes her wallet address and public key and sends it to Bob's client. Upon receipt, Bob's client replies with the necessary data needed by Alice's dApp client, such as Bob's wallet address, the address of his EdgeHRProxy smart contract instance, and the descriptions of existing health record types. It then invokes the authorizeCareProvider function (see Section 6) of the EdgeHR smart contract to authorize Alice to access and update his health records. The units of gas consumed for this operation are 10,590 which results in a transaction cost of USD 0.16.

When Alice examines Bob, she creates a medical record detailing his symptoms and the prescribed treatment. Her dApp client then encrypts this record, stores it locally, and sends it to Bob's client using the LAN connection. Subsequently, Alice's client interacts with the EdgeHR smart contract (through the EdgeHRProxy smart contract), invoking the addMetaData function to record a hash of these data, along with other relevant information (detailed in Section 6), ensuring the integrity and non-repudiation of the medical record. In addition, addMetaData maps in the smart contract, through the manageKindDescription function, the record type (kind), paired with the description "general visit". The total gas consumed amounts to 171,033, resulting in a final fee for the function execution of USD 2.55. If events are not emitted and the kind already exists, this function consumes a minimum of 92,856 gas, resulting in a cost of USD 1.38.

Upon receiving the medical record, Bob's client decrypts it using Alice's public key and re-encrypts it using his private key and then stores the record locally. While Bob's personal health information remains stored in both the storage devices of Bob and Alice, the secure and immutable logging of the hash on the blockchain via the EdgeHR smart contract provides a verifiable and tamper-evident record of the medical event.

Figure 3 shows a detailed visual representation of the interactions between Bob and Alice as they occur over time. It illustrates both Phase 1 and Phase 2 of the process described above.

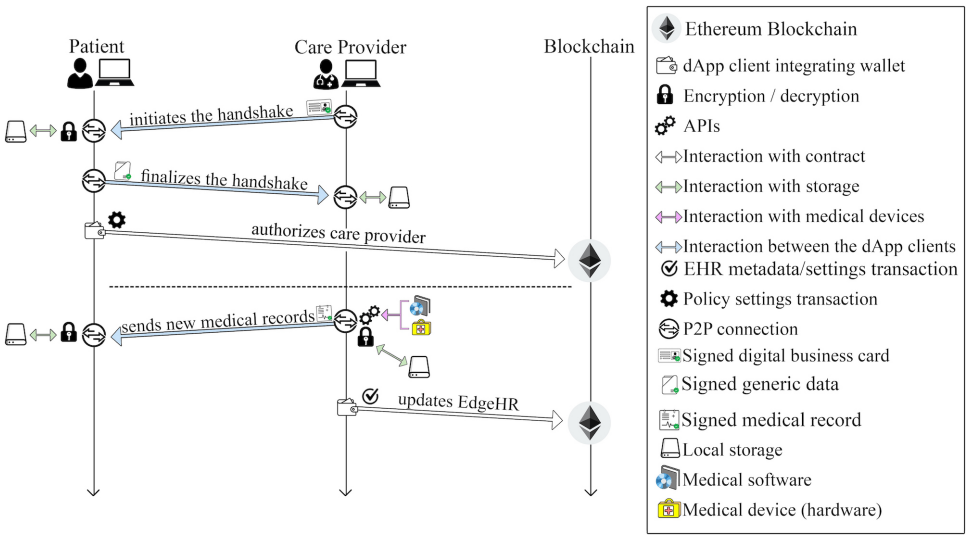


**Figure 3.** Interactions between a patient (Bob) and a care provider (Alice) when a first contact is established and new EHRs are generated.

### *4.2. Further Interactions between a Patient and a Specialist Care Provider*

Bob asks Charlie, a specialized physician, to schedule an appointment for a specialist consultation. However, Charlie needs to review some data from Bob's EHR. Initially, he

initiates a remote handshake by sending his digital business card to Bob via email. Upon receiving Charlie's message, Bob sends back the dApp client's response that includes his wallet address, the address of his EdgeHRProxy smart contract instance, and the description of the types of the existing EHRs. Bob then authorizes Charlie to access his records by invoking the authorizeCareProvider function of the EdgeHR smart contract. Charlie is now able to invoke the getSummary function of the EdgeHR smart contract to retrieve an overview of Bob's medical history, which includes the notes and descriptions, in the EHRs. This function, marked as `view`, keeps the state of the contract unchanged, and thus no gas is consumed. Then, Charlie identifies specific records he requires for a more detailed assessment and requests these records via email by sending a RecordAccessRequest. Upon receipt, Bob's client retrieves and decrypts the relevant records from local storage, re-encrypts them using Charlie's public key, and then sends them via email. Charlie receives and reviews the medical records and determines that Bob needs more specific medical tests before proceeding with the specialist visit. This decision generates a new medical record that documents these tests and the motivation.

Charlie encrypts this record using his private key and sends it to Bob's client via email. Bob's client receives the new record, decrypts it, re-encrypts it, and integrates it into his local EHR copy. Due to an unexpected internet outage, Charlie is unable to perform the transaction that would index this new record on-chain in Bob's EdgeHR smart contract. Bob's dApp client expects to find the processed care provider's transaction within a given timeout period (that the patient can set). Consequently, after a few days (when the waiting period is over), Bob's dApp client independently performs a transaction invoking the addMetaData function, sending to the smart contract the data related to the record previously created by Charlie and transmitted via email, thereby ensuring the new record is permanently recorded on the blockchain. When Charlie's internet connection has been restored, his dApp client, which had previously prepared the transaction but was unable to send it to the network, is finally able to broadcast it to the network. However, since Bob has already created a record with the proposed hash, the addMetaData function in the EdgeHR smart contract rejects the transaction, preventing redundant entries.

Figure 4 provides a detailed representation of the interactions between Bob and Charlie as they occur over time. It illustrates the process occurring via email, as described above.

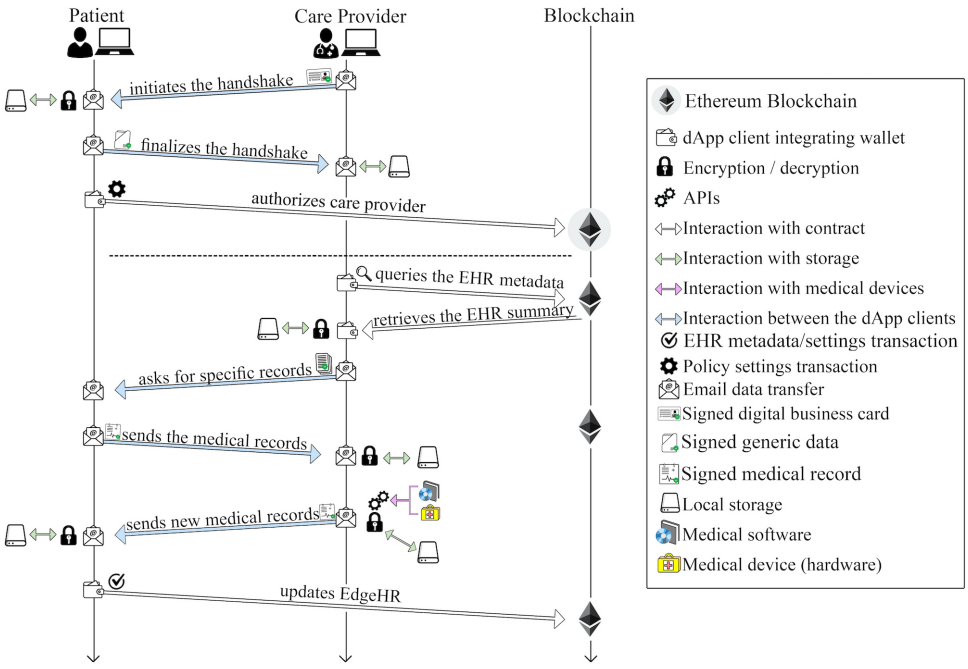


**Figure 4.** Interactions between a patient (Bob) and a specialized care provider (Charlie) accessing some existing EHRs and generating new EHRs.

## 5. Discussion

The proposed system aims to achieve several essential key features, for an effective EHR system, including decentralization, accessibility, enhanced security, patient control, enhanced privacy, interoperability, storage and cost efficiency, and optimized performance, which will be discussed in the following paragraphs.

### 5.1. Decentralization

True decentralization, not only from an architectural standpoint but also in terms of independence from one or more trusted entities, is fully realized only through a public blockchain. To meet the requirement for decentralization, we have concluded that Ethereum is the optimal solution, as it operates independently of any specific entity and allows any user to operate a node. Although becoming a validator in Ethereum requires staking 32 ETH [39], users have the option to delegate their stake and participate in a staking pool [40]. Despite this limitation, the number of nodes in a public blockchain like Ethereum is significantly higher compared to a consortium blockchain. According to Etherscan (https://beaconscan.com/validators#active accessed on 17 April 2024), there were a total of 977,948 active validators at the time of writing. This extensive network of nodes contributes to greater decentralization, enhancing security and reducing dependency on trusted entities. This aspect is particularly important in the context of healthcare data, where security and reliability are paramount. Adopting a public blockchain for EHRs offers real patient sovereignty, reduces risks associated with centralized control, and improves network integrity and auditability. However, the transition to a public blockchain, such as Ethereum, is not straightforward, as it necessitates addressing its inherent limitations, especially (i) the difficulty in ensuring all three aspects of scalability, security, and decentralization [41–44], (ii) the costs associated with transactions, and (iii) the potential bugs in the code. In the following subsections, we detail our solution and how it addresses these issues.

### 5.2. Accessibility and Enhanced Security

Accessibility and authentication security are crucial to ensuring that all stakeholders can interact with the system both easily and securely. Blockchain technology inherently provides robust authentication solutions that do not require any additional trusted authorities. This technology employs native account management, which is implemented using strong cryptographic mechanisms and relies on the consensus algorithms utilized by the validator nodes. Furthermore, the economic cost associated with transactions' execution acts as a deterrent against potential attacks, thereby adding an additional layer of security and preserving the integrity of the system. Regarding the implementation of wallet software, we believe that different versions are functionally equivalent. Wallet applications authenticate users and authorize transactions using private keys, from which the wallet addresses are derived. Typically, for any given blockchain, there are multiple wallet implementations available, including hardware wallet versions that enhance security since they are less susceptible to online attacks and malware. Hardware wallets store private keys within a physical device, making them more resilient to unauthorized access. They are capable of signing transactions without the need for a direct connection to a PC or other devices. Once signed, the transactions can then be transferred to the computer or device via USB or Bluetooth to be sent to the blockchain network.

In conclusion, the fundamental pillar of user access security in blockchain-based applications lies in the robust cryptography employed by the blockchain technology itself. The client of our dApp leverages this cryptography and utilizes the Web3 library through APIs that harness the native wallet functionalities of Ethereum. This ensures that the previously described security features are fully integrated into the dApp client.

### 5.3. Patient Control

The employment of smart contracts enables a dynamic and secure mechanism where users can directly control who has access to their health records and under what conditions,

ensuring that each patient is placed at the center of their healthcare data management. Even in cases where data are directly acquired by users, specifically from hard copies provided by care providers, EdgeHR still offers a solution allowing for the self-entry of data by patients, enhancing its adaptability to a wider range of data entry scenarios, giving patients unprecedented control over their personal medical information. This approach is unlike other previous proposals, which limit updates to care providers only, and their solutions are then possible when care providers are fully equipped to use the proposed blockchain-based EHR systems. Furthermore, interactions between patients and doctors are initiated directly by the stakeholders themselves without the need for prior authorization or management by a third party, reinforcing the autonomy and immediacy of patient care. Lastly, by storing health data directly on users' own devices, data remain genuinely in the hands of the stakeholders, avoiding reliance on centralized storage solutions and reducing vulnerability to external access or control

### *5.4. Enhanced Privacy*

One of the primary challenges when adopting the blockchain in an EHR system is related to privacy. Due to the transparent nature of public blockchains, all transactions are visible to anyone. This degree of openness, while beneficial for transparency, raises significant concerns when dealing with sensitive data, where confidentiality is paramount. This issue extends to compliance with regulatory standards, since the immutable nature of the blockchain may conflict with the requirements for data to be alterable or deletable. In our EHR model, to ensure that data are not constantly exposed to online vulnerabilities, we have implemented the paradigm of edge computing: data are stored off-chain, locally on the users's own devices, while only their hashes are saved on the blockchain.

Although these data could potentially be subject to modification, their integrity is ensured by the use of cryptographic hash functions, which generate the unique digital fingerprints stored on-chain and are reinforced by the care provider's digital signature. This approach also guarantees non-repudiation, ensuring that the originator cannot deny their authorship. Additionally, stored data are encrypted, ensuring that, in the event of unauthorized access, they remain indecipherable. This method adds an essential layer of accountability and trust to the system by ensuring that every transaction and data exchange is verifiable. It enables secure storage outside the blockchain and maintains a highly available indexing system on the blockchain for easy retrieval and verification, thereby effectively safeguarding health information and adhering to privacy standards.

### *5.5. Interoperability*

Interoperability is crucial for EHR systems. Although it depends on the data format rather than the infrastructure used to save these data, the edge computing paradigm allows for greater interoperability as the locally stored data can be re-written and modified according to various standards, ontologies, and methodologies that define clinical data models in a standardized and reusable way. This approach not only enhances EHR flexibility and system compatibility but also promotes the seamless exchange and utilization of health information across various environments, as patients and care providers can utilize format translators to revise the format of previously saved records to comply with different and new standards.

### *5.6. Storage and Cost Efficiency*

The necessity to integrate off-chain storage solutions introduces additional complexity and potential points of failure, as data must be securely and reliably synchronized with the blockchain. Through the strategies and mechanisms adopted in our methodology, the blockchain extends its inherent robustness and strengths across all components of the system, including external storage. By using the edge computing paradigm and storing data on user devices, as described in our approach, integrity, ownership, and non-repudiation of data are preserved without incurring the economic costs associated with blockchain

storage or external cloud services. This approach is similar to the one described in [12], where we previously provided a blockchain-based solution for the green energy market, described in the form of a design pattern, which also leverages edge computing to optimize data management. The cost for transaction processing and the overall throughput of public blockchains are also significant constraints, potentially becoming a critical bottleneck for blockchain-based distributed applications.

The dApp client's capability to postpone blockchain transactions to times beyond the conclusion of a medical visit or examination offers a significant benefit in terms of reducing transaction costs. This delay may also allow for the aggregation of multiple data entries into a single transaction, reducing the frequency and, consequently, the total cost of transactions, while also addressing the issue of limited throughput. The dApp client also features a real-time gas cost monitoring tool that tracks gas prices to detect periods of lower average costs, offering an opportunity to execute transactions more cheaply. This feature enhances the efficiency of blockchain interactions and is beneficial during fluctuating network conditions.

It is important to note that the initial expense of deploying smart contracts is incurred only once, and the total cost varies depending on network traffic at the time of deployment. Additionally, patients have the option, within the dApp client, to set a maximum fee limit, lower than the expected costs when an average gas price is utilized. The dApp then strategically delays the deployment until the transaction demand and the average gas prices decrease. This approach does lead to a delay in the deployment transaction processing, as validator nodes prioritize transactions with higher fees. However, choosing to monitor conditions and wait rather than initiating a transaction with a potentially lower fee in the hopes that it will eventually be processed provides greater control and predictability. This strategy helps avoid potential queue backlogs where transactions might not get processed for an extended period. Specifically, a transaction with a very low gas price might never be processed if it consistently falls below the minimum gas price accepted by validators. Such transactions can remain stuck in the mempool until they are either dropped due to node policies (which vary) or replaced by the sender with another transaction with a higher gas price—a method known as "fee bumping".

### *5.7. Smart Contracts and Optimized Performance*

The security and efficiency of smart contracts on a decentralized blockchain pose unique challenges. The immutable nature of blockchain technology ensures the permanence and reliability of the stored information, whether it involves transactions or user data. Once data are stored in the blockchain, altering or deleting them becomes virtually impossible. This characteristic extends to smart contracts as well and makes post-deployment modifications or bug fixes challenging. To overcome this limitation, our model leverages the Proxy design pattern, which is designed to enhance performance and address the inherent constraints of blockchain and smart contract development. These design patterns are described in Appendix A.

In addition, the creation of individual smart contract instances for each patient avoids accumulating excessive data in a single smart contract responsible for managing different patients, a scenario that would become unfeasible with a large number of users. Conversely, by providing each user with their own smart contract instance, it becomes possible to streamline the management of each patient's data, ensuring that the system remains scalable and efficient even as the user base grows. In this way, costs become proportional to the individual user's data volume, and each patient's information is isolated and protected within their individual contract. Furthermore, this separation of instances reinforces the system's patient-centric focus since the patient becomes the only owner (see Appendix A) of the smart contract, who can define access policies for their own data.

## 6. Smart Contract Details

This section outlines the structure and functionalities of the EdgeHR and EdgeHRProxy smart contracts implemented in Solidity [45], and includes snippets of the code of key functions. When the patient's client deploys its own EdgeHR and EdgeHRProxy instances, the constructors (see Listing 1) of the smart contracts are activated. They both employ the Owner design pattern (see Appendix A) and are designed to store the wallet address of the transaction's sender (the patient), as the owner of the contract.

**Listing 1.** Setting the state variable for the owner.

```
constructor() { owner = payable(msg.sender); }
```

`msg.sender` is the sender of the transaction and is stored in the owner variable, which is of the type `address`. This setup enables the smart contract to restrict specific operations to the authorized user by comparing the sender's address in transaction requests with the stored owner's address, utilizing the `onlyOwner` modifier (see Listing 2). A modifier in Solidity is a reusable portion of code that can be attached to functions to alter their behavior and is used to simplify and centralize validation or precondition checks. Such a mechanism ensures that the patient is the exclusive authority with the capability to set access policies for their EHR. Following the successful on-chain deployment of the EdgeHR smart contract, the EdgeHRProxy smart contract is then provided with the EdgeHR instance's address, implementing the Proxy design pattern (see Appendix A). The address of the EdgeHRProxy instance becomes the gateway for accessing the service, as it forwards all requests to the EdgeHR smart contract. This approach maintains a fixed access point for users, facilitating the flexible evolution of the contract's functionality; that is, the underlying logic of the service can be updated by deploying a new version of the smart contract and obtaining a new address, which is then stored in the proxy smart contract.

**Listing 2.** Some contract management functions.

```
modifier onlyOwner() { require(msg.sender==owner, 'must be owner'); _; }
function getOwner() public view returns(address) { return owner; }
function lock() public onlyOwner { _locked = 1; }
function unlock() public onlyOwner { _locked = 0; }
```

EdgeHR smart contract stores the references to EHRs in a hash table that maps the hash of each EHR to an instance of the HealthRecord structure shown in Figure 5, facilitating the indexing and retrieval of the records, which are stored locally by the dApp clients.

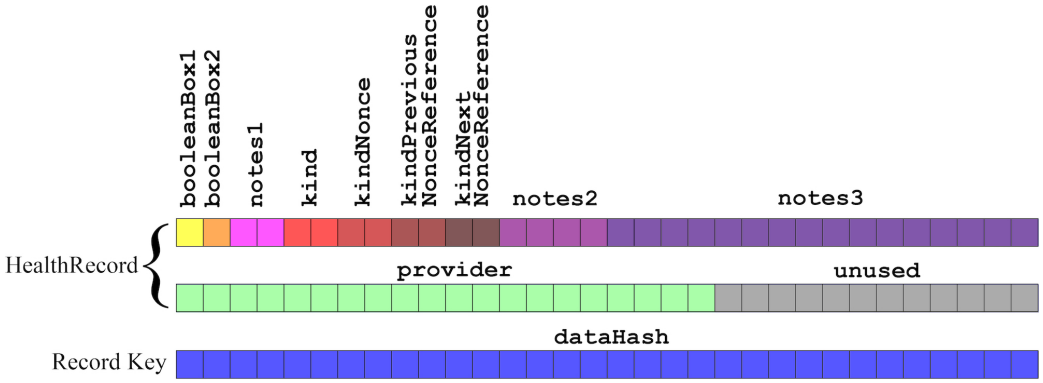


**Figure 5.** Visual representation of the HealthRecord data structure.

Figure 5 shows several boxes, where each box represents one byte of storage. The fields booleanBox1 and booleanBox2 are one byte, utilizing the `uint8` type. Fields note1, kind, kindNonce, kindPreviousNonceReference, kindNextNonceReference are two bytes each and are represented by the `uint16` type. The notes2 field occupies four bytes and is

defined as `uint32`, while notes3 is considerably larger, spanning 16 bytes and defined as `uint128`. The provider field, an `address`, uses 20 bytes. Lastly, the dataHash key, defined as `bytes32`, occupies 32 bytes.

In the HealthRecord struct, the variables have been designed, grouped together, and ordered by size (starting with the smallest and ending with the largest), to enable Solidity's compiler to perform variable packing within single 256-bit words (see Appendix A), maximizing space efficiency. This feature significantly reduces the storage allocation on the Ethereum blockchain, thereby lowering associated gas costs.

Each field serves a specific purpose, contributing to the management of health data: the booleanBox1 and booleanBox2 are two bytes allocated for storing Boolean variables, utilizing them as binary flags related to the EHR. This approach implements the BooleanBox design pattern [11] (see Appendix A), leveraging a compact and efficient method for Boolean data management in Solidity (the code for this library can be found in https://github.com/Crypto-Patterns/Solidity-Patterns, accessed on 9 May 2024). These two fields implement 16 Boolean variables, designed for internal use, that can provide additional context about the status of the recorded data. Medical professionals can customize them according to their preferences, with the exception of the first bit in booleanBox1, which is reserved for denoting whether the record had been deleted, as detailed in the explanation of the replaceMetaData functions. Examples of potential uses include flags for "Urgency of follow-up", to indicate if the data point to a condition that requires follow-up examinations; "Verified", denoting whether the data have been verified for accuracy; "Data completeness", showing whether the record is complete or if additional information is required; "Text notes by patient", distinguishing if the text in the notes fields, stored as binary numbers, was written by the care provider or the patient; "Emit events", to control event emission by record-updating functions; "Patient consent forms", indicating whether patient consent has been obtained for the particular medical procedure or data usage.

The three note fields, totaling 176 bits (22 bytes), are designed to store short text strings that serve multiple purposes. The text must be converted into binary, utilizing ASCII encoding, and segmented into 16, 32, and 128 bits by the client. Such notes can be used to represent short personalized remarks from the doctor or the patient, or to specify whether a specific data format, such as HL7, FHIR, or openHR, is used.

The kind is an identifier that classifies the type or category of the record, while kindNonce ensures uniqueness within each category. The unique identification of each record is achieved through the combination of both. Additionally, kindPreviousNonceReference serves as a reference to a previous record within the same kind, establishing a backward link. Conversely, kindNextNonceReference points to a subsequent record, forming a forward connection. This field can only be updated when a following record is added, and its nonce is known. These references facilitate the creation of a bidirectional linked list of records, enhancing the organization and contextual understanding of related health events or statuses within a particular category. The provider field represents the wallet address of the care provider responsible for generating the medical record. Finally, the dataHash represents the hash of the actual data.

Utilizing `uint16` for kind and kindNonce, rather than a whole 256-bit word, does not compromise the system's longevity. Although the maximum values for these `uint16` fields could theoretically be reached, considering the current growth rate of Ethereum (approximately 2,628,000 blocks per year) and a usage pattern of one new record per block, it would take several centuries for the combination of kind and nonce to reach their maximum values. This approach, therefore, ensures the longevity and practical utility of the system for the foreseeable future, striking a balance between storage efficiency and future scalability.

In the client application, the data are replicated using the same structure with the addition of an extra string field called blockStamp. The blockstamp denotes a reference to the block where the transaction, which updated the metadata, has been stored. This reference is represented by the unique hash of said block, providing a precise and verifiable

pointer to the block's location within the blockchain, which contains the transaction. This field also establishes a temporal marker based on the blockchain's timeline, synchronized with the blockchain's pace. It allows us to overcome issues associated with timestamps, which can be inaccurate due to potential clock tampering or malfunctions. This mechanism ensures traceability and integrity of the transaction history related to metadata updates.

The smart contract is equipped with functions that we distinguish into two groups: the first one allows the patient, as the contract's owner, to manage and customize the smart contract, while the second group is designed for interacting with the medical metadata.

### *6.1. Management and Customization Functions*

The proposed EdgeHR smart contract implements several management functions. The following shows the main ones, which are also reported in Listing 2. The getOwner function identifies the owner's address, representing the patient. The lock and unlock functions, restricted to the owner, serve as a security mechanism to halt the smart contract in the case of anomalies or when vulnerabilities are discovered, to mitigate potential (or further) risk. These two functions toggle the smart contract's active state, performing an emergency stop. In the halted state, the smart contract blocks the transactions, giving developers time to address the issue or fix any bugs. Listing 2 also provides the onlyOnwer modifier.

The authorizeCareProvider function (see Listing 3) allows the owner to manage access controls, specifying for different addresses an authorization level, mapped into a dictionary, or Solidity's `mapping`. The framework establishes four tiers of permissions: level 1 for data reading, level 2 for data writing, level 3 for adding data to records created by a different care provider, and level 4 for data modification by the original creator. The expiration of the authorization is defined by a specific number of blocks, set in the authorizationDurationInBlocks parameter, after which the authorization becomes invalid. Given tat Ethereum's block time is 12 s (i.e., a new block is minted every 12 s, except in unexpected conditions), 24 h corresponds to 7200 blocks, 10 days to 72,000 blocks, and 1 month (31 days) to 223,200 blocks.

**Listing 3.** Controlling access according to addresses' authorization level.

```
function authorizeCareProvider(address _address, uint128 _level,
            uint128 _blockDuration) external onlyIfUnlocked onlyOwner {
   require(_address != address(0), "address cannot be null");
   require(_level >=0 && _level <=4, "level must be from 0 to 4");
   PermissonRecord memory permission;
   permission.level = _level;
   permission.expirationBlock = _blockDuration + uint128(block.number);
   permissions[_address] = permission;
}
```

Consistency between metadata record types and the actual data stored locally is ensured by the "kind" variable. In the client, these values are paired with descriptive strings and are specific to each patient's EHR. Existing descriptions, which originated from earlier interactions with other care providers, are downloaded by the client's dApp when a new medical professional completes the handshake process. When new types and descriptions were not previously defined, they are added and then shared with any subsequent care providers that the patient interacts with. Before adding a new description that might duplicate an existing one, the dApp client assesses the similarity between the newly entered data and the descriptions already in the system by employing a string distance function. It then asks the medical professional for confirmation if whether it is truly necessary to add this new type/description pair. This indexing mechanism allows the system to manage a common consistent subset of record types and descriptions for each patient, shared with their care providers, accommodating personalized medical histories that reflect the unique and specific medical interactions a patient may experience.

Maintaining verbose descriptions of the metadata types in the smart contract is not operationally necessary, but it can be beneficial if the patient wishes to authorize third

parties, such as researchers or medical professionals, to access these descriptions, enabling them to conduct statistical analysis or scientific research using the descriptions of the metadata records (see Section 6.2).

The manageKindDescription function mirrors this mapping in the smart contract. Once set, a description becomes permanently mapped to the kind's index. If the user opts to store these details on the blockchain, the smart contract will use the data structure in Listing 4, a hash table, to map the record type indexes to their corresponding ASCII-encoded textual descriptions, allowing for 32 bytes of text. In this structure, the `uint16` serves as the index for the record type, while the `uint256` encodes the textual description in ASCII.

**Listing 4.** Hash table for mapping record type indexes to descriptions.

```
mapping (uint16 => uint256) private kindDescriptions;
```

Before a new health record is created on-chain, the function manageKindDescription (see Listing 5) verifies whether the type's description already exists in the mapping and determines the necessary actions based on its presence or absence. If a description is already linked to that type, the process continues without making any changes. If there is no description associated with that type, the function creates a new entry for that kind with the provided description. Finally, if a description for that type already exists but is different from the one currently provided, an event can be emitted to alert that the care provider has entered a description that is different from the expected one for that type.

**Listing 5.** Update EHR descriptions of types.

```
function manageKindDescription (uint16 _kind, uint256 _description,
                                uint8 _emitEvents) private view {
   require(_kind>0 && _kind<=kindDescriptionsProgressiveIndex+1, "wrong kind");
   if(_kind > kindDescriptionsProgressiveIndex) {
      //add the description
      kindDescriptions[_kind] = _description;
      kindDescriptionsProgressiveIndex++;
      if(_emitEvents == 1)
         emit KindDescriptionAdded(msg.sender, _kind, _description);
   }
   else {
      if(kindDescriptions[_kind] != _description) {
         //description mismatch
         if(_emitEvents == 1)
            emit KindDescriptionMismatch(msg.sender, kindDescriptions[_kind],
                                _description);
      }
   }
}
```

When care providers need to define the meaning of specific Boolean values set in the booleanBox field, they can utilize the setBooleansDescriptions function to specify what each flag represents. The function maps integers to ASCII-encoded strings inside a dictionary, describing the binary flags, and this dictionary is nested within another one, which employs `address` types as its keys, as shown in Listing 6, linking each entry to the function's caller. The function does not allow for overwriting existing entries.

**Listing 6.** Nested mapping for Boolean flag descriptions.

```
mapping(address => mapping(uint8 => uint256)) private booleansDescriptions;
```

The getBooleanDescription function allows for the retrieval of the descriptions associated with the booleanBoxes' flags. It can be executed by the owner or entities with reading permissions and requires the provider address and the index of the booleanBox flag as

input (see Appendix A), returning descriptions individually. This approach is also adopted to accommodate Solidity's limitations in handling dynamic arrays of strings.

### *6.2. Functions for Interactions with Metadata*

The functions addMetaData, appendMetaData, appendMetaDataToSameProviderRecords, appendMetaDataToDifferentProviderRecords, and replaceMetaData are available in multiple variants, utilizing polymorphism to cater to different users. Specifically, certain versions are exclusive to the owner, while others are accessible to care providers. The key differences among these variants stem from the level of authorization needed and the requirement to assign a value to the booleanBox and notes variables. When these parameters are unnecessary, invoking the more streamlined versions of the functions reduces data transmission, thereby enhancing performance and lowering gas costs.

The addMetaData function is accessible to the owner or entities authorized at level 2 and requires the hash of the actual data, their type (kind), and various parameters as inputs depending on the variant of the function. To prevent duplicate entries, the function first verifies that the record, uniquely identified by the provided dataHash, does not already exist. This verification is carried out using a required statement that ensures both the provider field and the kind field of the mapping entry default to uninitialized values, specifically `address(0)`, for provider and 0 for kind, as indicated in Listing 7.

**Listing 7.** Verification of entry initialization conditions.

```
require(healthRecords[_dataHash].provider == address(0) &&
          healthRecords[_dataHash].kind == 0, "the hash is already in the EHR");
```

If the record does exist, the function reverts the transaction; otherwise, if the supplied hash does not exist, the function initializes a new HealthRecord structure instance by setting up its fields. The kindNonce value for the new record is determined by incrementing the last known nonce value for the given kind, which is stored in the latestNonces mapping. The new record is added to the healthRecords mapping using the dataHash variable as the key, ensuring each record can be uniquely identified and accessed directly via the hash of its related medical data. Additionally, to enable a sequential access similar to an array, but without the considerable computational costs of utilizing dynamic arrays, a second mapping, called healthRecordsIndexes, is employed. This mapping links each record's index to its corresponding medical data hash, implementing the 'Mapping vs. Array' design pattern which optimizes storage space on the blockchain (see Appendix A). The ehrProgressiveIndex variable, incremented for each new record, serves as the key in this secondary mapping (starting from one since zero is reserved in place of null). Finally, if the emitEvents flag is set, the function emits a MetaDataAdded event (see Section 6.3), signaling the successful addition of a new record including details such as the care provider address (`msg.sender`) and the ehrProgressiveIndex. This event can be used by external observers to track updates to the EHR system in real time.

Upon successful addition, the function addMetaData returns the unique index and the kindNonce for the new record. The owner-specific versions of the function enable the patient to update records autonomously. By specifying the address that uniquely identifies the healthcare provider, the patient can ensure that EHRs are accurately attributed to the appropriate medical professional. This functionality is crucial in scenarios where the healthcare provider may not have the capability to execute or create the transaction themselves and empowers the patient to take a more active role in record management while preserving the traceability and accountability of each provider's contributions to EHRs. Listing 8 presents the most generic version of the addMetaData function, intended for care providers, which also accommodates the booleanBoxes and the notes.

**Listing 8.** Function recording useful metadata of EHRs.

```
function addMetaData(
   uint8 _booleanBox1, uint8 _booleanBox2, uint8 _emitEvents, uint16 _notes1,
   uint16 _kind, uint16 _kindPreviousNonceReference, uint32 _notes2,
   uint128 _notes3, bytes32 _dataHash, uint256 _description)
 external onlyIfUnlocked hasPermission(2) returns (uint16, uint16) {
   require(_kind > 0, "the kind must be specified");
   require(healthRecords[_dataHash].provider == address(0) &&
          healthRecords[_dataHash].kind == 0, "the hash is already in the EHR");
   if (booleanBoxGlobal.getFlag(1) && _description > 0)
      manageKindDescription(_kind, _description, _emitEvents);
   HealthRecord memory newRecord = HealthRecord({
      booleanBox1: _booleanBox1, booleanBox2: _booleanBox2, notes1: _notes1,
      kind: _kind, kindNonce: latestNonces[_kind] + 1,
      kindPreviousNonceReference: _kindPreviousNonceReference,
      kindNextNonceReference: 0, notes2: _notes2, notes3: _notes3,
      provider: msg.sender});
   latestNonces[_kind] = newRecord.kindNonce;
   healthRecords[_dataHash] = newRecord;
   ehrProgressiveIndex++;
   healthRecordsIndexes[ehrProgressiveIndex] = _dataHash;
   if(_emitEvents == 1)
      emit MetaDataAdded(msg.sender, ehrProgressiveIndex, _kind,
                         newRecord.kindNonce);
   return (ehrProgressiveIndex, newRecord.kindNonce);
}
```

The appendMetaData, appendMetaDataToSameProviderRecords, appendMetaDataToDifferentProviderRecords, and replaceMetaData functions operate similarly and can also be executed by the owner. Besides the basic inputs, they require a reference to a nonce of an existing record, enabling the creation of a continuity link between records. The appendMetaDataToSameProviderRecords functions operate in the same way and can be executed by an entity with level 2 permission. However, they can append metadata only to records that were originally created by the same entity that is invoking the function. The appendMetaDataToDifferentProviderRecords functions, accessible to the owner or entities with level 3 permission, operate with a broader scope. These functions enable one to append metadata to records that were not necessarily created by the same entity invoking the function.

The replaceMetaData functions, requiring the highest authorization level and, in the case of the owner, the care provider address, are designed to correct data errors. This includes modifications to booleanBoxes, notes, and the data hash, while the kind, kindNonce, kindPreviousNonceReference, and kindNextNonceReference fields cannot be altered. Additionally, care providers who have the necessary authorization can still only modify their own records and not any others. The functions can also be used to partially delete a record by setting some fields to zero. However, a duplicate of the old records persists within the dApp clients and must be manually deleted by the users. These functions should be used cautiously, but despite their sensitive nature, they are essential to enable care providers to rectify potential errors. Moreover, although the hashed data do not reveal the original information, the patient's right to delete them from the blockchain remains paramount.

The polymorphic getSummary function (see Listing 9) extracts notes and record type descriptions from the mappings, temporarily stores them in an array in memory, and returns them as output. This allows authorized care providers to retrieve a condensed high-level view of the medical records, thus offering an overview of the patient's medical history. The function allows to specify limits and offsets to support data pagination.

**Listing 9.** Function providing notes and descriptions.

```
function getSummary(uint16 _limit, uint16 _offset) external view onlyIfUnlocked
         returns(PreliminarySearchStruct[] memory) {
   uint256 effectiveLimit = uint16(min(_limit + _offset, ehrProgressiveIndex));
   require(effectiveLimit > _offset, "invalid limit or offset");
   uint256 extractedRecords = effectiveLimit - _offset;
   PreliminarySearchStruct[] memory PreliminarySearch =
        new PreliminarySearchStruct[](extractedRecords);
   uint16 index;//defaults to 0
   _offset++;
   while(_offset <= effectiveLimit) {
      bytes32 tempHash = healthRecordsIndexes[_offset];
      HealthRecordStruct memory tempRecord = healthRecords[tempHash];
      PreliminarySearchStruct memory newRecord = PreliminarySearchStruct({
         notes1: tempRecord.notes1, notes2: tempRecord.notes2,
         notes3: tempRecord.notes3,
         description: kindDescriptions[tempRecord.kind]
      });
      PreliminarySearch[index] = newRecord;
      index++;
      _offset++;
   }
   return PreliminarySearch;
}
```

The getSummaryByKind function scans the records stored within the smart contract to extract and set a summary of records for a certain category. Like the previous one, this function is accessible to the contract owner, authorized entities, or when the patient has explicitly opted to share their data. The getRecordByIndex function can be utilized by either the owner or an authorized entity to retrieve a record, enabling the reconstruction of a patient's medical history. This function facilitates the validation of stored information's integrity by enabling a comparison between the stored data hash and the hash of the actual data. The getRecordByHash function, accessible by the owner or authorized entities, retrieves specific record details, including the index based on the data hash provided to the function.

Patients can explicitly choose (by setting a Boolean flag in a global BooleanBox variable) to mirror the textual descriptions of their EHRs on-chain and publicly share them with non-authorized third parties such as medical professionals or researchers. Third parties can download data by invoking the getDescription function for evaluative or research purposes. If they identify relevant record descriptions and wish to request detailed EHRs, they can contact the patient using the contactPatient function, WHICH triggers an event that records their address and their email contact. Once notified, the patient can review the credentials of the institution or research center and opt to reply with an email message to facilitate further communication.

### *6.3. Events*

Events facilitate communication with observers outside the blockchain by triggering actions in the front-end interface (see Appendix A.4). Within the EdgeHR smart contract, all functions can optionally emit events. For those interacting with the EHR, this option is explicitly specified at the time of invocation through a parameter. For contract management functions, this feature is controlled by a global BooleanBox flag, which only the owner can set. These events document the actions taken on the records and identify the address responsible for the action, assisting in the reconstruction of the EHR history by providing a transparent and traceable footprint of changes. They also serve to facilitate communication with observers outside the blockchain by triggering actions in the front-end interface (see Appendix A.4) and notifying the owner about key activities or errors. Listing 10 shows a list of the events, along with a brief description of their purposes.

**Listing 10.** Events of the EdgeHR smart contract.

```
event KindDescriptionAdded(address _address, uint16 _kind, uint256 _descripion);
event KindDescriptionMismatch(address _address, uint256 _storedDescripion,
      uint256 _providedDescription);
event MetaDataAdded(address _address, uint16 _index, uint16 _kind,
      uint16 _kindNonce);
event MetaDataAppended(address _address, uint16 _index, uint16 _kind,
      uint16 _kindNonce, uint16 kindPreviousNonceReference);
event MetaDataReplaced(address _address, uint16 _index, uint16 _kind,
      uint16 _kindNonce);
event RequestContact(address _address, string _email);
event SummaryDownloaded(address _address, uint16 _first, uint16 _last);
event RecordRetrievedByIndex(address _address, uint16 _index);
event RecordRetrievedByHash(address _address, uint16 _index);
```

KindDescriptionAdded is emitted when a new description for a kind is added to the system, logging the address, kind identifier, and description details. KindDescriptionMismatch is emitted when there is a discrepancy between stored and provided descriptions for a kind; MetaDataAdded logs the addition of new metadata; MetaDataAppended is emitted when data are appended to an existing record; MetaDataReplaced is emitted when a record is replaced; RequestContact is triggered to send contact information to facilitate future interaction; SummaryDownloaded is emitted when a summary of records is downloaded; RecordRetrievedByIndex is emitted when a record is retrieved by its index; and RecordRetrievedByHash logs when a record is retrieved by hash.

## 7. Conclusions

While blockchain technology offers several promising features such as decentralization, transparency, security, and immutability, it also faces significant challenges, including issues related to scalability, privacy, and interoperability, which require careful consideration and resolution. The success and acceptance of blockchain-based applications may rest heavily on the extent to which these limitations are tackled. This article proposed a solution that enables the adoption of blockchain for sharing EHRs, with special attention to the feasibility of using public blockchains such as Ethereum. To empower patients with full control over their medical records, we introduced EdgeHR, a patient-centric electronic health record solution that aims at addressing the issues that arise from Ethereum's network limitations. We adopted the edge computing paradigm and integrated design patterns that help achieve desirable features in an EHR system, such as decentralization, security, scalability, accessibility, management flexibility, privacy, cost-effective storage solutions, optimized performance, interoperability, and patient control.

In EdgeHR, data processing and storage are delegated to the user's devices. The authentication process occurs using the same blockchain technology's mechanisms that also ensure data integrity. The EdgeHR smart contract is responsible for managing metadata, related to medical records, creating an index on the blockchain, accessible to patients and authorized care providers. Our system supports autonomous transaction management and the possibility of employing gasless transactions offering significant flexibility in EHR management and population. We have also proposed to use the digests of the blockchain's blocks as timestamps to easily align with the pace of block generation and order recorded data stored on client devices, effectively preventing tampering and desynchronization issues.

**Author Contributions:** Conceptualization, V.M. and E.T.; methodology, V.M. and E.T.; software, V.M.; validation, V.M. and E.T.; writing—original draft preparation, V.M.; writing—review and editing, V.M., G.P. and E.T.; supervision, G.P. and E.T. All authors have read and agreed to the published version of the manuscript.

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

**Data Availability Statement:** No new data were created or analyzed in this study. Data sharing is not applicable to this article.

**Acknowledgments:** The authors acknowledge the support of University of Catania PIACERI 2020/22 Project "TEAMS".

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

## Appendix A. Design Patterns for Smart Contracts

This Appendix describes design patterns that are useful in the development of smart contracts and help overcome some of the inherent limitations of Ethereum's blockchain technology.

### *Appendix A.1. Accessibility*

The following design patterns can be employed to simplify contract management, ensure a robust authorization mechanism, and enhance security.

- Ownership: Ownership pattern designates the contract deployer as its owner. When the contract is published, its constructor is executed automatically, capturing the deployer's address from the msg.sender property of the transaction and saving it to a storage variable. This mechanism allows restricting access to specific contract operations to the owner.
- Authorization [46]: this pattern consists of managing user identities (wallet addresses), with different permission levels, ensuring only entities with appropriate authorization can modify, add, or access EHR system data.
- Circuit Breaker [47]: this pattern allows the contract owner to lock all operations of the contract's functions in case of vulnerabilities or unforeseen issues, allowing time for issue resolution and enhancing the system's resilience against bugs, breaches, or attacks.

### *Appendix A.2. Immutability*

The Proxy, or Migration [11,48], pattern addresses blockchain immutability, which prevents the direct modification of a contract's code, ensuring a smooth transition for users while enabling contract updates and improvements. It employs a proxy contract that keeps a reference to the address of the actual contract. This strategy offers a solution for updating smart contracts, allowing the owner to deploy and provide a new version of the actual contract by updating such a reference. Client code interacts with the proxy smart contract, which redirects the execution to the new version of the actual contract.

### *Appendix A.3. Scalable Storage Solutions and Interoperability*

To maintain high performance levels without overloading the network, preserve data privacy, and ensure data integrity while minimizing on-chain storage and related costs, strategies such as off-chain data storage and on-chain anchoring of integrity proofs are adopted. The Data Anchoring pattern entails storing a compact fixed-sized hash of data on the blockchain and the actual data in online databases, using a reliable and change-sensitive hashing algorithm such as SHA256 [49]. This approach ensures that the EHR system can handle growing amounts of data efficiently and allows for tampering detection. In addition, this pattern can positively impact interoperability, albeit indirectly. When combined with international standards, such as HL7 (https://www.hl7.org/ accessed on 17 April 2024), ICD11 (https://www.who.int/standards/classifications/classification-of-diseases accessed on 17 April 2024), or LOINC (https://mmshub.cms.gov/measure-lifecycle/measure-specification/specify-code/LOINC accessed on 17 April 2024) that define specific formats for structured health data representation, data anchoring can ensure integrity while allowing for the exchange of data across various systems.

*Appendix A.4. Chain Boundedness*

The execution of smart contracts is limited to the blockchain's ecosystem; i.e., they cannot directly affect external machines or software components in the outside world. In this context, the Oracle patterns [10] play a pivotal role. Oracles can both supply external data to the blockchain (inbound) and share blockchain data with external systems (outbound), allowing smart contracts' logic to make decisions based on data sources outside the blockchain. This integration ensures that smart contracts used by the EHR system have access to relevant information both within and outside the blockchain.

*Appendix A.5. Cost Efficiency and Optimized Performance*

In the context of smart contracts, computational cost poses a unique challenge not found in traditional programming, since it carries a financial implication. Considering the high volume of transactions and data management tasks in EHR systems, optimizing performance and cost is crucial for the system's economic sustainability. Below are some design patterns that can be employed.

- Packing Variables [9]: the EVM operates on 256-bit words for memory allocation. To minimize gas consumption, this pattern recommends sequentially declaring variables of the same data type, enabling the Solidity compiler to efficiently pack these variables together and optimize memory usage.
- Boolean Box [11]: this pattern optimizes storage for Boolean values in the EVM, which allocates 256 bits for a single Boolean value. The Boolean Box pattern aggregates multiple Boolean values into one word as individual bits, significantly optimizing storage use and substantially reducing the associated gas costs. It employs bitwise operations to set, retrieve, or modify the individual Boolean states within this word.
- Uint256 [9]: using variables smaller than 256 bits incurs extra gas due to automatic conversion to uint256. Thus, when smaller integers cannot be packed efficiently, directly using uint256 variables is more gas-efficient.
- Mapping vs, Array [9]: this approach recommends using more gas-efficient hash tables over arrays for data lists.
- Fixed Size Array [9]: this strategy recommends using arrays of a predetermined size to save gas, as dynamic arrays incur extra memory management operations.
- Default Value [9]: this pattern avoids explicit initialization of variables when their intended value is zero; it is cost-efficient since all variables in Solidity automatically default to zero.
- Short Circuit [9]: this approach consists of optimizing logical operations by ordering expressions to avoid unnecessary evaluations, thus saving gas.
- Short Constant Strings [9]: this pattern reduces stored string lengths to under 32 bytes (256 bits) and lowers storage costs.

## References

1. Gunter, T.D.; Terry, N.P. The emergence of national electronic health record architectures in the United States and Australia: Models, costs, and questions. *J. Med. Internet Res.* **2005**, *7*, e383. [CrossRef] [PubMed]
2. Wang, X.; Sun, J.; Wang, Y.; Liu, Y. Deepen electronic health record diffusion beyond breadth: Game changers and decision drivers. *Inf. Syst. Front.* **2022**, 24, 537–548. [CrossRef]
3. Wood, G.; Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
4. Antonopoulos, A.M.; Wood, G. *Mastering Ethereum: Building Smart Contracts and Dapps*; O'Reilly Media: Sebastopol, CA, USA, 2018.
5. Xiong, H.; Chen, M.; Wu, C.; Zhao, Y.; Yi, W. Research on progress of blockchain consensus algorithm: A review on recent progress of blockchain consensus algorithms. *Future Internet* **2022**, *14*, 47. [CrossRef]
6. Daniel, J.; Sargolzaei, A.; Abdelghani, M.; Sargolzaei, S.; Amaba, B. Blockchain technology, cognitive computing, and healthcare innovations. *J. Adv. Inf. Technol.* **2017**, *8*, 194–198. [CrossRef]
7. Bayle, A.; Koscina, M.; Manset, D.; Perez-Kempner, O. When blockchain meets the right to be forgotten: Technology versus law in the healthcare industry. In Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI), Santiago, Chile, 3–6 December 2018; pp. 788–792.

8. Destefanis, G.; Marchesi, M.; Ortu, M.; Tonelli, R.; Bracciali, A.; Hierons, R. Smart contracts vulnerabilities: A call for blockchain software engineering? In Proceedings of the International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Campobasso, Italy, 20 March 2018; pp. 19–25.
9. Marchesi, L.; Marchesi, M.; Destefanis, G.; Barabino, G.; Tigano, D. Design patterns for gas optimization in ethereum. In Proceedings of the International Workshop on Blockchain Oriented Software Engineering (IWBOSE), London, ON, Canada, 18 February 2020; pp. 9–15.
10. Mühlberger, R.; Bachhofner, S.; Castelló Ferrer, E.; Di Ciccio, C.; Weber, I.; Wöhrer, M.; Zdun, U. Foundational oracle patterns: Connecting blockchain to the off-chain world. In Proceedings of the Business Process Management: Blockchain and Robotic Process Automation Forum, Seville, Spain, 13–18 September 2020; Springer: Cham, Switzerland, 2020; pp. 35–51.
11. Mandarino, V.; Pappalardo, G.; Tramontana, E. Some Blockchain Design Patterns for Overcoming Immutability, Chain-Boundedness, and Gas Fees. In Proceedings of the 3rd Asia Conference on Computers and Communications (ACCC), Shanghai, China, 16–18 December 2022; pp. 65–71.
12. Mandarino, V.; Pappalardo, G.; Tramontana, E. Proof of Flow: A Design Pattern for the Green Energy Market. *Future Internet* **2023**, *15*, 313. [CrossRef]
13. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
14. Khan, W.Z.; Ahmed, E.; Hakak, S.; Yaqoob, I.; Ahmed, A. Edge computing: A survey. *Future Gener. Comput. Syst.* **2019**, *97*, 219–235. [CrossRef]
15. Kavitha, R.; Kannan, E.; Kotteswaran, S. Implementation of cloud based Electronic Health Record (EHR) for Indian healthcare needs. *Indian J. Sci. Technol.* **2016**, *9*, 1–5. [CrossRef]
16. Ibrahim, A.; Mahmood, B.; Singhal, M. A secure framework for sharing electronic health records over clouds. In Proceedings of the International Conference on Serious Games and Applications for Health (SeGAH), Orlando, FL, USA, 11–13 May 2016; pp. 1–8.
17. Matos, D.R.; Pardal, M.L.; Adao, P.; Silva, A.R.; Correia, M. Securing electronic health records in the cloud. In Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems, Porto, Portugal, 23 April 2018; pp. 1–6.
18. Zhang, K.; Chen, T.; Chen, S.; Wei, L.; Ning, J. Lightweight, verifiable and revocable EHRs sharing with fine-grained bilateral access control. *Clust. Comput.* **2024**, 1–17. [CrossRef]
19. Ekblaw, A.; Azaria, A.; Halamka, J.D.; Lippman, A. A Case Study for Blockchain in Healthcare: "MedRec" prototype for electronic health records and medical research data. In Proceedings of the IEEE Open & Big Data Conference, Washington, DC, USA, 5–8 December 2016; Volume 13, p. 13.
20. Quaini, T.; Roehrs, A.; da Costa, C.A.; da Rosa Righi, R. A Model For Blockchain-Based Distributed Electronic Health Records. *IADIS Int. J. WWW/Internet* **2018**, *16*, 66–79. [CrossRef]
21. 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]
22. Haleem, A.; Javaid, M.; Singh, R.P.; Suman, R.; Rab, S. Blockchain technology applications in healthcare: An overview. *Int. J. Intell. Netw.* **2021**, *2*, 130–139. [CrossRef]
23. Mettler, M. Blockchain technology in healthcare: The revolution starts here. In Proceedings of the International Conference on e-Health Networking, Applications and Services (Healthcom), Munich, Germany, 14–16 September 2016; pp. 1–3.
24. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Tech. Rep., 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 9 May 2024).
25. Jiang, S.; Cao, J.; Wu, H.; Yang, Y.; Ma, M.; He, J. Blochie: A blockchain-based platform for healthcare information exchange. In Proceedings of the International Conference on Smart Vomputing (Smartcomp), Taormina, Italy, 18–20 June 2018; pp. 49–56.
26. Wang, Y.; Zhang, A.; Zhang, P.; Wang, H. Cloud-assisted EHR sharing with security and privacy preservation via consortium blockchain. *IEEE Access* **2019**, *7*, 136704–136719. [CrossRef]
27. 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]
28. Hang, L.; Choi, E.; Kim, D.H. A novel EMR integrity management based on a medical blockchain platform in hospital. *Electronics* **2019**, *8*, 467. [CrossRef]
29. Dubovitskaya, A.; Baig, F.; Xu, Z.; Shukla, R.; Zambani, P.S.; Swaminathan, A.; Jahangir, M.; Chowdhry, K.; Lachhani, R.; Idnani, N.; et al. ACTION-EHR: Patient-centric blockchain-based electronic health record data management for cancer care. *J. Med. Internet Res.* **2020**, *22*, e13598. [CrossRef]
30. Tith, D.; Lee, J.S.; Suzuki, H.; Wijesundara, W.; Taira, N.; Obi, T.; Ohyama, N. Application of blockchain to maintaining patient records in electronic health record for enhanced privacy, scalability, and availability. *Healthc. Inform. Res.* **2020**, *26*, 3–12. [CrossRef] [PubMed]
31. Capece, G.; Lorenzi, F. Blockchain and Healthcare: Opportunities and Prospects for the EHR. *Sustainability* **2020**, *12*, 9693. [CrossRef]
32. Kim, M.; Yu, S.; Lee, J.; Park, Y.; Park, Y. Design of secure protocol for cloud-assisted electronic health record system using blockchain. *Sensors* **2020**, *20*, 2913. [CrossRef]

33. Abdelgalil, L.; Mejri, M. HealthBlock: A framework for a collaborative sharing of electronic health records based on blockchain. *Future Internet* **2023**, *15*, 87. [CrossRef]
34. Zhang, P.; White, J.; Schmidt, D.C.; Lenz, G.; Rosenbloom, S.T. FHIRChain: Applying blockchain to securely and scalably share clinical data. *Comput. Struct. Biotechnol. J.* **2018**, *16*, 267–278. [CrossRef] [PubMed]
35. Pilares, I.C.A.; Azam, S.; Akbulut, S.; Jonkman, M.; Shanmugam, B. Addressing the challenges of electronic health records using blockchain and ipfs. *Sensors* **2022**, *22*, 4032. [CrossRef]
36. DeSalvo, K. *Connecting Health and Care for the Nation: A Shared Nationwide Interoperability Roadmap Draft Version 1.0.*; The Office of the National Coordinator for Health IT: Washington, DC, USA, 2015.
37. 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]
38. Mayer, H. ECDSA security in bitcoin and ethereum: A research survey. *CoinFaabrik* **2016**, *28*, 50.
39. Kim, C. Ethereum 2.0: How It Works and Why It Matters. Coindesk. 2020. Available online: https://www.coindesk.com/wp-content/uploads/2020/07/ETH-2.0-072120.pdf (accessed on 17 April 2024).
40. Gersbach, H.; Mamageishvili, A.; Schneider, M. Staking pools on blockchains. *arXiv* **2022**, arXiv:2203.05838.
41. Xie, J.; Yu, F.R.; Huang, T.; Xie, R.; Liu, J.; Liu, Y. A survey on the scalability of blockchain systems. *IEEE Netw.* **2019**, *33*, 166–173. [CrossRef]
42. Halpin, H. Deconstructing the decentralization trilemma. *arXiv* **2020**, arXiv:2008.08014 .
43. Zhou, Q.; Huang, H.; Zheng, Z.; Bian, J. Solutions to scalability of blockchain: A survey. *IEEE Access* **2020**, *8*, 16440–16455. [CrossRef]
44. Werth, J.; Berenjestanaki, M.H.; Barzegar, H.R.; El Ioini, N.; Pahl, C. A Review of Blockchain Platforms Based on the Scalability, Security and Decentralization Trilemma. In Proceedings of the Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023), Prague, Czech Republic, 24–26 April 2023; pp. 146–155
45. Iyer, K.; Dannen, C. *Building Games with Ethereum Smart Contracts*; Apress: New York, NY, USA, 2018.
46. Bartoletti, M.; Pompianu, L. An empirical analysis of smart contracts: Platforms, applications, and design patterns. In Proceedings of the Financial Cryptography and Data Security: Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, 7 April 2017; pp. 494–509.
47. Wohrer, M.; Zdun, U. Smart contracts: Security patterns in the ethereum ecosystem and solidity. In Proceedings of the International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Campobasso, Italy, 20 March 2018; pp. 2–8.
48. Worley, C.R.; Skjellum, A. Opportunities, challenges, and future extensions for smart-contract design patterns. In Proceedings of the Business Information Systems Workshops: BIS International Workshops, Colorado Springs, CO, USA, 8–10 June 2019; pp. 264–276.
49. Rachmawati, D.; Tarigan, J.; Ginting, A. A comparative study of Message Digest 5 (MD5) and SHA256 algorithm. *J. Phys. Conf. Ser.* **2018**, *978*, 012116. [CrossRef]