Privacy-Preserving Fine-Grained EMR Access Control for IoMT: A Hybrid RBAC-Smart Contract Scheme with Attribute-Based Authorization

Hongzhi Li, Dun Li, Gang Lv, Noel Crespi, Roberto Minerva, Wenhao Shao, Zheqing Zhang, Qishou Xia

***Abstract*—The widespread application of medical information systems has promoted the growth of personal electronic medical records (EMRs), which are typically produced in different medical institutions and stored in data centers. Consequently, data owners no longer retain control over their medical data, nor can they establish access control rules for their EMRs. Therefore, this study designs a patient-centered EMR access control system that integrates decentralized smart contracts and role-based access control (RBAC) to provide fine-grained data access control. In this system, we integrate a role-based access control model to achieve user-permission definition and adopt a personalized data access policy definition mechanism to achieve patient-centered data access control. The proposed system allows data owners to define a series of data access policies through smart contracts, achieving decentralized management of data access control permissions. In addition, we analyze the security features of this scheme and design a series of comparative experiments to evaluate the performance. The experimental results show that this system can efficiently achieve access control of personal electronic medical records and has higher reliability compared to traditional cloud-based EMR sharing systems.**

***Index Terms*—Access Control, RBAC, Electronic Medical Record (EMR), Smart Contract.**

I. Introduction

ELECTRONIC medical records (EMRs) are digital medical diagnostic records that contain detailed records of a patient's personal information, medical history, diagnosis, treatment plan, and medical events. EMRs have significant advantages in improving medical service efficiency, promoting medical information exchange, and optimizing the allocation of medical resources [1]. Furthermore, the sharing of EMRs can significantly encourage the exchange of healthcare research and treatment information, including internal sharing within medical institutions and cross-institutional sharing [2]-[4]. Medical cloud has become the main storage solution for EMRs due to its high scalability and the convenience of data sharing. However, this centralized storage model inherently transfers the custody of sensitive health data from patients to third-party cloud providers, effectively depriving patients of direct control over their own medical information. Once the medical cloud is compromised or the access permissions of EMRs are out of control, it may lead to the leakage of patient privacy and the loss of critical information [5], [6]. Moreover, without appropriate data access control mechanisms, the above-mentioned EMRs sharing may bring unauthorized access risks [7]. These risks fundamentally stem from the current data governance paradigm. Once medical records reside in a centralized cloud repository, patients lack the ability to define access rules for their own EMRs autonomously. Therefore, it is necessary to develop robust access control mechanisms that can effectively mitigate these risks while accommodating the dynamic nature of modern healthcare systems.

In contemporary research, public-key cryptography-based mechanisms have been extensively employed for access control and secure data exchange applications [8]–[10]. Notably, Elliptic Curve Cryptography (ECC) has been adopted to ensure confidentiality preservation in data access control systems for Vehicular Ad-hoc Networks (VANETs) [10]. Parallel developments have integrated asymmetric encryption with edge computing architectures to facilitate cross-domain message transmission [11], [12]. Nevertheless, the pairwise authentication requirements inherent to conventional asymmetric encryption limit its scalability for cloud-based data sharing environments. To address this challenge, Proxy Re-Encryption (PRE) has garnered substantial attention as an alternative cryptographic primitive. PRE operates on the premise of a semi-trusted proxy entity that mediates ciphertext transformations, thereby enabling directional data exchange while minimizing communication costs [13], [14]. By decoupling the data owner from direct involvement in decryption key management, PRE offers a pragmatic solution for scalable and secure cloud data sharing, particularly in multi-tenant environments where dynamic access control and privacy preservation are paramount. Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [15] has emerged as a prominent cryptographic mechanism for implementing data access control, primarily due to its capacity to enable data owners to enforce policy-driven access rules directly through ciphertext attributes. However, CP-ABE implementations exhibit inherent limitations, notably their dependence on computationally intensive bilinear pairing operations and substantial ciphertext expansion

This work was supported in part by Chizhou University High level Talent Research Startup Fund under Grant CZ2022YJRC09 and in part by the National Natural Science Foundation of China under Grant 62072005. (Corresponding Authors: Gang Lv, Dun Li.)

Hongzhi Li, Gang Lv, Zheqing Zhang, and Qishou Xia are with the College of Big Data and Artificial Intelligence, Chizhou University, Chizhou 247000, China.

Dun Li is with the Department of Industrial Engineering, Tsinghua University, China, and Samovar, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France. E-mail:lidunshmtu@outlook.com.

Noel Crespi, Roberto Minerva are with Telecom SudParis, Institut Polytechnique de Paris, Palaiseau, 91120, France.

Wenhao Shao is with the Department of Computer Science and Technology, National University of Defense Technology, Hunan, 410073, China. Email: wenhao.shao@gxu.edu.cn.

during encryption. These constraints hinder their practical deployment in resource-constrained devices. As a complementary paradigm, lightweight attribute-based access control methodologies have recently been proposed to achieve secure data sharing in IoT-enabled systems [16]. These access control-based approaches prioritize computational efficiency and minimal resource utilization while maintaining stringent security guarantees. Role-Based Access Control (RBAC) [17] and Attribute-Based Access Control (ABAC) [18] remain predominant frameworks for managing system permissions; these two models exhibit critical limitations in modern distributed applications. First, the operational efficiency of RBAC-based or ABAC-based schemes essentially depends on semi-trusted third-party validators for authorization decisions, introducing potential single points of failure. Second, these schemes lack native mechanisms for immutable behavioral auditing, thereby impeding real-time detection and forensic analysis of malicious activities. Smart contracts [19] leverage the inherent transparency and cryptographic immutability of decentralized ledgers to autonomously execute access policies and provide tamper-evident audit trails.

Contemporary research in Electronic Medical Record (EMR) security primarily focuses on using cryptographic access controls to protect patient privacy by minimizing permissions. While cloud-based architectures enhance operational efficiency and offer some security through encrypted storage, they suffer from two major shortcomings: (1) There are insufficient mechanisms for end-to-end integrity verification of distributed Electronic Health Records (EHRs), making shared medical data vulnerable to undetected tampering; and (2) There is an over-reliance on centralized trust models, which are prone to single points of failure. Recent advancements in decentralized cryptography tackle these issues by introducing innovative privacy-preserving access control architectures. Blockchain-enhanced frameworks generally allow for the simultaneous realization of three crucial security features: (a) fine-grained attribute-based access governance, (b) provably secure multi-layer encryption, and (c) automated integrity auditing via Merkle-tree anchored cryptographic proofs. Following this trend, we propose a novel hybrid access control architecture that integrates the RBAC paradigm and smart contracts to achieve dynamic policy enforcement. To address the inherent trust vulnerabilities in conventional healthcare data management systems, this study introduces a smart contract-augmented access control framework that seamlessly integrates blockchain transparency, automated auditing, and patient-centric policy enforcement to achieve collusion-resistant, cryptographically verifiable data sharing. Unlike prior approaches that rely on centralized third-party auditors (TPAs) or opaque policy enforcement mechanisms, our scheme leverages Ethereum-compatible smart contracts [20] to enforce tamper-resistant access control and integrity verification, while empowering patients with fine-grained and autonomous access control over their own EMRs.

Specifically, we introduce the main contributions of this study as follows.

1) We propose an EMR access control framework powered by combining smart contracts and RBAC to enable fine-grained data access control.
2) We design a dynamic accumulator without bilinear mappings to achieve efficient identity information management.
3) We construct a novel proxy re-encryption algorithm to ensure the confidentiality of EMRs during data sharing.
4) We analyze the security properties in terms of formal and informal approaches and further evaluate the performance (i.e., computational and communication overhead) of this scheme by implementing a prototype.

The rest of this article is organized as follows. Section II is the related work. Section III describes the system model and design goals. Section IV presents the preliminary knowledge. Section V depicts the proposed protocol in detail. Section VI analyzes the security characteristics of this scheme. Section VII shows the implementation and performance evaluation, and Section VIII concludes this work.

II. Related work

A. Cryptographic Access Control

Cryptographic primitives like Ciphertext-Policy Attribute-Based Encryption (CP-ABE) provide a theoretical foundation for fine-grained access control. For instance, Saha et al. [23] employed 0-1 coding and outsourced decryption to adapt CP-ABE for IoT terminals. However, the core computational burden of bilinear pairing operations remains a critical bottleneck [21], [22], hindering deployment in resource-constrained IoMT environments. Similarly, while the fine-grained access control and matchmaking encryption (PBAC-FG) proposed by Sun et al. [24] enhances scalability, it introduces significant complexity in policy management. In contrast to these cryptographically heavy approaches, our scheme adopts a hybrid architecture. We leverage a lightweight RBAC model for efficient permission verification and delegate the computationally intensive task of data re-encryption to the cloud server via a novel PRE algorithm, thereby offloading the burden from end-user devices.

B. Smart Contract Integration

Blockchain and smart contracts introduce decentralization and transparency to access control. Guan et al. [25] and Peng et al. [26] utilized smart contracts to automate workflows. A key limitation of these architectures, however, is their continued reliance on semi-trusted intermediary servers for critical data processing or authorization decisions [26], which reintroduces central points of failure. Potluri et al. [27] employed smart contracts for dynamic policy enforcement, yet their cryptographic core still depends on computationally expensive bilinear pairings. Our work addresses this by designing a suite of smart contracts (IMC, RAC, EACC) that manage the entire authorization chain—identity verification, role assignment, and permission checking—in a fully decentralized manner, eliminating the need for semi-trusted third parties. Furthermore, our cryptographic constructions avoid bilinear pairings.

C. RBAC Model on Blockchain and Policy Privacy

Integrating the mature RBAC model with blockchain is another promising direction, as explored by Bian et al. [32]. A significant, often overlooked vulnerability in such transparent systems is the privacy leakage of access control policies themselves. Storing role-permission mappings directly on-chain can expose sensitive information about medical institutional structures. To mitigate this, our scheme decouples policy storage from the blockchain. Detailed access policies are stored off-chain in IPFS, with only their integrity-assuring hash fingerprints anchored on the blockchain. This approach preserves the immutability and verifiability of policies without disclosing their sensitive content.

D. Assurance of Data Confidentiality and Integrity

Ensuring both the confidentiality and integrity of EMRs during sharing is paramount. While several schemes [26], [29], [31] employ Proxy Re-Encryption (PRE) to safeguard confidentiality, they often lack a robust, independently verifiable mechanism for end-to-end data integrity. The cloud server, acting as the PRE proxy, could potentially tamper with stored ciphertexts without detection. A distinct contribution of our work is the tight integration of confidentiality with provable integrity. We anchor the hash of the encrypted EMRs directly onto the blockchain during the data encryption phase. This provides an immutable and independently verifiable proof, allowing any data requester to detect tampering without relying on the cloud server or the access control logic.

In summary, our proposed hybrid RBAC-Smart Contract scheme is designed to address these limitations collectively. We provide a balanced and practical solution for secure and efficient EMR access control in IoMT through the collaborative use of bilinear unpaired dynamic accumulators for efficient identity management, a fully decentralized smart contract authorization suite, an off-chain policy storage mechanism for privacy, and a blockchain-anchored integrity verification mechanism.

III. System Model

A. System Architecture

As shown in Fig. 1, the system architecture comprises three critical components: EMR providers, a blockchain-enabled access control module, and EMR requesters. Generally, patient-oriented healthcare devices serve as primary data sources, generating and transmitting personal medical records. These key components are introduced as follows.

1) Patients: As the exclusive owners of Electronic Medical Records (EMRs), patients have control over their health data. They can exercise fine-grained permissions through smart contracts to manage the access and sharing of their data.
2) Healthcare Devices: The healthcare devices utilized are typically wearable and portable smart devices designed to monitor and collect patients' physiological information.

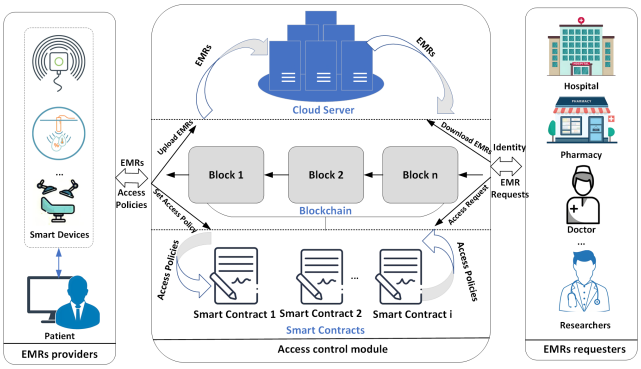


Fig. 1. System Architecture.

3) Hospitals: Hospitals serve as the primary providers of healthcare services, including both government-operated and private medical institutions.
4) Cloud Server: Cloud servers are used to store and manage encrypted Electronic Medical Records (EMRs) from medical institutions. These can be divided into private clouds maintained by medical institutions and public clouds managed by government agencies.
5) Smart Contract: We construct an EMR access control framework based on Role-Based Access Control (RBAC) using smart contracts. This framework implements a decentralized data access control mechanism.

In this study, the requesters of Electronic Medical Records (EMRs) include patients and authorized medical institutions, which primarily encompass the following roles:

1) Healthcare Institutions: These institutions often require access to EMRs for the same patient from different institutions. Consequently, they need to request EMRs from other collaborating healthcare facilities.
2) Doctor & Pharmacists: Doctors and pharmacists typically need access to patients' diagnosis and treatment records to develop appropriate treatment solutions.
3) Researchers: Researchers from medical research institutions need to acquire a substantial number of patient diagnosis and treatment records (EMRs) to facilitate the development of new drugs and conduct pathological research.

To address these threats, the design goals are formulated to ensure robust security and performance, as detailed in the following subsection.

B. Threat Model

In this study, we primarily address cyber-attack threats originating from both internal and external adversaries. First, cloud servers operate under an honest but curious (HBC) adversarial model. While they adhere to service agreements, they may attempt to infer private information. For data integrity verification, these servers could intentionally provide false proofs to hide instances of data corruption or loss. Second, if the access control policy implemented in a smart contract contains sensitive information, adversaries may exploit this information to compromise user privacy, which poses a threat

to system security. Finally, external cyber threats need to be effectively addressed. Malicious adversaries may intercept communications to reveal confidential information. Additionally, data requesters could forge authentication credentials to bypass access control mechanisms.

C. Design Goals

In this study, we propose a novel access control scheme integrated with data integrity checking to facilitate secure and reliable data sharing. The key design goals of this scheme are outlined as follows.

1) This scheme should ensure that unauthorized requesters cannot obtain any user privacy from the stored EMRs.
2) This scheme should allow authorized users to verify data integrity without accessing EMR plaintext.
3) This scheme should ensure that access control policies can resist unauthorized modifications from adversaries.
4) This scheme should ensure that all access activities are recorded and audited immutably through blockchain.
5) The proposed scheme should maintain acceptable computational and communication overhead.

IV. Preliminaries

A. Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is a commonly used model for managing permissions that regulates access to system resources by assigning users to specific roles. This hierarchical approach allows for efficient and scalable management of access permissions. As illustrated in Table I, the RBAC model consists of the following components.

TABLE I
DESCRIPTIONS OF RBAC

Element	Description
User	System users (e.g., employees, administrators).
Role	Collection of permissions (e.g., administrator, finance).
Permission	Operations on resources (e.g., "read file", "delete order").
Session	Session determines the currently available role.

B. Bilinear Mapping

Assume q is a large prime, $\mathbb{G}_1$, $\mathbb{G}_2$ are the two cyclic groups of order q. The bilinear mapping about $\mathbb{G}_1$ and $\mathbb{G}_2$ can be defined as $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. $\hat{e}$ has the following properties:

1) Bilinear: Assume $P, Q \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_q^*$, it has $\hat{e}\left(P^a, Q^b\right) = \hat{e}(P, Q)^{ab}$.
2) Non-degeneracy: $\hat{e}$ can not map all $P, Q \in \mathbb{G}_1$ to the unit element in $\mathbb{G}_2$, i.e., there exists $\hat{e}(P, Q) \neq 1$.
3) Computability: For any element $P, Q \in \mathbb{G}_1$, there exists a computable $\hat{e}(P, Q)$.

C. Proxy Re-encryption (PRE)

Proxy Re-Encryption (PRE) is a widely recognized public key encryption algorithm designed for data sharing. PRE enables third-party servers to transform ciphertext into another ciphertext that is encrypted with the target public key, all without accessing the plaintext. Typically, PRE involves the following key steps.

1) The data owner *Alice* uses her secret key K to encrypt the record M as $CT_{(M)}$ and stores $CT_{(M)}$ in proxy servers.
2) The data requester *Bob* sends its public key to *Alice*, and *Alice* utilizes its private key and the received public key to generate the re-encryption key $ReKey$.
3) The proxy server encrypts $CT_{(M)}$ as $CT'_{(M)}$ by using $ReKey$ and sends $CT'_{(M)}$ to *Bob*.
4) The data requester *Bob* decrypts $CT'_{(M)}$ using his private key, allowing him to access the original record M.

D. BLS Signature

BLS (Boneh-Lynn-Shacham) is an efficient digital signature algorithm commonly used in resource-constrained systems. The signature process involves the following functions.

- KeyGen: Select $s \in \mathbb{Z}_q^*$ as the private key and generate the corresponding public key as $pk = g_1^s$, where g_1 is a generator of $\mathbb{G}_1$.
- Sign: Compute the hash of message M as $h = H(M)$ and generate the signature as $\sigma = g_1^{h \cdot s} = g_1^{H(M) \cdot s}$.
- Verify: Check the received signature σ as $\hat{e}(\sigma, g_1) = \hat{e}\left(g_1^{H(M) \cdot s}, g_1\right) \stackrel{?}{=} \hat{e}\left(g_1^{H(M)}, pk\right)$.

E. Dynamic Accumulator

The RSA accumulator is a universal cryptographic tool that generates commitments for dynamically changing sets through modular exponentiation. The process for constructing an RSA accumulator is as follows:

1) Parameter Initialization: Generate an RSA modulus $N = pq$ where p and q are large primes. Then, select a random generator $g \in \mathbb{G}$.
2) Set Accumulation: For a set $L = \{x_1, \ldots, x_n\}$ that contains prime elements, compute the accumulator value as $c = g^{\sum_{i=1}^{n} x_i} \mod N$.
3) Witness Generation: For any element $x_i \in L$, generate a witness $w_i = g^{\sum_{j \neq i} x_j} \mod N$. This witness allows verification using the equation $w_i^{x_i} \equiv c \mod N$.
4) Adding an Element: When introducing a new prime x', update the accumulator to obtain the new value as $c_{\text{new}} = c \cdot g^{x'} \mod N$.

V. The Proposed Scheme

This section describes the scheme from four critical processes: *system initialization*, *identity management*, *access control framework*, and *EMRs sharing*. The frequently used notations are listed in Table II.

A. System Overview

Fig. 2 illustrates the complete process of generating and securely sharing Electronic Medical Records (EMRs), which can be divided into five key parts.

1) Patients generate health data during medical encounters, such as consultations, treatments, and hospitalizations.

TABLE II
KEY NOTATIONS

Notation	Description
E_i	Entity i
TA	Trusted authority
$Pubk_{(E_i)}$	The public key of E_i
IMC	Identity Manage Contract
RMC	Roles Management Contract
$RPSC$	Role Permission Setting Contract
$EACC$	EMR Access Control Contract
$EDSC$	EMR Decryption Smart Contract
SK	Session key
$CT_{(x)}$	Ciphertext for x
$UA_{(E_i)}$	User attributes of E_i
EMR	Electronic medical record
$SysID_i$	Pseudonym of E_i
PRE	Proxy ReEncryption
Fid	The file identity
AES	Advanced Encryption Standard
TS	Timestamp
$\mathcal{PY}_{(\mathcal{R})}$	Access policy
ΔT	Time threshold

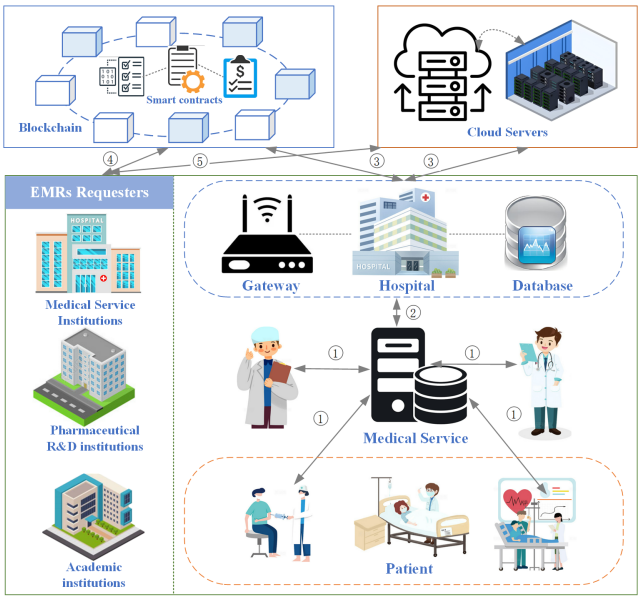


Fig. 2. EMRs generation and sharing process.

Medical staff input this data into the medical service system, which serves as the source for creating EMRs (marked ①).

2) The data owner encrypts the generated EMRs and stores them in the hospital database through the hospital's internal gateway, thereby completing the in-hospital data collection process (marked ②).
3) The encrypted EMRs are then transmitted to cloud servers. Along with this, the data owners submit corresponding data fingerprints and access control policies to the blockchain using smart contracts (marked ③).
4) EMR requesters, such as medical service institutions, pharmaceutical R&D institutions, and academic institutions, construct data sharing requests and submit them to the blockchain. This process relies on smart contracts to enable efficient data access control based on RBAC (marked ④).
5) Cloud servers perform the proxy re-encryption algorithm based on the permission credentials to facilitate secure sharing of EMRs (marked ⑤).

B. System Initialization

The system initialization is carried out by the system initiator TA, and the initialization procedure is described as follows.

1) TA randomly selects a large prime number $q \in Z_q^*$ and constructs two cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$ with the order q.
2) TA establishes a bilinear mapping as $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.
3) TA selects g_1 as a generator of $\mathbb{G}_1$ and chooses a hash function $H : \{0,1\}^* \times Z_q^* \to \{0,1\}^*$.
4) TA selects the BLS signature algorithm, denoted as $\mathrm{Sig}(\cdot)$, and adopts AES as the symmetric encryption method, denoted as $\mathrm{Enc}(\cdot)$.
5) TA randomly chooses $x \in Z_q^*$ as the system private key ssk and further computes the system public key as $spk = g_1^x$.
6) Finally, TA initializes the accumulator as $\triangle_a = g_1^{\lambda_0}$, where $\lambda_0 \in Z_q^*$ is a randomly selected number.

$PM = \{\mathbb{G}_1, \mathbb{G}_2, q, g_1, \hat{e}, \triangle_a, H(\cdot), \langle Enc(\cdot), Dec(\cdot)\rangle, \mathrm{Sig}(\cdot)\}$ will be finally published by TA as the public parameters.

C. Identity Management

1) Identity Registration: To enhance the reliability of systems, all involved parties should register their identities before sharing EMRs. Therefore, we incorporate smart contracts and cryptographic operations to manage identity information. To be specific, we first outline the identity registration process as follows.

1) E_i with identity $ID_i \in \{0,1\}^*$ randomly selects $\tau_i \in Z_q^*$ as its private key $Prik_{(E_i)}$ and further generates the public key as $Pubk_{(E_i)} = g_1^{\tau_i}$.
2) E_i generates a session key SK by running the Diffie-Hellman protocol with TA and further encrypts ID_i as $Enc(ID_i||Pubk_{(E_i)}, SK) \to CT_{(ID_i)}$.
3) TA decrypts the $CT_{(ID_i)}$ as $Dec(CT_{(ID_i)}, SK) \to \{ID_i||Pubk_{(E_i)}\}$ and further generates the pseudonym as follows:

$$SysID_i = g_1^{ssk} \oplus ID_i. \tag{1}$$

4) TA invokes the smart contract *Identity Manage Contract* (IMC) (i.e., Alg. 1) to update the dynamic accumulator $\triangle_a$ in the immutable ledger issueLedger. Subsequently, TA obtains the dynamic accumulator $\triangle_a$ and the corresponding witness pair as $\langle \triangle_a, W_i \rangle$ (lines 3-10).
5) TA generates the unique registration credential as:

$$Proof_{(ID_i)} = \mathrm{Sig}(H(SysID_i||Pubk_{(E_i)}), ssk). \tag{2}$$

6) TA sends $\langle SysID_i, Proof_{(ID_i)}, \triangle_a, W_i, TS \rangle$ to E_i through a secure channel.

Algorithm 1: Identity Manage Contract (IMC)

Input: $Pubk_{(E_i)}$, $SysID_i$, $\triangle_a$.
Output: $result$.

1 mapping (string ⇒ string) issueLedger;
2 `Identity registration:`
3 Require(msg.sender==TA);
`// Only TA has this permission.`
4 $H(Pubk_{(E_i)}||SysID_i) \rightarrow \lambda_i$;
5 $\triangle_a \cdot g_1^{\lambda_i} = g_1^{\sum_{k=0}^{i} \lambda_k} \rightarrow \triangle_a$;
6 $issueLedger.push(TS \Rightarrow \triangle_a)$;
7 $(\triangle_a / g_1^{\lambda_i}) \rightarrow W_i$;
8 $result \leftarrow (\triangle_a, W_i)$;
9 **return** $result$;
10 `Identity verification:`
11 $issueLedger.getTop() \rightarrow \triangle'_a$;
`// Get the latest accumulator value.`
12 $result \leftarrow \triangle'_a$;
13 **return** $result$;
14 `Identity revocation:`
15 Require(msg.sender==TA);
`// Only TA has this permission.`
16 $issueLedger.getTop() \rightarrow \triangle'_a$;
17 $H(Pubk_{(E_i)}||SysID_i) \rightarrow \lambda_i$;
18 $(\triangle'_a / g_1^{\lambda_i}) \rightarrow \triangle_a$;
19 **for** W_k *in* $(W_1, W_2, \cdots, W_n)$ **do**
20 **if** $k! = i$ **then**
21 $W_k^{new} = W_k / g_1^{\lambda_i}$;
22 $issueLedger.push(TS \Rightarrow \triangle_a)$;
23 $result \leftarrow (W_1^{new}, W_2^{new}, \cdots, W_n^{new})$;
24 **return** $result$;

2) Identity Verification: In the process of mutual communication, it is inevitable to verify the identity of relevant entities for security reasons. E_j executes the following identity verification process to confirm the legitimacy of E_i using the registration credential $Proof_{(ID_i)}$.

1) E_j verifies $Proof_{(ID_i)}$ by performing the method as:

$$Check_{(ID_i)} = \text{Verify}(spk, H(SysID_i||Pubk_{(E_i)}), Proof_{(ID_i)}) \quad (3)$$

2) If $\text{Check}_{(\text{ID}_\text{i})}$ is false, it indicates that E_i has not been authenticated by TA. Otherwise, E_j further invokes IMC (i.e., Alg. 1) to obtain the current accumulator $\triangle'_a$ (lines 11-13).
3) E_j verifies whether the registration information is valid by using the stored dynamic accumulator $\triangle'_a = g_1^{\sum_{k=0}^{n} \lambda_k}$ and the witness $\langle \triangle_a, W_i \rangle$. To be specific, E_j performs the following operations.
   a) Computes $(\triangle'_a / \triangle_a = g_1^{\sum_{k=i+1}^{n} \lambda_k}) \rightarrow \rho_i$.
   b) Computes $W_i \cdot \rho_i = g_1^{\sum_{k=0, k\neq i}^{n} \lambda_k} \rightarrow \rho_i^*$.
   c) Computes $H(Pubk_{(E_i)}||SysID_i) \rightarrow \lambda_i$.
   d) Verifies $g_1^{\lambda_i} \cdot \rho_i^* \stackrel{?}{=} \triangle'_a$.

Due to $g_1^{\lambda_i} \cdot \rho_i^* = g_1^{\lambda_i} \cdot g_1^{\sum_{k=0, k\neq i}^{n} \lambda_k}$ and $\triangle'_a = g_1^{\sum_{k=0}^{n} \lambda_k}$, if $g_1^{\lambda_i} \cdot \rho_i^* = \triangle'_a$, it indicates that E_i is currently a legitimate entity.

D. Roles Assignment

E_i invokes the procedure AssignRole(·) with its attributes $\text{UA}_{(\text{E}_\text{i})}$ to obtain its system role. The specific procedure is illustrated as follows.

AssignRole$(UA_{(E_i)}, Proof_{ID_i}) \rightarrow Role_{(E_i)}$: This method first verifies $Proof_{(ID_i)}$ by performing BLS signature verification as $\hat{e}(Proof_{(ID_i)}, g_1) \stackrel{?}{=} \hat{e}\left(g_1^{H(SysID_i||Pubk_{(E_i)})}, spk\right)$. After this verification, *Roles Assignment Contract* (RAC, i.e., Alg. 2) will be activated to allocate the role to E_i according to the predefined role assignment policy $\mathcal{PY}_{(\mathcal{R})}$.

$$\mathcal{PY}_{(\mathcal{R})} = \{Pid, Attris : \{k : v\}_{(1\cdots n)}, Role : \langle d, p, r \rangle\}, \quad (4)$$

where $Attris \in \mathcal{PY}_{(\mathcal{R})}$ represents the attributes of system roles, $Role$ is the corresponding system roles (e.g., doctors (d), pharmacists (p), and researchers (r)). In addition, $Pid \in \mathcal{PY}_{(\mathcal{R})}$ is the unique storage credential for $\mathcal{PY}_{(\mathcal{R})}$ in IPFS (InterPlanetary File System). An example of $\mathcal{PY}_{(\mathcal{R})}$ is illustrated in Fig. 3.

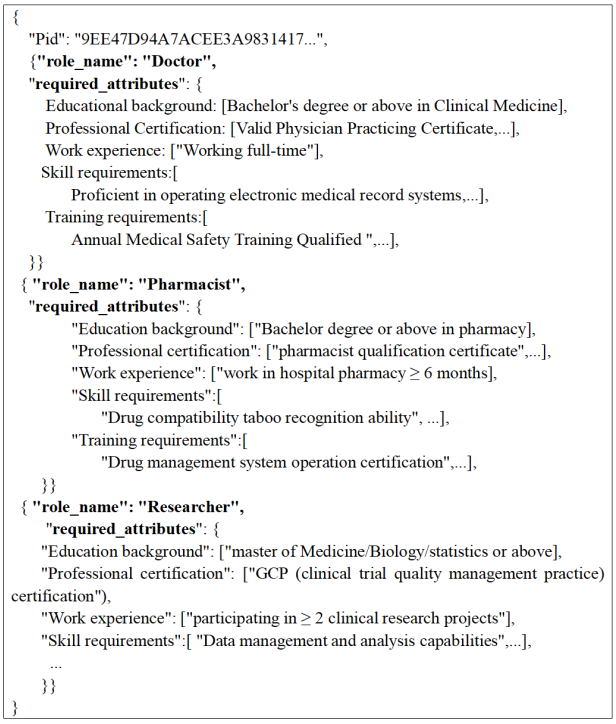


Fig. 3. An example of role assignment policies.

As shown in Alg. 2, RAC takes Pid and $UA_{(E_i)}$ as input parameters and defines a blockchain ledger RolesLedger to store the mapping relationship between entities and roles. Additionally, an IPFS client is defined to obtain the stored $Policy$ according to the input Pid (lines 3-4). If the policy is not found, it indicates that the required policy is not stored in IPFS. Otherwise, the obtained attributes $Policy.Attris$ will be used to search the matching attributes in $UA_{(E_i)}$ (lines 7-9). Finally, RAC determines whether E_i meets the obtained

$Policy$ by counting the number of matching attributes between $Policy.Attris$ and $UA_{(E_i)}$ (lines 10-13).

Algorithm 2: Roles Assignment Contract (RAC)

Input: Pid, $UA_{(E_i)}$, $Pubk_{(E_i)}$.
Output: $result$.

1 $count = 0$, $Policy = null$,
2 mapping(string ⇒ string) RolesLedger;
3 $client = ipfs\_httpclient.connect()$;
`// Connect to IPFS.`
4 $Policy = client.cat(Pid)$;
`// Download the target policy.`
5 **if** $Policy == null$ **then**
6 **return** $null$;
7 **for** $(key \Rightarrow attris)$ *in* $Policy.\{Attris\}$ **do**
8 **if** $UA_{(E_i)}$ *matches* $attris$ **then**
9 $count = count + 1$;
10 **if** $count == UA_{(E_i)}.size()$ **then**
11 $H(Pubk_{(E_i)}) \rightarrow key_i$;
12 $RolesLedger.push(key_i \Rightarrow Policy.Role)$;
13 **return** Policy.Role;
14 **return** $null$;

E. Access Control Framework

In this section, we elaborate this access control framework from three aspects: *permissions setting*, *permission granting*, and *permissions revocation*.

1) Permission Setting: To achieve fine-grained data access control, EMRs are structured as JSON tuples that include medical testing reports, diagnosis reports, and personal information. Here, we define EMRs as:

$$\mathcal{MR} = \{Fid, P, M, D\}. \tag{5}$$

As in (5), P represents the personal information of patients, M is the medical testing report, and D refers to the diagnosis report from doctors. To enable patients to manage their EMR access permissions, *Role Permission Setting Contract* (RPSC) is designed to allow EMR owners to set specific access permissions. Specifically, the corresponding workflow of access permission setting is described as follows.

1) E_i (data owner) constructs a permission setting request as:

$$\mathcal{PR} = \{Fid, Perm : \langle Role \Rightarrow (P, M, D)\rangle, Eid\}, \tag{6}$$

where $Fid \in \mathcal{PR}$ is the unique identity of $\mathcal{MR}$, $Perm$ is the corresponding permissions of the given $Role$. Also, $Eid = \langle Pubk_{(E_i)}, Sig_i\rangle$ is introduced to identify the request sender, containing the corresponding $Pubk_{(E_i)}$ and Sig_i.
2) E_i sends the generated $\mathcal{PR}$ to *Role Permission Setting Contract* (RPSC, Alg. 3), and IMC will be invoked to verify the legitimacy of E_i. After that, RPSC verifies the contained signature as $\text{Verify}(Pubk_{(E_i)}, Sig_i, H(\mathcal{PR})) \stackrel{?}{=} \text{True}$ (lines 4-7).
3) After completing the above verifications, RPSC will be used to set the access permission for the given $\mathcal{MR}$. RPSC first verifies the existence of $\mathcal{MR}$ by querying IPFS (lines 8-10). Afterward, $Perm$ will be iterated to construct *FilePermission* object and initialize a role-permission mapping EMRsRoleAccess (lines 11-20).

Algorithm 3: Role Permission Setting (RPSC)

Input: Fid, $Perm$, $Eid = \langle Pubk_{(E_i)}, Sig_i\rangle$.
Output: True or False.

1 **Struct** FilePermission {uint Personal; uint Medical; uint Diagnosis; }
2 mapping(string ⇒ FilePermission) RolePermission;
3 mapping(string ⇒ mapping) EMRsRoleAccess;
4 `Call the contract IMC:` $RPSC \leftarrow \triangle_a$;
5 `Perform identity verification using` $\triangle_a$`:` $RPSC \leftarrow Check_{ID_i}$;
6 **if** $!Check_{(ID_i)}$ **then**
7 **return** False;
`// Call IMC to realize identity authentication.`
8 $client = ipfs\_httpclient.connect()$;
9 **if** $!client.exist(Fid)$ **then**
10 **return** False;
11 **for** *(key ⇒ perm) in* $Perm$ **do**
12 FM=$FilePermission(perm.P, perm.M, perm.D)$;
13 $RolePermission.add(perm.Role \Rightarrow FM)$;
14 **if** $EMRsRoleAccess.get(Fid) == null$ **then**
15 $EMRsRoleAccess.add(Fid \Rightarrow RolePermission)$;
16 **else**
17 $EMRsRoleAccess.get(Fid) \rightarrow OriginPerm$;
18 **for** *(key ⇒ perm) in* $Perm$ **do**
19 $OriginPerm[key] = perm$;
20 $EMRsRoleAccess.update(Fid \Rightarrow OriginPerm)$;
21 **return** True;

2) Permission Granting: When E_i attempts to access an EMR $\mathcal{MR}_i$ with Fid, a data access request $\mathcal{AR}$ should be constructed as:

$$\mathcal{AR} = \{Fid, Pubk_{(E_i)}, Role_{(E_i)}, TS, Sig_i\}. \tag{7}$$

As mentioned in (7), Fid represents the unique identifier of the requested $\mathcal{MR}_i$. $Pubk_{(E_i)}$ and $Role_{(E_i)}$ are E_i's public key and role, respectively. In addition, Sig_i generated by E_i is used to ensure the integrity of $\mathcal{AR}$. The constructed $\mathcal{AR}$ is submitted to the smart contract *EMRs Access Control Contract* (EACC, Alg. 4), and EACC further cooperates with the aforementioned IMC and RPSC to realize the EMRs access control as follows.

1) EACC verifies the received $\mathcal{AR}$ by checking the contained signature Sig_i. If the signature verification fails, it indicates that $\mathcal{AR}$ is invalid; this procedure is terminated (lines 2-3).

2) EACC then invokes the contract IMC to verify the legitimacy of E_i by verifying the dynamic accumulator $\triangle_a$. If E_i is invalid, this procedure is terminated (lines 4-7).
3) EACC retrieves the role associated with E_i from the deployed Role Access Contract (RAC) as $Role_i^*$. It then compares the declared role $Role_i \in \mathcal{AR}$ with $Role_i^*$ to verify the authenticity of $Role_i \in \mathcal{AR}$ (lines 8-11).
4) EACC invokes RPSC with the given Fid to query the ledger EMRsRoleAccess and generates the corresponding permission proof $Proof_{(access)}$ (lines 12-16).

Algorithm 4: EMRs Access Control Contract (EACC)

Input: $Pubk_{(E_i)}$, Fid, $Role_i$, Sig_i.
Output: $Proof_{(access)}$.

1 mapping(string $\Rightarrow$ boolean) GrantLedger;
2 **if** !Verify$(Pubk_{(E_i)}, Sig_i, H(Fid||Role_i))$ **then**
3 **return** null;
4 `Call the contract IMC:` $EACC \leftarrow \triangle_a$;
5 `Perform identity verification with` $\triangle_a$`:` $EACC \leftarrow Check_{(ID_i)}$;
6 **if** $!Check_{(ID_i)}$ **then**
7 **return** null;
`// Call IMC to realize identity authentication.`
8 `Call RAC:` $EACC \leftarrow RolesLedger$;
9 $RolesLedger.get(H(Pubk_{(E_i)})) \rightarrow Role_i^*$;
10 **if** $Role_i^* != Role_i$ **then**
11 **return** null;
12 `Call RPSC:` $EACC \leftarrow EMRsRoleAccess$;
13 $EMRsRoleAccess.get(Fid) \rightarrow RolePermission$;
14 $RolePermission.get(Role_i^*) \rightarrow FilePerm$;
15 $\{FilePerm, Fid, Pubk_{(E_i)}, Expire, TS\} \rightarrow Proof_{(access)}$;
16 $GrantLedger.add(H(Proof_{(access)}) \Rightarrow True)$;
17 **return** $Proof_{(access)}$;

3) Permission Revocation: In practice, the permission revocation functionality enables a timely response to abnormal account situations by revoking permissions assigned to specific roles. For instance, when a doctor E_i is transferred from an administrative position to a regular position, their administrative privileges need to be revoked. This process can be illustrated through three cases of permission revocation: revocation of access proofs, revocation of roles, and revocation of identities. Here are the details of each case:

1) If E_j attempts to revoke the access credentials of E_i to a given EMR, E_j invokes the predefined blockchain ledger GrantLedger and reset the access credentials as $\mathrm{GrantLedger.update(H(Proof_{(access)}) \Rightarrow False)}$.
2) If E_i misuses role-based permissions to interfere with or damage EMRs, TA invokes the $RolesLedger$ on the blockchain and revokes the corresponding role assignment. Thus, E_i is no longer authorized to access any EMRs.
3) If E_i is found to be a malicious entity within the system, TA should revoke the identity of E_i by invoking the *Identity revocation* subroutine in IMC, and further performing the following operations.
   a) Call the deployed contract IMC and obtain the current dynamic accumulator $\triangle_a'$.
   b) Compute $H(Pubk_{(E_i)}||SysID_i) \rightarrow \lambda_i$.
   c) Compute $\triangle_a'/g_1^{\lambda_i} \rightarrow \triangle_a$ and update the dynamic accumulator as $\mathrm{issueLedger.push(TS \Rightarrow \triangle_a)}$.
   d) Update each witness as $W_k^{new} = W_k/g_1^{\lambda_i} = g_1^{\sum_{j=0,j\neq k,i}^{n} \lambda_j}$.

Through the combined use of the deployed smart contracts (i.e., EACC, RAC, and IMC), we can achieve the revocation of user permissions.

F. EMRs Sharing

To clarify the process of electronic medical record (EMR) sharing, we use the example of EMR sharing between entities E_i and E_j to illustrate the functionality of our proposed proxy re-encryption protocol. This process consists of five key parts: *EMRs Encryption*, *Permission Verification*, *EMRs ReEncryption*, and *EMRs Decryption*.

1) EMRs Encryption: The data owner E_i encrypts its own $\mathcal{MR}_i$ using its public key as follows.

1) E_i selects $y \in (0, q)$ and computes $C_1 = g_1^y$.
2) E_i computes the corresponding ciphertext of $\mathcal{MR}_i \in \{0,1\}^*$ as $C_2 = Pubk_{(E_i)}^y \oplus \mathcal{MR}_i = (g_1^{\tau_i})^y \oplus \mathcal{MR}_i$.
3) E_i computes the hash of C_2 as $H(C_2) \rightarrow hcode_{(\mathcal{MR}_i)}$ and stores the calculated $hcode_{(\mathcal{MR}_i)}$ in blockchain, obtaining the blockchain transaction id $Tid_{(\mathcal{MR}_i)}$.
4) E_i constructs $C_{\mathcal{MR}_i} = \{C_1, C_2, Tid_{(\mathcal{MR}_i)}, Fid_{(\mathcal{MR}_i)}\}$ as the ciphertext, where $H(C_1||C_2||Tid_{(\mathcal{MR}_i)}) \rightarrow Fid_{(\mathcal{MR}_i)}$. Subsequently, $C_{\mathcal{MR}_i}$ will be submitted and stored in the cloud server.

2) Permission Verification: Before sharing $\mathcal{MR}_i$, E_i must verify E_j's access permissions. The data requester E_j must first obtain the access permission credential $Proof_{(access)}$ from the EMRs Access Control Contract (EACC) as described in Section V-E.2. Specifically, E_j submits a valid access request $\mathcal{AR}$ to EACC, which, upon successful verification of E_j's role and permissions, issues $Proof_{(access)}$ as a tamper-resistant credential. This credential is cryptographically signed by EACC and recorded on the blockchain for auditability. Once in possession of $Proof_{(access)}$, E_j presents it to the data owner E_i during the permission verification phase to demonstrate its eligibility for accessing the requested EMR. This step ensures that E_i can independently verify E_j's access permissions without relying on a centralized authority. The detailed verification process is as follows.

1) E_j sends its access permission credential $Proof_{(access)}$ to entity E_i as $E_j \rightarrow E_i : \{Proof_{(access)}, TS, Sig_{(pf)}\}$. After receving the credential, E_i checks if $Proof_{(access)}$ has expired by comparing $Expire \in Proof_{(access)}$ with the current timestamp TS_{now}.
2) Since E_j generates the signature for $Proof_{(access)}$ using its private key, E_i verifies the received signature

as $\mathrm{Verify}(\mathrm{Sig}_{(\mathrm{pf})}, \mathrm{Proof}_{(\mathrm{access})}.\{\mathrm{Pubk}_{(\mathrm{E_j})}\}) \stackrel{?}{=} \mathrm{True}$ to check whether E_j is the owner of $Proof_{(access)}$.

3) E_i verifies the authenticity of $Proof_{(access)}$ by querying the decentralized ledger GrantLedger in EACC as $\mathrm{GrantLedger.get(H(Proof_{(access)}))} \stackrel{?}{=} \mathrm{True}$.
4) If all the verifications of 1), 2), and 3) pass, it indicates that E_j has the access permission to $\mathcal{MR}_i$. Otherwise, the sharing process should be terminated immediately.

Finally, E_i invokes the smart contract IMC and performs the *Identity verification* subroutine to check the registered identity of E_j as $E_i \rightarrow IMC : \{Pubk_{(E_j)}, TS\}$. After completing the above access permission verification of E_j, E_i performs the subsequent re-encryption algorithm.

3) EMRs ReEncryption: The data owner E_i collaborates with the cloud server (i.e., proxy server) to re-encrypt the requested $\mathcal{MR}_i$ as follows.

1) E_j sends its public key $Pubk_{(E_j)} = g_1^{\tau_j}$ to E_i, E_i then generates the re-encryption key as $ReKey_{i\rightarrow j} = Pubk_{(E_j)}^{\tau_i} = g_1^{\tau_i \cdot \tau_j}$.
2) E_i obtains C_1 from the proxy server and computes $C_1' = C_1^{\tau_i}$.
3) E_i computes $C_3 = C_1' \cdot ReKey_{i\rightarrow j} = g_1^{y\cdot\tau_i} \cdot g_1^{\tau_i\cdot\tau_j}$ and further sends C_3 to cloud servers.

4) Integrity Verification: E_j performs integrity verification on the received $C'_{\mathcal{MR}_i}$ as the following steps.

1) E_j obtains the ciphertext $C'_{\mathcal{MR}_i} = \{C_2, C_3, Tid_{(\mathcal{MR}_i)}\}$ from cloud servers.
2) E_j queries the hash value $hcode^*_{(\mathcal{MR}_i)}$ from blockchain using $Tid_{(\mathcal{MR}_i)} \in C'_{\mathcal{MR}_i}$.
3) E_j computes the hash of $C_2 \in C'_{\mathcal{MR}_i}$ as $H(C_2) \rightarrow hcode'_{(\mathcal{MR}_i)}$.
4) E_j checks whether $hcode'_{(\mathcal{MR}_i)} \stackrel{?}{=} hcode^*_{(\mathcal{MR}_i)}$. If $hcode'_{(\mathcal{MR}_i)} != hcode^*_{(\mathcal{MR}_i)}$, $C'_{\mathcal{MR}_i}$ should be discarded, and the data sharing procedure will be terminated.

5) EMRs Decryption: After validating $C'_{\mathcal{MR}_i}$, E_j sends it along with $Proof_{(access)}$ to *EMR Decryption Smart Contract* (EDSC, Alg. 5), EDSC then decrypts $C'_{\mathcal{MR}_i}$ as follows.

1) EDSC computes the partial decryption parameter as $\rho_{(i\rightarrow j)} = (Pubk_{(E_i)})^{Prik_{(E_j)}} = g_1^{\tau_i\cdot\tau_j}$ and further computes $\rho'_{(i\rightarrow j)} = C_3/\rho_{(i\rightarrow j)} = g_1^{y\cdot\tau_i}$.
2) EDSC obtains the target $\mathcal{MR}_i$ by computing $\rho'_{(i\rightarrow j)} \oplus C_2 = g_1^{y\cdot\tau_i} \oplus (g_1^{\tau_i})^y \oplus \mathcal{MR}_i = \mathcal{MR}_i$.
3) EDSC extracts the access permission for the obtained $\mathcal{MR}_i$ as $(Proof_{(access)}.FilePerm) \rightarrow FilePerm_{(E_j)}$.
4) If $(FilePerm.Personal == 0)$ then EDSC performs $H(\mathcal{MR}_i.P) \rightarrow P_h$. Similarly, EDSC verifies other fields of $\mathcal{MR}_i$, and further performs hash blinding on the relevant data fields (i.e., Medical, Diagnosis), achieving fine-grained EMRs sharing.

Finally, the processed $\mathcal{MR}_i$ using segmented hashing will be transmitted to E_j. The detailed data decryption process is summarized in Alg. 5.

Algorithm 5: EMR Decryption Contract (EDSC)

Input: $C'_{\mathcal{MR}_i}$, $Proof_{(access)}$, $Sig_{(pf)}$.
Output: $\mathcal{MR}_i$

if $!GrantLedger.get(H(Proof_{(access)})$ **then**
  return null;
if $!verify(Sig_{(pf)}, Proof_{(access)}.\{Pubk\})$ **then**
  return null; // $Proof_{(access)}$ does not belong to E_j.
$\rho_{(i\rightarrow j)} = (Pubk_{(E_i)})^{Prik_{(E_j)}} = g_1^{\tau_i\cdot\tau_j}$;
$\rho'_{(i\rightarrow j)} = \frac{C_3}{\rho_{(i\rightarrow j)}} = g_1^{y\cdot\tau_i}$;
$\rho'_{(i\rightarrow j)} \oplus C_2 = g_1^{y\cdot\tau_i} \oplus (g_1^{\tau_i})^y \oplus \mathcal{MR}_i = \mathcal{MR}_i$;
$Proof_{(access)}.FilePerm \rightarrow FilePerm_{(E_j)}$;
// Perform hash blinding on relevant fields based on the corresponding access permissions.
for *key* ***in*** $['Personal', 'Medical', 'Diagnosis']$ **do**
  if $FilePerm_{(E_j)}.key == 0$ **then**
    $H(\mathcal{MR}_i.key) \rightarrow \mathcal{MR}_i.key$;
return $\mathcal{MR}_i$;

VI. Security Analysis

In this section, we analyze the security of the proposed scheme to demonstrate that it meets the aforementioned security requirements. Additionally, we provide a formal security proof using BAN-logic [21] and the ProVerif tool [28].

A. Informal Security Analysis

We informally demonstrate that this scheme can effectively resist various attacks described in Section III-B. Furthermore, we conduct an informal security analysis to show that the proposed protocol is secure against additional attacks not covered in Section III.

Theorem 1. (Entity Anonymity). The proposed scheme preserves the anonymity property.

Proof. During the registration phase, the system identifier of each entity is calculated as $SysID_i = g_1^{ssk} \oplus ID_i$. Malicious attackers can only obtain the real identity ID_i, if the trusted authority's private key ssk is compromised.

Theorem 2. (Conditional Traceability). The proposed scheme preserves the conditional traceability property.

Proof. To protect against malicious activities that could compromise the protocol, it is essential that real identities are traceable. All generated system IDs are stored on an immutable blockchain ledger. The trusted authority (TA) can decrypt identities using its private key with the formula $SysID_i \oplus g_1^{ssk} \rightarrow ID_i$. Therefore, only the TA has the capability to trace the real identities of all entities.

Theorem 3. (Data Integrity). The proposed scheme ensures the integrity of EMRs.

Proof. To guarantee the integrity of EMRs, the data owner directly stores the hash of the ciphertext C_2 in the blockchain ledger during the data encryption stage. Furthermore, the transaction ID $Tid_{(M)}$ is packaged and stored in the cloud along with C_2. Therefore, data users can determine whether

the stored C_2 has been tampered with by comparing the on-chain hash value with the current hash value.

Theorem 4. (Data Confidentiality). The proposed scheme ensures the confidentiality of data.

Proof. To ensure the confidentiality of EMRs, the data owner E_i encrypts EMRs using its own public key as $C_2 = Pubk_{(E_i)}^{y} \oplus M = (g_1^{\tau_i})^y \oplus M$, and $y \in (0, q)$ is a selected random number. It is evident that as long as the parameter y remains confidential, the confidentiality of M can be maintained.

Theorem 5. (Secure Access Policy). The proposed scheme ensures the integrity of access policies.

Proof. The access policy, formatted in JSON, is submitted to IPFS, which maintains data integrity through the use of distributed hash tables (DHT) and Merkle Directed Acyclic Graphs (DAGs). Any modification made to the policy will alter its hash value, triggering a network-wide verification mechanism that prevents data tampering. Therefore, the stored access policy cannot be tampered with.

Theorem 6. (Session-key Security). The proposed scheme effectively prevents session-key computation attacks.

Proof. During the registration phase, user E_i and the Trusted Authority (TA) generate the session key using the Diffie-Hellman protocol. Specifically, the session key from E_i to TA is given by $K_{i \to TA} = (Y_{TA})^{x_i}$, and the session key from TA to E_i is given by $K_{TA \to i} = (Y_i)^{ssk}$. As a result, computing the session key is computationally infeasible for an adversary who does not possess the secret parameters x_i and ssk.

Theorem 7. (Resistance to Proof Forgery). The proposed scheme resists forgery of access credentials.

Proof. In this scheme, $Proof_{(access)}$ is the sole credential for data access permissions, determining whether access requests can be approved by the deployed access control mechanism. Since the hash of $Proof_{(access)}$ is stored in the immutable blockchain, any forged proof $Proof'_{(access)}$ can be detected by verifying its hash against the blockchain record. Furthermore, $Proof_{(access)}$ contains the user's public key $Pubk_i$, which prevents adversaries from impersonating authorized users and using their proofs to access EMRs.

Theorem 8. (Resistance to Impersonation Attacks). The proposed scheme is secure against the impersonation attack.

Proof. If an adversary $\mathcal{A}$ impersonates a legitimate entity E_i to construct requests (e.g., AR, PR) to perform operations allowed in the system, $\mathcal{A}$ should forge a digital signature σ' to replace the digital signature from E_i. Afterward, the recipients of requests will verify the requests by checking the contained signature. Obviously, the forged signature σ' will not pass the signature verification, and the requests will be discarded.

Theorem 9. (Resistance to Replay Attacks). The proposed scheme is secure against replay attacks.

Proof. Suppose an adversary $\mathcal{A}$ can capture transmitted messages. Even if $\mathcal{A}$ retransmits these messages, the recipients can verify their validity by analyzing the contained timestamp as $(TS_{now} - TS) \overset{?}{\leq} \triangle T$. Since the timestamp threshold $\triangle T$ is small, the proposed scheme is secure against replay attacks.

Theorem 10. (Resistance to Man-in-the-Middle Attacks). The proposed scheme is secure against man-in-the-middle (MITM) attacks.

Proof. Assume that an adversary $\mathcal{A}$ has the capability to intercept the transmitted request PR_i. $\mathcal{A}$ then attempts to modify PR_i and sends it to an entity E_i. To modify the request PR_i, $\mathcal{A}$ should generate a new signature Sig' for the modified request PR_i'. To generate a valid digital signature, $\mathcal{A}$ needs to obtain the corresponding private key from the public key, which is equivalent to Bob solving the discrete logarithm problem. This demonstrates that the proposed scheme is secure against man-in-the-middle (MITM) attacks.

B. Formal Security Analysis

1) Rules of BAN-logic: BAN-logic is a belief-based model logic that can be used to prove whether the proposed scheme can achieve the expected security requirements. Hence, we adopt BAN-logic to formally verify the security of the scheme, and summarize the related notations in Table III.

TABLE III
NOTATIONS USED IN BAN-LOGIC.

Notation	Description
$P \mid\equiv X$	P believes the message X.
$P \lhd X$	P receives the message X.
$P \mid\sim X$	P sent the message X.
$P \mid\Rightarrow X$	P controls the message X.
$\#(X)$	X is fresh (X is not a replayed message).
$\{X\}_K$	X is encrypted with key K.
$\{X\}_{K^{-1}}$	X is a signature signed with a private key K^{-1}.
$\langle X \rangle_Y$	X is sent combined with Y.
$P \overset{K}{\longleftrightarrow} Q$	P and Q share the secret key K.
$P \mid\equiv \overset{K}{\rightarrow} Q$	P believes Q's public key is K.

BAN-logic derives final beliefs from initial assumptions through rules, usually involving the following critical rules.

1) R1 (Message-Meaning Rule): If P believes that K is the public key of Q, and P received X signed with Q's private key K^{-1} (i.e., $\{X\}_{K^{-1}}$), then P believes Q sent X.
$$\frac{P \mid\equiv \overset{K}{\rightarrow} Q, P \lhd \{X\}_{K^{-1}}}{P \mid\equiv Q \mid\sim X}$$
2) R2 (Jurisdiction Rule): If P believes that Q has jurisdiction over X, and P believes that Q believes in X, then P believes in X.
$$\frac{P \mid\equiv Q \Rightarrow X, P \mid\equiv Q \mid\equiv X}{P \mid\equiv X}$$
3) R3 (Message-Signature Rule): If P knows Q's public key K and a signature $\{X\}_{K^{-1}}$, then P believes Q sent X.
$$\frac{P \lhd K, P \lhd \{X\}_{K^{-1}}}{P \mid\equiv Q \mid\sim X}$$
4) R4 (Nonce Verification Rule): If P believes that X is fresh and P believes that Q has declared X, then P believes that Q believes in X.
$$\frac{P \mid\equiv \#(X), P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

5) R5 (Freshness Rule): If X is fresh, then the composite message containing X is also fresh.

$$\frac{P \mid\equiv \#(X)}{P \mid\equiv \#(X,Y)}$$

6) R6 (Belief Rule): If P believes Q trusts the message (X,Y), then P believes Q trusts X.

$$\frac{P \mid\equiv Q \mid\equiv (X,Y)}{P \mid\equiv Q \mid\equiv X}$$

2) Idealization and initial assumptions: The idealized form of our scheme is as follows:

- M1 (Identity Registration): This message pair corresponds to the above registration process in Section V-C.
  1) $E_i \rightarrow TA : \{ID_i, Pubk_{(E_i)}\}_{K_{i\rightarrow TA}}$;
  2) $TA \rightarrow E_i : \{SysID_i, Pubk_{(E_i)}, TS\}|\{H(SysID_i ||Pubk_{(E_i)})\}_{spk^{-1}}$.
- M2 (Role Assignment Request & Response): This message pair corresponds to the role assignment process in Section V-D.
  1) $E_i \rightarrow RAC$: $\{UA_{(E_i)}, Proof_{(ID_i)}, TS\}|\{H(UA_{(E_i)} ||Proof_{(ID_i)}||TS)\}_{Pubk_{(E_i)}{}^{-1}}$;
  2) $RAC \rightarrow E_i$: $\{Role_{(E_i)}, TS\}|\{H(Role_{(E_i)}||TS) \}_{rac\_pk^{-1}}$.
- M3 (Access Proof Generation): This message pair corresponds to the permission-granting process defined in Section V-E.
  1) $E_i \rightarrow EACC$: $\{Fid, Pubk_{(E_i)}, Role_{(E_i)}, TS\}|\{ H(Fid|Pubk_{(E_i)}||Role_{(E_i)}||TS)\}_{Pubk_{(E_i)}{}^{-1}}$;
  2) $EACC \rightarrow E_i$:$\{Proof_{(access)}, Expire, TS\}|\{H(Proof_{(access)}||Expire||TS)\}_{sc\_pk^{-1}}$.

Based on the above idealized messages and our scheme description, we establish the following initial assumptions necessary for the BAN-logic analysis.

- S1: $E_i \mid\equiv \xrightarrow{spk} TA$.
- S2: $E_i \mid\equiv \xrightarrow{sc\_pk} EACC$.
- S3: $E_i \mid\equiv \xrightarrow{rac\_pk} RAC$.
- S4: $E_i \mid\equiv \xrightarrow{Pubk_{(E_j)}} E_j$.
- S5: $EACC \mid\equiv \xrightarrow{Pubk_{(E_j)}} E_j$.
- S6: $E_i \mid\equiv \#(TS)$.
- S7: $EACC \mid\equiv \#(TS)$.
- S8: $E_i \mid\equiv TA \Rightarrow (SysID_i, Proof_{(ID_i)})$.
- S9: $E_i \mid\equiv RAC \Rightarrow Role_{(E_i)}$.
- S10: $E_i \mid\equiv EACC \Rightarrow Proof_{(access)}$.

3) Proof Process: We formally analyze the security of our scheme from three aspects: the validity of registration credentials, the credibility of assigned roles, and the authenticity of access credentials.

(1) Validity of registration credentials.

1) From M1, E_i sees the message signed by TA: $E_i \lhd H(SysID_i||Proof_{(ID_i)}||\triangle_a||W_i||TS)_{spk^{-1}}$.
2) Using S1 and the Message-Meaning Rule (R1), we deduce that

$$E_i \mid\equiv TA \mid\sim H(SysID_i||Proof_{(ID_i)}||\triangle_a||W_i||TS). \tag{8}$$

3) From S6 and the Freshness Rule (R5), we have $E_i \mid\equiv \#(H(SysID_i||Proof_{(ID_i)}||\triangle_a||W_i||TS))$.
4) Applying the Nonce Verification Rule (R4) to the results of steps 2 and 3, we conclude that

$$E_i \mid\equiv TA \mid\equiv H(SysID_i||Proof_{(ID_i)}||\triangle_a||W_i||TS). \tag{9}$$

5) Given the properties of the hash function, we can infer $E_i \mid\equiv TA \mid\equiv (SysID_i, Proof_{(ID_i)}, \triangle_a, W_i, TS)$. Applying the Belief Rule (R6) gives us that

$$E_i \mid\equiv TA \mid\equiv Proof_{(ID_i)}, \tag{10}$$

$$E_i \mid\equiv TA \mid\equiv SysID_i. \tag{11}$$

6) Finally, using S8 and the Jurisdiction Rule (R2), we obtain the desired belief as

$$E_i \mid\equiv Proof_{(ID_i)}, \tag{12}$$

$$E_i \mid\equiv SysID_i. \tag{13}$$

Thus, the validity of the registration credentials is confirmed.

(2) Credibility of assigned roles.

1) From M2, E_i sees the role assignment message: $E_i \lhd \{Role_{(E_i)}, TS\}|H(Role_{(E_i)}|TS)_{raç\_pk^{-1}}$.
2) Using S3 and the Message-Meaning Rule (R1), we deduce that

$$E_i \mid\equiv RAC \mid\sim (Role_{(E_i)}, TS). \tag{14}$$

3) From S6 and the Freshness Rule (R5), we conclude that

$$E_i \mid\equiv \#(Role_{(E_i)}, TS). \tag{15}$$

4) According to (14) and (15), we apply the Nonce Verification Rule (R4) to conclude that

$$E_i \mid\equiv RAC \mid\equiv (Role_{(E_i)}, TS). \tag{16}$$

5) From (16), we apply the Belief Rule (R6) to conclude that

$$E_i \mid\equiv RAC \mid\equiv Role_{(E_i)}. \tag{17}$$

6) Finally, using S9 and (17), we apply the Jurisdiction Rule (R2) to obtain that

$$E_i \mid\equiv Role_{(E_i)}. \tag{18}$$

Hence, the credibility of assigned roles is formally analyzed.

(3) Authenticity of access permissions.

1) From M3, EACC sees E_i's request: $EACC \lhd H(Fid||Pubk_{(E_i)}||Role_{(E_i)}||TS)_{Pubk_{(E_i)}{}^{-1}}$.
2) Using S5 and R1, EACC believes E_i sent the request:

$$EACC \mid\equiv E_i \mid\sim (Pubk_{(E_i)}, Role_{(E_i)}, TS). \tag{19}$$

3) Considering S7 and (19), we apply the Freshness Rule (R5) to conclude that

$$EACC \mid\equiv \#(Pubk_{(E_i)}, Role_{(E_i)}, TS). \tag{20}$$

4) According to (19) and (20), we employ the Nonce Verification Rule (R4) to conclude that

$$EACC \mid\equiv E_i \mid\equiv (Pubk_{(E_i)}, Fid, Role_{(E_i)}). \tag{21}$$

5) From the second part of M3, we can conclude that

$$E_i \lhd \{H(Proof_{(access)}||TS)\}_{sc\_pk^{-1}} \quad (22)$$

6) Using S2 and the Message-Meaning Rule (R1), we deduce that

$$E_i \mid\equiv EACC \mid\sim (Proof_{(access)}, TS). \quad (23)$$

7) From S6 and the Freshness Rule (R5), we can conclude that

$$E_i \mid\equiv \#(Proof_{(access)}, TS). \quad (24)$$

8) Considering (23) and (24), we apply R4 (Nonce Verification Rule) to conclude that

$$E_i \mid\equiv EACC \mid\equiv (Proof_{(access)}, TS). \quad (25)$$

Using S10 and (25), we apply the Jurisdiction Rule (R2) to obtain that

$$E_i \mid\equiv (Proof_{(access)}, TS). \quad (26)$$

Therefore, the authenticity of the access permission credential is proven.

C. Formal security verification using Proverif

In this section, we adopt the widely accepted automated verification tool Proverif to verify the security of our scheme. Through a simulation study using the Proverif tool, we demonstrate that the proposed scheme can maintain data confidentiality, data integrity, and efficient data access control. Our proposed protocol will be specified to the Proverif tool in the form of a protocol model written in its dedicated modeling language. The protocol model consists of a set of processes. Each process represents the behavior of a protocol participant and comprises a sequence of operations. Since entities Patient, Doctor, Cloud, and AccessControlContract are communicating with one another in the proposed scheme, these entities are modeled as processes Patient, Doctor, Cloud, and AccessControlContract as shown in Fig. 4 and Fig. 5. During the access control process, the order in which events are triggered is an extremely important security objective. For instance, the data-access event in the doctor process can only be triggered after the predefined permission-grant event in the AccessControlContract process. Otherwise, it is likely to suggest that the proposed access control mechanism has been compromised. Hence, the correct sequence of events should be event (DoctorAccessedEMR) ==> event(AccessGranted(Doctor_pub)). Here, the expression DoctorAccessedEMR ==> AccessGranted(Doctor_pub) means that event DoctorAccessedEMR occurs after event AccessGranted.

Based on the proposed scheme, we summarize the sequence of events as follows.

1) Only legitimate users can request access to the stored EMRs. Therefore, the user authorized to access EMRs must be a legitimate user in the system, which can be expressed as: event(PermissionGranted(uid)) ==> (uid = Doctor_pub).
2) Only after obtaining authorization from AccessControlContract can doctors pass the verification process. This is expressed as: event(AccessGranted(Doctor_pub)) ==> event(PermissionGranted(Doctor_pub)).
3) Only after the requesting party has passed the verification from data owners can it obtain the requested EMR. This can be represented as: event(DoctorAccessedEMR) ==> event(AccessGranted(Doctor_pub).

```
(* Patient Process *)
let Patient =
  (* Data encryption *)
  new y:nonce;
  let C1 = exp(g, y) in
  let C2 = xor(exp(Patient_pub, y), PatientEMR) in
  let hoodeM = h(C2) in
  out(c, (hoodeM, C1, C2));
  (* Access control policy setup *)
  let policy = createPolicy(Doctor, Researcher) in
  out(c, sign(policy, Patient_priv)).

(* Doctor Process *)
let Doctor =
  (* Access request *)
  new reqNonce:nonce;
  let accessReq = (Doctor_pub, Fid, Doctor_role, reqNonce) in
  let sigReq = sign(accessReq, Doctor_priv) in
  out(c, (accessReq, sigReq));

  (* After receiving proof *)
  in(c, proof:proof);
  if verify(proof, h(accessReq), TA_pub) then
    (* Send public key for re-encryption *)
    out(c, Doctor_pub);
    (* Receive and verify re-encrypted data *)
    in(c, (C2_prime:cipher, C3:bitstring, Tid:hash));
    let storedHash = queryBlockchain(Tid) in
    if h(C2_prime) = storedHash then
      (* Decryption *)
      let rho = exp(Patient_pub, Doctor_priv) in
      let rho_prime = div(C3, rho) in
      let decrypted = xor(rho_prime, C2_prime) in
      event DoctorAccessedEMR(decrypted)
    else event IntegrityViolationDetected
  else event InvalidAccessAttempt.
```

Fig. 4. Patient and Doctor processes.

The results of the ProVerif code execution are presented in Fig. 6. This simulation involves a protocol with four parallel processes and five verification queries. As illustrated in Fig. 6, all four parallel processes were executed successfully, ensuring the confidentiality of Electronic Medical Records (EMRs)—meaning that no attacker can access sensitive information. Furthermore, Fig. 6 demonstrates that the sequence of events proceeded normally, indicating that the proposed access control scheme is reliable and meets the necessary security requirements.

VII. Performance Analysis

To validate the feasibility of our proposed scheme, we developed a prototype and conducted comprehensive performance evaluations.

```
(* Cloud Server Process *)
let Cloud =
   (* Store encrypted data *)
   in(c, (hoodeM:hash, C1:bitstring, C2:cipher));
   store EMR_Record = (C1, C2, hoodeM);
   (* Re-encryption service *)
   in(c, (C1_prime:bitstring, reKey:reKey));
   let C3 = mult(C1_prime, reKey) in
   out(c, C3).

(* Smart Contract Processes *)
let AccessControlContract =
   in(c, (req:(pubKey*id*role*nonce), sigReq:proof));
   if verify(sigReq, h(req), getPubKey(req)) then
      (* Check RBAC permissions *)
      if hasPermission(req, Patient_policy) then
         let accessProof = generateProof(req) in
         out(c, sign(accessProof, Contract_priv))
      else event PermissionDenied
   else event InvalidSignature.
```

Fig. 5. Cloud and AccessControlContract processes.

```
The event AccessUsed is executed at {28}.
A trace has been found.
RESULT not event(AccessUsed) is false.
-- Query not event(DoctorAccessedEMR) in process 1.
Translating the process into Horn clauses...
Completing...
Starting query not event(DoctorAccessedEMR)
RESULT not event(DoctorAccessedEMR) is true.
-- Query not event(VerifySig) in process 1.
Translating the process into Horn clauses...
Completing...
Starting query not event(VerifySig)
RESULT not event(VerifySig) is true.
-- Query event(PermissionGranted(uid[])) ==> uid[] = Doctor_pub[] in process 1.
Translating the process into Horn clauses...
Completing...
Starting query event(PermissionGranted(uid[])) ==> uid[] = Doctor_pub[]
RESULT event(PermissionGranted(uid[])) ==> uid[] = Doctor_pub[] is true.
-- Query inj-event(AccessGranted(Doctor_pub[])) ==> inj-event(PermissionGranted(Doctor_pub[])) in process 1.
Translating the process into Horn clauses...
Completing...
Starting query inj-event(AccessGranted(Doctor_pub[])) ==> inj-event(PermissionGranted(Doctor_pub[]))
RESULT inj-event(AccessGranted(Doctor_pub[])) ==> inj-event(PermissionGranted(Doctor_pub[])) is true.
-- Query inj-event(DoctorAccessedEMR) ==> inj-event(AccessGranted(Doctor_pub[])) in process 1.
Translating the process into Horn clauses...
Completing...
Starting query inj-event(DoctorAccessedEMR) ==> inj-event(AccessGranted(Doctor_pub[]))
RESULT inj-event(DoctorAccessedEMR) ==> inj-event(AccessGranted(Doctor_pub[])) is true.

----------------------------------------------------------------
Verification summary:

Query not attacker(PatientEMR[]) is true.

Query not event(AccessUsed) is false.

Query event(PermissionGranted(uid[])) ==> uid[] = Doctor_pub[] is true.

Query inj-event(AccessGranted(Doctor_pub[])) ==> inj-event(PermissionGranted(Doctor_pub[])) is true.

Query inj-event(DoctorAccessedEMR) ==> inj-event(AccessGranted(Doctor_pub[])) is false.

----------------------------------------------------------------
```

Fig. 6. Simulation results.

A. Experimental Environment

To evaluate the performance of this scheme, we developed a prototype based on Hyperledger Besu[1] and the Java Pairing-Based Cryptography (JPBC). This prototype consists of several medical terminal devices supported by single-board computers (Raspberry Pi 4 Model) and various EMRs generating and requesting terminals simulated through software. To be specific, the employed blockchain consists of 4 full-function nodes and employs the Clique PoA consensus mechanism, with a block generation time of 15 seconds. All nodes are deployed within the same local area network to reduce the impact of network latency on the experiments. To simulate real medical data, we generate three different scales of EMRs (5 KB, 50 KB, and 200 KB), structured in JSON format containing patient information, medical reports, and diagnostic records. Meanwhile, Alibaba Cloud is selected for deploying cloud-based applications. The specifications of these devices are described in Table IV.

TABLE IV
SPECIFICATIONS OF DEVICES.

Devices	CPU	OS	RAM	Disk
ThinkPad E14	Intel Core i7-10510U (4.6GHz)	Windows 10	64GB	256GB
Raspberry Pi 4	Cortex-A72 (1.5GHZ)	Ubuntu MATE 16.04	2GB	32GB
Alibaba ECS	Intel Core i7-6700K	Ubuntu 24.04	2GB	40GB

To be more general, we choose SHA-256 for hash computing, AES as the symmetric encryption method, and 160-bit ECC for generating BLS signatures. Here, we use Java to implement cryptographic operations[2] as shown in Table V.

TABLE V
EXECUTION TIME OF FUNDAMENTAL OPERATIONS (MS).

Operation	Symbol	Execution time
Point multiplication	t_{pm}	9.87
AES encryption	t_{ae}	4.21
AES decryption	t_{ad}	2.36
Bilinear mapping	t_{bm}	22.32
Hash (SHA-256)	t_h	3.51
BLS signing	t_{bs}	16.12
BLS verification	t_{bv}	17.37
ECDSA signing	t_{es}	11.23
ECDSA verification	t_{ev}	14.65

In this prototype, Hyperledger-Besu 23.10 is adopted to construct the blockchain network, and Ubuntu 24.04 is chosen as the underlying operating system. In addition, the involved smart contracts [3] (i.e., IMC, RAC, and RPSC, etc) are written with Solidity 0.5.8 and deployed in peer nodes. Table VI shows the running time of smart contracts. The execution time of each contract remains stable, not exceeding 20 ms, which is acceptable for healthcare applications.

TABLE VI
EXECUTION TIME OF SMART CONTRACTS (MS).

Smart Contract	Symbol	Max Cost	Min Cost	Average Cost
IMC	t_{imc}	13.78	5.23	8.46
RAC	t_{rac}	19.34	7.32	12.14
RPSC	t_{rpsc}	20.93	8.47	14.02
EACC	t_{eacc}	25.21	11.07	18.43

B. Comparative Schemes

To facilitate the performance analysis of our scheme, we introduce some state-of-the-art schemes [26], [29], [31], and [32] as comparative benchmarks. Here, we provide a brief introduction to these involved schemes as follows. (1) Peng et al.[26] designed a patient-centric EMR sharing scheme based on dual-blockchain and a tripartite authentication key

[1]https://www.kaleido.io/blockchain-platform/hyperledger-besu

[2]https://github.com/smu-lihongzhi/RBACSC/blob/main/RBACSC.java

[3]https://github.com/smu-lihongzhi/RBACSC/

agreement (TAKA) based on identity. They established a dual-blockchain system (MEDchain and SUPchain) to facilitate secure and precise access control for EMRs. (2) Zhao et al. [29] proposed BECAAC, a medical data sharing scheme that utilizes consortium blockchain, edge computing, proxy re-encryption, and group signatures. They developed a three-layer architecture consisting of physical, storage management, and application layers. This architecture features a "cloud-edge-local" storage platform, which enables data access control, ensures data confidentiality, maintains patient anonymity, and enhances accountability for malicious behaviors. (3) The authors of [31] proposed a consortium blockchain-based EMR sharing scheme that utilizes a novel proxy re-encryption and attribute-based control model. This approach aims to achieve personalized access while ensuring privacy. (4) Bian et al. [32] developed a medical record storage system based on a consortium blockchain, integrating Role-Based Access Control (RBAC), Advanced Encryption Standard (AES), and Fine-Grained Access Control with Storage Reduction (FASR). They utilized Hyperledger-Fabric to create a system that offers dynamic, efficient data access control by differentiating user roles. Additionally, they improved storage efficiency through FASR deduplication, ensuring the secure and efficient management of electronic medical records (EMRs). Table VII presents the comparison of the security properties of these schemes.

TABLE VII
SECURITY PROPERTIES.

Properties	[26]	[29]	[31]	[32]	Ours
Key Agreement	√	×	×	√	√
Anonymity	√	√	√	×	√
Data Confidentiality	×	√	√	√	√
Traceability	×	×	×	×	√
Identity Authentication	√	√	√	×	√
Data Integrity	√	×	×	×	√
Blockchain-based	√	√	√	√	√
Formal Security Proof	√	√	√	×	√
Resistance to Cyberattacks	√	√	√	√	√

Table VIII provides a comprehensive comparison of the key technical features among the benchmark schemes. As shown, all compared schemes adopt consortium blockchain as the underlying infrastructure, ensuring decentralized trust. Specifically, our scheme uniquely combines proxy re-encryption (PRE) with a dynamic accumulator for efficient identity management, while employing RBAC enhanced by smart contracts for permission verification. In contrast, schemes [26], [29], and [31] predominantly rely on attribute-based access control (ABAC), which introduces higher computational overhead during policy matching. Furthermore, our scheme distinguishes itself by implementing on-chain hash anchoring for end-to-end integrity verification, a feature not fully addressed in [26] and [32].

C. Analysis of Computational Overhead

We analyze the computational overhead of the proposed scheme by theoretically analyzing the contained cryptographic operations and the smart contract calls. The specific analysis process is described as follows.

- Identity registration: During the registration of entities, E_i performs one point-multiplication operation to generate $Pubk_{(E_i)}$ and further performs an identity encryption operation. Subsequently, TA performs one AES decryption, a smart contract invocation (IMC), and one digital signature. Therefore, we define the computational cost of this phase as $T_{(reg)} = t_{pm} + t_{ae} + t_{ad} + t_{imc} + t_{bs}$.
- Roles assignment: The method AssignRole(·) performs one signature verification for the received $Proof_{(ID_i)}$, and invokes the smart contract (RAC) to assign the corresponding role to E_i. Hence, the corresponding computational cost can be defined as $T_{(ras)} = t_{bv} + t_{rac}$.
- Permission granting: In this phase, E_i first constructs a data access request $\mathcal{AR}$ by performing one hash operation and a message signing. Aferward, E_i calls EACC to obtain its requested access permissions. Therefore, the computational cost can be further calculated as $T_{(peg)} = t_h + t_{bs} + t_{bv} + t_{eacc}$.
- EMRs sharing: During permission verification, E_j first constructs a request with its permission credential $Proof_{(access)}$ and a signature $Sig_{(pf)}$ for $Proof_{(access)}$. After that, the EMR owner E_i performs the permission verification operation by invoking two smart contracts (i.e., EACC and IMC) and checking $Sig_{(pf)}$. In addition, E_i performs two point multiplications and one hash operation during data re-encryption. In the EMR decryption stage, there exist two point multiplications and three hash operations. Therefore, the computational cost of the EMRs sharing phase is $T_{(esh)} = t_{bs} + t_{bv} + t_{imc} + t_{eacc} + 4t_{pm} + 4t_h$.

Therefore, the computational cost of our scheme can be expressed as $T_{total}^{(ours)} = T_{(reg)} + T_{(ras)} + T_{(peg)} + T_{(esh)} = 239.86$ (ms). It is noted that this computational cost T_{ours} contains the computational overhead of identity registration, role allocation, permission granting, and EMRs decryption. To further evaluate the performance in terms of computational overhead, some state-of-the-art schemes [26], [29], [31], and [32] are selected for comparisons. In Peng et al.'s scheme [26], the proposed protocol mainly involves user registration, EMR management, and EMR sharing phases. There exist two point multiplications and four hash operations during the registration phase, so the corresponding computational cost of this phase can be calculated as $T_{(reg)}^{(26)} = 2t_{pm} + 4t_h$. Similarly, the computational overhead of EMR management and EMR sharing are defined as $T_{(man)}^{(26)} = 2t_{bm} + 4t_{pm} + 7t_h + 2(t_{es} + t_{ev})$, $T_{(shr)}^{(26)} = 2t_{bm} + 4t_{pm} + 8t_h$, respectively. Consequently, the corresponding computational cost is defined as $T_{total}^{(26)} = 10t_{pm} + 4t_{bm} + 19t_h + 2(t_{es} + t_{ev}) = 305.83$ (ms). Zhao et al.'s protocol [29] mainly contains EMRs creating, EMRs sharing, and tracing phases, the computational cost of [29] can be calculated as $T_{total}^{(29)} = 4t_{bm} + 6t_h + 4t_{pm} + 3(t_{es} + t_{ev}) + 2(t_{ae} + t_{ad}) = 240.6$ (ms). In the scheme presented by Liu et al. [31], there exist two point multiplications, three hash operations during the registration process, four bilinear-mapping operations, two ECDSA signing operations, and eight hash operations during EMR sharing. Hence, we can define the computational cost as $T_{total}^{(31)} = 2t_{pm} + 4t_{bm} + 2(t_{es} + t_{ev}) + 11t_h = 354.17$

TABLE VIII
COMPARISON OF KEY TECHNICAL FEATURES

Scheme	Blockchain	Encryption	Access Control	Identity Management	Integrity Verification
Our Scheme	Consortium	PRE + AES + BLS	RBAC + Smart Contracts	Dynamic Accumulator	On-chain hash anchoring
Peng et al. [26]	Dual-Blockchain	TAKA + Symmetric	ABAC	Identity-based	Not specified
Zhao et al. [29]	Consortium	PRE + Group Signatures	ABAC	Group signatures	Edge computing based
Liu et al. [31]	Consortium	PRE + CP-ABE	ABAC	Attribute-based	Blockchain-based
Bian et al. [32]	Consortium	AES + FASR	RBAC	Role-based	Not specified

(ms). When it comes to Bian et al.'s scheme [32], there exist four AES encryption/decryption operations, five ECDSA signing operations, four point multiplications, and eight hash operations during EMRs access control. Therefore, we can calculate the corresponding computational cost as $T_{total}^{(32)} = 4(t_{ae}+t_{ad})+5(t_{es}+t_{ev})+4t_{pm}+8t_h = 223.24$ (ms). Table IX presents the comparison of computational cost.

TABLE IX
COMPARISON OF COMPUTATIONAL COSTS (MS).

Scheme	Computational Cost	Total
Peng et al.'s [26]	$10t_{pm}+4t_{bm}+19t_h+2(t_{es}+t_{ev})$	305.83
Zhao et al.'s [29]	$4t_{bm}+6t_h+4t_{pm}+3(t_{es}+t_{ev})+2(t_{ae}+t_{ad})$	240.6
Liu et al.'s [31]	$2t_{pm}+4t_{bm}+2(t_{es}+t_{ev})+11t_h$	354.17
Bian et al.'s [32]	$4(t_{ae}+t_{ad})+5(t_{es}+t_{ev})+4t_{pm}+8t_h$	223.24
Our Scheme	$5t_{pm}+3(t_{bs}+t_{bv})+5t_{pm}+5t_h+2t_{imc}+2t_{eacc}+t_{rac}$	239.86

We conduct simulations for this experiment based on the experimental environment outlined in Section VII-A. In this section, we compare the proposed scheme with the protocols of [26], [29], [31], and [32] in terms of the request-response latency. Specifically, we evaluate the data-sharing request processing capability by varying the number of concurrent access requests, legitimate users, and network nodes.

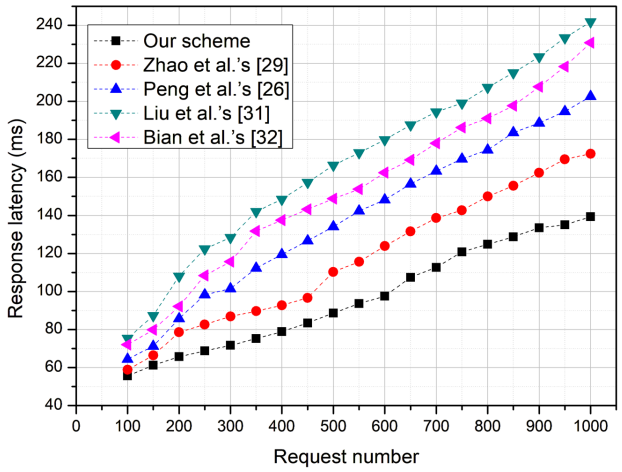


Fig. 7. Response latency with different access requests.

Given a fixed-size Electronic Medical Record (EMR) and a consistent network size, we varied the number of concurrent requests from 100 to 1000 and calculated the average response latency for each access request. The results are illustrated in Fig. 7. It is important to note that "access request" refers to authorized users requesting a specific EMR, excluding user registration and access authorization processes. As shown in Fig. 7, the response latency increases with the number of concurrent access requests. This rise in latency can be attributed to the increased network load and system demands that come with higher request volumes. Our approach achieves the lowest response latency due to the minimal cost of access permission verification. In comparison to the Attribute-Based Access Control (ABAC) model presented in references [26] and [29], the Role-Based Access Control (RBAC) model used in our scheme is more straightforward and efficient, as it does not necessitate the validation of numerous attributes. Additionally, the scheme referenced in [31] incurs the longest response latency due to the maximum number of time-consuming bilinear mapping operations it performs. While [32] utilizes a concise and efficient RBAC model, it lacks an identity authentication mechanism, leading to a significant number of signature verification operations during EMR sharing.

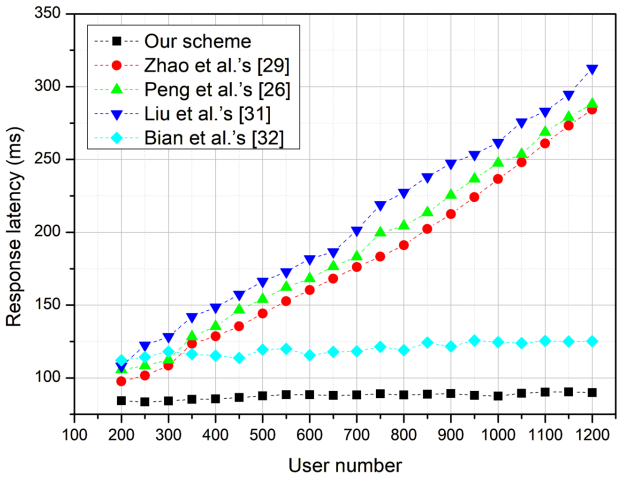
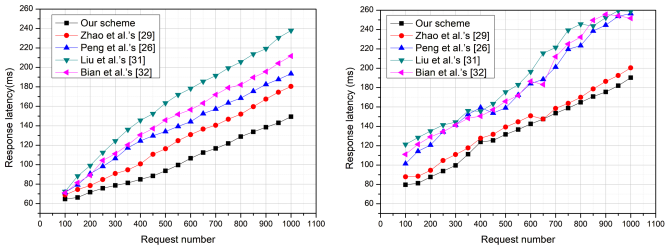


Fig. 8. Response latency with different number of legitimate users.

Additionally, we further conduct simulations to analyze the impact of different numbers of legitimate users on the response latency. Specifically, we increase the number of legitimate users in the system from 200 to 1200 while keeping the number of concurrent requests fixed at 200. As shown in Fig. 8, we observe that the three protocols (i.e., Peng et al.'s [26], Zhao et al.'s [29], and Liu et al.'s [31]) based on ABAC are significantly impacted by the number of legitimate users. As the number of users increases, the user attributes in the ABAC model also increase, resulting in higher latency. Meanwhile, the ABAC model-based authorization process involves retrieving user attributes and matching them against access policies. Consequently, this ultimately increases latency in the permission verification phase. In contrast, the RBAC-based



(a) Blockchain with twenty peer nodes (b) Blockchain with forty peer nodes

Fig. 9. Response latency with different number of blockchain peer nodes.

schemes (i.e., our scheme and Bian et al.'s [32]) demonstrate lower response latency with minimal fluctuations, as the role-based permission verification is independent of the number of legitimate users, and access permissions are only bound to roles. Even though the number of legitimate users in the system continues to increase, the response latency of our scheme remains relatively stable. Moreover, although both our scheme and [32] adopt the RBAC model, our scheme demonstrates superior latency performance through the integration of dynamic accumulators and distributed smart contracts for rapid permission verification.

As shown in Table VII, both our proposed scheme and the comparative solutions are implemented on blockchain systems. In these systems, the size of the network (the number of nodes) is a crucial factor that directly affects consensus efficiency and data synchronization latency in decentralized ledgers. To evaluate the scalability of our scheme, we maintain a constant number of legitimate users while varying the scale of the blockchain nodes. We analyze the impact of this variation on response latency under concurrent requests. As illustrated in Fig. 9, we measure the average response latency of the involved schemes for concurrent access requests with 20 and 40 peer nodes in the blockchain, respectively. When comparing Fig. 9 (a) with Fig. 9 (b), we observe that all blockchain-based schemes experience increased request response latency as the number of network nodes increases. For instance, in Zhao et al.'s scheme [29], when there are 500 concurrent requests and 20 network nodes, the average response latency is 117 ms. However, when the number of network nodes rises to 40, the average response latency increases to 140 ms. Similarly, other schemes, including our own, as well as those by Peng et al. [26] and Liu et al. [31], show a consistent trend where latency grows with the expansion of network nodes. In contrast, our scheme is less influenced by fluctuations in blockchain network nodes because we utilize the Proof-of-Authority (POA) consensus mechanism. This approach allows for blockchain verification through predetermined authority nodes, thereby simplifying the consensus process. Additionally, the implementation of dynamic accumulators helps to minimize retrieval delays for identity information during permission verification.

D. Communication Overhead

To evaluate the communication overhead of this scheme, we focus on the transmitted messages involved in the phases of identity registration, role assignment, permission granting, and EMR sharing. To calculate the communication cost, we assume that the size of identities, pseudonyms, and timestamps is 4 bytes. Furthermore, we set the average lengths of hash values, encrypted messages, ECDSA signatures, and BLS signatures to 32 bytes, 64 bytes, 64 bytes, and 96 bytes, respectively. Additionally, the length of elements in $\mathbb{G}_1$ and $\mathbb{G}_2$ is 48 bytes, while the size of an element in $\mathbb{Z}_q$ is 4 bytes. Table X presents the related notations used in the subsequent computation of communication overhead.

TABLE X
THE RELATED NOTATIONS (BYTE)

Notation	Length	Description
$\mathcal{S}_{ID}$	4	Length of identities
$\mathcal{S}_{PM}$	4	Length of pseudonyms
$\mathcal{S}_{TS}$	4	Length of timestamps
$\mathcal{S}_{HS}$	32	Length of hash value
$\mathcal{S}_{ECC}$	64	Length of ECDSA signature
$\mathcal{S}_{BLS}$	96	Length of BLS signature
$\mathcal{S}_{PK}$	48	Length of public key
$\mathcal{S}_{EM}$	64	Length of symmetric-encryption results
$\mathcal{S}_{G}$	48	Length of the element in $\mathbb{G}_1$ or $\mathbb{G}_2$
$\mathcal{S}_{Z_q}$	4	Length of the element in $\mathbb{Z}_q$

According to the above-mentioned parameter settings, our scheme mainly contains $\mathcal{M}_1 = \{CT_{(ID)}, TS\}$ and $\mathcal{M}_2 = \{SysID_i, Proof_{(ID_i)}, \triangle_a, W_i, TS\}$ during the registration. Hence the corresponding communication cost is $C^{reg}_{(ours)} = |CT_{ID}|+2|TS|+|SysID_i|+|Proof_{(ID_i)}|+|W_i|+|\triangle_a| = \mathcal{S}_{EM} + 2\mathcal{S}_{TS} + \mathcal{S}_{PM} + \mathcal{S}_{BLS} + 2\mathcal{S}_G = 268$ (bytes). In the role assignment phase, there exists two messages as: $\mathcal{M}_3 = \{UA_{(E_i)}, Proof_{(ID_i)}, TS, Sig_i\}$ and $\mathcal{M}_4 = \{Role, TS\}$, so the corresponding communication cost is $C^{ass}_{(ours)} = |UA_{(E_i)}|+|Proof_{(ID_i)}|+2|TS|+|Sig_i|+|Role|= \mathcal{S}_{EM}+2\mathcal{S}_{BLS}+2\mathcal{S}_{TS}+\mathcal{S}_{Z_q} = 272$ (bytes). When it comes to the permission granting procedure, there exists a data access request $\mathcal{AR} = \{Fid, Pubk_{(E_i)}, Role_{(E_i)}, TS, Sig_i\}$, and the length of $\mathcal{AR}$ is $|\mathcal{AR}|= \mathcal{S}_{HS}+\mathcal{S}_{PK}+\mathcal{S}_{Z_q}+\mathcal{S}_{BLS}+\mathcal{S}_{TS}$. In addition, the permission credentials $Proof_{(access)}$ is generated and transmitted to entities, we can further define the length as $|Proof_{(access)}|= 3\mathcal{S}_{Z_q} + \mathcal{S}_{HS} + \mathcal{S}_{PK} + 2\mathcal{S}_{TS} = 100$ (bytes). Thereby, the corresponding communication of the permission granting phase is $C^{pgt}_{(ours)} = |\mathcal{AR}|+|Proof_{(access)}|= 256$ (bytes). Finally, the communication overhead in the EMR sharing phase is composed of permission verification and ciphertext transmission (i.e., $\{Proof_{(access)}, TS, Sig_{(pf)}\}$ and $\{C_2, C_3, Tid_{(\mathcal{MR}_i)}\}$), and we define the communication cost as $C^{shr}_{(ours)} = 3\mathcal{S}_{Z_q}+\mathcal{S}_{HS}+\mathcal{S}_{PK}+3\mathcal{S}_{TS}+\mathcal{S}_{BLS}+2\mathcal{S}_G+\mathcal{S}_{HS} = 328$ (bytes). Therefore, the total communication cost of this scheme is $C_{(ours)} = \sum_{i\in reg,ass,pgt,shr} C^i_{(ours)} = 1124$ (bytes). Fig.10 presents the communication cost of the involved phases of our scheme.

In Peng et al.'s scheme [26], there are five rounds of message interaction with a total communication overhead of 1379 bytes (i.e., 11032 bits). With respect to the communication cost of Zhao et al.'s [29], there exist six messages, such as $\mathcal{M}^{(29)}_1 = \{CT_m, Pubk, Sig_{(ECC)}, TS\}$, $\mathcal{M}^{(29)}_2 = \{Sig_{(ECC)}, Resp \in \{0,1\}, TS, CT_x\}$, $\mathcal{M}^{(29)}_3 = \{{g_1}^k \in G, Sig_{(ECC)}, TS, HS\}$, $\mathcal{M}^{(29)}_4 = \{Ret \in Z_q, Sig_{(ECC)}, e_1 \in G, e_2 \in G, Pubk, TS\}$,

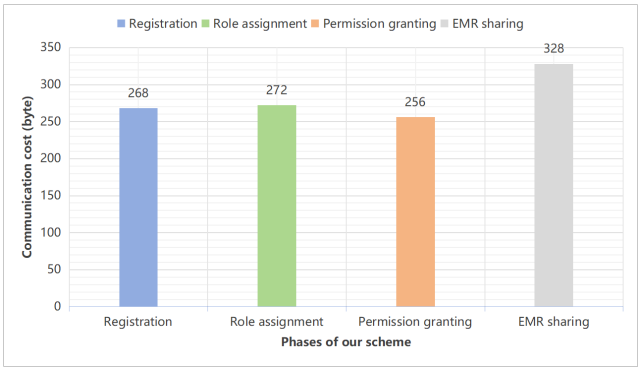


Fig. 10. Communication cost of our scheme.

$\mathcal{M}_5^{(29)} = \{Sig_{(ECC)}, Addr, Pubk, PM_i, TS\}$, and $\mathcal{M}_6^{(29)} = \{Pubk, CT_i, rand \in Z_q, w \in G, Sig_{(ECC)}, TS\}$. Hence, the communication cost of [29] is $C_{(29)} = 1064$ (bytes). In addition, the communication cost of Liu et al.'s [31] is around 1212 bytes, which includes the communication overhead of identity registration, data storage, and data access. Similarly, the communication cost of [32] is just about 873 bytes due to its lack of security properties such as data integrity and identity authentication. Fig. 11 shows the comparison results. We can observe that this proposed scheme exhibits a communication overhead that is comparable to that of Zhao et al.'s scheme [29], being only slightly higher than Bian et al.'s scheme [32]. Compared with schemes of [29] and [32], our scheme provides full security properties with acceptable communication overhead.

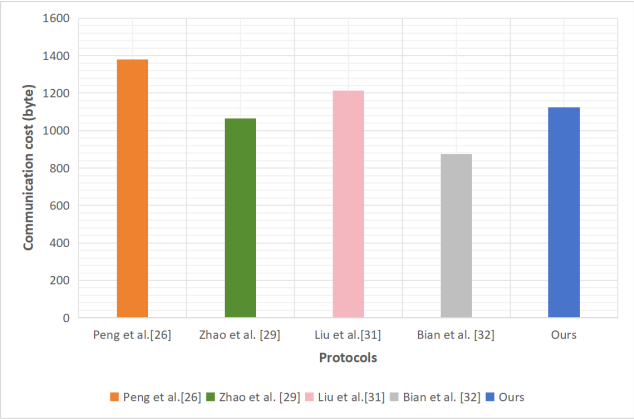


Fig. 11. Comparisons of communication overhead.

VIII. Conclusion

This work presents a patient-centered electronic medical record (EMR) access control scheme that integrates Role-Based Access Control (RBAC) with smart contracts. This approach enables fine-grained and decentralized management of data access. Additionally, we have designed a novel proxy re-encryption algorithm to ensure the confidentiality of EMRs during sharing. Formal security verification is conducted using BAN-logic and ProVerif, demonstrating essential security properties such as entity anonymity and data integrity. Our performance evaluation confirms that this hybrid RBAC-smart contract scheme is significantly more efficient than benchmark schemes, showing lower computational and communication overhead with stable response times. This proposal effectively mitigates vulnerabilities related to centralized trust. It also overcomes the privacy-preserving limitations of existing EMR systems. Despite the promising performance and security features demonstrated by our hybrid RBAC–smart contract scheme, it is important to acknowledge its inherent limitations, particularly regarding its dependency on blockchain performance. The current implementation relies on a consortium blockchain with a Proof-of-Authority (PoA) consensus, which, while efficient in controlled environments, may face scalability challenges in highly dynamic or large-scale IoMT deployments with massive concurrent access requests. In future research, we will focus on three main directions: 1) optimizing consensus mechanisms to minimize the impact of blockchain node scale, thereby enhancing scalability for large medical networks; 2) improving the efficiency of dynamic accumulators for extensive identity management and integrating lightweight cryptography for IoT-enabled healthcare systems; and 3) conducting large-scale real-world tests and exploring cross-chain interoperability to facilitate the sharing of EMRs across institutions.

References

[1] S. Reddi, P. M. Rao, P. Saraswathi, S. Jangirala, A. K. Das, S. S. Jamal, and Y. Park, "Privacy-preserving electronic medical record sharing for iot-enabled healthcare system using fully homomorphic encryption, iota, and masked authenticated messaging," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 9, pp. 10 802–10 813, 2024.

[2] J. Yu, W. Shen, and X. Zhang, "Cloud storage auditing and data sharing with data deduplication and private information protection for cloud-based emr," *Computers & Security*, vol. 144, p. 103932, 2024.

[3] D. Zhu, Y. Sun, N. Li, L. Song, and J. Zheng, "Secure electronic medical records sharing scheme based on blockchain and quantum key," *Cluster Computing*, vol. 27, no. 3, pp. 3037–3054, 2024.

[4] G. Wu, H. Wang, Z. Yang, D. He, and S. Chan, "Electronic health records sharing based on consortium blockchain," *Journal of Medical Systems*, vol. 48, no. 1, p. 106, 2024.

[5] K. Zhang, T. Chen, S. Chen, L. Wei, and J. Ning, "Lightweight, verifiable and revocable ehrs sharing with fine-grained bilateral access control," *Cluster Computing*, vol. 27, no. 7, pp. 9957–9973, 2024.

[6] G. Han, Y. Ma, Z. Zhang, and Y. Wang, "A hybrid blockchain-based solution for secure sharing of electronic medical record data," *PeerJ Computer Science*, vol. 11, p. e2653, 2025.

[7] S. Khan, M. Khan, M. A. Khan, L. Wang, and K. Wu, "Advancing medical innovation through blockchain-secured federated learning for smart health," *IEEE Journal of Biomedical and Health Informatics*, 2025.

[8] B. Li, L. Deng, Y. Mou, N. Wang, Y. Chen, and S. Li, "A pairing-free data sharing scheme based on certificateless conditional broadcast proxy re-encryption suitable for cloud-assisted iot," *IEEE Internet of Things Journal*, 2025.

[9] Y. Xu, H. Cheng, X. Liu, C. Jiang, X. Zhang, and M. Wang, "Pcse: Privacy-preserving collaborative searchable encryption for group data sharing in cloud computing," *IEEE Transactions on Mobile Computing*, 2025.

[10] M. Jamil, M. Farhan, F. Ullah, and G. Srivastava, "A lightweight zero trust framework for secure 5g vanet vehicular communication," *IEEE Wireless communications*, 2024.

[11] C. Wang, Y. Zhang, Q. Zhang, X. Xu, W. Chen, and H. Li, "Secas: Secure and efficient cross-domain authentication scheme based on blockchain for space tt&c networks," *IEEE Internet of Things Journal*, vol. 11, no. 16, pp. 26 806–26 818, 2024.

[12] R. Li, J. Cui, J. Zhang, L. Wei, H. Zhong, and D. He, "Blockchain-assisted revocable cross-domain authentication for vehicular ad-hoc networks," *IEEE Transactions on Dependable and Secure Computing*, 2025.

[13] F. Luo, H. Wang, W. Susilo, X. Yan, and X. Zheng, "Public trace-and-revoke proxy re-encryption for secure data sharing in clouds," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2919–2934, 2024.
[14] H. Pei, P. Yang, W. Li, M. Du, and Z. Hu, "Proxy re-encryption for secure data sharing with blockchain in internet of medical things," *Computer Networks*, vol. 245, p. 110373, 2024.
[15] X. Yao, J. Zhou, X. Du, and S. Zhang, "A cp-abe and iota based lightweight sensitive data access control scheme for iot," *IEEE Internet of Things Journal*, 2024.
[16] Q. Hu, M. Correia, and T. Jiang, "An efficient blockchain for decentralized abac policy decision point," *Future Generation Computer Systems*, vol. 166, p. 107732, 2025.
[17] B. Yang and H. Hu, "Resiliency analysis of role-based access control via constraint enforcement and mathematical programming," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 54, no. 7, pp. 4089–4100, 2024.
[18] S. Siyuan, L. Aodi, D. Xuehui, W. Xiaohan, and T. Ming, "Abac policy mining method for heterogeneous access control system: S. siyuan et al." *The Journal of Supercomputing*, vol. 81, no. 9, p. 1065, 2025.
[19] Hemani, D. Singh, and R. K. Dwivedi, "Designing blockchain based secure autonomous vehicular internet of things (iot) architecture with efficient smart contracts," *International Journal of Information Technology*, vol. 17, no. 2, pp. 1207–1223, 2025.
[20] J. Crisostomo, F. Bacao, and V. Lobo, "Machine learning methods for detecting smart contracts vulnerabilities within ethereum blockchain- a review," *Expert Systems with Applications*, p. 126353, 2025.
[21] A. Punia, P. Gulia, N. S. Gill, U. K. Lilhore, S. Simaiya, R. Alroobaea, H. Alsufyani, and A. M. Baqasah, "Quickmedblock: A framework for enhanced attribute-based access control using blockchain for ehr in cloud," *Peer-to-Peer Networking and Applications*, vol. 18, no. 5, p. 258, 2025.
[22] S. Wu, A. Zhang, Y. Gao, and X. Xie, "Patient-centric medical service matching with fine-grained access control and dynamic user management," *Computer Standards & Interfaces*, vol. 89, p. 103833, 2024.
[23] S. Saha, A. K. Das, M. Wazid, Y. Park, S. Garg, and M. Alrashoud, "Smart contract-based access control scheme for blockchain assisted 6g-enabled iot-based big data driven healthcare cyber physical systems," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 4, pp. 6975–6986, 2024.
[24] J. Sun, Y. Yuan, M. Tang, X. Cheng, X. Nie, and M. U. Aftab, "Privacy-preserving bilateral fine-grained access control for cloud-enabled industrial iot healthcare," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6483–6493, 2022.
[25] S. Guan, Y. Cao, and Y. Zhang, "Blockchain-enhanced data privacy protection and secure sharing scheme for healthcare iot," *IEEE Internet of Things Journal*, 2024.
[26] G. Peng, A. Zhang, and X. Lin, "Patient-centric fine-grained access control for electronic medical record sharing with security via dual-blockchain," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 6, pp. 3908–3921, 2023.
[27] S. Potluri, "Policy-aware secure data governance in distributed information systems using explainable ai models," *International Journal of AI, BigData, Computational and Management Studies*, vol. 6, no. 3, pp. 1–10, 2025.
[28] B. Wang, R. Jiang, X. Pu, and H. Zhang, "An on-chain and off-chain collaborative data sharing and access control model for electronic medical records," *The Journal of Supercomputing*, vol. 81, no. 2, p. 396, 2025.
[29] K. Zhao, Z. Bao, Y. Zhang, and H. Lei, "Becaac: A blockchain and edge computing-assisted access control scheme for medical data sharing," *IEEE Internet of Things Journal*, 2025.
[30] X. Qu, Z. Yang, Z. Chen, and G. Sun, "A consent-aware electronic medical records sharing method based on blockchain," *Computer Standards & Interfaces*, vol. 92, p. 103902, 2025.
[31] G. Liu, H. Xie, W. Wang, and H. Huang, "A secure and efficient electronic medical record data sharing scheme based on blockchain and proxy re-encryption," *Journal of cloud computing*, vol. 13, no. 1, p. 44, 2024.
[32] A. Bian, D. Han, M. Cui, and D. Li, "Medical record information storage scheme based on blockchain and attribute role-based access control," *Computer Science and Information Systems*, vol. 21, no. 3, pp. 807–830, 2024.