systems

MDPI

Article

Zero-Knowledge-Based Policy Enforcement for Privacy-Preserving Cross-Institutional Health Data Sharing on Blockchain

Faisal Albalwy [1,2]

[1] Department of Cybersecurity, College of Computer Science and Engineering, Taibah University, Madinah 42353, Saudi Arabia; fbalwy@taibahu.edu.sa
[2] Energy, Industry, and Advanced Technologies Research Center, Taibah University, Madinah 42353, Saudi Arabia

Abstract

This study presents ZK-EHR, a decentralized access control framework designed to enable secure and privacy-preserving sharing of encrypted electronic health records across institutional boundaries. Unlike existing blockchain-based EHR access control systems that expose user identities on-chain or lack cryptographic privacy guarantees, ZK-EHR decouples authorization from identity disclosure by integrating zk-SNARK-based proofs with blockchain smart contracts to verify policy compliance without revealing user roles, affiliations, or credentials. The framework employs three differentiated actor roles—Patient (Data Owner), Doctor (Care Provider), and Researcher (Authorized Analyst)—with distinct policy-driven access workflows, a custom Groth16 zero-knowledge circuit for role-based constraint enforcement, and a modular architecture combining on-chain verification with off-chain encrypted storage via IPFS. Concrete design proposals for access revocation and replay attack prevention are introduced to address operational security requirements. The system was evaluated under multiple operational and adversarial scenarios. Experimental results indicate consistent on-chain verification latency (approximately 390 ms), reliable rejection of tampered submissions, and per-verification gas consumption of 216,631 gas. A comparative analysis against representative baseline systems demonstrates that ZK-EHR uniquely combines identity anonymity, on-chain cryptographic policy enforcement, and auditable encrypted record retrieval. These findings establish the feasibility of zk-SNARK-based access control for decentralized, verifiable, and privacy-aware EHR management.

Keywords: zero-knowledge proofs; zk-SNARKs; electronic health records (EHR); blockchain; smart contracts; access control; privacy-preserving systems; verifiable computing

Academic Editor: Anas M. Alsobeh

Received: 23 February 2026
Revised: 24 March 2026
Accepted: 1 April 2026
Published: 2 April 2026

1. Introduction

The digital transformation of healthcare has led to the widespread adoption of electronic health records (EHRs), which promise significant benefits in terms of data accessibility, care coordination, and clinical decision-making [1–3]. However, as EHR systems have become more interconnected and interoperable, the complexity of managing privacy, security, and access control has grown significantly [4–6]. The ability to access patient data securely and reliably across hospital systems, national health networks, and research institutions is critical for real-time diagnosis, personalized treatment, and public health monitoring. However, cross-institutional access remains limited due to fragmented infrastructure, inconsistent authorization protocols, and deep-rooted concerns over data privacy and regulatory compliance [7–10].

These challenges manifest concretely in several healthcare scenarios where existing access models prove insufficient. For example, when a patient undergoing treatment at one hospital is referred to a specialist at a separate institution, record sharing typically requires manual consent workflows that expose the patient's identity and diagnosis to intermediary administrative staff [1,2]. In multi-site clinical trials, researchers must verify participant eligibility across institutions without accessing full medical records, yet current systems cannot enforce such selective disclosure without a trusted third party [11,12]. Similarly, during public health emergencies, aggregating de-identified data from hospitals operating heterogeneous systems demands verifiable, privacy-preserving exchange mechanisms that existing frameworks do not adequately support [4,13].

Current access control systems for EHRs often rely on a centralized architecture in which a hospital or trusted identity provider enforces user permissions. While these models are easier to manage within a single institution, they suffer from well-known limitations when scaled across multiple entities. First, they introduce single points of failure that are vulnerable to attacks or service outages [14,15]. Second, they require users to entrust sensitive identity and access credentials to intermediary systems, which can be compromised or misused [4,16,17]. Third, centralized models typically fail to provide verifiable logs or tamper-proof audit trails, which are essential for ensuring legal accountability and compliance with privacy regulations, such as HIPAA and GDPR [18–20]. These issues are further exacerbated in decentralized or federated healthcare environments, where there is no common trust anchor across all participants.

Blockchain technology has emerged as a potential solution to some of these challenges by offering a decentralized, immutable, and auditable infrastructure for managing access rights. By leveraging smart contracts, blockchain can enforce access rules transparently and without reliance on a centralized gatekeeper. Several studies have demonstrated the use of blockchain for managing patient consent, recording access logs, and validating data provenance in healthcare systems [21–25]. However, despite these advantages, blockchain alone cannot satisfy the strong privacy guarantees required for handling medical data. Transactions and contract states are typically visible on-chain, and even if identity information is hashed or pseudonymized, adversaries can often infer sensitive metadata through traffic analysis or linkage attacks [13,26–28]. For instance, MedRec [29] has shown that repeated interactions with certain types of medical smart contracts (e.g., oncology or mental health departments) can unintentionally expose sensitive health conditions based solely on usage patterns. Similarly, repeated accesses to oncology-related record hashes may suggest a cancer diagnosis, even without access to the underlying data. This creates a fundamental tension: How can we build a system that is both verifiable (i.e., transparent and auditable) and privacy-preserving (i.e., hides all sensitive user attributes)?

To address this, we introduce ZK-EHR, a novel blockchain-orchestrated access control framework that uses zero-knowledge proofs (zk-SNARKs) to enforce privacy-preserving access to encrypted EHRs. Our approach enables authorized parties (e.g., doctors, researchers) to prove that they have permission to access a specific patient's record—without disclosing any personally identifiable information (PII), sensitive attributes (e.g., role), or access credentials. The proof is verified by an on-chain smart contract that immutably records the authorization outcome and minimal audit metadata (e.g., timestamp and decision) on the blockchain. While each verification transaction is inherently associated with a pseudonymous blockchain address (msg.sender), ZK-EHR does not reveal real-world identities, roles, or credentials on-chain. Upon successful verification, the system enables permissioned retrieval of the encrypted EHR stored on the InterPlanetary File System (IPFS).

Zero-knowledge proofs (ZKPs) are powerful cryptographic constructs that allow one party (the prover) to convince another party (the verifier) that a statement is true without

revealing any underlying information. zk-SNARKs (Succinct Non-Interactive Arguments of Knowledge), in particular, are attractive due to their compact proof sizes and fast verification times. Although recent studies have applied zk-SNARKs to selected healthcare scenarios, such as anonymized device authentication, hybrid access models, and privacy-preserving insurance contracts [30–32], these implementations remain fragmented and are often application-specific. What remains lacking is a unified, scalable framework that enables privacy-preserving access control, verifiable authorization, and decentralized encrypted record retrieval across institutional boundaries. In this work, we address this gap by introducing ZK-EHR—an integrated architecture that brings together zk-SNARK authorization, blockchain-based verification, and IPFS-based EHR storage.

This paper makes the following key contributions:

- We propose a privacy-preserving architecture for secure EHR access that integrates zero-knowledge proof systems with blockchain-based smart contracts and decentralized file storage.
- We provide a formal security model and mathematical modeling of the access control logic, strictly proving the system's completeness, soundness, and zero-knowledge properties to ensure high-level cryptographic rigour.
- We design and implement a zk-SNARK circuit tailored for access authorization verification without identity leakage and deploy its verifier on an Ethereum compatible blockchain.
- We present a detailed architecture and interaction workflow, including sequence diagrams showing how requesting parties request and retrieve data while preserving confidentiality.
- We evaluate the system across key metrics, including proof generation time, verification latency, and IPFS upload latency, and validate robustness under adversarial scenarios (proof tampering, input integrity tests, and concurrent request handling), demonstrating feasibility in a simulated healthcare environment.

The rest of this paper is structured as follows: Section 2 reviews related literature on access control in EHR systems, blockchain-based models, and zk-SNARK applications in healthcare. Section 3 presents the ZK-EHR system architecture, role-differentiated access model, formal security analysis with proposed extensions for revocation and replay resistance, implementation details, and experimental setup. Section 4 reports the evaluation results across four operational and adversarial scenarios, including gas cost analysis. Section 5 discusses the findings in context, including a comparative analysis with existing approaches, security threat modelling, deployment considerations, and limitations. Section 6 concludes the paper and proposes future directions.

2. Literature Review

Prior work in privacy-preserving access control for EHRs spans traditional models, such as role-based access control (RBAC) and attribute-based access control (ABAC), blockchain-based systems, and more recently, zero-knowledge cryptography. This section reviews the evolution of these techniques, focusing on their limitations in supporting scalable, verifiable, and private access in decentralized healthcare environments.

2.1. Access Control in EHR Systems

Access control is a foundational requirement in EHR systems to ensure secure, policy-compliant access to sensitive patient data. Traditional models, such as RBAC and ABAC, have been widely adopted in healthcare infrastructures. However, these models exhibit several limitations when applied to complex, multi-institutional environments. RBAC, while easy to implement and manage, is often criticized for its lack of flexibility. For in-

stance, Singh et al. [33] proposed an advanced administrative model called RAMHAC to manage heterogeneous access control policies combining RBAC and ABAC. While their framework improves administrative scalability and includes formal security analysis, it remains vulnerable to the role-explosion problem—a condition where organisations must define an excessive number of fine-grained roles to accommodate every possible combination of access permissions. In large-scale or multi-institutional systems, this leads to increased administrative complexity, scalability bottlenecks, and policy inconsistencies, making RBAC increasingly difficult to manage.

Furthermore, RBAC lacks support for context-aware access decisions. To address these shortcomings, ABAC offers greater policy expressiveness by incorporating user, object, and environmental attributes into access decisions. Adda and Aliane [34] introduced the HoBAC model, which extends ABAC with hierarchical and dynamic attribute propagation. Although HoBAC provides high granularity and flexibility, especially for dynamic healthcare scenarios, its adoption is hindered by increased policy management complexity and computational overhead. OAuth-based access control has also been investigated in the healthcare domain. Saha and Neogy [35] developed a permission-based model leveraging OAuth combined with encryption and digital signatures to protect cloud-based personal health data. Despite supporting revocation and collusion resistance, their model depends on a centralized data administrator, contradicting the decentralization goals of modern EHR systems.

Recent work has explored combining ABAC with blockchain to improve transparency and cross-institutional trust. Oliveira et al. [11] proposed SmartAccess, a blockchain-based system that encodes ABAC policies as smart contracts on a permissioned ledger. Their framework enables dynamic, auditable, and context-aware access control. However, SmartAccess still inherits ABAC's complexity and lacks native mechanisms for privacy preservation or metadata protection. Despite these advancements, existing access control models still fall short of satisfying the stringent requirements of decentralized, privacy-aware healthcare environments. RBAC is insufficiently flexible, ABAC imposes administrative burdens, and OAuth introduces centralization. Even blockchain-based models, such as SmartAccess, lack native privacy guarantees. These gaps highlight the need for a unified architecture that supports decentralized, verifiable, and privacy-preserving access control—an objective that underpins the development of ZK-EHR.

2.2. Blockchain-Based EHR Access Models

The use of blockchain technology in EHR management has gained traction due to its potential to provide immutable audit trails, decentralized data governance, and tamper-resistant patient consent mechanisms. A wide range of frameworks leverage smart contracts and permissioned blockchains to automate access control and enable transparent data sharing across healthcare providers. However, despite these advantages, most current models fall short of guaranteeing strong privacy and identity protection, especially against metadata inference and public traceability.

Psarra et al. [12] proposed a proactive access control model using a permissioned Hyperledger Fabric network combined with fuzzy logic and LSTM for emergency scenarios. While their framework improves responsiveness and dynamic authorization, it relies on predefined user roles and permanently records access-related metadata on-chain, potentially exposing user behavior to inference attacks. Pampattiwar and Chavan [36] introduced a scalable blockchain-based model that integrates genetic algorithms with policy-based access mechanisms. Although the model emphasizes performance and data-level confidentiality, it lacks identity anonymization and relies on static ACLs, which limit policy flexibility and hinder revocation in dynamic environments.

Yuan et al. [37] proposed an access control framework that employs proxy re-encryption to secure patient data and facilitate controlled delegation through smart contracts—self-executing programs deployed on a blockchain that automatically enforce predefined rules and policies. While effective in data protection, this model assumes trusted intermediaries and exposes access behavior and policies on the blockchain, undermining privacy guarantees. Oliveira et al. [11] developed a blockchain-based ABAC system that uses smart contracts to enforce dynamic policies for cross-institutional EHR access. Though it offers contextual awareness and auditability, it inherits the complexity of ABAC and does not prevent information leakage from public policy evaluation and access logs. Despite their strengths, these blockchain-based access control models rely on transparent logs and static permission configurations; furthermore, they lack cryptographic privacy-preserving capabilities. This highlights the need for a new approach that combines verifiability with strong privacy guarantees—an objective addressed by the proposed ZK-EHR framework, which leverages zk-SNARKs to ensure provable private access control in decentralized healthcare systems.

2.3. zk-SNARKs in Healthcare Applications

ZKPs, particularly zk-SNARKs, have emerged as powerful cryptographic tools for enforcing privacy in digital systems. In healthcare, these techniques enable secure, policy-compliant access to EHRs without exposing sensitive information. Several studies have integrated zk-SNARKs with blockchain infrastructures to strengthen identity management, access verification, and data integrity in decentralized health ecosystems. However, these efforts remain limited in scope, offering partial solutions that do not fully address the privacy-preserving needs of end-to-end EHR access workflows.

Several identity-focused frameworks have demonstrated the utility of zk-SNARKs for secure authentication. For example, Bai et al. [31] proposed Health-zkIDM, a decentralized identity system that enables patient verification without disclosing personal details. Similarly, Luong and Park [38] introduced a blockchain-based identity management framework that ensures anonymous identity proofing using zk-SNARKs. While these systems offer strong privacy for user authentication, they do not support full access control or secure record retrieval, leaving critical gaps in the broader EHR access lifecycle.

Expanding into device-centric applications, Luong and Park [30] applied zk-SNARKs to IoT medical environments to authenticate data streams from connected health devices. However, this work is confined to real-time data authentication and does not extend to structured EHR access or policy enforcement. Likewise, Zhiguo et al. [39] proposed zk-AuthFeed, a protocol for authenticating external data feeds into smart contracts. Although it demonstrates efficient off-chain proof generation and on-chain verification, its application is limited to oracle systems and does not address record-level access control.

Other works have emphasized data privacy and sharing but lack cryptographic privacy guarantees. Chenthara et al. [40] introduced Healthchain, a blockchain-based EHR sharing system focused on secure storage, but it does not utilize zk-SNARKs or zero-knowledge proof mechanisms. Liu et al. [41] proposed a blockchain-aided data-sharing model that prioritizes access control and encryption but still exposes behavioral metadata and user identity information during access operations. More recently, two notable studies have explored zk-SNARKs in the context of full EHR access. A study by Myeong and Kim [42] presented a zk-SNARK-enabled Ethereum-based protocol for privacy-preserving health data sharing. The system offers efficient proof verification and supports FHIR-based interoperability. However, the architecture focuses solely on proof-based authorization and does not integrate identity verification, smart contract governance, or encrypted data access into a unified modular framework.

Collectively, these studies underscore a key gap in the literature: Existing zk-SNARK-based systems typically address isolated aspects of healthcare privacy, such as identity, device verification, and data encryption. However, they lack a comprehensive solution that combines verifiable access control with end-to-end privacy for encrypted health records within a single architectural framework. ZK-EHR addresses this gap directly by integrating zero-knowledge identity verification, policy-based proof validation, and encrypted record retrieval within a unified blockchain framework. Unlike prior approaches, ZK-EHR ensures that access rights are cryptographically verified, records remain encrypted until valid proof is submitted, and all access events are logged immutably—without leaking user identity or access patterns. This combination of privacy-by-default and verifiability-by-design distinguishes ZK-EHR as a novel, scalable, and secure architecture for decentralized healthcare access.

2.4. Theoretical Foundations of Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs) are cryptographic protocols that enable a prover to convince a verifier that a statement is true without revealing any information beyond the validity of the statement itself. Formally, a ZKP system for a relation $\mathcal{R}$ satisfies three properties: completeness (an honest prover always convinces the verifier), soundness (a dishonest prover cannot convince the verifier of a false statement except with negligible probability), and zero-knowledge (the verifier learns nothing beyond the truth of the statement). In the non-interactive setting, a single proof message suffices for verification without further interaction, making non-interactive ZKPs suitable for on-chain verification in blockchain environments.

Among non-interactive ZKP systems, zk-SNARKs (Succinct Non-Interactive Arguments of Knowledge) are particularly attractive for blockchain applications due to their compact proof size (approximately 128 bytes compressed or 256 bytes uncompressed for Groth16 over BN254, consisting of two $\mathbb{G}_1$ points and one $\mathbb{G}_2$ point) and constant-time verification independent of the computation size. The Groth16 proving system [43], which ZK-EHR employs, operates in three phases. During Setup, a circuit-specific structured reference string (SRS) is generated through a trusted ceremony such as the Powers of Tau protocol, producing a proving key and a verification key. During Prove, the prover uses the proving key, the public inputs, and the private witness to generate a succinct proof that the witness satisfies the circuit constraints. During Verify, the verifier checks the proof using the verification key and the public inputs through a constant number of elliptic curve pairing operations, without access to the witness.

The computational logic to be proven is expressed as an arithmetic circuit over a finite field $\mathbb{F}_p$, consisting of addition and multiplication gates. This circuit is compiled into a Rank-1 Constraint System (R1CS)—a set of equations of the form $A \cdot s \circ B \cdot s = C \cdot s$, where A, B, C are coefficient matrices and s is the combined public and private input vector. Each constraint enforces one multiplicative relationship, and the total constraint count directly determines proving time and memory requirements.

A critical design decision in ZK-EHR is the selection of the Poseidon hash function for in-circuit hashing. Unlike general-purpose hash functions such as SHA-256, which require thousands of R1CS constraints to encode their bitwise operations, Poseidon operates natively over the prime field $\mathbb{F}_p$ and is designed specifically for arithmetic circuit environments. Its algebraic structure—based on substitution-permutation networks with low-degree S-boxes over $\mathbb{F}_p$—results in substantially fewer constraints (typically 200–400 R1CS constraints for a 2-input hash depending on the parameterization (t, R_F, R_P), compared to 25,000 for SHA-256 [44]). This efficiency makes Poseidon the preferred choice for applications where in-circuit hashing is a performance bottleneck, as is the case in ZK-EHR's

policy compliance verification. For further cryptographic background, the reader is referred to [43,45].

3. Materials and Methods

This section presents the design, implementation, and evaluation methodology of the ZK-EHR framework. The architecture adopts a modular design comprising three principal layers: an off-chain policy evaluation and proof generation layer, an on-chain cryptographic verification layer, and a decentralized encrypted storage layer. The design is guided by three principles: (1) privacy by default—no user identity, role, or credential is disclosed during access authorization; (2) verifiability by design—every access decision is cryptographically verified on-chain and immutably logged; and (3) decentralized trust—no single entity controls access decisions. The framework targets access control for encrypted health records within an Ethereum-compatible blockchain deployment as its primary scope; cross-chain interoperability and production-grade key management are identified as future work directions (Section 5.5). The following subsections detail the system architecture and role model (Section 3.1), formal security model and proposed security extensions (Section 3.2), implementation details including the zk-proof lifecycle and smart contract integration (Section 3.3), and experimental setup and evaluation metrics (Section 3.4).

3.1. Proposed System: ZK-EHR Architecture

3.1.1. System Overview

The proposed system, ZK-EHR, is a privacy-preserving, blockchain-based access control framework designed for the secure and verifiable sharing of electronic health records (EHRs) across decentralized healthcare infrastructures. The system leverages zk-SNARKs to enable requesting parties to prove authorization to access sensitive medical data without disclosing their identity or access credentials.

The development of ZK-EHR was motivated by the limitations observed in current access control models that rely on centralized identity verification, static permission lists, or conventional blockchain-based smart contracts without privacy guarantees. While prior systems have addressed identity management, auditability, or data sharing separately, they have yet to integrate these capabilities into a unified architecture that ensures privacy-by-default and verifiability-by-design.

The core idea of ZK-EHR is to decouple access verification from identity disclosure while enabling seamless, decentralized enforcement of access control policies. When a requesting party (e.g., a doctor or researcher) requests access to an encrypted health record, the system consults a dedicated Policy Engine to evaluate whether the requesting party's private attributes—such as role (e.g., cardiologist), affiliated institution (e.g., hospital ID), and clearance level—satisfy the access conditions defined in the policy. These conditions may specify, for example, that access is granted only to cardiologists from certified institutions requesting data related to a specific department.

Based on this evaluation, a zero-knowledge proof is generated through the ZK circuit, demonstrating that the requester meets the policy without revealing any of the underlying attribute values. No user identity or role is revealed to the system. This proof serves as the authorization token and is submitted to an on-chain smart contract verifier. Upon successful verification, the contract permits access to the encrypted health record stored off-chain using IPFS or a similar decentralized storage backend.

The system's design ensures:

- Confidentiality by hiding identities and policies from both storage providers and blockchain observers.
- Integrity by enforcing policy-compliant access through verifiable proofs.

- Scalability through off-chain proof computation and lightweight on-chain verification.
- Auditability via immutable logging of authorized access without revealing sensitive metadata.

Figure 1 presents a high-level architectural view of ZK-EHR, depicting its major components and interactions, including the integration of the Policy Engine. The diagram begins with a requesting party (e.g., a doctor) submitting an access request, which is evaluated by the Policy Engine to determine whether the requesting party's attributes satisfy the predefined access conditions. These conditions are passed to the zk-SNARK Circuit Generator to produce cryptographic proof that enforces the policy without revealing any private information. The proof is submitted to a Verifier Smart Contract on the blockchain, which verifies its validity and records the authorization event. If successful, the blockchain signals access approval, allowing the requesting party to retrieve the corresponding encrypted health record from IPFS or another off-chain storage system. The following sections detail the internal components and operational workflow of the system.

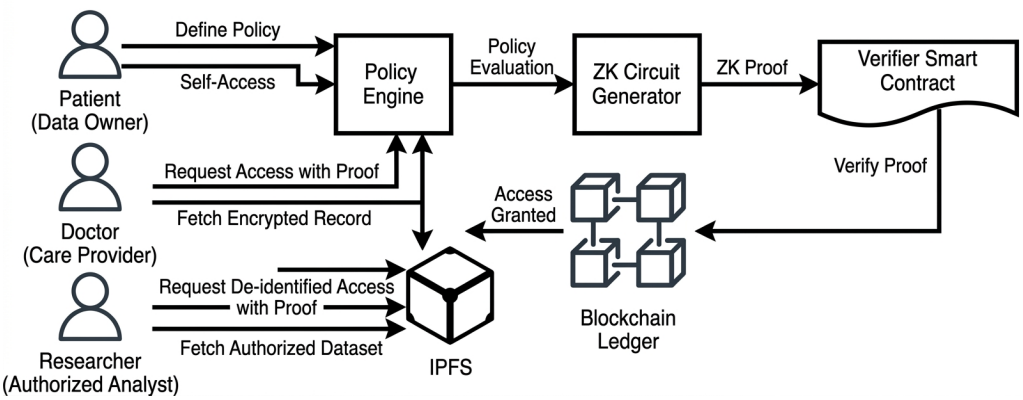

Figure 1. High-level architecture of ZK-EHR showing differentiated actor roles (Patient, Doctor, Researcher) and their interactions with the framework components.

3.1.2. Architectural Components

The ZK-EHR system comprises six core components that work in coordination to ensure secure, privacy-preserving, policy-compliant access to encrypted EHRs. Each component has a distinct role in the end-to-end process of request evaluation, proof generation, verification, and record retrieval. Table 1 summarizes these components and their corresponding functions.

Table 1. ZK-EHR System Component Summary. This table details the core functional modules of the proposed framework and their respective responsibilities.

Component	Description
Patient (Data Owner)	Owns health records stored on IPFS. Defines and publishes access policies specifying who may access their data and under what conditions. May also initiate self-access requests to retrieve their own encrypted records.
Doctor (Care Provider)	Initiates access requests to patient records for clinical purposes. Proves possession of required attributes (e.g., role = cardiologist, institution = Hospital A, clearance $\geq$2) via a zero-knowledge proof without revealing identity. Access scope is limited to records matching the policy conditions set by the patient.

Table 1. *Cont.*

Component	Description
Researcher (Authorized Analyst)	Requests access to de-identified or aggregate health data for research purposes. Proves institutional affiliation and ethics approval status via a zero-knowledge proof. Access scope may be restricted to de-identified subsets or aggregate views as defined by the data owner's policy.
Policy Engine	Evaluates access policies and transforms them into policy conditions.
zk-SNARK Circuit Generator	Generates a zk-SNARK proof that encodes policy compliance without exposing data.
Verifier Smart Contract	Verifies the zero-knowledge proof and authorizes access on-chain.
Blockchain Ledger	Logs verification results immutably to support auditability.
Encrypted Record Store (IPFS)	Stores encrypted EHRs off-chain and provides access upon authorization.

Patient (Data Owner): The Patient is the primary data owner in ZK-EHR. Each patient's encrypted health records are stored off-chain on IPFS, and the patient defines access policies governing who may retrieve them. A policy specifies the required attributes a requesting party must possess, for example, that the requester holds the role of cardiologist at a specific accredited institution with a minimum clearance level. The patient defines the policy conditions whose encoded representation is used as the public basis against which zero-knowledge proofs are verified on-chain. Patients may also initiate self-access requests, in which case the proof demonstrates ownership rather than third-party authorization. In the present implementation, the data owner's system is responsible for generating the per-record symmetric content key used to encrypt the record prior to off-chain storage.

Doctor (Care Provider): The Doctor is the primary clinical requester. When a doctor needs to access a patient's encrypted health record, the doctor initiates an access request. The doctor's identity attributes (e.g., role, institution, department, clearance level) are pre-bound to a verifiable credential issued by a trusted authority. Rather than submitting these attributes in plaintext, the doctor uses them as private inputs to the zk-SNARK circuit, generating a proof that the credential satisfies the patient's access policy. The proof is verified on-chain without revealing the doctor's identity or institutional affiliation.

Researcher (Authorized Analyst): The Researcher is a specialized requester whose access is typically restricted to de-identified or aggregate data subsets. Researchers prove institutional affiliation and ethics approval status through a zero-knowledge proof, analogous to the doctor's workflow but with potentially different policy constraints, for instance, access limited to de-identified diagnostic codes or contingent on an approved IRB protocol number. The researcher's proof circuit encodes these constraints, ensuring access is granted only to the authorized data scope.

Policy Engine: This module evaluates the access request against predefined policies, which may follow role-based, attribute-based, or context-aware models. The engine transforms the access control logic into Boolean policy conditions that are passed to the zk-circuit generator to be encoded into the zero-knowledge proof, ensuring that the access decision is enforced cryptographically. For example, a typical policy condition can be formalized as:

$$\mathcal{P}(r_u, d_u) = \begin{cases} 1 & \text{if } r_u = r_p \wedge d_u = d_p \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where r_u and d_u denote the role and department of the user, and r_p and d_p represent the required values defined in the policy. This logical expression is compiled into a constraint system and verified within the zero-knowledge circuit, allowing access decisions to be validated without revealing any private information on-chain.

Access policies in ZK-EHR are not static; data owners (Patients) may update their policies at any time to reflect changing consent preferences, institutional agreements, or regulatory requirements. A policy update proceeds as follows. First, the data owner formulates the revised policy conditions (e.g., extending access to a new department or raising the required clearance level). Second, the revised policy is hashed and the new hash $H(a'_{req})$ is published on-chain via a policy update transaction, replacing the previous policy hash associated with the relevant record or record group. Third, the Policy Engine is notified of the updated conditions so that future proof generation uses the new public inputs. Because the zk-SNARK circuit encodes a generic hash-equality check rather than hard-coding specific attribute values, policy updates that change attribute values (e.g., from clearance $\geq$2 to clearance $\geq$3) do not require circuit recompilation—only the public input (the policy hash) changes. However, policy updates that introduce new attribute types not currently supported by the circuit (e.g., a time-of-day restriction) would require circuit extension and recompilation. In-flight requests—proofs generated against the previous policy hash but not yet submitted—will be rejected by the Verifier Smart Contract because the on-chain policy hash will no longer match the proof's public input. This fail-safe behavior ensures that only proofs generated against the current policy are accepted.

Equation (1) illustrates a simple policy requiring exact matches on two attributes. In practice, healthcare access policies often involve multiple attributes, threshold conditions, and logical combinations. The circuit design underlying ZK-EHR supports such complexity through the composition of hash-equality constraints and arithmetic comparisons over the finite field.

A multi-attribute conjunction policy requiring that the requester holds a specific role at a designated institution with a minimum clearance level can be expressed as:

$$\mathcal{P}(r_u, d_u, c_u) = \begin{cases} 1 & \text{if } H(r_u, d_u) = H(r_p, d_p) \wedge c_u \geq c_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

This is encoded in the circuit as a hash-equality constraint for the role–department pair (using Poseidon) and an arithmetic comparison constraint ($c_u - c_{\min} \geq 0$, verified via a range-check gadget).

A conditional access policy differentiating by role—granting doctors full-record access and researchers access only to de-identified data—can be expressed as:

$$\mathcal{P}(r_u, scope) = \begin{cases} 1 & \text{if } (r_u = \text{Doctor} \wedge scope = \text{full}) \\ & \quad \vee\ (r_u = \text{Researcher} \wedge scope = \text{de-identified}) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

This is encoded as two parallel constraint branches with an OR-composition implemented through a selector variable, adding a small number of auxiliary constraints per branch.

A time-bounded access policy granting access only during a specified validity window can be expressed as:

$$\mathcal{P}(r_u, d_u, t) = \begin{cases} 1 & \text{if } H(r_u, d_u) = H(r_p, d_p) \wedge t_{\text{start}} \leq t_{\text{current}} \leq t_{\text{end}} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The time bounds are encoded as public inputs, and the circuit enforces arithmetic range constraints on the block timestamp. Since block timestamps are subject to minor manipulation by validators (typically within a 15-s window on Ethereum), the tolerance window should account for this variance in practice.

These examples demonstrate that ZK-EHR's circuit framework can express a broad range of real-world access policies. The constraint count scales linearly with the number of attributes and conditions, preserving Groth16 proving efficiency.

zk-SNARK Circuit Generator: This component constructs a cryptographic proof using the requester's attributes and the policy logic received from the Policy Engine. The proof demonstrates, in zero-knowledge, that the requester satisfies the access policy without exposing sensitive identity or policy information.

Verifier Smart Contract: Deployed on the blockchain, this contract receives the zk-SNARK proof and verifies its validity. It enforces access-control decisions without requiring access to the requester's identity or underlying plaintext data. Successful and unsuccessful verification outcomes are reflected through `AccessGranted` and `AccessDenied` events, enabling auditable access signaling without storing sensitive data on-chain.

Blockchain Ledger: Once verification is executed, the authorization outcome is reflected through the corresponding on-chain transaction and emitted access event. This provides auditability and accountability of access requests while preserving participant privacy. In the current system, the blockchain layer is used for proof verification and auditable access signaling rather than for storing plaintext medical data or content-encryption keys.

Encrypted Record Store (IPFS): The actual health records are stored off-chain in encrypted form using a decentralized storage system, such as IPFS. In the present implementation, each plaintext EHR file is encrypted locally using AES-256-GCM prior to upload. A fresh 256-bit symmetric content key and a unique initialization vector (IV) are generated for each record, and only the resulting ciphertext artifact is stored on IPFS.

Content-encryption keys remain strictly off-chain. Following successful on-chain proof verification, access authorization is reflected through the emitted verification event, after which the corresponding record key is released in wrapped form using the requester's public key via RSA-OAEP with SHA-256. The authorized requester unwraps the key locally and decrypts the retrieved ciphertext.

This design ensures that plaintext medical data and content-encryption keys are never exposed on-chain, preserving confidentiality while maintaining verifiable access control. Furthermore, the architecture supports differentiated access scopes across roles, consistent with the role-based policy model defined above.

3.1.3. Operational Workflow

The operational workflow of ZK-EHR comprises a sequence of interactions among its components that collectively ensure secure, policy-compliant, privacy-preserving access to encrypted EHRs. The process is initiated by the requesting party and proceeds through access evaluation, proof generation, on-chain verification, and record retrieval, as illustrated in Figure 1. Figure 2 further depicts this process as a time-ordered sequence of messages exchanged between system components. The core steps are described below:

1. Access Request Initiation: A requesting party (e.g., a doctor or researcher) initiates an access request to a specific EHR. The request includes relevant identity attributes (e.g., role, department, organization) without revealing the requesting party's full identity. These attributes are forwarded to the Policy Engine for access evaluation.
2. Policy Evaluation: The Policy Engine evaluates the requesting party's attributes against predefined access control rules. These rules may follow role-based or attribute-based models and are transformed into a policy condition statement suitable for

zk-SNARK circuit construction. The result is a logical expression defining access entitlement without leaking policy content.

3. Zero-Knowledge Proof Generation: The zk-Prover receives the evaluated policy conditions along with the requesting party's attributes and constructs a cryptographic proof. This proof demonstrates compliance with the access policy without disclosing the underlying data, the requesting party's identity, or the policy itself.
4. On-Chain Proof Verification: The generated proof is submitted to a Verifier Smart Contract deployed on the blockchain. This contract verifies the proof's validity using public verification keys and updates the blockchain ledger to log the results of the verification. No sensitive information is ever stored on-chain.
5. Access Grant and Record Retrieval: Upon successful verification, the blockchain returns an access granted signal to the requesting party. The requesting party then proceeds to fetch the corresponding EHR from the decentralized storage (e.g., IPFS). The record is decrypted locally using an authorized decryption key tied to the verified credentials.
6. Auditability and Accountability: Every verification event and access authorization is immutably logged on the blockchain, ensuring traceability without compromising requesting party privacy. This guarantees compliance with data protection standards and supports post-access auditing.

   Each proof verification in ZK-EHR is a stateless, independent blockchain transaction. The Verifier Smart Contract does not maintain session state, shared locks, or mutable counters between verification calls; it evaluates the submitted proof against the verification key and the provided public inputs, emits the appropriate event, and appends an access log entry. Consequently, multiple requesting parties may submit proofs for the same or different records simultaneously without creating race conditions or data conflicts. The Ethereum transaction ordering mechanism ensures deterministic sequencing of concurrent submissions. No additional synchronization mechanism beyond the blockchain's native transaction processing is required.

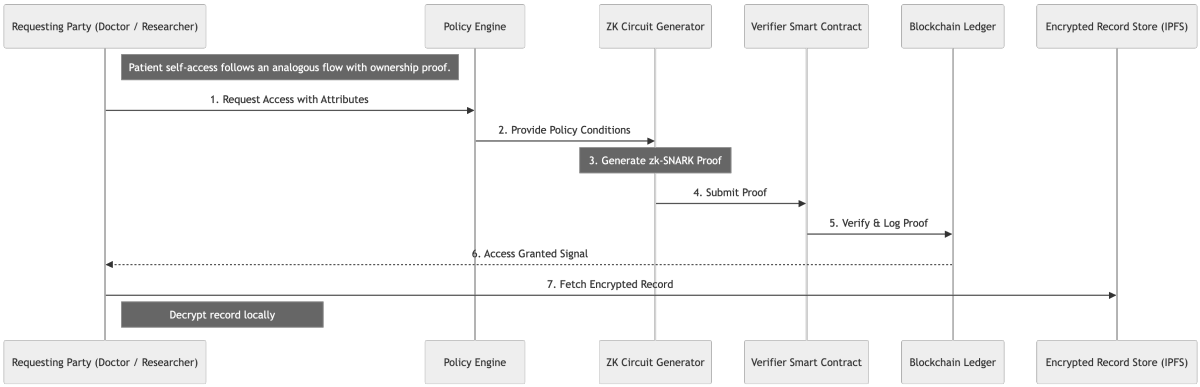

Figure 2. Sequence diagram of ZK-EHR operations showing the role-differentiated workflow, where a requesting party (Doctor or Researcher) submits a proof-based access request that is evaluated by the Policy Engine, verified on-chain, and followed by secure retrieval of encrypted records from IPFS. Solid arrows indicate request/response interactions, while dashed lines represent verification or acknowledgment signals. Patient self-access follows an analogous flow with ownership proof.

3.2. Formal System Model and Security Analysis

To provide a rigorous foundation for the ZK-EHR framework, we formalize the access control logic and analyze the security properties of the underlying zero-knowledge proof system.

Definition 1 (**Access Policy Formulation**). *Let $\mathcal{A}$ be the set of user attributes, where an attribute vector $a_u = \{r_u, d_u, \dots\} \in \mathcal{A}$ represents the user's role, department, and other credentials. Let $\mathcal{P}$ be the access policy defined by the data owner, specified as a set of required attribute values $a_{req} = \{r_{req}, d_{req}, \dots\}$. The access authorization function $F : \mathcal{A} \times \mathcal{A} \to \{0, 1\}$ is defined as:*

$$F(a_u, a_{req}) = \begin{cases} 1 & \text{if } H(a_u) = H(a_{req}) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $H : \{0,1\}^* \to \mathbb{F}_p$ is a collision-resistant cryptographic hash function (implemented via Poseidon in our circuit).

3.2.1. zk-SNARK Relation

The ZK-EHR system relies on a relation $\mathcal{R}$ for the arithmetic circuit C. We define the relation $\mathcal{R}$ over a finite field $\mathbb{F}_p$ as the set of pairs (x, w), where x is the public statement (policy constraints) and w is the private witness (user attributes).

$$\mathcal{R} = \{(x, w) \mid C(x, w) = 1\} \tag{6}$$

Here, the public input x corresponds to the hashed policy requirements $H(a_{req})$, and the private witness w includes the user's raw attributes a_u and their secret signing key sk.

3.2.2. Security Theorems

We assert that the ZK-EHR protocol satisfies the three fundamental properties of zero-knowledge proof systems: Completeness, Soundness, and Zero-Knowledge.

Theorem 1 (**Completeness**). *For any authorized user possessing valid attributes a_u such that $F(a_u, a_{req}) = 1$, the user can generate a proof π that is accepted by the Verifier smart contract with probability 1.*

Proof. The system employs the Groth16 proving scheme. If the user's attributes a_u satisfy the circuit constraints (i.e., the hash pre-image check holds), then the generated proof π satisfies the pairing equation $e(\pi_A, \pi_B) = e(\pi_C, \pi_\delta) \cdot e(x, \pi_\gamma)$ in the underlying elliptic curve groups. Since the circuit logic C faithfully implements the function F, an honest prover will always produce a valid witness w such that $(x, w) \in \mathcal{R}$. □

Theorem 2 (**Computational Soundness**). *For any adversary $\mathcal{A}$ not in possession of valid attributes satisfying the policy, the probability of generating a valid proof π is negligible, assuming the hardness of the Discrete Logarithm Problem (DLP) and the Knowledge of Exponent (KoE) assumption.*

Proof. This property follows from the soundness of the Groth16 protocol. A valid proof implies the knowledge of a witness w (i.e., the user's private attributes) that satisfies the circuit constraints. Since the attributes are bound to the user's identity through a digital signature check within the circuit, an adversary cannot forge a proof without knowing the corresponding private key or finding a hash collision for H, which is computationally infeasible. Thus, the system prevents unauthorized access. □

Theorem 3 (**Zero-Knowledge**). *The generated proof π reveals no information about the user's private attributes a_u or identity beyond the fact that they satisfy the policy $\mathcal{P}$.*

Proof. The zero-knowledge property is guaranteed by the randomization of the proof elements in the Groth16 construction. Specifically, the proof generation process involves

sampling random scalars $\delta, \gamma \in \mathbb{F}_p$ that mask the actual values of the witness w in the resulting elliptic curve points. Consequently, the distribution of proofs generated by a real prover is statistically indistinguishable from proofs generated by a simulator that does not know w. This ensures that the Verifier (the smart contract) learns only the single bit of information: that the policy is satisfied. □

3.2.3. Security Extensions: Revocation and Replay Resistance

The mechanisms described below build upon the circuit extensions detailed in Section 3.3.2; the reader may wish to consult that section first for the relevant signal structure.

The core Groth16-based protocol described in Sections 3.2.1 and 3.2.2 provides completeness, soundness, and zero-knowledge for individual proof verification events. To achieve comprehensive security in a deployed access control system, two additional properties are required: revocation resistance, ensuring that revoked credentials cannot produce accepted proofs, and replay resistance, ensuring that a valid proof cannot be resubmitted for unauthorized repeat access. The current prototype does not implement either mechanism, and both are acknowledged as limitations in Section 5.5. Below, we present concrete design proposals for each, demonstrating their feasibility and architectural compatibility with ZK-EHR. Full implementation and performance evaluation are deferred to future work.

For revocation resistance, we propose a two-layer mechanism. The first layer uses credential expiry: each credential includes an expiry timestamp, and the zk-SNARK circuit is extended with a constraint verifying that the current block timestamp falls within the credential's validity window. Expired credentials cannot produce a valid proof, as this property inherits directly from the soundness of the Groth16 protocol (Theorem 2): the expiry constraint is embedded in the circuit and cannot be bypassed without generating a proof for a false statement. The second layer addresses immediate revocation, such as when a doctor's hospital privileges are terminated before credential expiry. The Verifier Smart Contract maintains a lightweight on-chain revocation registry mapping credential identifiers to revocation status. Write access to this registry is restricted to authorized credential issuers via a permissioned function, preventing denial-of-service attacks through unauthorized revocation. Before performing the cryptographic pairing check, the contract queries this registry and rejects the proof if the credential has been flagged as revoked, regardless of cryptographic validity. Together, these layers provide both routine time-based expiry and emergency immediate revocation.

For replay resistance, we propose a nullifier-based mechanism. Each proof is accompanied by a unique nullifier—a deterministic, non-reversible value derived inside the zk-SNARK circuit from the prover's secret key and access request metadata. The nullifier is produced as a public output of the circuit, ensuring it is bound to the specific request without revealing private attributes. The Verifier Smart Contract maintains a registry of previously used nullifiers and rejects any proof whose nullifier has already been recorded. An adversary who replays a captured proof submits an identical nullifier, which is rejected. The adversary cannot produce a valid proof with a different nullifier for the same request without the prover's secret key, because the nullifier computation is enforced inside the circuit and the soundness of Groth16 (Theorem 2) prevents proofs with incorrectly computed outputs. The nullifier is request-specific and does not link across different access requests by the same requester, preserving unlinkability. This mechanism adds minimal overhead: one registry lookup and one conditional write per verification.

Both extensions integrate with existing ZK-EHR components without modifying the core Groth16 verification logic. The credential expiry constraint adds approximately 200 R1CS constraints, consistent with the circuit sizing in Section 3.3.2. The revocation and

nullifier registries are implemented as mappings in the Verifier Smart Contract, queried before the pairing check. Neither mechanism alters the zero-knowledge property: the revocation check occurs at the contract level, and the nullifier is derived from request-specific metadata without revealing private attributes or enabling cross-request linkage. These extensions are proposed at the design level. Their full integration, formal verification, and performance evaluation—including impact on gas costs and registry storage growth—are noted as future work.

3.3. Implementation Details

To realize the ZK-EHR framework in a reproducible and modular fashion, a fully functional prototype was developed using industry-standard tools for zero-knowledge proof generation, smart contract deployment, and decentralized storage. The implementation includes zk-SNARK circuits, Solidity verifier contracts, evaluation scripts, and performance measurement utilities. All source code and experimental assets are publicly available at GitHub [46] under the MIT License, ensuring transparency and facilitating reproducibility by the research community.

3.3.1. Development Environment Setup

To implement the ZK-EHR framework, a secure and modular local development environment was configured. The development was conducted on a macOS system with Apple Silicon (M1/M2 architecture), using a combination of cryptographic, blockchain, and decentralized storage tools.

The zero-knowledge proof circuits were designed and compiled using Circom v2.1.2, installed and built from source using Rust and Cargo to ensure compatibility with the latest circuit features. For proof generation and verification, the SnarkJS v0.6.11 library was employed, enabling Groth16 proving systems and seamless conversion of circuits into Solidity-compatible verifiers.

A local Ethereum-compatible blockchain was deployed using Ganache v7.9.2, providing deterministic account generation and fast block finality for testing. Smart contracts were developed in Solidity v0.8.28 and compiled using Hardhat v2.24.3, which also facilitated local deployments and unit testing with Mocha/Chai and Ethers.js v6.14.0.

For decentralized storage, IPFS (Kubo v0.18.1) was installed locally and configured to run as a daemon. Encrypted EHR records were uploaded to IPFS, and their corresponding content identifiers (CIDs) were used as off-chain references. During access requests, these CIDs were fetched by scripts integrated with the ipfs-http-client package. Table 2 illustrates the toolchain used across different layers of the system.

Table 2. Development Toolchain and Technology Stack.

Component	Tool/Version	Purpose
ZK Circuit Design	Circom v2.1.2	Circuit definition and compilation
ZK Proof Handling	SnarkJS v0.6.11	Setup, proof generation, verifier export
Blockchain Environment	Ganache v7.9.2	Local Ethereum-compatible testing
Smart Contract Framework	Hardhat v2.24.3	Compilation, deployment, testing
Contract Language	Solidity v0.8.28	Writing verifier contract
Testing Tools	Mocha, Chai, Ethers v6	Contract and script validation
Decentralized Storage	IPFS (Kubo v0.18.1)	Secure record hosting

3.3.2. zk-Proof Lifecycle

The enforcement of privacy-preserving access within ZK-EHR hinges on a structured zero-knowledge proof lifecycle. This lifecycle encapsulates all stages, from initial setup to on-chain validation, enabling authorized requesting parties to prove compliance with access control policies without disclosing their identities or the policies themselves. The design adheres to a privacy-by-default philosophy grounded in succinct non-interactive zero-knowledge arguments of knowledge (zk-SNARKs). The full lifecycle comprises four main phases:

1. Setup phase: A trusted setup is conducted using the Powers of Tau protocol to generate a circuit-agnostic structured reference string (SRS). This common reference string ensures that all subsequent proofs maintain soundness and zero-knowledge properties. The process is executed only once per circuit and culminates in the generation of the proving and verification keys.
2. Input encoding: When an access request is initiated, the requester's attributes (e.g., role, department, authorization status) are encoded as private inputs, while the policy criteria are encoded as public inputs to the circuit. This separation allows the prover to demonstrate policy compliance without revealing sensitive data. At the design level, in-circuit policy compliance is enforced by evaluating whether the hash of the requester's attributes matches the hash of the required policy condition. This is formalized as follows:

$$\text{isAuthorized} = \begin{cases} 1, & \text{if } H(r_u, d_u) = H(r_p, d_p) \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

   where H denotes a cryptographic hash function (e.g., Poseidon) embedded within the circuit. The terms r_u and d_u denote the user's role and department, while r_p and d_p represent the required access policy attributes. This logical expression is compiled into the constraint system, enabling on-chain policy enforcement without revealing any private input.
3. Proof generation: Using the compiled circuit, the proving key, and the prepared inputs, a zk-SNARK proof is generated locally by the requester. The proof succinctly attests that the hidden attributes satisfy the public policy conditions while revealing nothing beyond the validity of the claim.
4. On-chain verification: The proof, along with the corresponding public inputs, is submitted to a smart contract verifier deployed on the blockchain. The contract uses the verification key to validate the proof and emits an event signaling successful authorization. Crucially, this process logs the access event immutably without exposing requesting party identity or policy content.

The ZK-EHR access control circuit implements the authorization logic described above using the Circom circuit language compiled to a Groth16-compatible R1CS. The current prototype circuit accepts three private scalar inputs—`role`, `department`, and `authorized`—and produces one public output, `isAccessGranted`. Access is granted when all three inputs match the policy-defined values; specifically, the circuit checks `role` $= 1$, `department` $= 2$, and `authorized` $= 1$ via three equality sub-circuits, and the output is the product of these Boolean results: `isAccessGranted` $= (\texttt{role} = 1) \cdot (\texttt{department} = 2) \cdot (\texttt{authorized} = 1)$. The compiled prototype contains 5 R1CS constraints and was verified using Circom v2.2.2 and SnarkJS v0.7.5. Proof generation for this circuit was evaluated over 21 iterations (Section 4.1); the arithmetic mean of the logged proof-generation times is 113.27 ms (Table 3), although this average is influenced by two early warm-up runs, and the post-warm-up average across the remaining iterations is approximately 76 ms.

Table 3. Summary of performance metrics across 20 iterations in Scenario 1.

Metric	Mean (ms)	Std. Dev (ms)	Min (ms)	Max (ms)
Proof Generation Time	113.27	118.05	59.84	467.94
Verification Time	377.67	5.38	369.75	391.63
IPFS Upload Time	19.99	6.26	8.35	29.54

To support the security mechanisms proposed in Section 3.2.3 and to enable richer policy enforcement, several circuit extensions are envisioned. First, the fixed equality checks would be replaced with Poseidon-based attribute hashing: the requester's attribute vector would be hashed in-circuit and compared against a policy hash published on-chain, enabling flexible policies without hard-coded constants. This would require approximately 200–400 additional R1CS constraints per Poseidon invocation. Second, a credential-binding constraint would verify that a commitment to the requester's attributes is bound to the requester's secret key, preventing credential sharing (∼400 additional constraints). Third, an expiry-verification constraint would enforce that the current block timestamp falls within the credential's validity window using an arithmetic range-check gadget (∼200 additional constraints). Fourth, a nullifier computation would produce a unique, request-specific public output derived from the prover's secret key and request metadata, enabling the replay prevention mechanism described in Section 3.2.3. With all proposed extensions, the total circuit size is projected to be on the order of one thousand R1CS constraints, remaining well within the efficient proving range for Groth16.

This zk-proof lifecycle transforms traditional access control into a cryptographically verifiable, privacy-respecting process. Unlike centralized systems that rely on credential disclosure and role tokens, ZK-EHR enforces policies in zero knowledge, thus maintaining the confidentiality of both requesting parties and the rules governing access.

3.3.3. Smart Contract Integration

To enforce verifiable and privacy-preserving access control within the ZK-EHR framework, a Verifier smart contract was developed and deployed on an Ethereum-compatible blockchain. This contract acts as the on-chain enforcement mechanism responsible for validating zero-knowledge proofs submitted by requesting parties seeking access to EHRs. Its operation relies exclusively on cryptographic validation, eliminating the need for identity disclosure or centralized authorization mechanisms.

The Verifier contract is automatically generated from the compiled zk-SNARK circuit using the Groth16 proving system. It includes embedded elliptic curve parameters representing the verification key derived during the trusted setup phase. The contract exposes a single entry-point function, typically named verifyProof, which accepts the zk-proof and a set of public signals as inputs. Internally, it executes a series of elliptic curve pairing checks to validate the proof.

At a high level, the contract's logic can be abstracted by the following decision function:

$$\text{isValid} = \begin{cases} 1 & \text{if Verify}(vk, \vec{p}, \pi) = \text{true} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where π is the zk-proof, $\vec{p}$ is the vector of public inputs, and vk is the verification key.

The underlying cryptographic condition follows the Groth16 pairing-based scheme and is formalized as:

$$\text{verifyProof}(vk, \pi, \vec{x}) = \begin{cases} \text{true} & \text{if } e(\pi_A, \pi_B) = e(vk_\alpha + x \cdot vk_X, vk_\beta) \wedge e(\pi_C, vk_\delta) = e(vk_s, vk'_s) \\ \text{false} & \text{otherwise} \end{cases} \quad (9)$$

where $e(\cdot, \cdot)$ denotes the bilinear pairing function, and the terms π_A, π_B, and π_C represent elements of the zk-proof. The pairing checks ensure that the submitted proof is consistent with the proving and verification keys and that the encoded witness satisfies the arithmetic constraints defined in the circuit. The logical structure of this process is outlined in Algorithm 1, which presents the contract's behavior when a proof and its associated public inputs are submitted for verification.

Algorithm 1 Core verification logic of the Verifier smart contract

```
function VERIFYPROOF(proof, publicSignals)
    vk ← loadVerifyingKey()
    isValid ← snarkjsVerify(vk, publicSignals, proof)
    if isValid then
        emit AccessGranted(msg.sender)
        storeAccessLog(msg.sender, timestamp)
        return true
    else
        emit AccessDenied(msg.sender)
        return false
    end if
end function
```

The verification process begins by referencing a circuit-specific verification key and proceeds to validate the submitted zk-proof against the public inputs. Successful verification triggers the emission of an `AccessGranted` event and logs the access attempt on-chain. Conversely, a failed attempt results in an `AccessDenied` event. This event-based logging ensures immutable auditability of all authorization decisions without revealing identities or access rules.

Figure 3 provides a snapshot of the Solidity code used for implementation. The contract was deployed and tested using the Hardhat and Ganache frameworks. Tests were conducted across various scenarios to ensure correctness, gas efficiency, and resistance to tampered or malformed proofs. The contract is invoked during the final stage of the ZK-EHR access workflow, authorizing data retrieval upon successful verification.

This integration strategy achieves a clear separation of concerns: sensitive policy logic and identity attributes remain off-chain, while cryptographic enforcement occurs on-chain. By binding access control to zk-proof validity rather than static identities or roles, the Verifier contract plays a central role in realizing the ZK-EHR system's goals of decentralization, verifiability, and privacy preservation.

3.3.4. End-to-End Integration Scenario

To demonstrate the practical operation of the ZK-EHR framework, this section outlines a complete end-to-end scenario that captures the interaction between the requesting party, the zk-proof generation logic, the Verifier smart contract, and the off-chain encrypted record store. The objective is to ensure that access to sensitive health records is granted only upon verifiable, policy-compliant proof—without disclosing the identity or attributes of the requester.

Figure 3. Snapshot of the Solidity Verifier contract.

The integration scenario comprises five sequential stages:

1. **Access request initiation:** A healthcare provider initiates an access request for a specific EHR. The requester holds a set of attributes (e.g., role, department, affiliation) that are not explicitly revealed. These attributes are encoded as private inputs to the zero-knowledge circuit, while the required access conditions are expressed as public inputs.
2. **Proof generation (off-chain):** Using the locally compiled zk-circuit and the proving key, the requester generates a zk-SNARK proof demonstrating that their attributes satisfy the predefined access policy. This proof, along with its public inputs, is produced entirely off-chain using the SnarkJS toolkit. The resulting artifacts (`proof.json` and `public.json`) contain no sensitive or identifiable information.
3. **On-chain verification:** The generated proof and associated public inputs are submitted to the Verifier smart contract. The contract validates the proof using its embedded verification key and emits an `AccessGranted` event if the proof is valid. This on-chain verification provides an immutable, audit-friendly decision without revealing the requester's identity.
4. **Record retrieval (off-chain):** Upon receiving the access approval signal, the requester fetches the corresponding EHR from the decentralized storage system (e.g., IPFS). The record is identified by a known CID and remains encrypted throughout transmission. Decryption occurs locally, using an authorized key issued through a secure, off-chain channel.
5. **Audit and logging:** All access attempts, whether successful or denied, are logged immutably on-chain via event emission. These logs contain minimal metadata (e.g., timestamp and status) to ensure verifiability while preserving user anonymity. Optional off-chain logs may also be maintained for local audit trails.

Figure 4 illustrates the complete end-to-end integration flow in the ZK-EHR framework. The interaction diagram captures the coordinated interaction across system components, including the requester, zk-Prover, Verifier smart contract, IPFS, and audit log. It emphasizes that access control is enforced cryptographically, while policy logic and identity remain protected off-chain.

This scenario highlights the separation of roles in the system: Cryptographic enforcement and auditability occur on-chain, while attribute evaluation and sensitive logic remain off-chain. This architecture supports decentralized access control with verifiability and strong privacy guarantees.

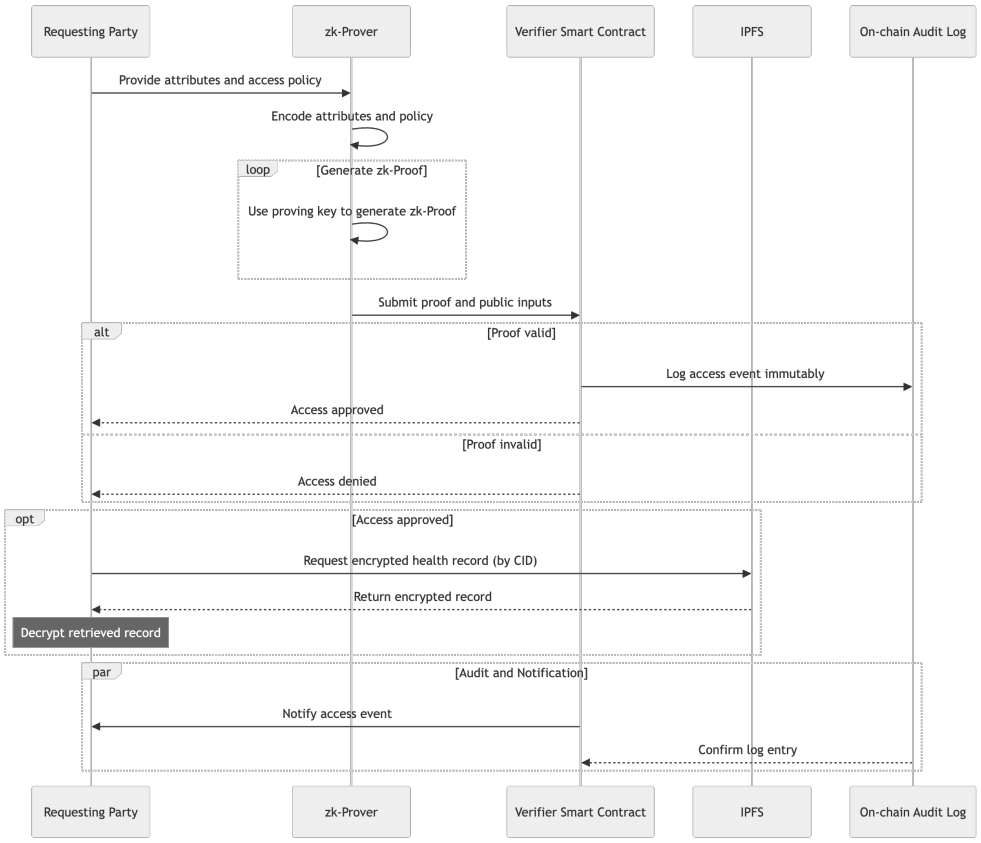

Figure 4. Sequence diagram of the ZK-EHR access control workflow. Solid arrows indicate request/response interactions, while dashed lines represent verification or signaling steps.

3.4. Experimental Setup and Metrics

3.4.1. Evaluation Setup

The evaluation of the ZK-EHR framework was conducted to assess its effectiveness, reliability, and scalability under various usage scenarios. The goal was to verify that the proposed system can enforce privacy-preserving access control based on zero-knowledge proofs while maintaining acceptable latency and operational integrity across typical and adversarial conditions.

All experiments were performed on a macOS Ventura 13.5 system equipped with an Apple M1 processor (8-core CPU, 8-core GPU) and 16 GB RAM. The development and testing stack included Node.js v18.18.0, Hardhat v2.24.3 for smart contract deployment and interaction, Ganache CLI v7.9.2 for running a local Ethereum-compatible testnet, and the snarkjs library (v0.6.11) for generating and verifying zk-SNARK proofs. The verifier contract was compiled from a Groth16 proof system, exported using snarkjs, and deployed using Hardhat.

The IPFS (Kubo v0.18.1) decentralized storage system was run locally to support encrypted file uploads and retrievals. All IPFS-related interactions were conducted using the `ipfs-http-client` JavaScript library. Encrypted EHRs were prepared in various sizes to test both functionality and storage layer scalability.

Each scenario was executed through dedicated JavaScript scripts organized in the `/evaluation/` directory. The scripts measured proof generation latency, smart contract verification time, upload latency, and success/failure status. Timing data were collected using `performance.now()` from the Node.js `perf_hooks` module, and the results were logged to CSV files for subsequent statistical analysis and visualization.

3.4.2. Performance Metrics

A set of core performance metrics was defined to quantitatively assess the ZK-EHR framework. These metrics were selected to evaluate the computational cost, responsiveness, and integrity of proof-based access enforcement under different operational scenarios.

- Proof generation time (ms): This metric captures the time required to generate a zk-SNARK proof from a given set of private attributes and public policy inputs using the snarkjs toolkit. It reflects the client-side computational overhead before a proof is submitted for access.
- Verification time (ms): The smart contract's execution time to verify the submitted proof and public inputs. This value is essential for assessing the on-chain cost of privacy-preserving access enforcement.
- IPFS upload latency (ms): The time needed to upload an encrypted health record to the local IPFS node. This metric gauges the responsiveness of the decentralized storage layer.
- Proof validity outcome: A binary status indicating whether the proof was accepted (valid) or rejected (invalid) by the Verifier contract. This enables the evaluation of robustness against malformed or tampered inputs.
- Error detection and rejection behavior: Observed outcomes in scenarios involving corrupted, incomplete, or manipulated proof components or public inputs. This qualitative metric evaluates the contract's resilience and correctness under adversarial conditions.

Each metric was computed over multiple iterations where applicable, and aggregated statistics, including mean, standard deviation, and outliers, were used to support deeper performance analysis across scenarios.

3.4.3. Evaluation Scenarios

The evaluation of ZK-EHR was structured around four core usage scenarios designed to capture both expected operational behavior and system robustness under edge cases. Each scenario was implemented as a self-contained script and executed independently to assess the system's response in terms of latency, correctness, and resilience.

- Scenario 1—Authorized Access with Valid Proof: This baseline test measured system performance during typical usage, where a valid zk-proof is submitted with correct public inputs and successfully verified by the Verifier contract. The test reflects the expected operational case in a secure, error-free environment.
- Scenario 2—Proof Tampering: This adversarial test explored the system's behavior when malicious users attempt to submit structurally valid but altered zk-proofs. Modifications included bit-level changes to different components (e.g., π_a, π_b, π_c, and public inputs). The goal was to verify the Verifier contract's ability to reject manipulated proofs.
- Scenario 3—Corrupted or Missing Public Inputs: This test assessed the framework's robustness in detecting malformed input vectors. Scenarios included truncated input

arrays, swapped element orders, or completely empty public inputs. The objective was to evaluate whether the contract would reject submissions with an incomplete or invalid verification context.

- Scenario 4—Concurrent Requests: To evaluate the system's behavior under parallel load, 50 proof submissions were triggered concurrently in batched waves. This tested the scalability and determinism of the Verifier contract and smart contract infrastructure under realistic multiuser workloads.

Each scenario contributes a different dimension to the evaluation framework, ranging from correctness and efficiency to fault tolerance and scalability.

4. Results

This section presents the experimental outcomes across the defined scenarios.

4.1. Scenario 1: Authorized Access with Valid Proof

This scenario evaluated the ZK-EHR framework's performance under optimal conditions using 20 consecutive authorized access attempts with identical user attributes and system inputs. The results are summarized in Table 3, which reports the mean, standard deviation, minimum, and maximum values for three critical performance metrics: proof generation time, smart contract verification time, and IPFS upload latency.

The average proof generation time was 113.27 ms, with a standard deviation of 118.05 ms; this indicates some fluctuation in local computation time. In contrast, the verification time remained highly consistent, averaging 377.67 ms, with a deviation of just 5.38 ms; this demonstrates the deterministic execution of the Groth16-based on-chain validation. Upload latency to IPFS was modest, with an average of 19.99 ms and a standard deviation of 6.26 ms; this reflects slight variability, likely attributable to transient system or network delays.

All proofs were successfully validated (valid = true), and no iterations failed during generation, verification, or storage. The distributions of the collected measurements are shown in Figure 5, using box plots to highlight metric spread and outlier behavior. The results confirm that ZK-EHR maintains reliable and auditable performance in typical authorized access workflows.

Figure 5. Distribution of performance metrics in Scenario 1.

To further evaluate the storage layer scalability of the ZK-EHR framework, a supplementary test was conducted in which encrypted medical files of varying sizes were uploaded to IPFS. These files included a small text-based medical record (200 KB), an annotated PDF (1 MB), a diagnostic X-ray image (5 MB), and a multislice MRI scan (10 MB). Each file was uploaded once to a local IPFS node, and the upload latency was measured on the client side. As IPFS relies on content-based addressing, repeated uploads of identical files may bypass full reprocessing or benefit from local caching, which could distort timing accuracy. Therefore, only first-time uploads were considered to reflect realistic usage conditions, such as when newly encrypted EHR records are introduced to the system.

The results are summarized in Table 4 and visualized in Figure 6. Upload times remained consistently under 300 ms across all file sizes, indicating that the system can handle the inclusion of large and complex medical records without introducing significant latency. This further reinforces the framework's readiness for real-world deployment scenarios.

Table 4. Upload latency to IPFS for encrypted EHR files of varying sizes.

File Type (Size)	Upload Time (ms)
Small Text Record (200 KB)	113.23
Annotated PDF (1 MB)	84.75
X-Ray Image (5 MB)	184.33
MRI Scan (10 MB)	273.01

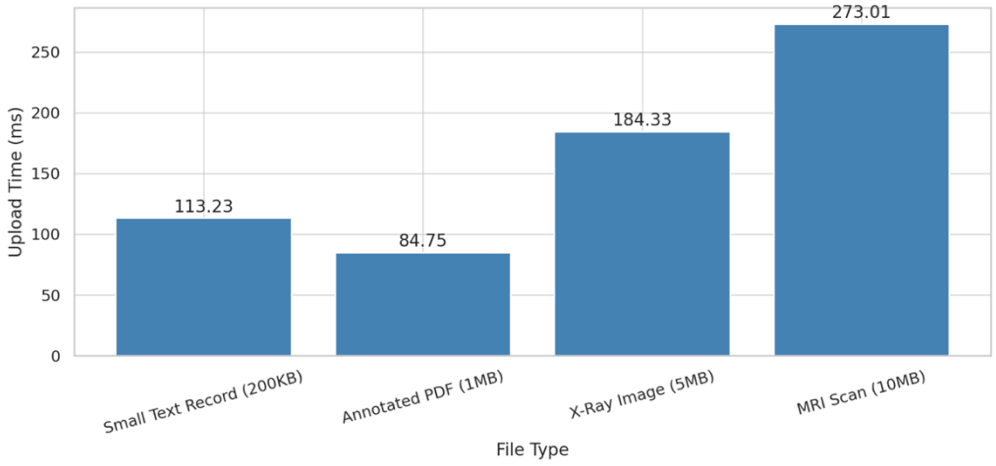

Figure 6. IPFS upload latency measured for encrypted EHR files of increasing size.

4.2. Scenario 2: Proof Tampering

To assess the robustness of the ZK-EHR system against integrity violations, this scenario introduced multiple forms of tampering to the proof and public inputs before submission. The goal was to validate the contract's ability to reject proofs that are structurally valid but cryptographically inconsistent with the original setup.

Five test cases were evaluated: modifying individual components, such as $\pi_a[0]$ and $\pi_c[1]$; altering a public input value; swapping the π_b block structure; and submitting an untouched valid proof. The verification outcomes and timings are summarized in Table 5.

Table 5. Verification results for tampered and valid zk-proofs.

Scenario	Modification Target	Validity	Output	Verif. Time (ms)
2.a	Tampered $\pi_a[0]$	Invalid	Rejected	66.12
2.b	Tampered $\pi_c[1]$	Invalid	Rejected	166.0
2.c	Modified public[0]	Invalid	Rejected	538.59
2.d	Swapped π_b structure	Invalid	Rejected	34.85
2.e	Valid Proof (no modification)	Valid	Accepted	393.28

As expected, all manipulated proofs were successfully rejected by the Verifier contract, while the valid proof passed. Notably, the verification time for invalid proofs ranged from 34.85 ms to 538.59 ms, demonstrating variability depending on the type of inconsistency. The valid proof was accepted at 393.28 ms, consistent with prior benchmarked values. This confirms the contract's resilience to tampering attempts and its reliable enforcement of cryptographic correctness.

4.3. Scenario 3: Corrupted or Missing Public Inputs

This scenario evaluated the system's behavior when the public input vector, which is used during proof generation, is altered or misconfigured prior to on-chain verification. Although the submitted zero-knowledge proof remained structurally valid, the mismatch between the expected and actual public inputs was expected to result in proof rejection.

Four cases were tested: altering the value of `public[0]`, applying a misconfigured mutation to the public input vector, submitting an empty array, and using the original valid input. As shown in Table 6, corrupted and empty inputs were correctly rejected by the Verifier contract. Upon re-examination of the evaluation setup, the originally labeled "missing public input" case does not correspond to a genuinely truncated public-input vector in the current implementation. The checked-in circuit/verifier pair exposes a single public signal, and the mutation applied in Scenario 3.b preserved the original valid one-element input rather than constructing a malformed vector. The observed acceptance in this case therefore reflects an evaluation-script artifact rather than tolerance of incomplete inputs. This behavior does not indicate any weakness in the underlying Groth16 verification scheme. In the current fixed-size verifier interface, incorrect public inputs are rejected during verification, and true length mismatches are rejected at the ABI boundary.

Table 6. Verification outcomes for proofs submitted with corrupted, missing, or valid public inputs.

Scenario	Input Condition	Validity	Output	Verif. Time (ms)
3.a	Corrupted `public[0]`	Invalid	Rejected	402.71
3.b	Misconfigured input (unchanged valid vector)	Valid	Accepted	375.24
3.c	Empty public input	Invalid	Rejected	0.77
3.d	Valid public input	Valid	Accepted	399.55

The valid case was accepted as expected, with a verification time of 399.55 ms, consistent with prior benchmarks. In addition, the corrupted and empty inputs were handled correctly, reinforcing the system's baseline reliability against malformed input vectors.

4.4. Scenario 4: Concurrent Requests

To evaluate the system's concurrency performance, 50 parallel proof verification requests were submitted in batched waves of 10 using asynchronous execution. This model simulated realistic multiuser interaction with the smart contract. Each batch was followed by a short delay to prevent resource saturation and emulate expected usage patterns in decentralized systems.

As shown in Figure 7, all verification attempts were successful, confirming that the Verifier contract handled concurrent submissions without error. The average verification time was 3625.08 ms with a standard deviation of 55.57 ms, suggesting consistent response times across concurrent requesting parties. The narrow variance further indicates predictable performance even under a moderate load.

These results validate the contract's ability to sustain parallel access without functional degradation. However, the absolute latency is notably higher than in single-request scenarios, which may be attributed to the synchronous nature of Ganache or resource contention within the local execution environment.

4.5. Summary of Results

The evaluation results confirm that the ZK-EHR framework provides verifiable, privacy-preserving access control with consistent performance across a variety of operational and adversarial conditions. Each scenario yielded insights into specific dimensions of the system's behavior, summarized as follows:

- Correctness and stability (Scenario 1): Under normal conditions, all authorized access attempts were successfully validated. Proof generation and on-chain verification times were stable, with IPFS upload latency remaining under 30 ms. This confirms the reliability and low overhead of the system during typical use.
- Tamper resistance (Scenario 2): All manipulated proofs (whether through alteration of π_a, π_c, π_b, or public inputs) were correctly rejected by the Verifier contract, demonstrating the circuit's robustness and the smart contract's ability to enforce strict cryptographic verification.
- Input integrity (Scenario 3): Malformed or incomplete public input vectors generally resulted in rejection, except in the case of truncated arrays, which revealed a subtle vulnerability in the input validation process. This observation indicates that future versions of the circuit may benefit from explicit input length constraints.
- Scalability (Scenario 4): The Verifier contract successfully processed 50 concurrent requests with consistent outcomes. While verification times were higher compared to the single-request baseline (∼3600 ms vs. ∼390 ms), the system maintained correctness without failure or divergence, highlighting its operational resilience under load.

These results collectively affirm that ZK-EHR achieves its design goals of decentralization, privacy, verifiability, and auditability. The system performs reliably across edge cases and scales well under concurrent access, with areas for future optimization identified in verifier efficiency and input validation.

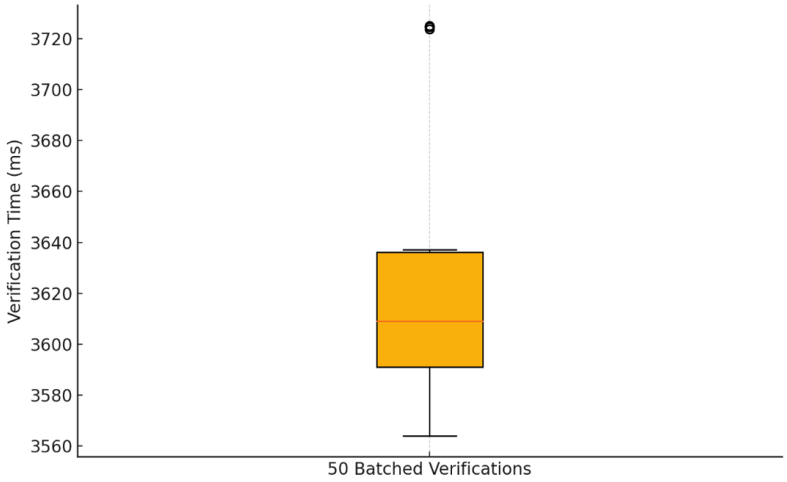

Figure 7. Verification time distribution for 50 batched concurrent proof submissions using the ZK-EHR Verifier contract.

4.6. Gas Cost Analysis

To assess the on-chain operational cost of ZK-EHR, gas consumption was measured using `eth_estimateGas` on the deployed Groth16 Verifier Smart Contract in the Hardhat built-in EVM. The `verifyProof` function is a `view` function that performs the Groth16 pairing check and returns a boolean result. In the current prototype, the verifier contract does not emit events or write to storage; it performs only the cryptographic verification. Table 7 summarizes the gas usage.

The measured gas consumption of 216,631 gas falls within the typical range of 200,000–300,000 gas for Groth16 proof verification on Ethereum [43]. At an illustrative gas price of 20 gwei and an ETH price of approximately $2500 the per-verification cost is approximately $0.011, confirming economic viability for routine healthcare access control. In a production deployment with event emission and storage writes for audit logging,

an additional 20,000–50,000 gas per request would be expected, bringing the projected total to approximately 240,000–270,000 gas.

These measurements were obtained using the Hardhat built-in EVM. On public Ethereum mainnet or Layer 2 networks, actual costs will depend on network congestion, gas price dynamics, and the deployment chain.

Table 7. Gas consumption per access request on the ZK-EHR Verifier Smart Contract (current prototype).

Operation	Gas Consumed
Proof verification (Groth16 pairing check)	216,631
Access log recording (event + storage)	N/A *
Total per access request	216,631

* Not applicable in the current prototype verifier.

5. Discussion

5.1. Summary of Evaluation Outcomes

The results obtained across all evaluation scenarios demonstrate that the ZK-EHR framework fulfills its core design goals of secure, private, and verifiable access control for EHRs. Each scenario was carefully designed to reflect a distinct operational context, and the findings offer several important insights.

In Scenario 1 (Authorized Access), the system achieved stable performance. All valid proofs were accepted and consistently low verification and upload times were achieved. This confirms the system's baseline readiness for real-world use cases with minimal latency overhead. The subtest on file scalability further showed that encrypted records of up to 10 MB could be uploaded to IPFS within 300 ms, confirming the system's storage layer robustness.

Scenario 2 (Proof Tampering) validated the system's resistance to manipulated inputs. All intentionally modified zk-SNARK components resulted in rejection by the verifier contract, demonstrating that the circuit design and verification logic offer effective protection against tampering.

Scenario 3 (Corrupted or Missing Public Inputs) revealed a subtle input vulnerability. While some malformed inputs were correctly rejected, the originally labeled truncated input case was found to be an evaluation artifact rather than a genuine malformed input scenario. This raised the need for more explicit input validation, either within the circuit or at the smart contract interface.

Scenario 4 (Concurrent Requests) highlighted the system's ability to handle multiple parallel verifications. When executed in unregulated bulk, the network stack became saturated, causing high latency ($\sim$18 s). However, when submitted in structured batches, the verification times stabilized ($\sim$3.6 s), confirming that ZK-EHR can support concurrent access scenarios when deployed with basic scheduling constraints.

Overall, the system shows strong alignment with its intended objectives, demonstrating functional correctness, privacy preservation, resistance to tampering, and acceptable levels of performance under varying conditions. However, areas such as input sanitation, parallel throughput optimization, and integration resilience remain open to further refinement.

5.2. Comparison with Existing Approaches

As summarized in Table 8, existing blockchain-based access control systems for EHRs exhibit various limitations in privacy, scalability, and policy enforcement. Among non-ZKP approaches, SmartAccess [11] implements ABAC via on-chain smart contracts but exposes user identities and lacks encrypted retrieval. Psarra et al. [12] combine role-based logic with fuzzy machine learning on Hyperledger Fabric, improving responsiveness

but without identity anonymization or cryptographic proof verification. Pampattiwar and Chavan [36] introduced a dual-blockchain architecture with genetic-algorithm-based scalability, though without zero-knowledge mechanisms. Yuan et al. [37] proposed proxy re-encryption for encrypted retrieval, but their model permits institutional access without patient authorization and does not support identity anonymity.

Among ZKP-based systems, Health-zkIDM [31] and its variant [30] apply zk-SNARKs for identity authentication but lack support for encrypted record retrieval or policy-aware access enforcement. Myeong and Kim [42] proposed a zk-SNARK-enabled Ethereum-based protocol with strong privacy guarantees and FHIR-based interoperability; however, their architecture focuses on proof-based identity verification and does not integrate smart contract governance, encrypted data access, or policy-level enforcement into a unified framework. In contrast, ZK-EHR combines zero-knowledge policy proofs, on-chain Groth16 verification, and IPFS-based encrypted retrieval into an integrated architecture, and is the only system in this comparison that explicitly evaluates adversarial scenarios including proof tampering, malformed inputs, and concurrent request handling.

To enable direct quantitative comparison, Table 8 focuses on three representative baseline systems for which performance metrics are available in the published literature: SmartAccess [11], representing ABAC-based blockchain access control; Psarra et al. [12], representing permissioned Hyperledger Fabric with machine-learning-augmented authorization; and Myeong and Kim [42], the most directly comparable zk-SNARK-based EHR system.

Table 8. Comparative analysis of ZK-EHR against representative baseline systems. Qualitative features assessed from published descriptions. Quantitative metrics reported where available; "—" = not reported or not applicable.

Feature/Metric	Smart Access [11]	Psarra et al. [12]	Myeong & Kim [42]	ZK-EHR
Privacy model	ABAC	Context-aware ABAC	zk-SNARK (data)	zk-SNARK (policy)
Identity anonymity	No	No	Yes	Yes
On-chain enforcement	Smart contract	SC + ML	SC + ZKP	zk-proof
Encrypted retrieval	No	Yes (IPFS)	No	Yes (IPFS)
Revocation	Yes	Yes	No	Proposed
Replay protection	No	No	No	Proposed
Platform	GoQuorum	HLF	Ganache	Hardhat
Proof gen. (ms)	—	—	76–278	∼113
Verif. time (ms)	—	—	21–43	∼378
Gas (units)	164,122 [a]	—	81,400 [b]	216,631 [c]

[a] ABAC policy evaluation; not ZKP verification. [b] ZKP verification via libsnark (Table 3 in [42]). [c] Groth16 pairing check via snarkJS (Section 4.6).

The quantitative comparison with Myeong and Kim [42]—the most directly comparable system, as both employ zk-SNARKs on Ethereum—reveals complementary performance profiles. ZK-EHR achieves proof generation of ∼113 ms (Table 3), faster than [42]'s moderate-complexity scenario (278 ms). However, ZK-EHR's verification time (∼378 ms) exceeds [42]'s 21–43 ms, attributable to measurement differences: ZK-EHR's timing includes full Hardhat EVM call overhead, while [42] reports contract-internal execution only. ZK-EHR's gas cost (216,631) exceeds [42]'s 81,400, reflecting the use of snarkJS-generated Groth16 verifiers versus libsnark. SmartAccess [11] reports 164,122 gas for ABAC policy evaluation, providing a cross-architecture reference point.

Regarding revocation, both SmartAccess [11] and Psarra et al. [12] provide implemented context-based revocation mechanisms. ZK-EHR's proposed revocation design (Section 3.2.3) advances beyond contract-level enforcement by embedding credential expiry directly in the zk-SNARK circuit, but remains at the design stage. The comparison has inherent limitations: refs. [11,12] are not ZKP-based, so proof generation and verification metrics are not applicable. Ref. [12] uses Hyperledger Fabric, avoiding gas costs

entirely, while ref. [11] operates on permissioned Ethereum with deterministic finality. Cross-architecture gas comparisons should be interpreted cautiously, as the measured operations serve different functions.

5.3. Security Threat Model and Trust Assumptions

The ZK-EHR framework adopts a zero-knowledge–based design to minimize reliance on centralized trust while preserving the privacy of medical records and access decisions. Nevertheless, the security guarantees of the system depend on a well-defined threat model and a set of operational assumptions.

The system assumes that an honest requesting party generates zk-proofs using valid private attributes and the correct access circuit. The proof is then verified by a smart contract that is assumed to be deployed without tampering, using a correctly generated verification key. The verifier contract is stateless, deterministic, and immune to external manipulation once deployed.

The threat model considers three classes of adversaries:

- External adversaries: Unauthorized parties attempting to forge or submit invalid zk-proofs to gain access to protected records.
- Internal adversaries: Parties with valid credentials who attempt to misuse access rights by submitting outdated, replayed, or misrepresented proofs.
- Passive observers: Entities monitoring the blockchain or IPFS metadata to infer sensitive information or access patterns (e.g., timing correlations, frequency of access, or repeated requests originating from the same pseudonymous blockchain address).

ZK-EHR defends against these threats through multiple cryptographic and architectural safeguards:

- Proof integrity is enforced through zk-SNARKs. Any manipulation of proof components (e.g., π_a, π_b, π_c) results in failure during verification. Scenario 2 (tampering) confirms that this defense is effective.
- Input correctness is partially enforced at the circuit level. Scenario 3 (corrupted public inputs) demonstrates the system's ability to reject malformed data, although improvements to input schema validation are recommended.
- Replay resistance is not enforced by default but can be supported by embedding a nonce or timestamp within the public input vector. This would allow verifiers to detect reused proofs.
- Privacy against passive observers is maintained by encrypting all EHR content before IPFS upload, and using content-addressable hashes (i.e., CIDs) that do not reveal requesting party attributes. However, as metadata remains observable, future versions may integrate access-controlled IPFS gateways or obfuscated hash mechanisms. In addition, address linkability (i.e., correlating multiple requests to the same on-chain sender address) is inherent to public blockchains and falls outside the zero-knowledge guarantees of ZK-EHR; it can be mitigated operationally via address rotation, relayers/meta-transactions, or additional privacy layers.

The system's cryptographic soundness depends on the assumption that the trusted setup (Powers of Tau) is performed honestly and securely. This one-time ceremony underlies the integrity of the proving and verification keys and must be managed transparently.

Overall, ZK-EHR defends effectively against the most active and passive threats relevant to decentralized healthcare data sharing, although further enhancements could extend protection to support revocation, input encoding constraints, and proof freshness guarantees.

5.4. Practical Deployment Considerations

Deploying the ZK-EHR framework in real-world healthcare environments introduces several operational challenges and requirements that must be carefully addressed to ensure system reliability, usability, and compliance with institutional standards.

First, the performance of zero-knowledge proof generation and on-chain verification must align with the infrastructure capabilities of the target healthcare setting. While the proof generation process occurs off-chain and is computationally intensive, its latency ($\sim$100–400 ms) remains within acceptable limits for most clinical access scenarios. Smart contract verification, in turn, executes deterministically and consistently on Ethereum-compatible networks, with measured average gas costs aligning with those of lightweight verification operations.

Second, integrating IPFS as a decentralized storage layer provides content addressability and distribution benefits but raises concerns regarding data availability, garbage collection, and persistence. To ensure record accessibility, healthcare institutions must either operate their own IPFS nodes or use pinning services that guarantee file retention over time. Additionally, IPFS itself does not offer native access control; all data protection relies on encryption prior to upload, placing the burden of key management on the system or the requesting party.

Third, the integration of zk-SNARK tooling (e.g., Circom, SnarkJS) and Ethereum development frameworks (e.g., Hardhat) requires development and operational expertise that may not be readily available within traditional healthcare IT departments. A production-ready version of ZK-EHR would likely require a user interface for request generation, identity attribute encoding, and decryption key provisioning—features that must be abstracted from the underlying cryptographic operations to support adoption by nontechnical users.

Finally, system auditability and compliance with regulations, such as HIPAA or GDPR, must be considered. Although the framework does not expose personally identifiable information (PII), access logs and verifier events are recorded immutably on-chain. This design supports retrospective auditing but may require governance protocols to interpret and manage these logs in sensitive environments.

In summary, while ZK-EHR is technically viable for deployment, achieving operational sustainability and regulatory compliance necessitates careful system engineering, user onboarding strategies, and integration with existing medical record management workflows.

5.5. Limitations and Future Work

Despite the strengths demonstrated by the ZK-EHR framework, several practical and architectural limitations remain, offering avenues for future enhancement. One key limitation of the current prototype is the absence of a fully implemented revocation mechanism. To address this at the design level, Section 3.2.3 proposes a two-layer approach combining credential expiry timestamps verified within the zk-SNARK circuit with an on-chain revocation registry maintained by the Verifier Smart Contract. However, once a requesting party has successfully retrieved and decrypted a file stored on IPFS, there is currently no built-in way to retract data access retroactively, as the content-addressable and persistent nature of IPFS means that a party in possession of the decryption key can retrieve the record indefinitely. Addressing this residual gap may additionally require integrating proxy re-encryption or time-bound access policies at the cryptographic level. Full implementation and performance evaluation of the proposed revocation mechanism are deferred to future work.

A further limitation of the current system is the absence of native replay protection. Section 3.2.3 proposes a nullifier-based prevention mechanism in which each proof is accompanied by a unique, deterministic nullifier derived inside the zk-SNARK circuit and

tracked in an on-chain registry. This design is architecturally compatible with the existing verification logic and adds minimal overhead. However, the mechanism is not yet implemented; until deployed, a previously valid proof could be resubmitted for unauthorized repeat access. Full implementation and evaluation of the nullifier mechanism are deferred to future work.

Although the current implementation now supports off-chain record encryption and wrapped key release, a comprehensive operational key-management framework remains beyond the present scope of this work. In the present system, records are encrypted using AES-256-GCM with per-record symmetric keys, and successful proof verification enables off-chain release of wrapped record keys using RSA-OAEP with SHA-256. Future work may extend this design through secure transport hardening, multi-user lifecycle management, revocation-aware key rotation, HSM/KMS-backed custody, enterprise IAM integration, and operational recovery procedures.

The evaluation process also revealed a scenario-labeling issue in the handling of malformed public inputs. In the current implementation, the checked-in circuit/verifier pair exposes a single public signal, and the originally labeled "missing input" case did not construct a genuinely truncated vector; instead, it preserved the original valid one-element input. This reflects an evaluation-script artifact rather than a weakness in the underlying Groth16 verification scheme. Future refinement should ensure closer alignment between evaluation scenarios and the implemented public-input structure.

The framework also inherits a critical limitation from its underlying zk-SNARK protocol—Groth16 requires a trusted setup via the Powers of Tau ceremony. While the current implementation assumes that this setup has been performed securely, any compromise during this phase may allow adversaries to generate valid-looking proofs illegitimately. Although alternative proof systems with transparent setups exist (e.g., Plonk and Halo2), they introduce other tradeoffs in terms of performance and circuit complexity.

From an experimental standpoint, all performance evaluations were conducted using a local Ethereum testnet (Ganache) and a local IPFS node. While these environments are sufficient for demonstrating feasibility and behavior under controlled conditions, they do not fully capture gas consumption, latency, or resilience on public networks. Future evaluations should include tests on mainnet-compatible chains, such as Polygon or Arbitrum, as well as decentralized IPFS clusters with variable availability.

Looking ahead, several enhancements can strengthen ZK-EHR. These include enriching the public input schema with session-specific fields, extending the current design toward a comprehensive key-management and revocation framework, integrating off-chain access control proxies with on-chain verification, and supporting multipolicy circuits. In addition, user experience improvements, such as graphical tools for proof generation and secure key handling, are crucial for practical adoption in nontechnical clinical environments.

6. Conclusions

This study presented ZK-EHR, a privacy-preserving, blockchain-based access control framework for encrypted electronic health records. The framework integrates zk-SNARK authorization proofs with on-chain Groth16 verification and off-chain IPFS-based encrypted storage, enabling policy-compliant access without disclosing user identities or credentials. Three differentiated actor roles—Patient (Data Owner), Doctor (Care Provider), and Researcher (Authorized Analyst)—are supported through distinct access scopes enforced by a custom Groth16 zero-knowledge circuit for role-based constraint enforcement, with Poseidon used for efficient in-circuit hashing in extensible policy variants.

Experimental evaluation across four scenarios confirmed the framework's operational viability. Under standard conditions, proof generation averaged 113 ms and on-chain

verification remained consistent at approximately 378 ms with low variance (Table 3). All tampered proofs were correctly rejected by the Verifier Smart Contract (Table 5), and input integrity tests confirmed that corrupted and empty public inputs were reliably rejected; the originally labeled "missing input" case was traced to an evaluation-script artifact in which the mutation preserved the original valid one-element public input (Table 6). Under concurrent load, the contract processed 50 parallel requests without failure. Gas cost analysis measured 216,631 gas per proof verification (Table 7), consistent with typical Groth16 deployments and suggesting feasibility for routine healthcare access control, subject to network fee conditions. A literature-based comparative analysis against three representative baseline systems (Table 8) demonstrated that ZK-EHR is the only framework in the comparison providing simultaneous identity anonymity, cryptographic policy enforcement via zero-knowledge proofs, and encrypted record retrieval.

To address security requirements identified during review, concrete design proposals for credential-based access revocation and nullifier-based replay prevention were proposed (Section 3.2.3), demonstrating architectural compatibility with the existing Groth16 verification logic. These extensions add minimal circuit overhead (approximately 200 additional R1CS constraints for credential expiry, with comparable overhead projected for nullifier computation) and do not compromise the zero-knowledge property.

Limitations remain, including the need for full implementation and empirical evaluation of the proposed security mechanisms, testing on public blockchain networks with realistic consensus delays, and development of governance frameworks for multi-institutional deployment. Nevertheless, the results demonstrate that zk-SNARK-based access control is a practical and extensible approach to secure, privacy-preserving healthcare data sharing across institutional boundaries.

Funding: This scientific paper is derived from a research grant funded by Taibah University, Madinah, Kingdom of Saudi Arabia, with grant number (1159-15-447).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: The author gratefully acknowledges the support of the Energy, Industry, and Advanced Technologies Research Center, Taibah University, Madinah, Saudi Arabia, for funding this research project.

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

EHR	Electronic Health Record
ZK-EHR	Zero-Knowledge Electronic Health Record
ZKP	Zero-Knowledge Proof
zk-SNARK	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge
IPFS	InterPlanetary File System
ABAC	Attribute-Based Access Control
RBAC	Role-Based Access Control
ACL	Access Control List
CID	Content Identifier
SRS	Structured Reference String
PII	Personally Identifiable Information

IoT	Internet of Things
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act

References

1. Adekunle Oyeyemi, A.; Jeremiah Olawumi, A.; Rawlings, C.; Chioma Anthonia, O.; Oloruntoba, B. The impact of electronic health records on patient care and outcomes: A comprehensive review. *World J. Adv. Res. Rev.* **2024**, *21*, 1446–1455. [CrossRef]
2. Menachemi, N.; Collum, T. Benefits and drawbacks of electronic health record systems. *Risk Manag. Healthc. Policy* **2011**, *4*, 47–55. [CrossRef] [PubMed]
3. Queen Elizabeth, E.; Jane Osareme, O.; Evangel Chinyere, A.; Opeoluwa, A.; Ifeoma Pamela, O.; Andrew Ifesinachi, D. The impact of electronic health records on healthcare delivery and patient outcomes: A review. *World J. Adv. Res. Rev.* **2023**, *21*, 451–460. [CrossRef]
4. Carlos Ferreira, J.; Elvas, L.B.; Correia, R.; Mascarenhas, M. Enhancing EHR Interoperability and Security through Distributed Ledger Technology: A Review. *Healthcare* **2024**, *12*, 1967. [CrossRef]
5. Zilong, D.; Alobaedy, M.; Ibrahim, M.N.H.; Huang, X. A Blockchain Technology Framework to Enhance Security and Interoperability of Electronic Healthcare Records. *Int. J. Acad. Res. Bus. Soc. Sci.* **2025**, *15*, 1156–1170. [CrossRef]
6. Joshi, H.; Joseph, S. Standardization and Interoperability: Federated Learning's Impact on EHR Systems and Health Informatics. *Adv. Health Inf. Sci. Pract.* **2025**, *1*, UBYM3803. [CrossRef] [PubMed]
7. Fridas, A.; Bourouliti, A.; Touramanidou, L.; Ivanova, D.; Votis, K.; Katsaounis, P. Advanced Digital System for International Collaboration on Biosample-Oriented Research: A Multicriteria Query Tool for Real-Time Biosample and Patient Cohort Searches. *Computers* **2025**, *14*, 157. [CrossRef]
8. Schutte, N.; Bogaert, P.; Saso, M.; Abboud, L.; Van Oyen, H. How can population health research benefit from a European Health Data Infrastructure? *Eur. J. Public Health* **2021**, *31*, ckab164.126. [CrossRef]
9. Chepkirui, M.; Dellicour, S.; Musuva, R.; Odero, I.; Omondi, B.; Omondi, B.; Onyango, E.; Barsosio, H.; Otiso, L.; Okomo, G.; et al. Missed opportunities for digital health data use in healthcare decision-making: A cross-sectional digital health landscape assessment in Homa Bay county, Kenya. *PLoS Digit. Health* **2025**, *4*, e0000870. [CrossRef]
10. Chainarong, D.; Thirasak, K.; Chuaphanngam, T.; Fugkeaw, S. Xdhds: Cross-Domain Healthcare Data Sharing Using Self-Soverign Id and Blockchain. In *2025 17th International Conference on Knowledge and Smart Technology (KST)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6. [CrossRef]
11. De Oliveira, M.T.; Reis, L.H.A.; Verginadis, Y.; Mattos, D.M.F.; Olabarriaga, S.D. SmartAccess: Attribute-based access control system for medical records based on smart contracts. *IEEE Access* **2022**, *10*, 117836–117854. [CrossRef]
12. Psarra, E.; Apostolou, D.; Verginadis, Y.; Patiniotakis, I.; Mentzas, G. Permissioned blockchain network for proactive access control to electronic health records. *BMC Med. Inform. Decis. Mak.* **2024**, *24*, 303. [CrossRef]
13. Qi, M.; Wang, Z.; Han, Q.L.; Zhang, J.; Chen, S.; Xiang, Y. Privacy Protection for Blockchain-Based Healthcare IoT Systems: A Survey. *IEEE/CAA J. Autom. Sin.* **2024**, *11*, 1757–1776. [CrossRef]
14. Tawfik, A.M.; Al-Ahwal, A.; Eldien, A.S.T.; Zayed, H.H. ACHealthChain blockchain framework for access control and privacy preservation in healthcare. *Sci. Rep.* **2025**, *15*, 16696. [CrossRef]
15. Thantharate, P.; Thantharate, A. ZeroTrustBlock: Enhancing Security, Privacy, and Interoperability of Sensitive Data through ZeroTrust Permissioned Blockchain. *Big Data Cogn. Comput.* **2023**, *7*, 165. [CrossRef]
16. Hassidim, A.; Korach, T.; Shreberk-Hassidim, R.; Thomaidou, E.; Uzefovsky, F.; Ayal, S.; Ariely, D. Prevalence of sharing access credentials in electronic medical records. *Healthc. Inform. Res.* **2017**, *23*, 176–182. [PubMed]
17. SooHoo, S.; Keller, M.S.; Moyse, H.; Robbins, B.; McLaughlin, M.; Arora, A.; Burger, A.; Huang, L.; Huang, S.C.; Goud, A.; et al. Accessing patient electronic health record portals safely using social credentials: Demonstration pilot study. *JMIR Form. Res.* **2022**, *6*, e29647. [CrossRef] [PubMed]
18. Rahman, M.S.; Khalil, I.; Mahawaga Arachchige, P.C.; Bouras, A.; Yi, X. A novel architecture for tamper proof electronic health record management system using blockchain wrapper. In *Proceedings of the 2019 ACM International Symposium on Blockchain and Secure Critical Infrastructure*; ACM: New York, NY, USA, 2019; pp. 97–105.
19. Saravanan, T.; Pugalenthi, R. Progressive Temporal Blockchain Framework for Ensuring Secure Management and Access of EHR. In *2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*; IEEE: Piscataway, NJ, USA, 2023; pp. 1–9. [CrossRef]

20. Ahmad, A.; Saad, M.; Njilla, L.; Kamhoua, C.; Bassiouni, M.; Mohaisen, A. BlockTrail: A Scalable Multichain Solution for Blockchain-Based Audit Trails. In *ICC 2019–2019 IEEE International Conference on Communications (ICC)*; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6. [CrossRef]
21. Rohini, K.R.; Rajakumar, P.S.; Geetha, S. Smart Patient Consent Management Model for Health Information Exchange Based on Blockchain Technology. *J. Comput. Sci.* **2024**, *20*, 730–741. [CrossRef]
22. Kakade, S.V.; Tiple, B.; Sansuddi, A.S.; Kokate, M.D.; Alate, M.M.; Chavan, S. Blockchain-Based Medical Record Sharing in Healthcare IoT: Building Trust and Transparency through Secure Provenance Tracking. *J. Electr. Syst.* **2023**, *19*, 43–52. [CrossRef]
23. Shruti, S.; Alapatt, B.; George, J. MediCrypt: A Model with Symmetric Encryption for Blockchain Enabled Healthcare Data Protection. In *2024 IEEE International Conference on Contemporary Computing and Communications (InC4)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–6. [CrossRef]
24. Musamih, A.; Salah, K.; Jayaraman, R.; Arshad, J.; Debe, M.; Al-Hammadi, Y.; Ellahham, S. A Blockchain-Based Approach for Drug Traceability in Healthcare Supply Chain. *IEEE Access* **2021**, *9*, 9728–9743. [CrossRef]
25. Liu, L.; Zhang, D. A Bernstein-type Inequality for High Dimensional Linear Processes with Applications to Robust Estimation of Time Series Regressions. *arXiv* **2021**, arXiv:2109.10354. [CrossRef]
26. Ali, A.; Al-rimy, B.A.; Alsubaei, F.S.; Almazroi, A.A.; Almazroi, A.A. HealthLock: Blockchain-Based Privacy Preservation Using Homomorphic Encryption in Internet of Things Healthcare Applications. *Sensors* **2023**, *23*, 6762. [CrossRef] [PubMed]
27. Alsadhan, A.; Alsalamah, H.; Alhogail, A. A Blockchain-Based Privacy-Preserving Model for IoMT Medical Systems. In *2024 6th International Conference on Blockchain Computing and Applications (BCCA)*; IEEE: Piscataway, NJ, USA, 2024; pp. 16–21. [CrossRef]
28. Ran, L.; Peng, C.; Xu, D.; Tan, W.; Lin, Z. Blockchain Privacy Disclosure Risk Assessment Scheme Based on Improved Paillier Algorithm. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*; IEEE: Piscataway, NJ, USA, 2022; pp. 1881–1887. [CrossRef]
29. Azaria, A.; Ekblaw, A.; Vieira, T.; Lippman, A. MedRec: Using Blockchain for Medical Data Access and Permission Management. In *2016 IEEE Open and Big Data Conference (OBD)*; IEEE: Piscataway, NJ, USA, 2016; pp. 25–30.
30. Luong, D.A.; Park, J.H. Privacy-Preserving Blockchain-Based Healthcare System for IoT Devices Using zk-SNARK. *IEEE Access* **2022**, *10*, 55739–55752. [CrossRef]
31. Bai, T.; Hu, Y.; He, J.; Fan, H.; An, Z. Health-zkIDM: A healthcare identity system based on fabric blockchain and zero-knowledge proof. *Sensors* **2022**, *22*, 7716.
32. Maheshwari, V.; Prasanna, M. Privacy-preserving authentication for 5G healthcare with HBZKP: Hierarchical blockchain-based zero knowledge proof for secure edge devices. *Ain Shams Eng. J.* **2025**, *16*, 103463. [CrossRef]
33. Singh, M.P.; Sural, S.; Vaidya, J.; Atluri, V. A role-based administrative model for administration of heterogeneous access control policies and its security analysis. *Inf. Syst. Front.* **2024**, *26*, 2255–2272. [CrossRef]
34. Adda, M.; Aliane, L. HoBAC: Fundamentals, principles, and policies. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *11*, 5927–5941. [CrossRef]
35. Saha, S.; Neogy, S. *Permission Based Access Control for Healthcare Systems*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 43–56.
36. Pampattiwar, K.; Chavan, P. A secure and scalable blockchain-based model for electronic health record management. *Sci. Rep.* **2025**, *15*, 11612. [CrossRef]
37. Yuan, W.-X.; Yan, B.; Li, W.; Hao, L.-Y.; Yang, H.-M. Blockchain-based medical health record access control scheme with efficient protection mechanism and patient control. *Multimed. Tools Appl.* **2023**, *82*, 16279–16300. [CrossRef]
38. Luong, D.A.; Park, J.H. Privacy-preserving identity management system on blockchain using Zk-SNARK. *IEEE Access* **2023**, *11*, 1840–1853. [CrossRef]
39. Wan, Z.; Zhou, Y.; Ren, K. zk-AuthFeed: Protecting data feed to smart contracts with authenticated zero knowledge proof. *IEEE Trans. Dependable Secur. Comput.* **2022**, *20*, 1335–1347.
40. Chenthara, S.; Ahmed, K.; Wang, H.; Whittaker, F.; Chen, Z. Healthchain: A novel framework on privacy preservation of electronic health records using blockchain technology. *PLoS ONE* **2020**, *15*, e0243043.
41. Liu, J.; Fan, Y.; Sun, R.; Liu, L.; Wu, C.; Mumtaz, S. Blockchain-aided privacy-preserving medical data sharing scheme for e-healthcare system. *IEEE Internet Things J.* **2023**, *10*, 21377–21388. [CrossRef]
42. Myeong, G.E.; Ram, K.S. Blockchain Based Zero Knowledge Proof Protocol For Privacy Preserving Healthcare Data Sharing. *J. Technol. Inform. Eng.* **2025**, *4*, 171–189. [CrossRef]
43. Groth, J. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology—EUROCRYPT 2016*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9666, pp. 305–326.
44. Ben-Sasson, E.; Chiesa, A.; Genkin, D.; Tromer, E.; Virza, M. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *Advances in Cryptology—CRYPTO 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 90–108.

45. Sun, X.; Yu, F.R.; Zhang, P.; Sun, Z.; Xie, W.; Peng, X. A Survey on Zero-Knowledge Proof in Blockchain. *IEEE Netw.* **2021**, *35*, 198–205. [CrossRef]
46. Albalwy, F. *ZK-EHR: Zero-Knowledge Access Control Framework for EHRs*; GitHub Repository; GitHub, Inc.: San Francisco, CA, USA, 2025. Available online: https://github.com/FbalwyTU/ZK-EHR (accessed on 6 July 2025).