A Patient-Centric Blockchain Framework for Secure Electronic Health Record Management: Decoupling Data Storage from Access Control

Tanzim Hossain Romel[1], Tanberul Islam[1], Kawshik Kumar Paul[1], Maisha Rahman[1], and Dr. Abu Sayed Md. Latiful Hoque[1]

[1]Department of Computer Science & Engineering, Bangladesh University of Engineering & Technology, Dhaka, Bangladesh

Abstract

We present a patient-centric architecture for electronic health record (EHR) sharing that separates content storage from authorization and audit. Encrypted FHIR resources are stored off-chain; a public blockchain records only cryptographic commitments and patient-signed, time-bounded permissions using EIP-712. Keys are distributed via public-key wrapping, enabling storage providers to remain honest-but-curious without risking confidentiality. We formalize security goals (confidentiality, integrity, non-repudiable authorization, and auditability) and provide a Solidity reference implementation. On-chain costs for permission grants average **78,000 gas** (L1), and end-to-end access latency for 1 MB records is **1.3–2.0 s**, dominated by storage retrieval. Layer-2 deployment reduces costs by an order of magnitude. We discuss metadata privacy and regulatory considerations (HIPAA/GDPR), demonstrating a practical route to restoring patient control while preserving security properties required for sensitive clinical data.

Keywords: Blockchain, Electronic Health Records, Access Control, Healthcare Privacy, Smart Contracts, FHIR, Cryptographic Protocols

1 Introduction

The digitization of healthcare has transformed medical practice, enabling evidence-based decision-making and population health management at unprecedented scales. However, health records remain trapped in organizational silos with incompatible systems. When patients seek care from multiple providers or relocate, critical medical history becomes inaccessible, leading to duplicated tests, adverse drug interactions, and suboptimal treatment decisions.

1.1 The Centralization Problem

Contemporary health information exchange architectures exhibit three fundamental weaknesses. First, they create single points of failure where system compromise can affect millions of patient records. Major healthcare data breaches have exposed the medical information of hundreds of millions of individuals [1]. Second, centralized systems require patients to trust intermediary organizations with unfettered access. While policies constrain behavior, insider threats persist, and audit logs maintained by audited entities offer limited assurance. Third, patients exercise minimal control over sharing, contradicting principles of autonomy and informed consent.

1.2 Blockchain as an Architectural Primitive

Blockchain technology addresses specific weaknesses through replicated, append-only ledgers where transactions are cryptographically verified rather than institutionally authorized. However, naive blockchain application introduces problems: storing protected health information on public blockchains violates privacy through transparency and immutability, and transaction costs make large document storage impractical.

The key insight is architectural separation: encrypted records reside in off-chain storage optimized for large objects, while blockchain serves exclusively as authorization layer and integrity mechanism. This exploits complementary strengths while avoiding respective weaknesses.

1.3 Contributions

1. **Formal Architecture:** We specify how off-chain encrypted storage combined with on-chain access control achieves confidentiality against curious storage providers, integrity verification, and patient-controlled authorization with cryptographic non-repudiation.

2. **Reference Implementation:** Complete Ethereum smart contract handling record registration, permission granting through signed

messages, time-bounded access with revocation, and auditability through event logs.

3. **Healthcare Integration:** Integration with HL7 FHIR standards, showing how FHIR resources serve as plaintext while supporting de-identified data release.

4. **Performance Evaluation:** Comprehensive evaluation including on-chain measurements, end-to-end latency analysis, storage overhead characterization, and privacy metrics.

5. **Threat Analysis:** Systematic threat landscape analysis, identifying which adversaries the system resists and deployment requirements for residual risks.

1.4 Paper Organization

Section II surveys related work. Section III establishes technical foundations. Section IV formalizes system and threat models. Section V presents architecture and workflows. Section VI provides the smart contract implementation. Section VII gives security analysis. Section VIII reports performance evaluation. Section IX addresses privacy and regulatory compliance. Section X discusses limitations and future work. Section XI concludes.

2 Related Work

2.1 Early Blockchain Health Systems

MedRec, proposed by Azaria et al., pioneered blockchain for medical records using Ethereum to log access permissions [2, 3]. However, MedRec relies on healthcare providers to host and enforce access controls, essentially replacing one trusted party with another. Our architecture eliminates residual trust by encrypting records with patient-controlled keys.

Peterson et al. proposed blockchain audit trails for health information exchange [4], but assumed honest storage without cryptographic protection. Yue et al. used attribute-based encryption [5], but required trusted key generation centers and complex attribute management. Our design uses simple public-key operations compatible with existing wallet infrastructure.

Griggs et al. demonstrated smart contracts for remote patient monitoring [6]. Comprehensive surveys [7–9] identified recurring challenges: scalability, privacy protection, regulatory compliance, and standardization—issues we address systematically.

2.2 Cryptographic Approaches

Benaloh et al. demonstrated patient-controlled encryption for Microsoft HealthVault [10]. Fisher et al. proposed PGP-based encryption for healthcare [11]. These systems lacked blockchain's tamper-evident logging and depended on centralized providers. Our work combines patient-controlled encryption with blockchain auditability.

Proxy re-encryption allows semi-trusted proxies to transform ciphertexts without seeing plaintext. While elegant, it introduces operational complexity. We use simpler key wrapping, with proxy re-encryption as future optimization for frequent re-sharing scenarios.

2.3 Recent Blockchain Proposals

Wang et al. proposed fine-grained access control for decentralized storage [12], similar in spirit but lacking patient-controlled key management and using password-based rather than cryptographic authorization. Omar et al. developed privacy-friendly cloud platforms [13] providing auditability but storing unencrypted off-chain data.

FHIRChain [14, 15] integrated FHIR with blockchain authorization. ACTION-EHR explored specialized architectures for cancer care [16]. Several systems proposed token-based access control. Our approach refines this using EIP-712 signatures, providing stronger security (harder to forge), better UX (no gas costs for patients), and replay protection through nonces.

3 Technical Preliminaries

3.1 Authenticated Encryption with Associated Data

We use AES-256 in Galois/Counter Mode (AES-GCM), providing authenticated encryption with associated data (AEAD) [17, 18]. For plaintext M, symmetric key $SymmK$, nonce N, and associated data AD, encryption produces ciphertext C and authentication tag T:

$$(C, T) = \text{AES-GCM-Enc}(SymmK, N, M, AD) \tag{1}$$

Decryption with verification:

$$M = \text{AES-GCM-Dec}(SymmK, N, C, T, AD) \tag{2}$$

AEAD provides IND-CCA2 security: adversaries cannot distinguish ciphertexts from random strings or forge valid tags without keys [19]. We use 256-bit keys, 96-bit nonces, and 128-bit tags.

3.2 Public-Key Encryption

We use Elliptic Curve Integrated Encryption Scheme (ECIES) for key wrapping [20, 21]. To share $SymmK$ with recipient having public key PK_{rcpt}:

$$W = \text{ECIES-Enc}(PK_{rcpt}, SymmK) \tag{3}$$

Unwrapping with corresponding private key SK_{rcpt}:

$$SymmK = \text{ECIES-Dec}(SK_{rcpt}, W) \tag{4}$$

ECIES uses ephemeral Diffie-Hellman for key establishment, providing IND-CCA2 security. We use secp256k1 curve (256-bit keys) matching Ethereum's cryptographic primitives.

3.3 Cryptographic Hash Functions

Content digests use Keccak-256, producing 256-bit hashes. For message M:

$$d = H(M) = \text{Keccak256}(M) \tag{5}$$

Keccak-256 provides collision resistance: finding $M \neq M'$ where $H(M) = H(M')$ is computationally infeasible.

3.4 Digital Signatures

We use Elliptic Curve Digital Signature Algorithm (ECDSA) over secp256k1. For message m and private key SK:

$$\sigma = \text{Sign}_{SK}(m) \tag{6}$$

Verification with corresponding public key PK:

$$\{0, 1\} = \text{Verify}_{PK}(m, \sigma) \tag{7}$$

ECDSA provides existential unforgeability under chosen message attacks: adversaries cannot forge valid signatures without private keys.

3.5 EIP-712 Typed Structured Data

EIP-712 [22] defines typed structured message signing, providing replay protection and human-readable signatures. Messages include domain separator binding to specific contract and chain:

$$\sigma = \text{Sign}_{SK}(\text{EIP712-Hash}(\text{domain}, \text{message})) \tag{8}$$

Domain separator includes contract address and chain ID, preventing cross-contract and cross-chain replay attacks. Nonces prevent within-contract replay.

3.6 Ethereum and Smart Contracts

Ethereum is a decentralized platform for smart contracts—programs executing on blockchain with consensus-verified state transitions [23]. Key properties:

- **Consensus:** Proof-of-stake validates transactions
- **State Persistence:** Contract storage persists across transactions
- **Event Emission:** Logs create off-chain indexable audit trails
- **Gas Metering:** Computational costs paid in gas units

3.7 HL7 FHIR

Fast Healthcare Interoperability Resources (FHIR) is a standard for exchanging healthcare information electronically. FHIR defines resources (patients, observations, medications) with REST APIs and JSON/XML representations. We use FHIR R4 for record format, enabling integration with existing healthcare systems.

3.8 Content-Addressed Storage

We use InterPlanetary File System (IPFS) [24], providing content-addressed distributed storage. Files are identified by cryptographic hashes (CIDs). Content-addressing enables integrity verification: retrieved data's hash must match the CID. IPFS provides:

- **Content Deduplication:** Identical files share storage
- **Distributed Retrieval:** Files retrieved from any hosting node
- **Immutable Addressing:** CIDs uniquely identify content

Alternative storage (S3, Storj, Filecoin) can substitute IPFS, maintaining architecture properties if content-addressed or separately verified against on-chain digests.

4 System and Threat Model

4.1 Entities

- **Patient (P):** Controls access to their records. Has keypair (SK_P, PK_P) for signing authorizations and encryption keypair (SK_P^{enc}, PK_P^{enc}) for receiving wrapped keys.
- **Recipients (R_i):** Healthcare providers requesting access. Each has keypairs (SK_i, PK_i) for transaction signing and (SK_i^{enc}, PK_i^{enc}) for receiving wrapped keys.
- **Storage Provider (S):** Stores encrypted records. Can be IPFS nodes, cloud storage, or hybrid systems. Assumed honest-but-curious.
- **Blockchain Network (B):** Public Ethereum network maintaining authorization ledger. Assumed adversarially robust with sufficient decentralization.

4.2 Adversarial Capabilities

We assume a *Dolev-Yao* network adversary controlling communication channels: can intercept, modify, delay, or inject messages. However, the adversary cannot break cryptographic primitives (AES-256, secp256k1, Keccak-256).

Storage Provider: Honest-but-curious—follows protocol but attempts to read stored data. Can access all on-chain data and stored ciphertexts. Cannot obtain decryption keys without valid authorization.

Unauthorized Recipients: May attempt accessing records without permissions. Can observe blockchain state and submit unauthorized transactions. Cannot forge signatures or break encryption.

Out of Scope: We do *not* protect against: (1) Patient key compromise—adversary obtaining SK_P can sign arbitrary authorizations; (2) Malicious patients willingly sharing records inappropriately; (3) Side-channel attacks on cryptographic implementations; (4) Denial-of-service attacks on blockchain or storage.

4.3 Trust Assumptions

- **Blockchain Security:** We assume Ethereum maintains consensus security. With sufficient validators and stake, byzantine actors cannot forge or reorder transactions.

- **Cryptographic Primitives:** We assume AES-256-GCM, ECIES/secp256k1, ECDSA/secp256k1, and Keccak-256 remain secure against classical adversaries with polynomial-time resources.

- **Client Environment:** We assume patients and recipients operate secure devices with private key protection (hardware wallets, secure enclaves). Software wallets risk key theft.

- **Storage Availability:** We assume storage providers maintain availability. Redundant storage (multiple IPFS nodes, cloud backup) mitigates single-provider failure.

4.4 Security Goals

Definition 1 (Confidentiality)**.** *No adversary without valid authorization can learn plaintext health records. Formally: For all polynomial-time adversaries $\mathcal{A}$ without $SymmK$ or valid wrapped key W, advantage in distinguishing encrypted records from random strings is negligible.*

Definition 2 (Integrity)**.** *Any tampering with stored records is detectable. Formally: For all adversaries $\mathcal{A}$, probability of producing (C', T', N', AD') passing verification $H(C'\|T'\|N'\|AD') = d$ where $(C', T') \neq (C, T)$ is negligible.*

Definition 3 (Authorization Authenticity)**.** *Access requires cryptographically verifiable patient consent. Formally: No adversary $\mathcal{A}$ without SK_P can produce valid permission signature σ passing EIP-712 verification.*

Definition 4 (Non-Repudiation)**.** *Patients cannot deny granting permissions they signed. Formally: Given valid $(rid, addr_{rcpt}, exp, W, \sigma)$ with verified signature, patient P cannot credibly claim they didn't sign.*

Definition 5 (Auditability)**.** *All access grants and revocations are immutably recorded. Formally: For all operations, blockchain emits events creating tamper-evident log entries queryable by any observer.*

5 Architecture and Workflows

5.1 System Architecture

Figure 1 illustrates the architecture. The patient's client encrypts records locally before upload. Storage providers (IPFS, S3) host only ciphertext. The blockchain maintains metadata: storage pointers, content digests, and permissions. Recipients obtain pointers and keys through on-chain authorization.

5.2 Key Workflows

Before interacting with the system, patients and recipients must complete initial setup by generating keypairs and registering their encryption public keys on-chain. Figure 2 illustrates this initialization process.

5.3 Workflow 1: Record Creation and Registration

The patient creates a health record:

1. **Generate Key:** Patient generates random symmetric key $SymmK \in \{0,1\}^{256}$

2. **Prepare FHIR Resource:** Convert health data to FHIR JSON format M. Optionally compute associated data AD containing metadata (resource type, timestamp).

3. **Encrypt:** Generate random nonce $N \in \{0,1\}^{96}$. Compute $(C, T) = \text{AES-GCM-Enc}(SymmK, N, M, AD)$

4. **Upload:** Submit (C, T, N, AD) to storage provider S, obtaining content-addressed pointer ptr (e.g., IPFS CID).

5. **Digest:** Compute $d = H(C\|T\|N\|AD)$

6. **Wrap Key:** Wrap key for self: $W_P = \text{ECIES-Enc}(PK_P^{enc}, SymmK)$

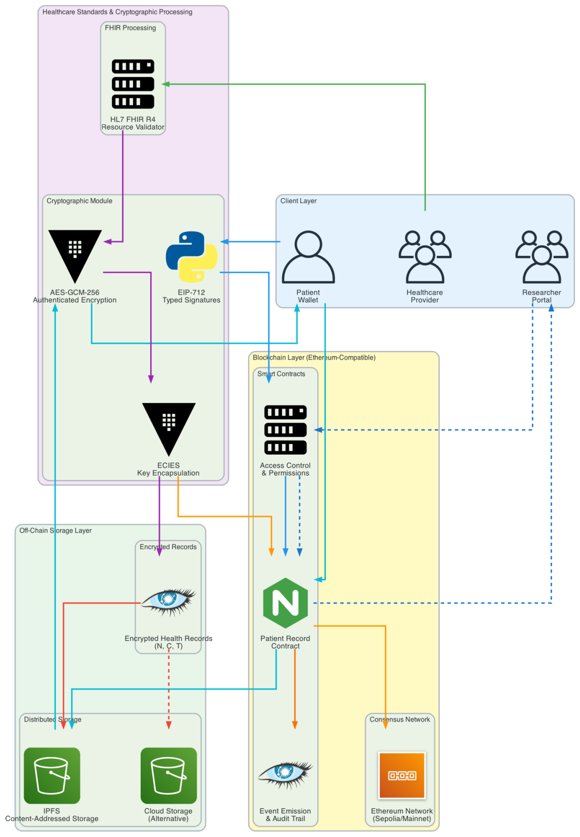
Healthcare Standards & Cryptographic Processing
FHIR Processing
HL7 FHIR R4 Resource Validator
Cryptographic Module
AES-GCM-256 Authenticated Encryption
EIP-712 Typed Signatures
ECIES Key Encapsulation
Client Layer
Patient Wallet
Healthcare Provider
Researcher Portal
Blockchain Layer (Ethereum-Compatible)
Smart Contracts
Access Control & Permissions
Patient Record Contract
Event Emission & Audit Trail
Consensus Network
Ethereum Network (Sepolia/Mainnet)
Off-Chain Storage Layer
Encrypted Records
Encrypted Health Records (N, C, T)
Distributed Storage
IPFS Content-Addressed Storage
Cloud Storage (Alternative)

Figure 1: System architecture separating encrypted storage from blockchain authorization

7. **Register:** Submit transaction calling `addRecord`(ptr, d, W_P) on smart contract. Contract stores (ptr, d, W_P) and emits `RecordAdded`(rid, d, ptr) event.

The record ID rid is returned, serving as identifier for future operations. The encrypted blob (C, T, N, AD) resides off-chain, while blockchain stores only (ptr, d, W_P).

5.4 Workflow 2: Permission Grant via Signed Message

The patient grants access to recipient R:

1. **Obtain Recipient Key:** Patient queries key registry for PK_R^{enc}. If not registered, recipient must first register their encryption public key.

2. **Wrap Key:** Patient computes $W_R = \text{ECIES-Enc}(PK_R^{enc}, SymmK)$

3. **Choose Expiration:** Select expiration timestamp exp (e.g., 90 days from now).

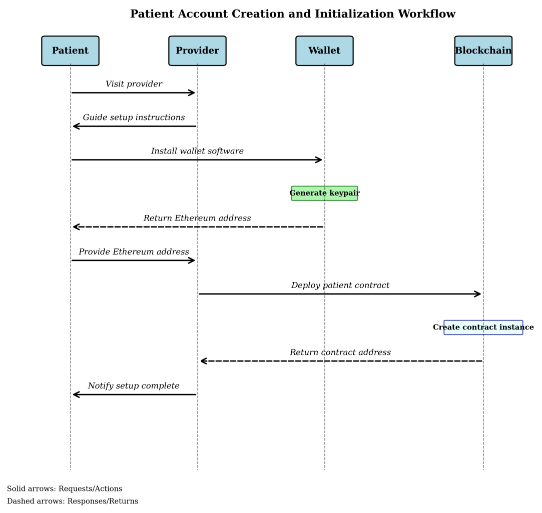

Figure 2: Patient and recipient account creation workflow showing keypair generation and public key registration

4. **Construct Message:** Build EIP-712 typed message:

$$\begin{aligned} msg = \{&\text{recordId} : rid, \\ &\text{grantee} : addr_R, \\ &\text{expiration} : exp, \\ &\text{wrappedKey} : W_R, \\ &\text{nonce} : n\} \end{aligned}$$

where n is patient's current nonce.

5. **Sign:** Compute $\sigma = \text{Sign}_{SK_P}(\text{EIP712-Hash}(msg))$

6. **Transmit:** Send (rid, exp, W_R, σ) to recipient R off-chain (email, secure messaging).

7. **Recipient Submits:** Recipient calls `grantPermissionBySig`(rid, exp, W_R, σ). Contract:

- Recovers signer from σ
- Verifies signer is patient
- Verifies nonce hasn't been used
- Increments patient's nonce
- Stores permission: $\text{permissions}[rid][addr_R] = (exp, false, W_R)$
- Emits `PermissionGranted`$(rid, addr_R, exp)$

This design has crucial properties: (1) Patient pays no gas—recipient submits transaction; (2) Signature provides non-repudiation; (3) Nonce prevents replay; (4) Permission is time-bounded.

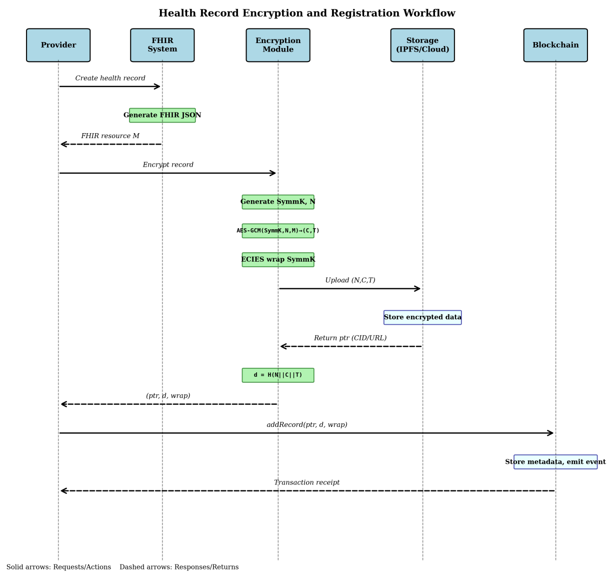

Figure 3: Record creation and encryption workflow showing local encryption, off-chain storage, and on-chain registration

5.5 Workflow 3: Record Access

Authorized recipient R accesses a record:

1. **Query Metadata:** Call `getRecordMetadata`(rid). Contract checks `hasValidPermission`(rid) for `msg.sender`. If valid, returns (ptr, d).
2. **Retrieve Ciphertext:** Fetch (C, T, N, AD) from storage using ptr.
3. **Verify Integrity:** Compute $d' = H(C\|T\|N\|AD)$. Check $d' = d$. If mismatch, abort (tampered data).
4. **Obtain Wrapped Key:** Query `permissions`$[rid][addr_R]$ to get W_R.
5. **Unwrap Key:** Compute $SymmK = \text{ECIES-Dec}(SK_R^{enc}, W_R)$
6. **Decrypt:** Compute $M = \text{AES-GCM-Dec}(SymmK, N, C, T, AD)$. If authentication fails, abort (tampered or wrong key).
7. **Parse FHIR:** Parse plaintext M as FHIR JSON and display in EHR system.

End-to-end security: Storage provider sees only (C, T, N, AD)—ciphertext. Network adversary observing retrieval sees only encrypted data. Only recipient with W_R and SK_R^{enc} can obtain M.

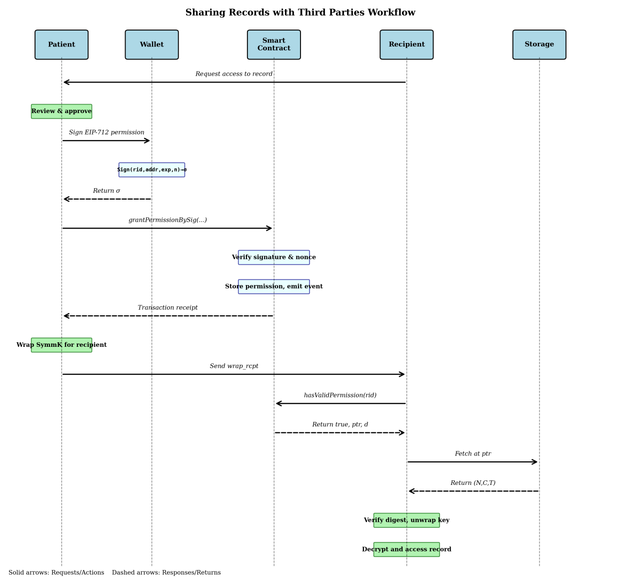

Figure 4: Permission grant workflow showing patient signature, off-chain transmission, and on-chain authorization

5.6 Workflow 4: Permission Revocation

Patient revokes access:

1. **Submit Revocation:** Patient calls `revokePermission`$(rid, addr_R)$. Contract sets $\text{permissions}[rid][addr_R].\text{revoked} = true$ and emits `PermissionRevoked`$(rid, addr_R)$.
2. **Future Access Blocked:** Subsequent `getRecordMetadata`(rid) calls by $addr_R$ fail permission check.

Limitation: Revocation prevents future access but cannot retract already-decrypted plaintext. If recipient downloaded M before revocation, they retain that data. This is fundamental to cryptographic access control and must be clearly communicated to patients.

Key Rotation for Forward Security: For stronger guarantees when trust relationships end, patient can rotate keys: generate new $SymmK'$, re-encrypt record, upload new ciphertext, update on-chain metadata with new (ptr', d'). Previous $(ptr, d, SymmK)$ become obsolete. Revoked recipients cannot access new version.

6 Smart Contract Implementation

6.1 Design Rationale

The smart contract serves three roles: (1) **Metadata Registry**—storing storage pointers and content digests; (2) **Authorization Engine**—verifying permissions before metadata release; (3) **Audit Log**—emitting events for all operations.

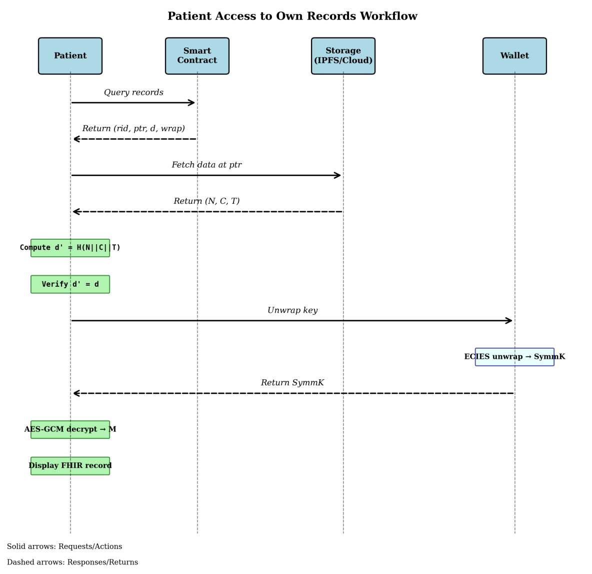

Figure 5: Record access workflow showing authorization check, ciphertext retrieval, integrity verification, and decryption

```
contract PatientHealthRecords is ERC721, EIP712 {
    address public immutable patient;

    struct RecordMetadata {
        string storagePointer;
        bytes32 contentDigest;
        bytes wrappedKeyOwner;
        uint64 createdAt;
    }

    struct Permission {
        uint64 expiration;
        bool revoked;
        bytes wrappedKey;
    }

    mapping(uint256 => RecordMetadata) private
         _records;
    mapping(uint256 => mapping(address => Permission))
        public permissions;
    mapping(address => uint256) public nonces;
}
```

Listing 1: Smart Contract Data Structures

```
function addRecord(
    string memory ptr,
    bytes32 digest,
    bytes memory wrappedKey
) external onlyPatient returns (uint256) {
    _recordIds.increment();
    uint256 rid = _recordIds.current();

    _records[rid] = RecordMetadata({
        storagePointer: ptr,
        contentDigest: digest,
        wrappedKeyOwner: wrappedKey,
        createdAt: uint64(block.timestamp)
    });

    _safeMint(patient, rid);
    emit RecordAdded(rid, digest, ptr);
    return rid;
}
```

Listing 2: Record Registration Function

We use ERC-721 non-fungible tokens where each record is a token owned by the patient. This provides: (1) Standardized ownership model; (2) Compatibility with existing Ethereum tooling; (3) Natural patient-as-owner semantics. However, we override transfer functions to make records non-transferable.

6.2 Core Data Structures

6.3 Record Registration

Gas cost: ∼180,000 gas first record (cold storage), ∼165,000 gas subsequent records.

6.4 Signature-Based Permission Grant

Gas cost: ∼78,000 gas. Signature verification costs ∼3,000 gas; storage costs dominate.

6.5 Permission Verification and Metadata Access

6.6 Permission Revocation

Gas cost: ∼31,000 gas.

6.7 Security Properties

Read Gating: `getRecordMetadata` enforces authorization via `hasValidPermission`. Only patient or valid grantees obtain (ptr, d).

Transparency Trade-off: Storage pointers *ptr* and digests d are public in events and contract storage. This is acceptable because pointers reference encrypted content. Without wrapped keys, adversaries obtain only ciphertext. Read-gating provides UX consistency, not confidentiality—security relies entirely on encryption and permission-controlled key distribution.

Non-Transferability: We override `transferFrom` and `approve` to revert, preventing accidental record transfers. Medical records should not be tradable assets.

Replay Protection: Nonces prevent signature replay. Each patient has an incrementing nonce; signatures include current nonce and are valid only once.

Time-Bounded Access: All permissions have expiration timestamps. Expired permissions fail `hasValidPermission` checks automatically.

```
function grantPermissionBySig(
    uint256 rid,
    uint64 expiration,
    bytes memory wrappedKey,
    bytes memory signature
) external validRecordId(rid) {
    // Recover signer
    address signer = _recoverSigner(
        rid, msg.sender, expiration,
        wrappedKey, nonces[patient], signature
    );
    require(signer == patient, "Invalid signature");

    // Check expiration is future
    require(expiration > block.timestamp,
        "Expiration must be future");

    // Increment nonce (prevent replay)
    nonces[patient]++;

    // Store permission
    permissions[rid][msg.sender] = Permission({
        expiration: expiration,
        revoked: false,
        wrappedKey: wrappedKey
    });

    emit PermissionGranted(rid, msg.sender, exp);
}

function _recoverSigner(
    uint256 rid, address grantee, uint64 exp,
    bytes memory wk, uint256 nonce, bytes memory sig
) internal view returns (address) {
    bytes32 structHash = keccak256(abi.encode(
        PERMISSION_TYPEHASH,
        rid, grantee, exp,
        keccak256(wk), nonce
    ));
    bytes32 hash = _hashTypedDataV4(structHash);
    return ECDSA.recover(hash, sig);
}
```

Listing 3: Permission Grant with Signature

7 Security Analysis

7.1 Confidentiality Against Storage Providers

Theorem 1 (Storage Provider Confidentiality)**.** *Assuming AES-GCM provides IND-CCA2 security and ECIES provides IND-CCA2 security, no honest-but-curious storage provider $\mathcal{S}$ can distinguish encrypted health records from random strings with non-negligible advantage.*

Proof Sketch. By contradiction. Suppose adversary $\mathcal{S}$ has non-negligible advantage ϵ in distinguishing encrypted records. Storage contains (C, T, N, AD) where (C, T) are outputs of AES-GCM-Enc$(SymmK, N, M, AD)$. By IND-CCA2 security of AES-GCM, (C, T) is computationally indistinguishable from random strings without $SymmK$. Since $\mathcal{S}$ cannot obtain $SymmK$ (wrapped keys W are ECIES ciphertexts, also indistinguishable from random without private keys), $\mathcal{S}$ cannot distinguish (C, T) from random. This contradicts the assumption of advantage $\epsilon > 0$. □ □

```
function hasValidPermission(uint256 rid)
    public view validRecordId(rid)
    returns (bool) {
    // Patient always has access
    if (msg.sender == patient) return true;

    Permission memory p = permissions[rid][msg.sender
        ];
    return !p.revoked &&
           p.expiration > 0 &&
           p.expiration >= block.timestamp;
}

function getRecordMetadata(uint256 rid)
    external view validRecordId(rid)
    returns (string memory ptr, bytes32 digest) {
    require(hasValidPermission(rid),
        "Not authorized");

    RecordMetadata memory rec = _records[rid];
    ptr = rec.storagePointer;
    digest = rec.contentDigest;
}
```

Listing 4: Permission Check and Metadata Retrieval

```
function revokePermission(
    uint256 rid, address grantee
) external onlyPatient validRecordId(rid) {
    require(permissions[rid][grantee].expiration > 0,
        "No permission to revoke");

    permissions[rid][grantee].revoked = true;
    emit PermissionRevoked(rid, grantee);
}
```

Listing 5: Permission Revocation

7.2 Integrity Verification

Theorem 2 (Tamper Detection)**.** *Assuming Keccak-256 is collision-resistant, any modification to stored records is detected with probability $\geq 1 - 2^{-128}$.*

Proof Sketch. On-chain digest $d = H(C\|T\|N\|AD)$ commits to encrypted blob. To pass verification, adversary must produce (C', T', N', AD') where $H(C'\|T'\|N'\|AD') = d$ with $(C', T') \neq (C, T)$. This requires finding collision in Keccak-256. With 256-bit output and collision resistance, success probability is $\leq 2^{-128}$ (birthday bound). □ □

Authentication Tag: AES-GCM's tag T provides additional integrity protection. Even if adversary finds hash collision, forging valid tag without $SymmK$ has negligible probability (AES-GCM provides 128-bit authentication security).

7.3 Authorization Authenticity

Theorem 3 (Signature Unforgeability)**.** *Assuming ECDSA over secp256k1 provides existential unforgeability under chosen message attacks (EUF-CMA), no adversary without patient's private key SK_P can forge valid permission signatures with non-negligible probability.*

Table 1: Threat Coverage

Threat	**Defense**	**Residual Risk**
Storage snooping	Encryption	None
Data tampering	Digest verification	None
Unauthorized access	On-chain authz	None
Permission forgery	EIP-712 signatures	None
Replay attacks	Nonces	None
Audit log tampering	Blockchain immutability	None
Patient key theft	—	High
Malicious patient	—	Inherent
Storage unavailability	Redundancy	Low
DoS on blockchain	—	Low (fees deter)

Table 2: Cryptographic Operation Latency

Operation	**Mean (ms)**	**95th % (ms)**
AES-GCM Enc (1 KB)	0.42	0.58
AES-GCM Enc (100 KB)	2.1	2.7
AES-GCM Enc (1 MB)	18.3	22.1
AES-GCM Enc (10 MB)	181.5	205.3
AES-GCM Dec (1 MB)	16.8	20.5
ECIES Key Wrap	3.2	4.1
ECIES Key Unwrap	3.5	4.3
ECDSA Sign (EIP-712)	2.8	3.6
ECDSA Verify	3.1	3.9
Keccak256 (1 MB)	12.1	14.8

Proof Sketch. Suppose adversary $\mathcal{A}$ forges signature σ' for message $(rid, addr_R, exp, W_R, n)$ that passes verification. By EUF-CMA security of ECDSA, this occurs with negligible probability without SK_P. EIP-712 domain separator binds signature to specific contract and chain, preventing cross-contract/cross-chain replay. Nonce prevents same-message replay. □ □

7.4 Replay Attack Prevention

Nonces provide replay protection:

Property 1 (Nonce Monotonicity). *For each patient, nonce n increases monotonically with each permission grant. Signature σ for $(rid, addr_R, exp, W_R, n)$ is valid only if current nonce equals n. After successful grant, nonce increments to $n + 1$, invalidating all signatures with nonce $\leq n$.*

Time-Based Replay: Expiration timestamps prevent long-term replay. Even if adversary captures signature, using it after expiration fails `require(expiration > block.timestamp)` check.

7.5 Auditability

Property 2 (Audit Trail Completeness). *All state-changing operations emit events: `RecordAdded`, `PermissionGranted`, `PermissionRevoked`, `RecordUpdated`. Events are permanently stored in blockchain logs, queryable by any observer.*

Blockchain immutability ensures events cannot be deleted or modified after confirmation. Patients, regulators, or auditors can reconstruct complete access history by filtering events for specific records or addresses.

7.6 Threat Analysis Summary

Table 1 summarizes threat coverage.

8 Performance Evaluation

8.1 Experimental Setup

Blockchain: Ethereum Sepolia testnet, September-October 2024. Transactions via Web3.js v4.2.1.

Storage: IPFS (go-ipfs v0.18) on 8-core, 16 GB RAM server. AWS S3 for comparison. Client in Dhaka, Bangladesh; IPFS nodes in Singapore; S3 in us-east-1.

Client: Intel Core i7-1165G7 @ 2.80GHz, 16 GB RAM, Ubuntu 22.04. Web Crypto API for AES-GCM, eth-crypto for ECIES/ECDSA.

Workload: Synthea v3.2.0 generated FHIR R4 resources. Record sizes: 1 KB (observations) to 10 MB (imaging reports). 50 trials per measurement for mean and 95th percentile.

8.2 Cryptographic Operations

Table 2 shows operation latencies.

AES-GCM achieves ∼55 MB/s throughput, linear in plaintext size. Hardware AES acceleration (AES-NI) provides these speeds on modern processors. Public-key operations (ECIES, ECDSA) take 3-4 ms regardless of record size—operate on fixed-size keys/hashes. For typical sharing (one signature, one key wrap), total cryptographic overhead ¡10 ms, negligible vs network latency.

8.3 On-Chain Gas Costs

Table 3 reports gas consumption.

Layer-2 solutions reduce gas consumption 10-13× through batching and off-chain computation.

Cost Breakdown: `grantPermissionBySig` costs: signature verification (∼3,000 gas), storage writes (∼40,000 gas for new permission), state updates (∼20,000 gas), events (∼10,000 gas), execution overhead (∼5,000 gas).

Economic Viability: For patients managing dozens of providers, Layer-2 costs are negligible compared to clinical workflow expenses. Healthcare institutions or in-

Table 3: Smart Contract Gas Consumption

Operation	Gas
Ethereum Mainnet (L1)	
Contract Deployment	2,341,829
addRecord (first)	183,742
addRecord (subsequent)	166,542
grantPermissionBySig	78,331
revokePermission	31,204
Layer-2 (Arbitrum One)	
addRecord	14,392
grantPermissionBySig	6,127
Layer-2 (zkSync Era)	
addRecord	11,243
grantPermissionBySig	5,894

Table 4: End-to-End Access Latency (1 MB Records)

Component	Mean (ms)	95th % (ms)
Blockchain query	245	312
IPFS retrieval	1,087	1,523
Integrity verification	12	15
Key unwrapping	4	5
AES-GCM decryption	17	21
Total (IPFS)	**1,365**	**1,876**
S3 retrieval	423	589
Total (S3)	**701**	**942**

surance can subsidize L1 costs; L2 enables direct patient payment models with substantially lower costs.

8.4 End-to-End Access Latency

Table 4 shows total record access time.

Storage retrieval dominates latency. IPFS exhibits higher variance due to distributed nature—retrieval from distant/slow peers. S3 provides consistent performance through CDN. Cryptographic operations contribute ¡5% total latency.

Scalability: For 10 MB records, total latency increases to ∼3.5 s (IPFS) or ∼2.1 s (S3). Encryption/decryption scale linearly but remain small fraction. For typical clinical documents (10-500 KB), latency stays <1 s.

8.5 Storage Overhead

Encryption overhead: AES-GCM adds 12 bytes (nonce) + 16 bytes (tag) = 28 bytes per record. For 1 MB plaintext: overhead 0.003%. Content-addressing (IPFS CID) adds 32-byte identifier. Total storage penalty negligible.

On-chain storage per record: 32 bytes (digest) + 50 bytes (storage pointer string) + 65 bytes (wrapped key) $\approx$ 147 bytes. At 20,000 gas/kB storage cost, per-record storage cost ∼3,000 gas—small fraction of total transaction cost.

9 Privacy and Regulatory Compliance

9.1 Metadata Privacy

Public Information: Storage pointers *ptr* and content digests *d* are public on-chain, visible in events and contract storage. This transparency is *by design*—pointers reference exclusively *encrypted* content. Without wrapped keys, adversaries obtain only ciphertext.

Metadata Leakage: On-chain data reveals: (1) Which addresses participate in health data sharing; (2) How many records each patient has; (3) When records are created/accessed; (4) Which recipients have permissions.

Mitigation: Recipients concerned about linkability should use fresh addresses per patient via HD wallet derivation (BIP-32/BIP-44). Patients can use mixing services or privacy-preserving layer-2s (zkSync) to obscure transaction origins. However, complete metadata privacy contradicts auditability—trade-off between transparency and privacy must be balanced per deployment requirements.

9.2 HIPAA Compliance

Health Insurance Portability and Accountability Act (HIPAA) mandates safeguards for protected health information (PHI). Our system addresses HIPAA requirements:

Administrative Safeguards: Patients define access policies through cryptographic permissions. Audit logs track all authorizations. Risk analysis identifies vulnerabilities (key management, storage availability).

Physical Safeguards: Encrypted storage prevents PHI exposure during theft/breach. Hardware wallets protect private keys.

Technical Safeguards: Authentication (ECDSA signatures), encryption (AES-256), integrity (Keccak-256), audit trails (blockchain events), transmission security (TLS for off-chain communication).

Access Control: Role-based access through permission grants. Patient controls who accesses what, when. Emergency access mechanisms (Section X.C) balance safety with privacy.

9.3 GDPR Compliance

General Data Protection Regulation (GDPR) grants data subjects rights over personal data. Our architecture supports GDPR requirements:

Right to Access: Patients always have permission to their records. Can query blockchain for metadata and retrieve encrypted data anytime.

Right to Portability: FHIR format enables data export. Patients can download encrypted records, decrypt with their keys, and transfer to other systems.

Right to Erasure: Permission revocation prevents future access. Key rotation with storage deletion (removing old ciphertexts) approximates "forgetting." However, blockchain immutability means on-chain metadata (pointers, digests, permission logs) persist. This tension between GDPR's "right to be forgotten" and blockchain's immutability requires careful legal interpretation—possibly treating on-chain data as audit logs exempt from erasure under regulatory compliance grounds.

Data Minimization: Only encrypted data and authorization metadata stored. No PHI on blockchain.

Consent Management: EIP-712 signatures provide explicit, informed consent for each data sharing event.

9.4 De-Identified Data for Research

Healthcare research requires large datasets while protecting privacy. We support de-identification pipelines creating research-safe datasets.

De-Identification Methods: We implement Safe Harbor method (HIPAA) removing 18 identifier categories: names, addresses (except ZIP regions), dates (except year), phone numbers, emails, SSNs, medical record numbers, account numbers, certificate numbers, vehicle identifiers, device identifiers, URLs, IP addresses, biometric identifiers, full-face photos, and other unique identifiers.

Pipeline Architecture: Patients select records for research contribution. De-identification service: (1) Decrypts records using patient-authorized temporary key; (2) Applies transformations (identifier removal, quasi-identifier generalization, date shifting); (3) Re-encrypts with research consortium key; (4) Uploads to research database with new permissions.

Performance: Evaluation on 1,000 Synthea FHIR bundles (mean 127 KB): de-identification takes mean 47.3 ms/record (95th: 68.5 ms). Processing 1M records requires ∼13 hours sequential, parallelizable to <1 hour with 32-core cluster.

Re-Identification Risk: Even de-identified data carries re-identification risk through quasi-identifiers. We apply k-anonymity ($k \geq 5$) [25] and evaluate with ARX Data Anonymization Tool. Results show re-identification risk <0.2% for generalized datasets.

10 Discussion and Future Work

10.1 Deployment Barriers

Integration Challenges: Most EHR systems don't natively support blockchain. Middleware or API gateways can bridge, but introduce operational dependencies. Standards development and vendor cooperation are essential.

User Experience: Non-technical patients need intuitive interfaces hiding cryptographic complexity. Wallet software must balance security (key protection) with usability (avoiding lock-out). Social recovery mechanisms—where trusted contacts help restore access—show promise but need careful design to avoid vulnerabilities.

Economic Models: Gas costs require sustainable funding. Options: (1) Healthcare institutions subsidize as infrastructure cost; (2) Insurance covers as benefit; (3) Patients pay directly (raises equity concerns); (4) Tiered models with basic access funded by institutions.

10.2 Institutional Key Management

Per-recipient key wrapping becomes impractical for large institutions (hospitals with hundreds of staff). We propose **institutional guardian keys**:

Design: Organizations represented by single encryption key managed institutionally. Patient wraps $SymmK$ once for hospital's key. Hospital maintains internal access control determining which staff can decrypt. Reduces on-chain operations from $O(n)$ per institution (where n = staff count) to $O(1)$.

Implementation: Institution deploys guardian contract: (1) Registers encryption public key; (2) Maintains staff roster; (3) Provides `requestDecryption`$(rid, patientContract)$ for staff; (4) Verifies caller is authorized staff; (5) Uses institutional private key (in HSM) to unwrap $SymmK$ and re-wrap for requesting clinician.

Trade-offs: Trades patient-controlled fine-grained access for scalability/usability. Patients trust institution to enforce internal policies rather than controlling individual clinicians directly. Matches real-world practice: patients authorize "my hospital" rather than enumerating every provider. On-chain audit still records institutional-level grants/revocations.

10.3 Emergency Access

Medical emergencies sometimes require immediate access when patient cannot grant permission (unconscious, incapacitated). Current design prioritizes patient control, omitting emergency access.

Two-Physician Multisignature Pattern: Emergency access requires signatures from two independent attending physicians. Contract implements `emergencyGrantAccess` accepting two physician signatures, record ID, justification code, and two-hour expiration. Contract verifies both signatures against pre-registered emergency physicians, validates justification, confirms expiration is exactly two hours, and records emergency grant with `EmergencyAccessGranted` event.

Immediately triggers off-chain notification to patient (SMS, email, push) including which physicians accessed which records, justification, and timestamp. Permission auto-expires after two hours. Patient can proactively revoke before expiration. All emergency accesses permanently recorded with special flags for compliance review.

Accountability Through Receipts: Emergency access must post on-chain receipt within 24 hours via `confirmEmergencyAccess`$(rid, justificationHash)$. Receipt costs ∼35,000 gas but ensures all retrievals appear in immutable audit log.

10.4 Advanced Cryptographic Enhancements

Proxy Re-Encryption: Allows patients to delegate re-encryption to semi-trusted proxy, enabling efficient re-sharing without patient remaining online or re-encrypting. Introduces operational complexity but could improve usability for frequent re-sharing.

Attribute-Based Encryption: Encodes access policies directly in ciphertexts, automatically enforcing conditions like "any cardiologist in my hospital." Introduces computational overhead and complex key management.

Zero-Knowledge Proofs: Enables proving permission validity without revealing access control policy. Provider could prove they have valid access without disclosing which permission grant they use, obscuring metadata about sharing patterns.

10.5 Cross-Chain Interoperability

Healthcare is global involving diverse stakeholders potentially operating on different blockchains. Cross-chain interoperability protocols could enable permission grants on one blockchain recognized on another, or aggregate records across multiple blockchains.

Polkadot [26] provides heterogeneous multi-chain framework through relay chains and parachains. Cosmos offers Inter-Blockchain Communication for sovereign blockchain interoperation. Atomic swaps or blockchain bridges could enable interoperability. However, each approach introduces complexity and trust assumptions requiring careful evaluation for sensitive medical data.

10.6 Formal Verification

Smart contracts manage safety-critical assets (health data). Bugs have severe consequences. Formal verification tools (Certora, K Framework, Coq) can mathematically prove contracts satisfy specified properties, providing stronger assurance than testing. Future work includes formal verification of our contract against confidentiality, integrity, and authorization properties.

10.7 Long-Term Data Stewardship

Digital estate planning mechanisms could allow patients to designate heirs or archival repositories for records. Medical research could benefit from posthumous data donation through advance directives recorded on-chain. Decentralized autonomous organizations (DAOs) could distribute governance across stakeholders with on-chain voting on protocol upgrades.

11 Limitations

Key Compromise: If patient's SK_P is stolen, adversary can sign arbitrary authorizations. Multi-signature wallets, hardware wallets, and social recovery reduce risk but add complexity.

Revocation Limits: Revocation prevents future access but cannot retract already-decrypted plaintext—fundamental limitation of cryptographic access control.

Storage Availability: Depends on off-chain infrastructure. Storage provider failure makes records inaccessible until restored. Redundant storage mitigates but increases cost.

Transaction Costs: Gas prices fluctuate. High congestion makes L1 expensive. L2 solutions reduce costs but introduce additional trust assumptions (optimistic rollup fraud proofs, zkRollup trusted setup).

Scalability: Per-recipient key wrapping doesn't scale to very large recipient lists (e.g., sharing with 1,000+ researchers). Institutional guardian keys and proxy re-encryption address some cases, but massive-scale sharing requires additional techniques.

Metadata Privacy: On-chain transparency reveals sharing patterns. While not exposing PHI, metadata can enable inference attacks (frequency analysis, timing correlation). Complete metadata privacy contradicts auditability—fundamental trade-off requiring deployment-specific balance.

12 Conclusion

We presented a patient-centric blockchain architecture for health record management that cryptographically enforces access control while maintaining practical performance. By separating encrypted storage from on-chain authorization, we achieve confidentiality against curious storage providers, tamper-evident audit trails, and patient sovereignty over medical data.

Our Ethereum implementation demonstrates feasibility with reasonable gas costs (78,000 gas per permission grant on L1, 6,000 gas on L2) and latency (1.3–2.0 s end-to-end for 1 MB records). Performance analysis shows cryptographic overhead is negligible compared to network/storage latency. Layer-2 deployment reduces costs 10-13×, making the system economically viable at scale.

Security analysis establishes that standard cryptographic primitives (AES-256-GCM, ECIES, ECDSA) composed correctly provide desired properties: confidentiality, integrity, authorization authenticity, and auditability. Integration with FHIR standards enables interoperability with existing healthcare systems.

The architecture addresses key challenges in healthcare data management: eliminates trusted intermediaries, provides cryptographic rather than policy-based access control, creates immutable audit trails, and restores patient agency over sensitive medical information. While not solving all healthcare IT problems (key management, emergency access, metadata privacy require ongoing research), this work establishes a foundation for truly patient-controlled health information exchange.

Future work includes formal verification of smart contracts, enhanced privacy through zero-knowledge proofs, cross-chain interoperability for global health data networks, and user studies evaluating real-world adoption barriers. The path forward requires collaboration among cryptographers, healthcare informaticists, policymakers, and patient advocates to realize the vision of patient-empowered, secure, and interoperable health data infrastructure.

Acknowledgments

This work was supported by the Department of Computer Science & Engineering at Bangladesh University of Engineering & Technology. We thank the Ethereum Foundation and IPFS community for open-source tools enabling this research.

References

[1] J. R. Vest and L. D. Gamm, “Health information exchange: Persistent challenges and new strategies,” *J. Am. Med. Inform. Assoc.*, vol. 17, no. 3, pp. 288–294, May 2010.

[2] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, “MedRec: Using blockchain for medical data access and permission management,” in *Proc. IEEE OBD*, Aug. 2016, pp. 25–30.

[3] A. Ekblaw, A. Azaria, J. D. Halamka, and A. Lippman, “A case study for blockchain in healthcare: MedRec prototype,” in *Proc. IEEE OBD*, Aug. 2016.

[4] K. Peterson, R. Deeduvanu, P. Kanjamala, and K. Boles, “A blockchain-based approach to health information exchange networks,” in *Proc. NIST Workshop Blockchain Healthcare*, 2016.

[5] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, “Healthcare data gateways: Found healthcare intelligence on blockchain,” *J. Med. Syst.*, vol. 40, no. 10, p. 218, Oct. 2016.

[6] K. N. Griggs et al., “Healthcare blockchain system using smart contracts,” *J. Med. Syst.*, vol. 42, no. 7, p. 130, June 2018.

[7] K. M. Hasib et al., “Blockchain-based electronic health records management: A comprehensive review,” *IEEE Access*, vol. 10, pp. 11411–11434, 2022.

[8] C. C. Agbo, Q. H. Mahmoud, and J. M. Eklund, “Blockchain technology in healthcare: A systematic review,” *Healthcare*, vol. 7, no. 2, p. 56, Apr. 2019.

[9] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, “Blockchain distributed ledger technologies for biomedical and health care applications,” *J. Am. Med. Inform. Assoc.*, vol. 24, no. 6, pp. 1211–1220, Nov. 2017.

[10] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, “Patient controlled encryption: Ensuring privacy of electronic medical records,” in *Proc. ACM CCSW*, Nov. 2009, pp. 103–114.

[11] B. Fisher et al., “PGP-HCCA: Pretty good privacy for health care client applications,” in *Proc. IEEE ICHI*, Sept. 2013.

[12] S. Wang, Y. Zhang, and Y. Zhang, “A blockchain-based framework for data sharing with fine-grained access control,” *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[13] A. A. Omar et al., “Privacy-friendly platform for healthcare data in cloud based on blockchain,” *Future Gener. Comput. Syst.*, vol. 95, pp. 511–521, June 2019.

[14] P. Zhang, J. White, D. C. Schmidt, G. Lenz, and S. T. Rosenbloom, “FHIRChain: Applying blockchain to securely share clinical data,” *Comput. Struct. Biotechnol. J.*, vol. 16, pp. 267–278, 2018.

[15] P. Zhang and D. C. Schmidt, “Blockchain for health data and its potential use in health IT and health care,” *JAMA*, vol. 319, no. 23, pp. 2363–2364, June 2018.

[16] D. C. Nguyen et al., “ACTION-EHR: Patient-centric blockchain-based electronic health record data management,” *J. Netw. Comput. Appl.*, vol. 178, p. 102987, Mar. 2021.

[17] NIST, "Advanced Encryption Standard (AES)," FIPS PUB 197, Nov. 2001.

[18] M. Dworkin, "Recommendation for block cipher modes: GCM and GMAC," NIST SP 800-38D, Nov. 2007.

[19] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis," in *Proc. ASIACRYPT*, Dec. 2000, pp. 531–545.

[20] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle Diffie-Hellman assumptions and DHIES," in *Proc. CT-RSA*, Apr. 2001, pp. 143–158.

[21] V. Shoup, "A proposal for an ISO standard for public key encryption," ISO/IEC JTC 1/SC27, Dec. 2001.

[22] R. Schiff et al., "EIP-712: Typed structured data hashing and signing," Ethereum Improvement Proposal, Sept. 2017.

[23] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Yellow Paper, 2014.

[24] J. Benet, "IPFS - content addressed, versioned, P2P file system," *arXiv:1407.3561*, July 2014.

[25] L. Sweeney, "k-anonymity: A model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002.

[26] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," White Paper, 2016.