A Blockchain-Based Role-Based Access Control Gateway for Secure Electronic Health Record Access and Tamper-Evident Auditing

Osamah Saleh, Shouki A. Ebad
Department of Computer Science-College of Science, Northern Border University, Arar, Saudi Arabia[1]

***Abstract*—Electronic Health Record (EHR) systems store sensitive patient information and require strong access control and reliable audit records. Traditional centralized Role-Based Access Control (RBAC) systems may be vulnerable to unauthorized access, privilege escalation, audit-log modification, and undetected changes to stored records. This study developed a lightweight blockchain-based RBAC gateway integrated with OpenMRS. The system used three roles, Admin, Doctor, and Patient, and one Solidity smart contract to manage role assignments, Doctor–Patient permissions, access decisions, trusted record hashes, and blockchain audit events. A Node.js gateway acted as the policy-enforcement point, while SQLite was used as a centralized audit-log baseline. The system was evaluated through authorized-access, unauthorized-access, privilege-escalation, audit-log tampering, EHR data-tampering, gas-use, and throughput experiments. All 50 measured authorized requests were allowed and completed the expected OpenMRS retrieval workflow, while all 50 unauthorized-access attempts were blocked before reaching OpenMRS. The SQLite decision could be changed from deny to allow, while the corresponding blockchain event remained unchanged. The trusted-hash experiment detected all 10 simulated EHR modifications. Median end-to-end latency was 31.8 ms for denied requests and 468.9 ms for authorized requests. Sequential smart contract throughput averaged 15.0850 TPS for authorized access and 15.4214 TPS for unauthorized access. These findings show that the proposed gateway can improve access-control enforcement, detect off-chain record changes, and provide tamper-evident audit evidence without storing full EHR records on-chain.**

Keywords*—Electronic health records; role-based access control; blockchain; smart contracts; OpenMRS; audit logs; access control; cybersecurity

I. Introduction

Electronic Health Record (EHR) systems are widely used by healthcare organizations to store, manage, and exchange patient information in digital form. EHRs can include personal information, diagnoses, medications, allergies, laboratory results, treatment plans, and clinical notes. EHR systems can improve healthcare quality, efficiency, documentation, and access to clinical information [1]. However, because EHR systems include sensitive data that should only be accessible by authorized users, using digital health records also poses significant cybersecurity and privacy threats [2].

Healthcare organizations must protect the confidentiality, integrity, and availability of patient records. Confidentiality means that medical information is only available to permitted users. Integrity means that data and system records cannot be changed without authorization. Availability means that authorized healthcare users can access the required information when it is needed. These security goals are important in healthcare because unauthorized access or manipulation of patient information can cause privacy violations, incorrect medical decisions, and loss of trust in the healthcare system [3]. Healthcare software projects can also face significant design and implementation challenges, particularly when system requirements, architecture, and stakeholder needs are not adequately aligned [21].

Access control is one of the main security mechanisms used to protect EHR systems. Role-Based Access Control (RBAC) is commonly used because it assigns permissions to roles rather than directly to individual users. Users then receive permissions according to their assigned organizational role [4]. For example, an administrator may manage user roles, a doctor may access the records of assigned patients, and a patient may access only their own medical record. RBAC supports the principle of least privilege because each user should receive only the permissions required to perform their work [3,4].

Many traditional RBAC implementations depend on a centralized server or database that manages role assignments, access decisions, and audit logs. This creates a single point of failure. If an external attacker or malicious insider gains sufficient privileges, the attacker may access medical records without permission, assign higher roles, or modify audit logs to hide evidence of misuse. Centralized logs are important for monitoring and incident investigation, but their value is reduced if a privileged attacker can alter or delete them [5].

To ensure accountability, audit logs must document who requested access, what resource was requested, when the request was made, and whether access was approved or denied. According to NIST log-management guidance, security logs support monitoring, incident response, troubleshooting, and forensic investigation [5]. However, storing audit evidence only in a conventional centralized database may not provide strong protection against an administrator or attacker who has direct access to that database.

Blockchain technology provides a possible method for improving audit-log integrity. A blockchain is a distributed ledger in which transactions are grouped into blocks and linked using cryptographic hashes. This structure makes previously recorded transactions difficult to modify without detection [6]. Smart contracts can also automatically enforce access-control

rules and record access decisions as blockchain events. Therefore, blockchain can be used as a tamper-evident audit layer while the full medical records remain outside the blockchain.

This study develops a lightweight blockchain-based RBAC gateway integrated with OpenMRS. OpenMRS is used as the off-chain EHR platform containing synthetic patient records. A Node.js gateway acts as the policy-enforcement point between users and OpenMRS. Before an EHR record is retrieved, the gateway sends an access request to a Solidity smart contract deployed on a local Ethereum-compatible blockchain. The smart contract checks the requester's role and doctor–patient permission and records the access decision in a tamper-evident blockchain audit trail.

The system applies Admin, Doctor, and Patient roles with explicit Doctor–Patient permissions. The implementation also uses a centralized SQLite audit log as a comparison baseline. The system was evaluated through authorized-access requests, unauthorized-access attempts, privilege-escalation attempts, audit-log tampering experiments, EHR data-tampering experiments, gas-use measurements, and sequential throughput testing.

The main contributions of this study are as follows:

- A lightweight smart contract-based RBAC model for controlling access to synthetic EHR records stored in OpenMRS.
- An explicit Doctor–Patient permission model in which Doctors can access only assigned Patient records.
- A blockchain audit mechanism that records both successful and denied access requests.
- A gateway architecture that prevents unauthorized OpenMRS record retrieval before the EHR platform is contacted.
- An experimental comparison between tamper-evident blockchain audit records and a mutable centralized SQLite audit log.
- A security evaluation covering unauthorized access, privilege escalation, audit-log tampering, and EHR data tampering.
- On-chain EHR hash verification for detecting simulated off-chain record modification.
- A performance evaluation covering blockchain latency, end-to-end latency, gas usage, and sequential smart contract throughput.

II. Related Work

Several studies have investigated blockchain-based access control, auditability, and medical-record management. These studies show that blockchain can support tamper-evident access records and automated authorization, but they differ in access-control model, blockchain platform, storage design, and evaluation approach.

Ullah et al. proposed a blockchain-enabled framework for EHR access auditing using Purpose-Based Access Control (PBAC) and smart contracts [7]. Their framework records access events in an immutable audit trail and verifies whether an access request is consistent with the defined access purpose. The study mainly focuses on audit accountability and policy verification. In contrast, the present study uses a simpler RBAC model and evaluates specific security scenarios, including unauthorized access, privilege escalation, centralized audit-log tampering, and EHR data tampering.

Joshi and Mahajan proposed MrBlock, a secure and interoperable EHR-management framework based on Hyperledger Fabric [8]. MrBlock combines RBAC and Attribute-Based Access Control (ABAC) with smart contracts and uses the Raft consensus mechanism. Their evaluation reported lower average latency and higher throughput than a PBFT-based configuration. MrBlock provides a broader healthcare-sharing framework, while the present work focuses on a lightweight gateway architecture integrated with OpenMRS and uses one smart contract for access decisions, audit events, and trusted record hashes.

Taloba and Rayan proposed a privacy-preserving medical-data management framework using blockchain-enabled encrypted RBAC [9]. Their system combines smart contracts, encrypted storage, blockchain, and IPFS to reduce unauthorized access, data leakage, and privilege misuse. This approach provides a more comprehensive privacy-management architecture. The present work instead focuses on a smaller proof of concept in which OpenMRS remains the off-chain EHR platform, while blockchain is used only for access-control decisions, trusted hashes, and tamper-evident audit evidence.

Yaqub et al. developed a blockchain-enabled policy-based access-control mechanism for preventing unauthorized EHR access [10]. Their model uses smart contracts to enforce policy and consent rules and record access decisions. Although this approach also uses blockchain for automated authorization, it is based on PBAC rather than RBAC. The present study uses three explicit RBAC roles and Doctor–Patient permission mappings so that the authorization rules can be evaluated through clear and repeatable security test cases.

Tahir et al. proposed a blockchain-based healthcare-records management framework focused on security, privacy, and interoperability [11]. Their work addresses broader healthcare-record management problems and explains how blockchain can reduce centralized trust and improve integrity. The current study has a narrower scope and concentrates specifically on access enforcement, audit-log comparison, record-integrity verification, and measurable security testing in an OpenMRS-based environment.

MedRec, proposed by Azaria et al., is one of the early blockchain-based systems for medical-data access and permission management [12]. MedRec demonstrated that blockchain can be used to manage permissions and maintain an auditable history of medical-record access while the medical records themselves remain outside the blockchain. This separation between on-chain permission evidence and off-chain EHR data is also reflected in the present design. However, MedRec was designed as a broader medical-record permission-

management system and was not evaluated using the specific attack scenarios considered in this study.

Table I compares the reviewed systems with the proposed approach.

TABLE I. COMPARISON OF RELATED BLOCKCHAIN-BASED EHR APPROACHES

Study	Access-control model	Blockchain/ platform	EHR storage approach	Audit mechanism
Ullah et al. [7]	PBAC	Blockchain with smart contracts	EHR remains outside audit layer	Immutable access audit trail
MrBlock [8]	RBAC + ABAC	Hyperledger Fabric / Raft	Distributed EHR-management framework	Blockchain-based transaction records
Taloba and Rayan [9]	Encrypted RBAC	Blockchain + smart contracts + IPFS	Encrypted distributed/off-chain storage	Tamper-resistant access records
Yaqub et al. [10]	PBAC	Blockchain / smart contracts	Off-chain EHR with policy enforcement	Blockchain access-decision records
Tahir et al. [11]	Smart-contract-based control	Blockchain healthcare framework	Hybrid healthcare-record management	Blockchain-based auditability
MedRec [12]	Permission management	Ethereum-based blockchain	Medical records remain off-chain	Blockchain permission/access history
Proposed system	RBAC with Doctor–Patient mapping	Local Ethereum-compatible blockchain	OpenMRS off-chain; trusted hashes on-chain	Blockchain audit events plus SQLite comparison baseline

The comparison shows that most previous studies focus on broader EHR-sharing, interoperability, privacy, or policy-management frameworks. Fewer studies combine a lightweight RBAC gateway, a real open-source EHR platform, direct comparison between blockchain and a mutable centralized audit log, attack-based security testing, and trusted off-chain record-hash verification within the same proof of concept. The novelty of the present study is therefore not the use of blockchain or RBAC individually, but the integration of these mechanisms into a focused OpenMRS gateway that is evaluated through explicit security attacks and measurable audit, integrity, latency, gas, and throughput results.

III. MATERIALS AND METHODS

A. Research Design

This study followed a design and experimental evaluation approach. A lightweight blockchain-based Role-Based Access Control (RBAC) gateway was designed, implemented, and tested using a local laboratory environment.

The implementation did not replace the internal access-control system of OpenMRS. Instead, it introduced an external security gateway between the requesting user and the OpenMRS REST API. All EHR access requests passed through this gateway before any patient record was retrieved.

B. System Architecture

The system consisted of five main components:

- An OpenMRS EHR platform containing synthetic patient records.
- A Node.js and Express gateway acting as the policy-enforcement point.
- A Solidity smart contract deployed on a local Ethereum-compatible blockchain.
- A centralized SQLite audit-log database used as a comparison baseline.
- Automated scripts used to configure the environment, run experiments, and collect results.

Fig. 1 presents the interaction among the gateway, smart contract, OpenMRS, and SQLite baseline. The detailed access process is described in Section III-A. The main trust relationships and assumed attacker capabilities are defined in the following threat model.

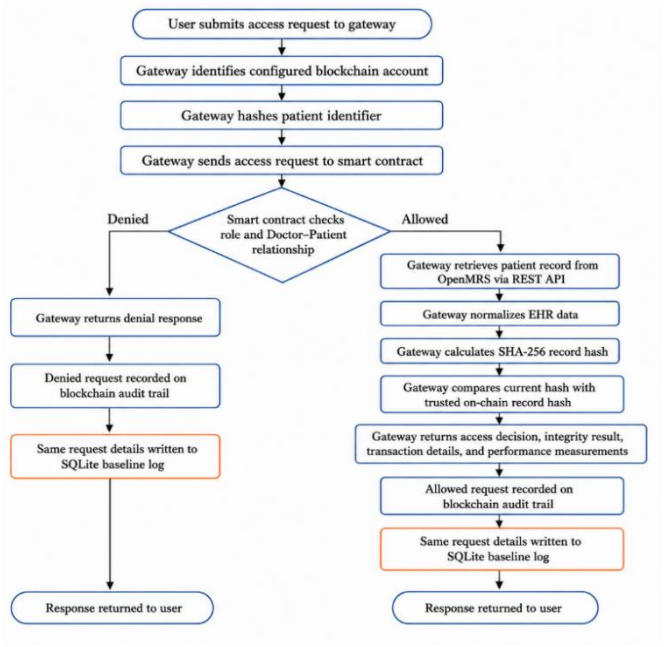

Fig. 1. Workflow of the proposed blockchain-based RBAC gateway for authorized and denied EHR access requests.

C. Threat Model and Trust Assumptions

The prototype assumes that the local blockchain network and deployed smart contract operate correctly and that previously recorded blockchain events cannot be modified through the normal application interfaces. The admin blockchain account is treated as trusted because it is responsible for assigning roles, granting Doctor–Patient permissions, linking Patient accounts, and registering trusted EHR hashes.

The gateway is also treated as a trusted enforcement component in the current proof of concept. It maps configured users to blockchain accounts, submits access requests to the smart contract, retrieves authorized records from OpenMRS, performs record hashing, and writes the centralized SQLite audit entry. Compromise of the gateway is therefore outside the protection provided by the current prototype and could allow request contents or user mappings to be manipulated. The trust

boundaries, trusted components, sensitive credentials, and evaluated attacker paths are illustrated in Fig. 2.

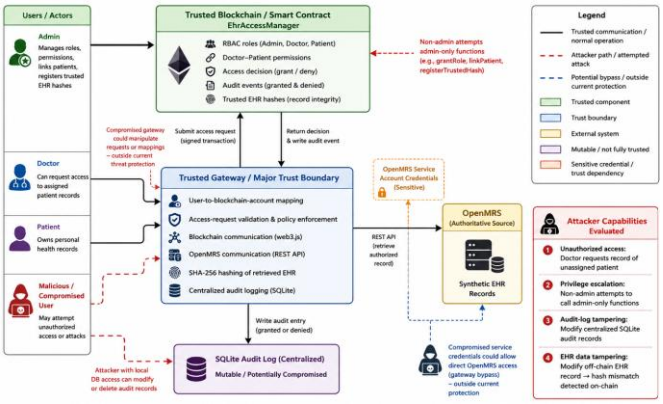

Fig. 2. Trust boundaries, trusted components, sensitive credentials, and potential attacker paths in the proposed blockchain-based RBAC gateway architecture.

OpenMRS is treated as the authoritative off-chain source of the synthetic EHR data, while the OpenMRS service credentials are considered sensitive trusted credentials. The gateway uses these credentials only after blockchain authorization. If the service credentials are compromised and OpenMRS can be accessed directly, an attacker could bypass the blockchain gateway. This limitation is considered part of the system trust boundary.

Doctor and Patient blockchain accounts are not assumed to be trusted. They may submit unauthorized access requests or attempt operations outside their assigned permissions. The smart contract is responsible for enforcing the defined RBAC policy against these accounts.

The SQLite audit log is intentionally treated as a mutable and potentially compromised component. An attacker with sufficient local database access may modify its contents. The blockchain audit trail provides the independent reference used to detect such modification.

The evaluated attacker model therefore includes: 1) a Doctor attempting unauthorized access to an unassigned Patient record; 2) a non-admin account attempting privilege escalation; 3) an attacker modifying the centralized SQLite audit record; and 4) modification of an off-chain EHR representation that causes a mismatch with the trusted on-chain hash. The current evaluation does not assume compromise of the blockchain consensus mechanism, Admin private key, gateway host, or OpenMRS service credentials.

D. Security Properties

The security properties evaluated in this study are defined as follows. Access-control enforcement means that an access request is granted or denied according to the implemented RBAC role and Doctor–Patient permission rules. Tamper-evidence means that modification of the centralized SQLite audit record can be detected by comparing it with the corresponding blockchain audit event. Record integrity verification means that the SHA-256 hash calculated from the current canonicalized EHR record matches the trusted hash registered on-chain. Privilege-escalation prevention means that a non-admin account cannot successfully perform Admin-only role-management operations.

The term secure access is used in this study only to describe access that has passed the implemented authorization checks. It does not imply complete end-to-end security of the gateway, OpenMRS platform, user authentication, service credentials, or blockchain infrastructure (Table II).

TABLE II. MAPPING OF SECURITY PROPERTIES TO EXPERIMENTS

Security property	Experiment	Validation criterion
Access-control enforcement	Authorized-access test	Assigned Doctor request is allowed according to RBAC policy
Access-control enforcement	Unauthorized-access test	Unassigned Doctor request is denied before OpenMRS retrieval
Privilege-escalation prevention	Privilege-escalation test	Non-admin role-management request is rejected, and role remains unchanged
Tamper-evidence	Audit-log tampering test	SQLite modification creates a discrepancy with the unchanged blockchain audit event
Record integrity verification	EHR data-tampering test	Modified EHR content produces a hash mismatch with the trusted on-chain hash
Record integrity verification	JSON canonicalization test	Equivalent JSON representations produce the same hash, while changed clinical content produces a different hash
Record provenance	Versioned record-update test	New trusted state increments the version and links to the previous trusted hash
Workflow outcome consistency	Failure and partial-operation tests	Outcome distinguishes blockchain authorization from EHR retrieval success or failure
Duplicate-request prevention	Duplicate-request test	Repeated requestId is rejected before a second access operation is processed

E. Gateway Identity and Request Integrity

In the current proof of concept, the gateway uses a predefined mapping between synthetic actor names and blockchain accounts. This mapping is stored as trusted local configuration and is used to select the blockchain account that submits each access request. Therefore, the prototype assumes that the gateway configuration has not been modified and that the actor-to-account mapping is correct.

The current implementation does not cryptographically bind an external human identity to the submitted request payload. A compromised gateway could therefore alter the actor mapping, patient identifier, or requested action before submitting the transaction to the smart contract. This limitation is outside the protection provided by the present proof of concept.

In a production deployment, users should first be authenticated using a secure identity mechanism. Access requests should then be cryptographically signed using the user's wallet or another trusted signing mechanism. The signed request should include the user identity, patient identifier, requested action, nonce, and timestamp. The gateway should verify the signature, nonce, and timestamp before submitting the request to the smart contract. This would bind the authenticated

user to the request contents and reduce the risk of gateway-level request manipulation and replay attacks.

F. Development and Experimental Environment

The prototype was implemented in a local and controlled testing environment. OpenMRS was selected because it is an open-source medical-record platform that provides patient-management functions and REST API support [13]. The blockchain access-control logic was implemented using Solidity [14] and deployed through the Hardhat development environment [15].

The gateway communicated with the smart contract, retrieved authorized records from OpenMRS, calculated record hashes, wrote centralized audit logs, and collected experimental measurements.

SQLite was used as the centralized audit-log baseline because it provides a lightweight relational database that can be stored and modified locally [16].

The experimental environment is detailed in Table III.

TABLE III. EXPERIMENTAL ENVIRONMENT AND SOFTWARE TOOLS

Component	Tool or specification
EHR platform	OpenMRS Reference Application 3.7.0
EHR database	MariaDB
Smart-contract language	Solidity 0.8.24
Local blockchain framework	Hardhat 3.3.0
Blockchain library	Ethers.js 6.16.0
Gateway runtime	Node.js 22.16.0
Gateway framework	Express 5.2.1
Centralized audit database	SQLite
Hash algorithm	SHA-256
Testing scripts	JavaScript and PowerShell
Operating system	Microsoft Windows 11 Home 64-bit, version 10.0.26200 (Build 26200)
Processor	Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz, 6 cores / 12 logical processors
Memory	16 GB RAM

G. Smart-Contract Design

One smart contract named EhrAccessManager was developed to manage roles, patient relationships, access decisions, record hashes, and blockchain audit records. The contract represented the Admin, Doctor, Patient, and unassigned account states.

The blockchain account that deployed the contract automatically received the Admin role. The Admin was allowed to assign and revoke roles, link a Patient account to a patient record, grant or revoke Doctor access to a Patient, and register a trusted EHR record hash.

The main smart contract functions were:

- assignRole, which assigned a role to a blockchain account;
- revokeRole, which removed an assigned role;
- linkPatientRecord, which linked a Patient account to a hashed patient identifier;
- grantDoctorPatientAccess, which granted a Doctor permission to access a specific Patient;
- revokeDoctorPatientAccess, which removed this permission;
- storePatientRecordHash, which created a new versioned EHR commitment containing the current record hash and a link to the previous trusted hash;
- verifyPatientRecordHash, which compared a calculated record hash with the trusted on-chain hash;
- requestAccess, which evaluated an access request and created an audit event;
- canAccess, which returned the expected access decision without changing the blockchain state;
- getAuditEvent, which returned a previously stored audit record.
- getLatestPatientRecordCommitment, which returned the latest version number, record hash, previous hash, timestamp, and updater address;
- getPatientRecordVersionCount, which returned the number of trusted record versions registered for a patient.

Role-management and trusted-hash functions were protected using an Admin-only modifier. A non-admin blockchain account attempting to call these functions caused the transaction to be rejected.

H. Access-Control Policy

The implemented access policy used three active roles: Admin, Doctor, and Patient. The implemented access rules are summarized in Table IV.

TABLE IV. IMPLEMENTED RBAC PERMISSIONS

Role	Access permission
Admin	Access all patient records and manage roles, permissions, patient links, and trusted hashes
Doctor	Access only explicitly assigned Patient records
Patient	Access only the record linked to the Patient's blockchain address
None	No access to patient records

The Doctor–Patient mapping provided more specific control than allowing every Doctor to access every medical record. This supported the least-privilege principle because a Doctor received access only to assigned Patients.

I. Gateway Access Workflow

The gateway received access requests containing an actor name, patient identifier, and requested action. The actor name represented a synthetic user configured in the controlled experiment. Each configured actor was associated with a blockchain account and private key stored in the local testing environment.

For every access request, the gateway performed the following steps:

- It validated the actor name, patient identifier, and requested action.
- It selected the blockchain account associated with the actor.
- It converted the patient identifier into a bytes32 hash using the Ethereum Keccak-256 hashing function.
- It submitted the access request to the smart contract.
- It read the resulting access decision and blockchain audit event.
- If access was denied, it returned an HTTP denial response and did not contact OpenMRS.
- If access was granted, it retrieved the corresponding patient record from OpenMRS.
- It canonicalized the returned JSON data using a deterministic serialization procedure before hashing.
- It generated a SHA-256 hash of the canonicalized record.
- It compared the calculated hash with the trusted record hash stored on-chain.
- It wrote an equivalent audit entry to SQLite.
- It recorded latency, gas usage, transaction-fee information, and supporting evidence.

Before calculating the SHA-256 hash, the gateway converted the retrieved EHR JSON into a deterministic representation. Object keys were sorted recursively in lexical order, while array element order was preserved. The resulting structure was serialized as compact JSON without additional whitespace and encoded using UTF-8 before the SHA-256 digest was calculated. This procedure ensured that differences in object-key ordering or formatting did not produce different hashes for logically equivalent records [18].

SHA-256 was used because it produces a fixed 256-bit digest suitable for detecting changes in the EHR record [17].

The gateway used one OpenMRS service account to retrieve approved records. Therefore, the blockchain and gateway provided the research access-control layer, while OpenMRS remained the off-chain EHR platform. This design was suitable for the local proof of concept but was not intended to represent a complete production authentication system.

J. Synthetic EHR Dataset

Only synthetic and non-identifying patient information was used. The experimental dataset contained five synthetic patient records stored in OpenMRS, as summarized in Table V. Each logical patient identifier was mapped to an OpenMRS UUID. The records contained simplified information such as diagnosis and medication.

The patient identifiers used in blockchain operations were hashed before being stored or included in access-control mappings. The full medical records were not stored on-chain. Only hashed identifiers, access decisions, role-related information, record hashes, timestamps, and audit information were placed on the blockchain.

TABLE V. SYNTHETIC EHR RECORDS USED IN THE EXPERIMENTS

Patient identifier	Example diagnosis	Example medication
P001	Hypertension	Amlodipine 5 mg
P002	Type 2 diabetes	Metformin 500 mg
P003	Asthma	Salbutamol inhaler
P004	Hyperlipidemia	Atorvastatin 20 mg
P005	Ischemic heart disease	Aspirin 81 mg

K. Trusted Record-Hash Registration and Verification

To support EHR integrity verification, the Admin first retrieved an approved patient record from OpenMRS through a trusted registration operation. The gateway canonicalized the returned JSON record and calculated its SHA-256 hash. The Admin then stored this value in the smart contract as the trusted hash for that patient identifier.

The trusted hash was not automatically replaced during ordinary access requests. This was necessary because automatically storing the latest hash could allow an unauthorized modification to become accepted as the new trusted state.

During every successful EHR access, the gateway calculated the current record hash and compared it with the trusted on-chain value. A mismatch indicated that the current OpenMRS record differed from the previously registered trusted version.

A legitimate update to a patient record required a separate Admin-controlled registration operation that created a new trusted commitment while preserving the previous version in the on-chain history.

L. Versioned Record Commitments

The trusted EHR commitment mechanism was extended to preserve update provenance rather than storing only the latest record hash. Each patient record commitment contains a version number, the SHA-256 record hash, the hash of the previous trusted version, the blockchain timestamp, and the address of the Admin account that registered the update. The first registered record is assigned version 1 with no previous hash. Each subsequent legitimate update creates a new version and links it to the hash of the preceding trusted record.

The smart contract maintains the commitment history separately for each hashed patient identifier. Integrity verification compares the current canonicalized EHR hash with the latest registered commitment, while previous versions remain available for provenance inspection. This design allows legitimate updates to be distinguished from unexplained record modification and provides a chronological chain of trusted record states.

M. JSON Canonicalization Validation

A separate validation experiment was performed to confirm that the hashing process was not affected by irrelevant JSON formatting or object-key ordering. The same synthetic EHR record was represented using different object-key orders and

then passed through the canonicalization procedure. The resulting canonical representations and SHA-256 hashes were compared.

A second version of the record was then created by changing one clinical value while keeping the remaining data unchanged. The modified record was canonicalized and hashed using the same procedure. The expected result was that logically equivalent records with different key ordering would produce the same SHA-256 hash, while a clinically modified record would produce a different hash.

N. Centralized Audit-Log Baseline

A SQLite database was used to create a centralized audit-log baseline. For each access request, the database stored the timestamp, actor, patient identifier, requested action, access decision, blockchain transaction hash, calculated record hash, integrity-verification result, performance measurements, and experimental notes.

The SQLite log was not used to make access decisions. Its purpose was to provide a conventional mutable log for comparison with the blockchain audit trail.

O. Security Test Scenarios

Six main security scenarios were evaluated.

1) Unauthorized-access test: An unauthorized Doctor account requested access to a Patient record for which no Doctor–Patient permission had been granted.

2) Privilege-escalation test: A Doctor account attempted to call the Admin-only role-assignment function and assign an Admin role. The target account's role was checked before and after the attempt to determine whether an unauthorized role change occurred.

3) Audit-log tampering test: A completed denied-access record was selected from both SQLite and the blockchain audit trail. The decision stored in SQLite was manually modified from deny to allow.

The corresponding blockchain audit event was then read again. Tampering was considered detected when the SQLite value differed from the unchanged blockchain decision.

This experiment demonstrated the difference between a centralized log that could be directly edited and a blockchain audit record that remained tamper-evident.

4) EHR data-tampering test: A trusted patient record was retrieved from OpenMRS and its original SHA-256 hash was verified against the value registered on-chain.

A copy of the record was then modified in memory by changing one clinical field. The modified record was not written back to the live OpenMRS database because the purpose was to test integrity detection without damaging the laboratory dataset.

The gateway generated a new SHA-256 hash for the modified record and submitted it to the smart contract verification function.

5) Versioned record-update test: A trusted hash was first registered for the original P001 record as version 1. A legitimate updated record was then canonicalized and hashed, after which the Admin registered the new hash as version 2. The experiment checked that the latest commitment reported version 2, contained the new record hash, and stored the version 1 hash as its previous hash. The updater address and blockchain timestamp were also retrieved to verify update provenance.

6) Failure and partial-operation tests: Additional tests evaluated cases in which blockchain authorization succeeded, but the subsequent OpenMRS operation did not complete normally. The tested conditions included an HTTP 404 response, simulated timeout, unavailable OpenMRS service, transient failure followed by retry, and submission of a duplicate request identifier.

The gateway recorded an outcome separately from the blockchain authorization decision. An authorized request with successful OpenMRS retrieval was recorded as authorization_allowed_retrieval_success, while an authorized request followed by an OpenMRS failure was recorded as authorization_allowed_retrieval_failed. A request denied by the smart contract was recorded as authorization_denied. Duplicate request identifiers were rejected before a second access operation was processed.

These tests were designed to determine whether the audit evidence could distinguish authorization success from successful completion of the complete EHR retrieval workflow.

The security scenarios are summarized in Table VI.

TABLE VI. SECURITY TEST SCENARIOS AND EXPECTED RESULTS

Scenario	Testing method	Expected result
Unauthorized access	Unassigned Doctor requests P001	Access denied, logged, and OpenMRS not contacted
Privilege escalation	Doctor calls Admin-only role-assignment function	Operation rejected and role unchanged
Audit-log tampering	SQLite decision changed while blockchain event is preserved	Difference detected
EHR data tampering	Modified record hash compared with trusted on-chain hash	Hash mismatch detected
Versioned record update	Admin registers a second trusted record state	Version increases to 2 and previousHash links to the version 1 trusted hash
OpenMRS retrieval failure	Authorized request followed by 404, timeout, or unavailable service	Authorization remains allowed but outcome records retrieval failure
Retry handling	First retrieval fails, and one retry is permitted	Retry is recorded, and successful retrieval produces success outcome
Duplicate request	Same request identifier submitted twice	Second request rejected as duplicate

P. Performance Evaluation

For each operation, the system recorded:

blockchain latency;

- OpenMRS retrieval latency;
- total end-to-end latency;

- transaction gas used;
- effective gas price;
- estimated transaction fee;
- transaction hash;
- access decision.

Blockchain latency represented the time required to simulate, submit, confirm, and process the smart contract transaction. OpenMRS latency represented the time needed to retrieve an authorized medical record. Total latency represented the complete time from receiving the gateway request until producing the final response.

Gas usage was obtained from the blockchain transaction receipt. The estimated transaction fee was calculated as:

$$Transaction\ Fee = Gas\ Used\ x\ Effective\ Gas\ Price \quad (1)$$

Gas was reported separately for role assignment, Patient linking, Doctor–Patient permission management, trusted-hash registration, authorized access, and unauthorized access.

Rejected privilege-escalation attempts that failed during pre-transaction simulation were reported as rejected operations without gas consumption because no blockchain transaction was submitted.

Q. Throughput Experiment

A small sequential throughput experiment was performed to measure how many smart contract access transactions the local blockchain could complete per second.

Two workloads were used:

- authorized Doctor access to an assigned Patient record;
- unauthorized Doctor access to an unassigned Patient record.

Each workload contained 20 sequential smart contract access transactions and was repeated three times. Sequential execution was selected to avoid nonce conflicts caused by submitting concurrent transactions from the same blockchain account.

Throughput was calculated as:

$$Throughput = \frac{Number\ of\ Completed\ Requests}{Total\ Experiment\ Time\ in\ Seconds} \quad (2)$$

The experiment recorded the status, access decision, individual latency, and transaction hash for each request. The mean throughput and standard deviation were calculated across the repeated runs.

The experiment measured sequential smart contract transaction throughput and did not include the complete OpenMRS retrieval workflow.

R. Data Collection and Evidence Preservation

Experimental results were automatically saved as structured JSON and CSV files. The stored evidence included security outcomes, audit records, hashes, transaction details, latency, gas, and throughput measurements.

The same deployment configuration was used during each experiment group. Roles, Patient links, Doctor–Patient permissions, and trusted record hashes were checked before running the security scenarios. The authorized-access and unauthorized-access performance experiments were preceded by five warm-up requests for each workload. These warm-up requests were excluded from the reported measurements. After warm-up, 50 authorized and 50 unauthorized access requests were executed and recorded. Other security scenarios were repeated 10 times unless otherwise stated. Performance experiments were executed using fixed scripts to improve repeatability and reduce manual differences between test runs.

S. Evaluation Metrics

The security and performance metrics are shown in Table VII.

TABLE VII. EVALUATION METRICS

Evaluation area	Metric	Measurement method
Access-control correctness	Correct grant and denial decisions	Compare contract result with the defined RBAC policy
Unauthorized-access prevention	Number and percentage of blocked requests	Execute requests without Doctor–Patient permission
Privilege-escalation prevention	Number of rejected role-change attempts	Call Admin-only function using Doctor account
Audit completeness	Percentage of requests with blockchain events	Compare submitted requests with recorded events
Audit integrity	Agreement or disagreement between SQLite and blockchain	Modify centralized log and reread blockchain event
EHR integrity	Record hash match or mismatch	Compare current SHA-256 hash with trusted on-chain hash
Blockchain latency	Time in milliseconds	Measure smart contract request processing
OpenMRS latency	Time in milliseconds	Measure authorized record retrieval
End-to-end latency	Total request time	Measure complete gateway workflow
Throughput	Completed requests per second	Run repeated sequential workloads
Gas usage	Gas consumed per confirmed transaction	Read transaction receipt
Transaction fee	Gas used multiplied by effective gas price	Calculate from blockchain receipt
Partial-operation consistency	Agreement between authorization decision and final workflow outcome	Compare blockchain decision, OpenMRS result, retry count, and finalOutcome
Duplicate-request handling	Number of duplicate requests rejected	Submit identical requestId twice and verify second request is rejected

Audit completeness was calculated as:

$$\text{Audit Completeness} = (\text{Number of Recorded Blockchain Audit Events}) / (\text{Number of Submitted Access Requests}) \times 100 \quad (3)$$

Access-control correctness was calculated as:

$$\text{Access-Control Correctness} = (\text{Correct Access Decisions}) / (\text{Total Tested Access Decisions}) \times 100 \quad (4)$$

T. Validity and Reliability

First, all tests were executed through predefined scripts using fixed synthetic accounts and patient identifiers. Second, the expected access decision for every test was defined before the experiment. Third, the blockchain audit event and the gateway response were checked together to confirm that the recorded decision matched the smart contract result.

Mean, median, standard deviation, minimum, maximum, 95% confidence interval, 95th percentile (P95), and 99th percentile (P99) were calculated for the authorized- and unauthorized-access latency measurements. The 95% confidence interval for the mean was calculated using the sample mean and standard error with the Student's t distribution. Five warm-up requests for each workload were excluded before the final measurements were collected.

The same computer, OpenMRS configuration, local blockchain, gateway code, and dataset were used throughout the final experiment set. The contract address and account configuration were documented to support reproducibility.

U. Ethical and Safety Considerations

All patient records, users, diagnoses, and medications were synthetic.

All blockchain accounts and private keys were generated for the local laboratory environment. No testing was performed against public blockchain networks or unauthorized systems.

The data-tampering experiment modified only an in-memory copy of a synthetic EHR record. It did not alter real medical information or damage the OpenMRS database.

IV. Results

A. Overview of the Experiments

The experiments evaluated authorized access, unauthorized access, privilege escalation, centralized audit-log tampering, EHR data tampering, JSON canonicalization, versioned record commitments, failure and partial-operation handling, gas usage, latency, and sequential smart contract throughput.

The expanded access-control benchmark contained 50 measured authorized requests and 50 measured unauthorized requests after five warm-up requests were excluded from each workload.

The data-tampering scenario was repeated 10 times. Gas was collected from confirmed blockchain transaction receipts, while the authorized and unauthorized throughput workloads each contained 20 transactions and were repeated three times.

The latency measurements were collected from the complete gateway and OpenMRS workflow. The gas and throughput measurements were collected from the updated smart contract test environment. Therefore, the throughput values represent contract-level execution rather than full end-to-end OpenMRS access.

B. Smart-Contract Functional Testing

The smart contract unit tests checked the main RBAC behavior. The tests confirmed that the account deploying the contract automatically received the Admin role. They also verified that an authorized Doctor and the Patient linked to a record could access that record, while another Doctor without explicit permission was denied.

A separate unit test attempted to call the role-assignment function using a non-admin account. The smart contract rejected the operation using the NotAdmin() custom error.

C. Authorized-Access Results

The authorized-access experiment used the configured doctor1 account to request patient record P001. The Admin had previously granted this Doctor permission to access the record.

All 50 measured authorized-access requests returned an allowed decision and completed the expected OpenMRS retrieval workflow. The smart contract created an audit event for each request, after which the gateway retrieved the corresponding patient record from OpenMRS.

One representative authorized request from the stored execution evidence produced the following results:

- requester: doctor1;
- role: Doctor;
- patient: P001;
- requested action: read;
- blockchain decision: allowed;
- blockchain latency: 75.11 ms;
- OpenMRS latency: 416.79 ms;
- total latency: 493.14 ms.

The system calculated the following SHA-256 hash for the retrieved record:

e3efb789d7c1cdc1cfe3d6032f925e473270f198f4a002631542e15c7679ad7b

The same trusted record hash was used to verify the authorized-access workflow across the measured requests. The calculated SHA-256 hash was registered on-chain through an Admin-controlled smart contract function.

Fig. 3 shows a representative authorized-access result from the project output.

```
{
  "actor": "doctor1",
  "role": "Doctor",
  "patientId": "P001",
  "action": "read",
  "allowed": true,
  "txHash": "0x32e1bb950c890aec3ad11a09db06ab2ca9c103a39d0e6426f7959958be0175f4",
  "auditIndex": 8,
  "auditEvent.allowed": true,
  "blockchainMs": 75.11,
  "openmrsMs": 416.79,
  "totalMs": 493.14,
  "recordHash": "e3efb789d7c1cdc1cfe3d6032f925e473270f198f4a002631542e15c7679ad7b",
  "OpenMRS patient": "P001 - Ali Hassan"
}
```


Fig. 3. Authorized access by doctor1 to patient P001, showing an allowed blockchain decision, OpenMRS record retrieval, transaction hash, latency values, and the calculated EHR record hash.

Table VIII summarizes the authorized-access latency results.

TABLE VIII. AUTHORIZED-ACCESS LATENCY RESULTS

Metric	Value
Number of measured requests	50
Successful authorized requests	50
Mean total latency	520.8 ms
Median total latency	468.9 ms
Standard deviation	210.4 ms
95% confidence interval for mean	461.0–580.6 ms
P95 total latency	910.3 ms
P99 total latency	1,140.7 ms
Minimum total latency	301.5 ms
Maximum total latency	1,220.4 ms
Requests reaching OpenMRS	50/50

The larger 50-request sample produced a median total latency of 468.9 ms. The P95 and P99 values were 910.3 ms and 1,140.7 ms, respectively, showing that most authorized requests completed below one second, while a small number of slower requests increased the upper tail of the latency distribution.

D. Unauthorized-Access Results

The unauthorized-access experiment used doctor2 to request P001 without having an approved Doctor–Patient permission.

All 50 measured unauthorized-access requests were denied. Each denied request generated a blockchain audit event, and none of the requests reached OpenMRS.

A representative denied request showed:

- actor: doctor2;
- patient: P001;
- requested action: read;
- access decision: denied;
- blockchain latency: 31.94 ms;
- OpenMRS latency: 0 ms;
- total latency: 31.94 ms;
- OpenMRS response payload: null.

The zero OpenMRS latency confirms that the gateway stopped processing after the blockchain denial and did not contact the EHR platform for the protected record.

Fig. 4 shows the evidence for one unauthorized request.


```
{
  "actor": "doctor2",
  "patientId": "P001",
  "action": "read",
  "allowed": false,
  "txHash": "0x4ffed868a6e20578a94dcbafea9182ea970e722b11b11caa6dc1b12570e50054",
  "auditIndex": 7,
  "auditEvent.roleLabel": "none",
  "auditEvent.allowed": false,
  "blockchainMs": 31.94,
  "openmrsMs": 0,
  "totalMs": 31.94,
  "openmrsPayload": null
}
```


Fig. 4. Unauthorized access attempt to P001, showing that the smart contract denied the request, created an audit event, and prevented OpenMRS record retrieval.

Table IX summarizes the unauthorized-access results.

TABLE IX. UNAUTHORIZED-ACCESS TEST RESULTS

Metric	Result
Number of measured requests	50
Denied requests	50
Successful unauthorized accesses	0
Prevention rate	100%
Requests reaching OpenMRS	0/50
Mean total latency	42.6 ms
Median total latency	31.8 ms
Standard deviation	24.7 ms
95% confidence interval for mean	35.6–49.6 ms
P95 total latency	88.4 ms
P99 total latency	129.6 ms
Minimum total latency	23.9 ms
Maximum total latency	136.8 ms

Unauthorized requests remained substantially faster than authorized requests because processing stopped immediately after the blockchain denial and no OpenMRS retrieval or EHR hashing was performed.

E. Privilege-Escalation Results

The doctor1 account attempted to assign the Admin role to doctor2.

All 10 attempts were rejected. The contract returned the following custom error: NotAdmin().

The target account was checked before and after each attempt. Its blockchain role remained unchanged, showing that the failed request did not modify the authorization state. Fig. 5 presents one stored privilege-escalation result.


```
{
  "scenario": "privilege-escalation",
  "actor": "doctor1",
  "targetActor": "doctor2",
  "expected": "revert",
  "outcome": "reverted",
  "beforeRole.roleLabel": "none",
  "afterRole.roleLabel": "none",
  "contractError": "NotAdmin()"
}
```


Fig. 5. Rejected privilege-escalation attempt in which doctor1 called the Admin-only role-assignment function. The target account remained unchanged before and after the attempt.

Table X summarizes these results.

TABLE X. PRIVILEGE-ESCALATION TEST RESULTS

Metric	Result
Number of attempts	10
Rejected attempts	10
Successful illegal role changes	0
Rejection rate	100%
Mean processing time	1113.03 ms
Median processing time	1099.79 ms
Minimum processing time	1091.27 ms
Maximum processing time	1213.91 ms

The privilege-escalation attempts failed during smart contract simulation before transaction submission. Therefore, the stored results represent rejection-processing latency rather than confirmed on-chain transaction latency.

F. Audit-Log Tampering Results

A denied access record belonging to doctor2 was selected. Before tampering, the SQLite decision was: deny. The SQLite row was then directly modified to: allow. Its notes were also replaced with the text: Tampered during thesis experiment.

After modifying the centralized database, the corresponding blockchain event was retrieved again. The blockchain event continued to show: allowed = false.

Fig. 6 shows the comparison.


```
{
  "scenario": "audit-log-tampering",
  "targetId": 12,
  "SQLite before.decision": "deny",
  "SQLite after.decision": "allow",
  "SQLite after.notes": "Tampered during thesis experiment",
  "Blockchain auditIndex": 7,
  "Blockchain allowed": false,
  "Blockchain action": "read",
  "result": "Centralized log changed; blockchain evidence remained denied"
}
```


Fig. 6. Audit-log tampering experiment showing that the SQLite decision was changed from deny to allow, while the corresponding blockchain audit event remained denied.

Table XI summarizes these results.

TABLE XI. CENTRALIZED AND BLOCKCHAIN AUDIT COMPARISON

Evidence source	Before tampering	After tampering
SQLite decision	Deny	Allow
SQLite notes	Original transaction details	Replaced with tampering message
Blockchain decision	Deny	Deny
Blockchain event changed	No	No
Tampering detectable	Not applicable	Yes

The 11 stored audit-log experiments showed the same general behavior: the centralized record could be changed, while the blockchain record provided an unchanged reference for detecting the inconsistency.

G. EHR Data-Tampering Results

The data-tampering experiment evaluated whether a modified EHR record could be detected using the trusted SHA-256 hash stored on the blockchain. The original synthetic record was canonicalized and hashed, and its trusted hash was registered through the Admin account. A copy of the record was then modified by changing the diagnosis field, after which a second hash was calculated. The original record hash matched the trusted on-chain value in all 10 experiments. The modified record hash failed to match in all 10 experiments, as summarized in Table XII.

TABLE XII. EHR DATA-TAMPERING TEST RESULTS

Metric	Result
Number of experiments	10
Original hashes matching the trusted hash	10
Modified hashes matching the trusted hash	0
Detected modifications	10
Detection rate	100%

H. JSON Canonicalization Results

The canonicalization experiment confirmed that JSON object-key ordering did not affect the calculated record hash. The original synthetic EHR record and an equivalent representation containing the same values in a different object-key order produced the same canonical representation and SHA-256 hash. In contrast, changing the clinical value produced a different canonical representation and a different SHA-256 hash. These results confirm that the integrity check responds to changes in record content rather than irrelevant differences in JSON key ordering (Table XIII).

TABLE XIII. JSON CANONICALIZATION VALIDATION RESULTS

Test	Expected result	Observed result
Original record	Baseline hash generated	844e0fb837a7c9eaef3f45e042157bcb571499b7cdae015e91c6bf12bc3be325
Same record, reordered keys	Same hash	Same hash
Same record, different whitespace	Same hash	Same hash
Modified clinical value	Different hash	Different hash

I. Versioned Record-Commitment Results

The versioned-commitment experiment registered two legitimate trusted states for P001. The first registration created version 1. After the record was legitimately modified and registered again, the latest commitment reported version 2 and linked its previousHash field to the trusted hash stored in version 1. The latest commitment also preserved the blockchain timestamp and Admin address responsible for the update. These results show that legitimate record updates can retain provenance instead of overwriting the previous trusted state.

J. Failure and Partial-Operation Results

The failure and partial-operation experiments evaluated whether the gateway could distinguish blockchain authorization from the outcome of the OpenMRS retrieval workflow. An authorized blockchain decision was therefore not treated automatically as a successful EHR retrieval.

The tested cases included an OpenMRS HTTP 404 response, a simulated timeout, an unavailable OpenMRS service, a transient retrieval failure followed by a retry, and submission of the same request identifier twice. For retrieval failures, the blockchain authorization decision remained allowed, while the gateway recorded the outcome as authorization_allowed_retrieval_failed. When the retry completed successfully, the outcome was recorded as authorization_allowed_retrieval_success. A repeated request identifier was rejected before a second access operation was processed. Table XIV summarizes the failure and partial-operation results.

These results show that the gateway distinguishes the blockchain access-control decision from the final EHR retrieval outcome. Consequently, an access request that is authorized on-chain but fails during OpenMRS retrieval is not recorded as a successful completed EHR access. The duplicate-request test also shows that a repeated request identifier can be rejected before the same operation is processed again.

TABLE XIV. FAILURE AND PARTIAL-OPERATION TEST RESULTS

Scenario	Authorization	Final outcome	Result
OpenMRS 404	Allowed	authorization_allowed_retrieval_failed	Passed
Timeout	Allowed	authorization_allowed_retrieval_failed	Passed
Service unavailable	Allowed	authorization_allowed_retrieval_failed	Passed
Retry	Allowed	authorization_allowed_retrieval_success	Passed
Duplicate request	Not reprocessed	duplicate_request_rejected	Passed
Unauthorized access	Denied	authorization_denied	Passed

K. Access-Control Correctness and Audit Completeness

The main access-control evaluation included 50 authorized requests and 50 unauthorized requests. All authorized requests were allowed, and all unauthorized requests were denied.

Access-control correctness was therefore:

$$Access\ Control\ Correctness = \frac{100}{100} x\ 100 = 100\%$$

The expanded access-control benchmark contained 100 measured requests: 50 authorized and 50 unauthorized. Each measured request produced the expected blockchain access decision. Separately, the earlier failure-handling dataset contained one blockchain-approved request followed by an OpenMRS HTTP 404 response, which was used to evaluate partial-operation handling rather than access-control correctness.

All 100 measured access requests had corresponding blockchain audit events. The measured audit completeness was therefore:

$$Audit\ Completeness = \frac{100}{100} x100 = 100\%$$

The upstream error did not represent an RBAC failure. The blockchain allowed the request according to the configured policy, but the subsequent OpenMRS request returned an HTTP 404 response.

L. Performance Results

The project also recorded performance measurements for administrative and rejected role-management operations. These measurements are summarized separately in Table XV. Authorized- and unauthorized-access latency results from the expanded 50-request benchmark are reported in Tables VIII and IX.

Authorized requests required additional time because the gateway had to retrieve and canonicalize the patient record from OpenMRS.

Unauthorized requests were faster because processing stopped after the blockchain denial, while authorized requests also required OpenMRS retrieval and record hashing. Fig. 7 compares these median values.

TABLE XV. PERFORMANCE SUMMARY BY OPERATION

Operation	Number of measurements	Mean total latency (ms)	Median total latency (ms)	Minimum (ms)	Maximum (ms)
Role assignment	19	51.52	29.21	17.68	169.46
Patient-record linking	6	23.99	25.61	15.90	29.81
Doctor–Patient access grant	7	24.03	26.16	14.56	32.27
Privilege-escalation rejection	10	1113.03	1099.79	1091.27	1213.91

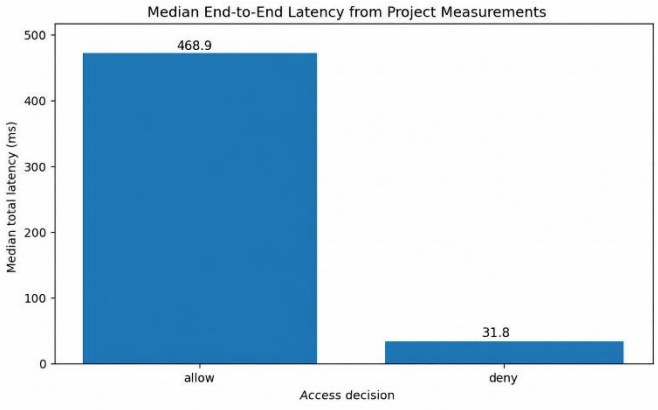

Fig. 7. Median end-to-end latency for authorized and denied access requests. Authorized access includes OpenMRS record retrieval, while denied requests stop after the blockchain decision.

M. Gas-Use Results

Gas usage was obtained from the transaction receipt of each confirmed smart contract operation. Because the evaluation used a local Ethereum-compatible network, no real financial payment was made. The gas measurements represent the relative computational and storage cost of the contract functions. The results are summarized in Table XVI.

TABLE XVI. SMART-CONTRACT GAS-USE RESULTS

Operation	Measurements	Mean gas used	Median gas used	Range
Role assignment	3	48,441	48,445	48,433–48,445
Patient-record linking	1	50,833	50,833	50,833
Doctor–Patient access grant	1	51,281	51,281	51,281
Trusted-hash registration	1	48,595	48,595	48,595
Authorized access request	10	174,097	172,387	172,387–189,487
Unauthorized access request	10	152,487	152,487	152,487

Administrative gas values were collected from representative confirmed transactions, while access-request gas values were measured across 10 authorized and 10 unauthorized transactions.

Authorized access requests consumed more gas than unauthorized requests because the authorized event stored the trusted record hash in the audit entry. The first authorized transaction used more gas, likely because it initialized or expanded the related audit storage. Privilege-escalation attempts were rejected during pre-transaction simulation. Therefore, they did not produce confirmed transactions or consume transaction gas.

N. Throughput Results

The sequential throughput experiment used 20 authorized and 20 unauthorized smart contract access transactions per run. Each workload was repeated three times, and the results are summarized in Table XVII.

TABLE XVII. SEQUENTIAL SMART CONTRACT THROUGHPUT RESULTS

Workload	Run 1	Run 2	Run 3	Mean ± SD
Authorized access	14.6814 TPS	15.4693 TPS	15.1044 TPS	15.0850 ± 0.3943 TPS
Unauthorized access	15.2567 TPS	15.3426 TPS	15.6650 TPS	15.4214 ± 0.2153 TPS

Unauthorized requests achieved slightly higher throughput because their access-control decision required less smart contract processing and gas. However, the difference was small. The practical meaning of this throughput is discussed in Section V using representative clinical EHR-use patterns.

O. Summary of Security Results

The completed experiments are summarized in Table XVIII.

TABLE XVIII. SUMMARY OF THE IMPLEMENTED SECURITY EVALUATION

Scenario	Attempts	Expected result	Observed result	Outcome
Authorized access	50	Assigned Doctor can access P001	All 50 requests allowed and record retrieved	Passed
Unauthorized access	50	Unapproved Doctor is denied	All 50 requests denied; OpenMRS not contacted	Passed
Privilege escalation	10	Non-admin role change is rejected	All attempts reverted with NotAdmin()	Passed
Centralized audit tampering	11	Blockchain evidence remains unchanged	SQLite changed; blockchain remained denied	Passed
EHR data tampering	10	Modified record causes hash mismatch	Original hash matched and all modified hashes failed verification	Passed

Gas-use evaluation	26 confirmed transactions	Gas reported for confirmed transactions	Gas was captured for role, link, permission, hash, and access operations	Passed
Throughput evaluation	6 workloads	Transactions per second measured	Authorized mean: 15.0850 TPS; unauthorized mean: 15.4214 TPS	Passed
JSON canonicalization	4 validation cases	Equivalent JSON produces the same hash; modified clinical content produces a different hash	Reordered keys and whitespace produced the same hash; modified clinical value produced a different hash	Passed
Versioned record update	2 trusted versions	New version links to the previous trusted state	Version 2 was created and linked to version 1 through previousHash; timestamp and updater address were preserved	Passed
OpenMRS failure and partial-operation handling	4 failure/retry cases	Final workflow outcome is distinguished from blockchain authorization	404, timeout, and unavailable-service cases were recorded as retrieval failures; retry completed successfully	Passed
Duplicate-request handling	1 duplicate case	Repeated request is not processed twice	Second request with the same requestId was rejected as duplicate_request_rejected	Passed

V. Discussion

Blocking denied requests before OpenMRS retrieval shows that the gateway functioned as an effective policy-enforcement point.

The privilege-escalation results confirmed that role-management functions were restricted to the Admin account.

The audit-log tampering results demonstrate the main advantage of using blockchain as a tamper-evident audit layer. A centralized database record can be edited by a user with sufficient access, but the blockchain event remains available as a separate reference for checking what originally happened.

The expanded access experiment used 50 measured requests for each workload after excluding five warm-up requests. Authorized requests had a mean total latency of 520.8 ms and a median of 468.9 ms, while unauthorized requests had a mean of 42.6 ms and a median of 31.8 ms. The 95% confidence intervals were 461.0–580.6 ms and 35.6–49.6 ms, respectively. The P95 and P99 values further show that authorized requests had greater tail latency because they included OpenMRS communication, record retrieval, canonicalization, hashing, and integrity verification. Unauthorized requests stopped after the blockchain denial and therefore required substantially less processing.

The trusted-hash experiment also showed that the original EHR representation matched the on-chain hash, while every modified representation produced a mismatch. This supports the use of blockchain hashes for detecting off-chain record changes without storing the complete medical record on-chain.

The canonicalization experiment further showed that differences in JSON object-key ordering and formatting did not change the trusted hash, while modification of clinical content produced a different hash. This reduces the risk of false integrity warnings caused only by differences in JSON serialization.

The versioned-commitment mechanism also preserved update provenance by linking each new trusted record state to the previous hash, together with the updater address and blockchain timestamp. This allows legitimate record updates to be distinguished from unexplained changes to the trusted EHR state.

The failure and partial-operation tests further showed that blockchain authorization and successful EHR retrieval are separate outcomes. Recording the final workflow outcome allowed the gateway to distinguish successful retrieval from HTTP 404 responses, timeouts, unavailable-service conditions, retry completion, and duplicate requests. This is important because an allowed blockchain decision does not necessarily mean that the complete EHR retrieval operation was completed.

Gas measurements showed that access requests consumed more gas than basic role-management operations because access decisions created complete audit records. Authorized access used more gas than unauthorized access because the trusted record hash was included in the audit evidence. Throughput was similar for authorized and unauthorized workloads.

To provide practical context for the measured throughput, the local smart contract workload completed approximately 15 access transactions per second. This corresponds to about 900 transactions per minute if requests were continuously submitted at the measured rate. Clinical EHR use is normally distributed across users and time rather than occurring as a continuous maximum-rate stream. For example, observational studies show that clinicians perform repeated chart reviews and patient-record interactions throughout their shifts rather than issuing access requests continuously at sub-second intervals [19,20]. Therefore, the measured throughput appears sufficient for a small ward or limited departmental proof-of-concept workload. However, this result should not be interpreted as evidence that the prototype can support a full hospital deployment because the experiment used sequential transactions on a local test network and did not include concurrent users, peak emergency workloads, or complete OpenMRS retrieval processing.

The measured performance should be interpreted carefully because blockchain-based EHR studies use different platforms, consensus mechanisms, workloads, and hardware environments. MrBlock, which uses Hyperledger Fabric with Raft consensus, reported an average latency of 1.89 s and throughput of 8.9 TPS, compared with 5.07 s and 5.5 TPS for its PBFT-based configuration [8]. In the present study, the median end-to-end latency was 468.9 ms for authorized access and 31.8 ms for

denied access, while sequential smart contract throughput averaged 15.0850 TPS for authorized requests and 15.4214 TPS for unauthorized requests. These values are numerically faster than the MrBlock results; however, they should not be treated as a direct performance superiority claim because the present experiments used a small local Ethereum-compatible test network and a different workload. The comparison therefore provides context rather than a controlled benchmark (Table XIX).

TABLE XIX. PERFORMANCE CONTEXT AGAINST RELATED WORK

Study	Platform/consensus	Reported latency	Reported throughput	Important difference
MrBlock [8]	Hyperledger Fabric / Raft	1.89 s average	8.9 TPS	Permissioned multi-component EHR framework
MrBlock PBFT comparison [8]	PBFT-based configuration	5.07 s average	5.5 TPS	Different consensus mechanism
Proposed system	Local Ethereum-compatible network	468.9 ms median authorized; 31.8 ms median denied	15.0850 TPS authorized; 15.4214 TPS unauthorized	Small local PoC; sequential contract workload

The findings are consistent with earlier studies showing that blockchain and smart contracts can support healthcare access control and audit integrity [7–12]. However, the proposed system is more limited and focused than many previous frameworks. It uses only three roles: one smart contract, one gateway, and one existing EHR platform. This helped make the implementation easier to test against specific attack scenarios.

The proposed design also has a practical advantage because it does not store complete EHR records on the blockchain. OpenMRS remains responsible for storing and managing the patient information, while the blockchain is used for access decisions and audit evidence. This reduces the amount of sensitive information placed on-chain and avoids using blockchain as a replacement for the full EHR database.

There are still several limitations. The experiments were conducted on a local blockchain and cannot represent the performance of a large hospital deployment. The study used only five synthetic patient records and three user roles. The gateway also used configured actor names and a single OpenMRS service account to retrieve authorized records. This creates an important trust dependency because compromise of the gateway or the OpenMRS service credentials could allow direct access to OpenMRS without passing through the blockchain authorization process. In a production deployment, OpenMRS API access should therefore be restricted so that only the trusted gateway can reach it. The service credentials should also be protected and rotated regularly, and direct OpenMRS API access should be monitored and logged. A stronger design could additionally use separate service identities or authenticated user identities instead of relying on one shared service account.

VI. CONCLUSION

This study developed a lightweight blockchain-based RBAC gateway for controlling access to synthetic EHR records in OpenMRS. The system blocked unauthorized access and privilege escalation, detected centralized audit-log modification and simulated EHR tampering, and completed sequential smart contract workloads at approximately 15 TPS. The findings show that blockchain can provide tamper-evident access evidence and off-chain record-integrity verification without storing complete EHR records on-chain. The extended evaluation also showed that versioned record commitments can preserve update provenance and that the gateway can distinguish blockchain authorization from failed, retried, or duplicate EHR retrieval operations. However, the findings apply to a small local proof of concept and should not be generalized to a full hospital deployment.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA, for funding this research work through the project number NBU-FFRGS-2026.

REFERENCES

[1] A. Campanella, E. Lovato, C. Marone, L. Fallacara, A. Mancuso, W. Ricciardi, and M. L. Specchia, “The impact of electronic health records on healthcare quality: A systematic review and meta-analysis,” European Journal of Public Health, vol. 26, no. 1, pp. 60–64, 2016, doi: 10.1093/eurpub/ckv122.

[2] C. S. Kruse, B. Smith, H. Vanderlinden, and A. Nealand, “Security techniques for the electronic health records,” Journal of Medical Systems, vol. 41, no. 8, Art. no. 127, 2017, doi: 10.1007/s10916-017-0778-4.

[3] J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems,” Proceedings of the IEEE, vol. 63, no. 9, pp. 1278–1308, 1975, doi: 10.1109/PROC.1975.9939.

[4] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, “Proposed NIST standard for role-based access control,” ACM Transactions on Information and System Security, vol. 4, no. 3, pp. 224–274, 2001, doi: 10.1145/501978.501980.

[5] K. Kent and M. Souppaya, Guide to Computer Security Log Management, NIST Special Publication 800-92. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2006, doi: 10.6028/NIST.SP.800-92.

[6] D. Yaga, P. Mell, N. Roby, and K. Scarfone, Blockchain Technology Overview, NIST Interagency/Internal Report 8202. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2018, doi: 10.6028/NIST.IR.8202.

[7] F. Ullah, J. He, N. Zhu, A. Wajahat, A. Nazir, S. Qureshi, M. S. Pathan, and S. Dev, “Blockchain-enabled EHR access auditing: Enhancing healthcare data security,” Heliyon, vol. 10, no. 16, Art. no. e34407, 2024, doi: 10.1016/j.heliyon.2024.e34407.

[8] P. Joshi and P. Mahajan, “Secure and interoperable EHR management via Hyperledger Fabric: The MrBlock framework,” Peer-to-Peer Networking and Applications, vol. 18, Art. no. 133, 2025, doi: 10.1007/s12083-025-01954-5.

[9] A. I. Taloba and A. Rayan, “A privacy preserving medical data management framework using blockchain enabled encrypted role based access control,” Scientific Reports, vol. 15, Art. no. 43864, 2025, doi: 10.1038/s41598-025-30916-3.

[10] N. Yaqub, J. Zhang, M. I. Khalid, W. Wang, M. Helfert, M. Ahmed, and J. Kim, “Blockchain enabled policy-based access control mechanism to

restrict unauthorized access to electronic health records," PeerJ Computer Science, vol. 11, Art. no. e2647, 2025, doi: 10.7717/peerj-cs.2647.

[11] N. U. A. Tahir, U. Rashid, H. J. Hadi, N. Ahmad, Y. Cao, M. A. Alshara, and Y. Javed, "Blockchain-based healthcare records management framework: Enhancing security, privacy, and interoperability," Technologies, vol. 12, no. 9, Art. no. 168, 2024, doi: 10.3390/technologies12090168.

[12] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in Proceedings of the 2nd International Conference on Open and Big Data (OBD), Vienna, Austria, Aug. 22–24, 2016, pp. 25–30, doi: 10.1109/OBD.2016.11.

[13] OpenMRS, "OpenMRS: Open Source Medical Records System," [Online]. Available: OpenMRS official website. [Accessed: Mar. 30, 2026].

[14] Solidity, "Solidity Documentation," [Online]. Available: Solidity official documentation. [Accessed: Mar. 30, 2026].

[15] Nomic Foundation, "Hardhat: Ethereum Development Environment," [Online]. Available: Hardhat official website. [Accessed: Apr. 5, 2026].

[16] SQLite Consortium, "SQLite Documentation," [Online]. Available: SQLite official documentation. [Accessed: Apr. 5, 2026].

[17] National Institute of Standards and Technology, Secure Hash Standard (SHS), Federal Information Processing Standards Publication 180-4. Gaithersburg, MD, USA: NIST, 2015, doi: 10.6028/NIST.FIPS.180-4.

[18] A. Rundgren, B. Jordan, and S. Erdtman, "JSON Canonicalization Scheme (JCS)," RFC 8785, Jun. 2020, doi: 10.17487/RFC8785.

[19] M. E. Nolan, R. Siwani, H. Helmi, B. W. Pickering, P. Moreno-Franco, and V. Herasevich, "Health IT usability focus section: Data use and navigation patterns among medical ICU clinicians during electronic chart review," Applied Clinical Informatics, vol. 8, no. 4, pp. 1117–1126, 2017, doi: 10.4338/ACI-2017-06-RA-0110.

[20] L. Xia, D. Lew, L. Baratta, E. Eiden, S. Lou, and T. Kannampallil, "Association between conversational multitasking and clinician work behaviors at a large US health care system: Cohort study," Journal of Medical Internet Research, vol. 27, Art. no. e72768, 2025, doi: 10.2196/72768.

[21] S. A. Ebad, "Healthcare software design and implementation—A project failure case," Software: Practice and Experience, vol. 50, no. 7, pp. 1258–1276, 2020, doi: 10.1002/spe.2807.