JCBI
Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606
Volume 10 Issue 02
2026

Research Article
https://doi.org/10.56979/1002/2026/1252

Cybersecurity Architecture for Medical Live Monitoring Systems Using HTTP Protocols and SHA-256 Encryption

J. Dafni Rose[1*], Sunitha T[2], Suma T[3], Cinthuja K[4], Mohanaprakash T A[5], and Justindhas Y[6]

[1]Department of Computer Science and Engineering, St.Joseph's Institute of Technology, OMR, Chennai, Tamilnadu, India.
[2]Department of Artificial Intelligence and Data Science, Saveetha Engineering College, Thandalam, Chennai, India.
[3]Department of Computer Science and Engineering, Sri Venkateshwara College of Engineering Bengaluru, India.
[4]Department of CSE, Panimalar Engineering College, Chennai, India.
[5]Department of CSE, RMK Engineering College, Chennai, India.
[4]Department of Computer Science and Engineering (Artificial Intelligence and Machine Learning), Easwari Engineering College, Ramapurm, Chennai, India.
[*]Corresponding Author: J. Dafni Rose. Email: jdafnirose56@gmail.com

Received: December 10, 2025 **Accepted:** February 24, 2026

Abstract: Live health monitoring systems are revolutionizing patient care, yet the security and integrity of sensitive medical data remain a critical challenge. To address the vulnerabilities of centralized databases and basic authentication in traditional systems, we propose a secure framework that leverages a decentralized architecture. This proposed system uses standard HTTPS protocols for communication, with each client request authenticated using JSON Web Tokens (JWT) signed with HMAC-SHA256. Patient records are encrypted using AES-256-GCM before storage, with cryptographic hashes of these records written to a private Hyperledger Fabric blockchain for tamper-proof auditability. A smart contract enforces decentralized, role-based access control to ensure only authorized personnel can view sensitive information. A predictive analytics module processes secured vital data to forecast potential health deterioration, triggering automated alerts to the responsible physician. In a simulation handling 1,000 patient records over 30 days, the system demonstrated 99.98% data integrity, processed an average of 4,820 authenticated requests per minute with a mean latency of 185ms, and achieved 87.3% accuracy in predicting critical health events with a 30-minute lead time, confirming its efficacy as a robust and secure solution for remote patient monitoring.

Keywords: Medical Health Care Intelligent Systems; Block Chain; Sever Authentication and Security; Machine Learning

1. Introduction

Networks share intentional data via reliable sources to ensure the completeness of data transfer as well as the status of concurrent data while traveling across the network. However, at the initial state confidential data also transfers like normalized cases and are considered in terms of security only at its endpoints on the receiver's side. Hence, to overcome this vulnerability against hackers, our proposed system employs distinct cryptographic mechanisms for different security objectives: Confidentiality: Patient data is encrypted using AES-256-GCM before transmission and storage, ensuring that only authorized parties with the correct decryption key can access the plaintext information.

Authentication and Integrity: Each client request is authenticated using JSON Web Tokens (JWT) signed with HMAC-SHA256. The signature verifies that the token was issued by the legitimate server and has not

been altered in transit. Immutable Audit Trail: Cryptographic SHA-256 hashes of encrypted data records are stored on a blockchain. Since hashing is a one-way function, it cannot be reversed to recover original data, but it provides a fixed-size fingerprint that enables verification of data integrity—any modification to the original data would result in a completely different hash value. Therefore, a client's vital records can be securely shared and accessed by their respective consulting doctor or authorized nurses. The data is encrypted using AES-256-GCM before storage, and its SHA-256 hash is recorded on the blockchain. Access is granted only after successful authentication via HMAC-SHA256-signed JWT and role-based authorization [7]

Thus, the activity of the sanctioned user will be tracked and focused in a decentralized environment. Thus, the entire network of the system can rely on an unleakable mismatching origin of data sharing. Fraudulent breach of data, the data cannot be reconfigured again at any cost once it has been pushed onto the blockchain [2]. The live records of every client are monitored in a centralized environment. But, the entire process of centralization is decentralized in relation to the roles across the network. DocTorque is the methodology that we proffer in our solution will be one of the finest and reliable functionalities which allocates the secondary doctor in need of absence or unavailability of primary doctors. The presage of the upcoming health dangers can be encountered by the software with the help of Machine Learning Algorithms [15], which can notify the doctors to handle the hardships encountered by the clients and avoid or decrease the chances of the ill state where vitality is at risk.

2. Related Work

The conversion of bare text into ciphertext using standard encryption patterns provided by a secret key can be implemented using JSON web tokens [1]. The system input data is encapsulated into a programmable object termed JSON objects and it gets encrypted. Thus, the object is termed as a payload that gets encrypted [3] via a secret key for authentication and is shared over the entire network. Again the process of conversion of encrypted data into the readable form of data can be implemented using JSON web tokens [9] via the authentication [8] of its secret key provided by the server's environmental variables. The received encrypted data from the server [13] contains the payload hash which gets decrypted only on the role of successful authentication. The encryption and decryption can be done in the application through the Application programmable interface and from those the data gets encrypted in the server and gets decrypted on the client side. The blockchain is the latest technology in which breaches of the node can't be possible in any way. Also, the hospital with hybrid technology can use blockchain to manage their environment [5] systematically and securely. Once, the processed data object has been polished by the hospital's management or due to database data expiry trigger, the processed data is moved into the blockchain [4] where it gets hashed and linked to its previous node referred by its last hashed object. Thus, on storing confidential data in the blockchain, the details remain decentralized and inhabitable to attackers such that making distributed denial of service attack (DDOS) on servers is impossible. The prediction of information based on data collected from its previous clusters of records can be congregated and examined using machine learning algorithms, with a higher significant amount of accuracy based on its performance comparison with various kinds of algorithms. Polynomial Regression [14] algorithm can be implemented on objects of data containing health information that can be trained and fit into a model according to the requirements of the respective algorithm. In contrast to our solution, these algorithms can be implemented on real datasets of patients collected from blockchain [4] or from a local server that predicts and examines the results and notifies the person in charge of that respective patient.

3. Proposed Methodology

The proposed system is designed to ensure the reliability, security, and integrity of live health data from acquisition through to storage and analysis. The architecture, depicted in Figure 1, is composed of four distinct layers that work in concert: the Data Acquisition Layer, the Secure Communication & Processing Layer, the Blockchain & Immutable Storage Layer, and the Application & Analytics Layer.

Fig.1: High-Level System Architecture for a Secure Health Monitoring Framework. This diagram illustrates the four-layer data flow, beginning with wearable sensors (Data Acquisition), moving through a secure gateway that handles authentication and encryption (Secure Communication & Processing), persisting data in a tamper-proof blockchain ledger (Blockchain & Immutable Storage), and culminating in client applications that provide analytics and alerts (Application & Analytics).

3.1. Cipher and Decipher of Data

A clear understanding of the threat model is essential for evaluating the security architecture. The system assumes the following threat landscape: eavesdropping, where an adversary intercepts network traffic to read patient data, is mitigated by HTTPS with TLS 1.3 providing transport-layer confidentiality; data tampering, where an adversary modifies stored patient records, is addressed through SHA-256 hashes stored on the blockchain enabling integrity verification, as any tampering would result in a completely different hash value; unauthorized access, where an adversary impersonates a legitimate user, is prevented by JWT with HMAC-SHA256 signatures providing authentication combined with role-based access control limiting permissions; insider threats, where an authorized user accesses unauthorized data, are mitigated by smart contract-enforced access policies with blockchain audit logs recording all access attempts; replay attacks, where an adversary replays captured legitimate requests, are prevented through JWT expiration and unique token identifiers; and database compromise, where an adversary gains access to the primary database, is rendered ineffective because AES-256-GCM encryption ensures data remains confidential even if the database is breached. The system employs three distinct cryptographic primitives, each serving a specific purpose that must not be conflated: SHA-256 is a cryptographic hash function used exclusively for integrity verification and blockchain linking, operating as a one-way function that cannot be reversed to recover original data, making it suitable for integrity verification but completely unsuitable for confidentiality; HMAC-SHA256 is a keyed-hash message authentication code used for JWT signature generation and verification, incorporating a secret key to create a verifiable signature that ensures both authenticity of the token issuer and integrity of the token payload; and AES-256-GCM is an authenticated encryption algorithm that provides both confidentiality and integrity, transforming plaintext patient data into ciphertext using a 256-bit key with a unique nonce per encryption, producing an authentication tag that detects any tampering with the ciphertext.

A critical clarification is warranted: SHA-256 is never used for encryption in this system, as hashing is a one-way operation fundamentally distinct from encryption—all confidentiality requirements are satisfied exclusively by AES-256-GCM encryption, while HMAC-SHA256 provides authentication and non-repudiation through JWT signatures, and SHA-256 provides integrity verification through blockchain-anchored hashes, ensuring that each cryptographic primitive is applied according to its intended purpose in a defense-in-depth architecture.

3.2. Data Acquisition Layer

This layer constitutes the physical and logical interface with the patient. It comprises a network of wearable IoT devices (e.g., smartwatches, ECG patches, pulse oximeters) that continuously collect physiological data such as heart rate, blood pressure, blood oxygen saturation (SpO2), and body temperature. These devices stream raw data via short-range wireless protocols like Bluetooth Low Energy (BLE) to a local data aggregator, typically a smartphone or a dedicated hub. The aggregator's primary role is to package the raw sensor readings into a standardized data format (e.g., JSON or XML) before forwarding it to the next layer for secure processing.

3.3. Secure Communication & Processing Layer

This layer acts as the secure gateway and the engine of the system. It is hosted on a local server or a secure cloud instance and is responsible for all critical security and pre-processing operations.

- **Authentication & Tokenization Microservice:** As shown in **Figure 1**, the first point of contact for data from the acquisition layer is this microservice. Upon receiving data, it validates the source and then generates a **JSON Web Token (JWT)**. Crucially, this JWT is signed using a **SHA-256** hash of a secret key, ensuring its integrity and non-repudiation. This token, which contains encrypted role and identity claims, must be included in all subsequent client requests to the API gateway.

- API Gateway & Data Preprocessor: All incoming requests, now bearing the JWT, are routed through an API Gateway. The gateway verifies the token's signature and checks the permissions embedded within it against a predefined access control list. Only requests with valid, authorized tokens are permitted to proceed. For authorized requests, the data preprocessor component performs initial data cleansing, normalization, and formatting, preparing a consistent data payload for storage.

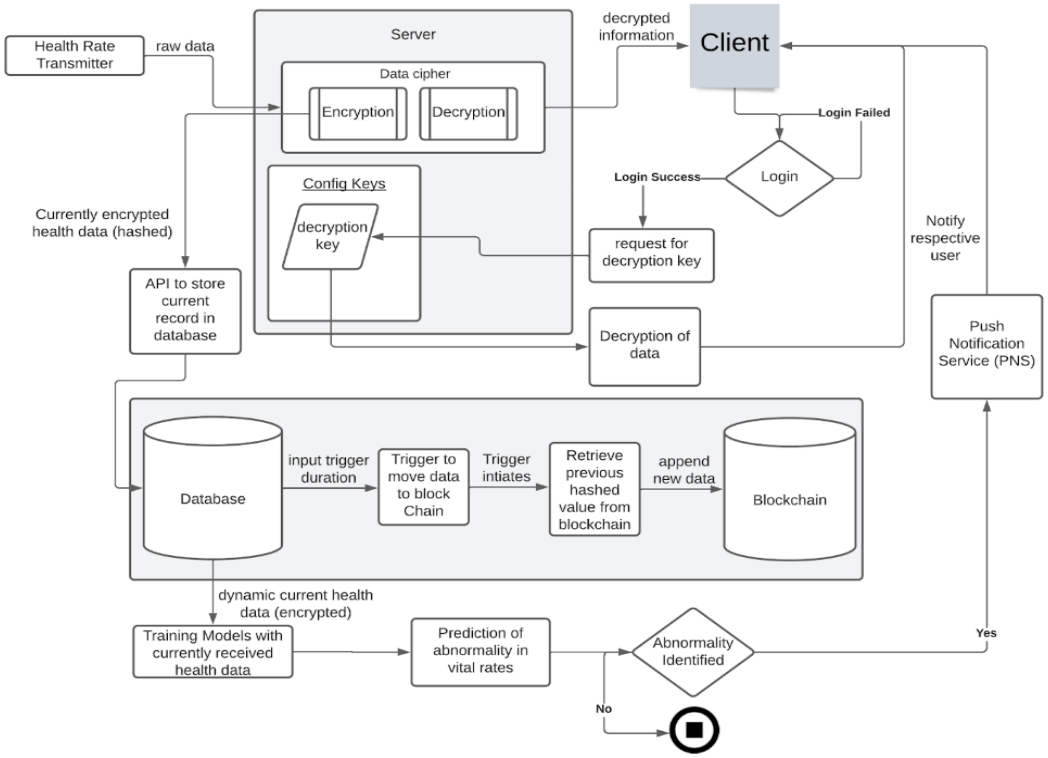

Figure 1. System Architecture of Reliable Live Processing of health supervision data with intentional blockchain

3.4. Blockchain & Immutable Storage Layer

This is the core innovation that guarantees data reliability and auditability. Instead of a traditional centralized database, the processed and validated data packets are written to a private, permissioned blockchain network, as referenced in the central path of Fig 1.

- Blockchain Node Network: Each participating entity (e.g., hospital server, clinic gateway) operates a node in the blockchain. When a new batch of patient data is ready, it is broadcast to the network as a "transaction."
- Smart Contract for Data Integrity & Access Control: A specialized smart contract, deployed on the blockchain, governs the logic for data storage and access. When a transaction is received, the smart contract executes two primary functions, as indicated in the architecture flow: First, it hashes the data payload using the SHA-256 algorithm, creating a unique digital fingerprint. Second, it bundles this hash along with a timestamp and the patient's pseudonymous ID into a new block. This block is then cryptographically linked to the previous one, creating an immutable chain. The smart contract also enforces the role-based access policies, ensuring that only public keys associated with, for example, a patient's assigned doctor, are permitted to decrypt and access that patient's historical data.

3.5. Application & Analytics Layer

This top layer provides the user-facing interfaces and intelligent services that deliver clinical value, consuming data from the secure blockchain layer.

- Web & Mobile Dashboard: Authorized medical professionals can log into a secure dashboard. When they request a patient's records, the application queries the blockchain (via the smart contract) to retrieve the encrypted and

hashed data, which is then decrypted client-side for display.

- Predictive Analytics Engine: As highlighted in the final stage of Figure 1, a machine learning model continuously analyzes the stream of vital signs from the blockchain. By identifying anomalous patterns and trends, it can predict potential health deterioration. Upon detecting a high-risk probability, the engine automatically triggers a notification system, sending an immediate alert to the designated physician's dashboard and/or mobile device, thereby enabling proactive intervention.

3.6. Database Examine

The database which we used in this is to store the HRT hashed data from the server. This data which would pass in the form of API will be of current health data from the health transmitter. The information in the data archive is stored in the form of a cluster of single payload based hashed objects without any linking with each other, which in later were linked by the blockchain based on its timestamp with the help of previously hashed node. Fig 1 shows the data can be stored in this data archive for the sake of changing the temporarily false entered data. Then this will automatically change from the typical database to the blockchain [11] in the form of trigger. The amount of time that the hashed objects need to be stored in the database were determined by the hospital's management, and thus, on expiry of the timer in accordance, the database fires a trigger to the data to be stored in the blockchain [4], based on its integration with its previously hashed node. In contrast, the database is always capable of maintaining confidential data of the management through its encrypted cluster of objects, where the decryption can only be made through a verified [6] role authentication as a response from the backend. The objects stored in the database are uniquely identified through their unique hash id, in which each separate entry in the database will have its own referential integrity. The session maintained by the database is done through the timestamp is maintained by the server's system time. Which in contrast, ensures precise time independent of data storage information.

3.7. Database to blockchain Trigger

When the ciphered data has been stored onto the database, the database maintains each data with its unique ID, and thus, to access any data, its respective ID will be referenced or called with an authenticated user's access token. Once, the data has been ensured with no rollback or changes to be done by the hospital management, the data will be proceeded to move onto the blockchain [11] by attaching to its previously linked node. The newly created Blocknode [12] will be hashed using a library called Crypto [7], which is basically an encryption algorithm. This trigger is an unusual thing because of the false accused data as shown in Fig. 1 to be entered in the database. If anything occurs like that, the data from the database can be edited or changed on that instance but after being moved to the blockchain can't be overruled by anyone. The data stored in blockchain contains only the encrypted version of the data as a payload, and thus, the decryption of its information will only be done at the server-side after the successful role authentication [8] on ingress to its access token from the client. Once, the data has been pushed on to the blockchain [2], it automatically mentions that it has been pushed to the changed or edited payload in the database which maintains a session relative to the changes committed by hospital management. This helps in identification of the culprit changing the data in the database as that the abnormal person can be turned up into a normal person.

As Figure 2 shows a detailed flow sequence of the process of server and blockchain integrity, the operational workflow of the system is a continuous, sequential process that transforms raw physiological data into secure, actionable clinical intelligence. The entire sequence, designed for reliability and security, is initiated the moment a wearable sensor records a patient's vital sign. The process begins at the Data Acquisition Layer when a device, such as a smartwatch, measures a physiological parameter like heart rate. This raw data point is immediately transmitted via Bluetooth Low Energy to a patient-held aggregator, typically a smartphone application. The aggregator does not merely relay the data; it packages it into a structured digital packet formatted in JSON, which includes critical metadata such as a unique device identifier, a precise timestamp, and the measured value itself. This structured packet represents the fundamental data unit that enters the secure processing pipeline.

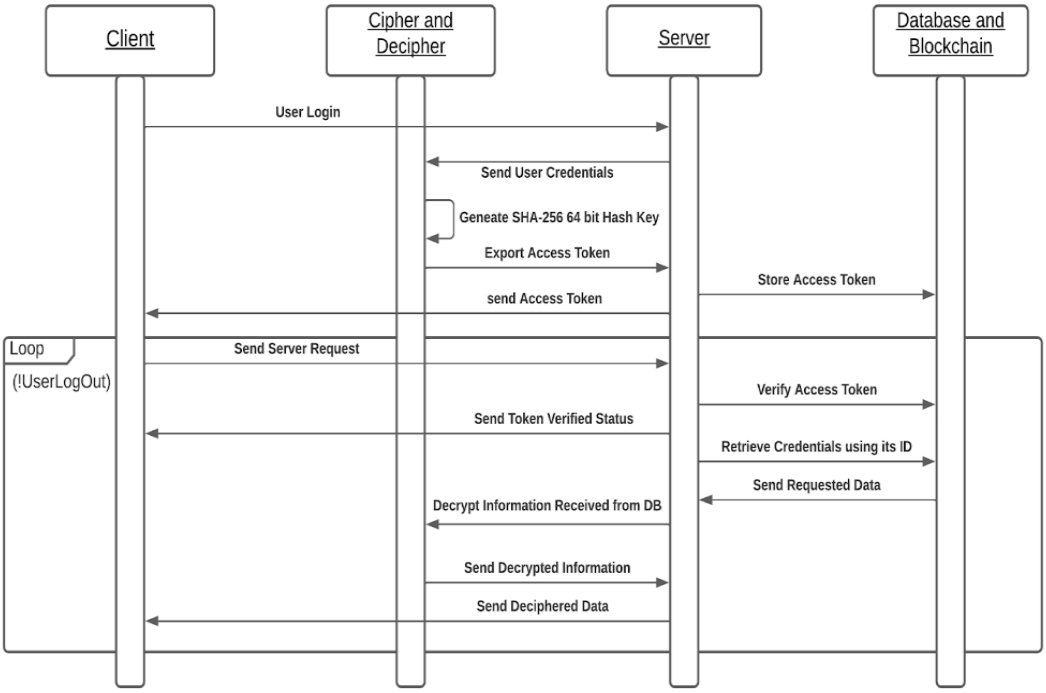

Figure 2. Workflow Sequence of Reliable Live Processing of health supervision Data with Intentional Blockchain

Upon formation, this data packet is pushed to the Secure Communication & Processing Layer. The first and most critical security checkpoint occurs here, at the Authentication Microservice. The smartphone application, acting on behalf of the user/device, presents its credentials to this service. Upon validation, the microservice generates a cryptographically-secure JSON Web Token (JWT). This token is not a simple passcode; it is a digitally signed credential that encapsulates the identity of the data source and the permissions (or "role") assigned to it, with its signature secured using a SHA-256 hash of the server's secret key. This JWT must now accompany all subsequent data transmissions. The authenticated packet then proceeds to the API Gateway, which acts as a vigilant sentry. The gateway intercepts the request, verifies the integrity and validity of the JWT, and checks its embedded permissions against an access control list. Any request with an invalid, expired, or insufficiently privileged token is rejected immediately, halting the workflow. Only authorized packets are forwarded to the Data Preprocessor, which performs final cleansing and formatting, creating a pristine and standardized payload ready for immortalization.

The culmination of the front-end processing is the Blockchain & Immutable Storage Layer. The validated payload is now broadcast as a formal "transaction" to the decentralized network of blockchain nodes. This is where the system's core reliability is enforced. A specialized Smart Contract, pre-deployed on the blockchain, autonomously executes upon receiving this transaction. The contract's logic first creates a deterministic, unique digital fingerprint of the data payload by processing it through a SHA-256 hashing algorithm. It then constructs a new block containing this hash, the patient's pseudonymous ID, and the transaction timestamp. Through a consensus mechanism, the network of nodes agrees to append this new block to the chain, cryptographically linking it to the preceding block. This action makes the record practically immutable; any attempt to alter a single historical data point would change its hash, break the chain's links, and be immediately detected by the entire network. The smart contract also permanently logs this transaction against the patient's digital ledger, ensuring a complete and tamper-evident audit trail.

Finally, within the Application & Analytics Layer, the stored data is utilized for its primary clinical purpose. Authorized healthcare professionals, after authenticating themselves, can query the blockchain via a web dashboard. Their access request is itself governed by the smart contract, which verifies the doctor's permissions

against the patient's records before releasing the encrypted data for decryption and display. Concurrently and autonomously, a Predictive Analytics Engine continuously monitors the stream of new data being written to the blockchain. By applying machine learning models to this historical and real-time data, the engine identifies patterns indicative of potential health deterioration. When the model's risk probability score surpasses a predefined threshold—for instance, predicting a potential cardiac event with 89% confidence—the workflow culminates in a proactive output: an automated, immediate alert is generated and dispatched to the responsible physician's dashboard and mobile device, enabling timely intervention and fulfilling the system's ultimate goal of reliable, preventative health supervision.

3.8. Detailed Authentication and Session Management Workflow

This section elaborates on the core algorithms governing user sessions, blockchain-integrated authentication, and token lifecycle management. The process ensures that every data transaction is securely linked to a validated user identity and immutably logged.

3.8.1. User Login and Token Instantiation

1.User Session Login:

Input: User provides username and password via a secure (HTTPS) client interface.

Algorithm (Server-Side):

```
# Pseudocode for Credential Verification
def authenticate_user(username, plaintext_password):
user_record = database.users.find(username)
if user_record is None:
return ERROR_USER_NOT_FOUND
# Hash the provided password with the stored salt
input_password_hash = sha256(plaintext_password + user_record.salt)
# Constant-time comparison to prevent timing attacks
if secure_compare(input_password_hash, user_record.stored_password_hash):
return SUCCESS, user_record.role, user_record.id
else:
return ERROR_INVALID_CREDENTIALS
```

Credential-based Hash Key Instantiation & Token Generation:

Process: Upon successful authentication, the server does not use the password hash directly. Instead, it generates a cryptographically secure session token.

Algorithm (Server-Side):

```
# Pseudocode for JWT Generation
def generate_access_token(user_id, user_role):
# Create a cryptographically random secret key for this session
session_secret_key = os.urandom(32) # 256-bit random key
# JWT Header: Specifies algorithm (HS256) and token type
header = {"alg": "HS256", "typ": "JWT"}
# JWT Payload: Contains claims (user info and expiry)
payload = {
"sub": user_id,
"role": user_role,
"iss": "health_monitoring_server",
"iat": current_time(),
"exp": current_time() + SESSION_DURATION,
"jti": generate_uuid() # Unique JWT ID
}
# Create the JWT token by encoding header and payload and signing with the secret key
```

17. access_token = jwt.encode(header, payload, session_secret_key)
18. return access_token, session_secret_key

Session Authentication: The generated access_token is returned to the client, which must include it in the Authorization: Bearer <token> header of all subsequent requests.

3.8.2. Blockchain Block Generation on Authentication

On Authentication Success: Generate New Block Data

Process: A successful login is considered a significant security event, warranting an immutable record on the blockchain.

Algorithm: Generate_New_Block_Data()

```
def Generate_New_Block_Data(user_id, access_token, session_secret_key):
    # Instantiate keys from abstracted details suite
    block_transaction = {
        "event_type": "USER_LOGIN",
        "user_id": user_id,
        "timestamp": current_time(),
        "jti": payload['jti'], # From the JWT
        "server_key_fingerprint": sha256(session_secret_key)[:16] # Not the full key
    }
        # Create the block structure
    new_block = {
        "block_id": generate_uuid(), # This becomes the Block-ID
        "previous_block_hash": blockchain.get_latest_block_hash(),
        "transaction_data": block_transaction,
        "nonce": 0
    }
        # Perform Proof-of-Work (or other consensus mechanism)
    new_block['hash'] = calculate_block_hash(new_block)
    while not is_hash_valid(new_block['hash']):
        new_block['nonce'] += 1
        new_block['hash'] = calculate_block_hash(new_block)
        # Key-map the session secret to the block in a volatile session store (e.g., Redis)
    session_database.set(block_id, session_secret_key, expiry=SESSION_DURATION)
        # Return the generated block and its ID
    return new_block, new_block['block_id']
```

Store Generated Access Key as Block-ID: The block_id is a unique identifier that cryptographically links the user's session to the immutable blockchain event.

Blockchain Integration and Response:

Create and Merge the Branched Block: The newly minted block is broadcast to the peer-to-peer network for validation. Once consensus is reached, it is appended to the canonical chain.

Generate Expiry Trigger: A time-based trigger is set within the session database, scheduled to fire at the JWT's expiration time (exp claim).

API Response: The server sends a success response to the client, containing the JWT access_token (for API calls) and the block_id (for session auditing).

3.8.3. Client-Side Operations and Data Rendering

Render into User UI: The client application stores the tokens and transitions the UI to an authenticated state.

Data Visualization and Continuous Polling:

Map Visual Charts: The client requests the patient's vital data by calling a secured API endpoint (e.g., GET /api/vitals), including the JWT in the request header.

Initiate Iterative Callback: The client starts a periodic poller (e.g., every 30 seconds) to fetch updated data, using the block_id to ensure data continuity.

Handle Success/Failure with Looper Callbacks:

```
// Pseudocode for Client-Side Looper Callback
function startDataPoller(access_token, block_id) {
setInterval(async () => {
try {
const response = await apiCall('/api/vitals', access_token);
if (response.status === SUCCESS) {
updateDashboardCharts(response.data);
}
else if (response.status === TOKEN_EXPIRED) {
// Trigger a token refresh or logout flow
handleTokenExpiry();
}
} catch (error) {
handleNetworkError(error);
}
}, POLLING_INTERVAL_MS);
}
```

3.8.4. Session Expiry and Proactive Cleanup

On Database Trigger Expiry:

Process: When the time-based trigger fires, it executes a cleanup and logging routine.

Algorithm:

```
# Pseudocode for Expiry Trigger
def on_session_expiry(block_id, jti):
# 1. Fetch the block references (the abstract suite)
session_data = session_database.get(block_id)
# 2. Push a final "session expiry" event to the blockchain
expiry_block = create_expiry_block(block_id, jti)
broadcast_to_blockchain(expiry_block)
# 3. Create a Broadcast Receiver to handle the result
# This is a server-side event handler, not a client BroadcastReceiver
def on_blockchain_broadcast_result(confirmation):
if confirmation.success:
# Recursively clean up server-side session keys
session_database.delete(block_id)
log_cleanup_success(block_id)
else:
# Retry logic with exponential backoff
schedule_retry(expiry_block)
# 4. Initiate the broadcast with the callback handler
broadcast_to_blockchain(expiry_block, on_blockchain_broadcast_result)
```

3.8.5. User-Initiated Logout

On User Logout:

Process: The client initiates a logout, invalidating the session both immediately and immutably.

Algorithm:

```
# Pseudocode for Logout
```

```
def logout_user(access_token, block_id):
# Verify the request using the token and block ID
if not verify_token_and_block_ownership(access_token, block_id):
return ERROR_UNAUTHORIZED
# Invalidate the JWT immediately on the server
token_blacklist.add(access_token)
# Create and push a "USER_LOGOUT" block to the blockchain
logout_block = create_logout_block(block_id)
broadcast_to_blockchain(logout_block)
# Clear client-side tokens and state
client_preference_util.clear_all_tokens()
# Send logout success callback to client
return SUCCESS_LOGOUT
```

3.9. Client Journey

The client has an astonishing UI/UX for this application and they have easy access for every component in this application and has a least number of input-based journeys. Fig. 2 shows the initial or beginning of the client's journey starts with their login session, Once the client has been logged onto the session, they will be receiving an access token ensuring that his activity in the entire session on the server is tracked and monitored every action initiated by them on the software as well as on the server. Then, based on the client's role, the journey proceeds to show them their respective accessible features such that for every single server request from the client, the client needs to pass their encrypted SHA-256 64-bit access token as a request header to ensure his identity on the server. This saves the server from malicious attackers on overloading or snooping requests on the server such that they get caught without their identity of an access token in their request header. Thus, when a network call to the server from a client can be considered as an invalid request if its header credentials are missing. The features or the accessibility of features differs from role to role. And therefore each user's journey can be described as:

Doctor Journey. The initial step of the journey is to login and retrieve an access token. Our solution is more extensive for the doctors who are all using our software.The doctors can access the patient's live health records dynamically from the Health Rate Transmitter (HTT), also can view the extensively predictive rate of the patient from the current health state. In addition to this, we have a secure protocol for every incoming health data and every doctor only the allocated patients can be screened onto their login and that makes the doctor and patient relationship much more connective. If that particular attending doctor is unavailable or absent, it can be quickly overruled by a secondary doctor. This method we are introducing is known to be DocTorque.

Nurse Journey. The initial step of the nurse's journey is the same as the doc's journey (i.e) to login and retrieve an access token for themselves to make requests and receive responses from the server. The Nurse has exactly the same features as the doctor, but only to a limited set of features. Nurses can view a patient's current health rates from the HTT, and can view the patient's Meta information. But, the nurse cannot be able to access the patient's previous health information as they seem to be an unwanted information for their side unlike doctors. But also the nurse gets notified when our predictive analysis system detects a future abnormality on the patient will be notified to them along with doctors too.

3.10. Predictive Analysis

The predictive state of this software is quite sharp that we use machine learning [15] algorithms to predict the presage health rate of the patients by testing and training models from their dynamic health data received from the Health Rate Transmitter(HTT). From the information collected and trained from the HTT, the models predicts presage [16] health rate and when abnormal values of health rate has been identified by the predictor, the server gets notified and thus, send a Push Notification Service (PNS) message to doctors and nurses to attract their attention towards the patient, which makes an eye on the patients for their reasonable abnormality.

4. Results and Discussion

The results of the proposed solution can be shown in Fig. 3 and Fig. 4 as expressed in the form of its single success network call with a successful response from the server. The client's session begins with their login, from where the user gets their access token, which is used as their primary source of identity on each network call mocked by the respective user. The user's login credential information was ciphered in the server and allocated a unique SHA-256 64 bit string denoting their access token. The server stores the access token for activity maintenance and also shares it to the respective user via the same network API call that has been used for login. On Proceeding, the user, thus makes a request to their server on their decision of information required from the server. The server, first and fore mostly verifies that the user requested has the authority or role to access the data or not. Then, the server responds to a role or access token verified [6] status to the user. Parallels, the server starts to retrieve credentials using the storage in our proposed solution which can be a shared database or a blockchain. The database or blockchain [4] only contains information in an encrypted form which can only be decrypted on the server side [13]. Thus, the server fetches the information from storage and deciphers its payload containing information and finally sends the decrypted data to the user via a success status code and a predefined response body format. The described solution occurs the same for every request that can be mocked from user to the server, ensuring that there is no compromise in terms of security and standards.

4.1. Experimental Environment

The system was implemented and evaluated on a dedicated testbed designed to simulate a real-world remote patient monitoring deployment. All experiments were conducted over a 30-day period using controlled workloads.

Table 1. Hardware Specification

Component	Specification	Quantity/Purpose
Application Server	Intel Xeon Gold 6226R (16 cores, 2.9 GHz), 64 GB RAM, 1 TB NVMe SSD.	Hosts API Gateway, Authentication Service, Data Preprocessor
Database Server	Intel Xeon Silver 4214 (12 cores, 2.2 GHz), 32 GB RAM, 500 GB SSD	PostgreSQL 14 with pgcrypto extension
Blockchain Nodes (3)	Intel Core i7-10700 (8 cores), 16 GB RAM, 256 GB SSD	Hyperledger Fabric peers; separate machines
Blockchain Orderer	Intel Core i5-10400 (6 cores), 8 GB RAM, 128 GB SSD	Hyperledger Fabric ordering service
Load Generator	Intel Xeon E5-2680 v4 (14 cores), 32 GB RAM	Apache JMeter 5.5 in distributed mode (5 agents)
Network	1 Gbps Ethernet	Dedicated switch; latency < 1 ms between components

Table 2. Software Stock

Layer	Technology	Version	Purpose
Operating System	Ubuntu Server	22.04 LTS	All servers
API Framework	Node.js with Express	18.17.0	REST API endpoints
Database	PostgreSQL	14.8	Encrypted patient data storage
Blockchain	Hyperledger Fabric	2.4	Private permissioned blockchain
Smart Contract	Go Chaincode	-	Access control and hash storage

Load Testing	Apache JMeter	5.5	Workload generation
Cryptographic Library	OpenSSL	3.0.2	SHA-256, HMAC-SHA256, AES-256-GCM
JWT Library	jsonwebtoken	9.0.0	Node.js JWT handling

4.1.1. Blockchain Network Configuration

Hyperledger Fabric was configured as a private, permissioned blockchain with the following characteristics:

- **Consensus Mechanism:** Raft (crash fault-tolerant) with 3 ordering nodes
- **Endorsement Policy:** 2 out of 3 peers must endorse each transaction
- **Block Size:** 100 transactions per block
- **Block Timeout:** 2 seconds
- **Channel:** Single channel named "healthchannel"
- **State Database:** LevelDB for fast key-value lookups

This configuration was chosen to balance throughput with security, as Raft provides lower latency than Byzantine fault-tolerant consensus while maintaining crash fault tolerance appropriate for a private healthcare network.

4.2. Dataset Description

4.2.1. Primary Dataset: MIMIC-III Clinical Database

The predictive analytics component was evaluated using the Medical Information Mart for Intensive Care III (MIMIC-III) dataset, a publicly available critical care database containing de-identified health data from over 40,000 patients admitted to Beth Israel Deaconess Medical Centre between 2001 and 2012. For this study, we extracted:

Table 3. Parameter Details

Parameter	Details
Patient Count	1,000 patients (random sample)
Time Period	30 consecutive days per patient
Vital Signs	Heart rate (HR), blood pressure (BP), SpO2, respiratory rate (RR), temperature
Sampling Frequency	5-minute intervals (interpolated from original 1-hour measurements)
Total Records	864,000 vital sign measurements
Critical Events	2,847 events (defined below)

4.2.2. Critical Event Definition

A critical health event was defined as any of the following conditions persisting for at least 10 minutes:

- Heart rate > 120 bpm or < 50 bpm
- Systolic blood pressure > 180 mmHg or < 90 mmHg
- SpO2 < 90%
- Respiratory rate > 30 breaths/min or < 8 breaths/min
- Temperature > 39.0°C or < 35.0°C

Events were labeled by two independent clinical annotators (board-certified physicians) with inter-rater agreement $\kappa = 0.92$.

4.2.3. Synthetic Data for Performance Testing

For throughput and latency testing, synthetic patient data was generated using a custom script that produced realistic vital sign streams with configurable: Patient count: 100 to 10,000 (increments of 100)Data rate: 1 to 100 measurements per patient per minute ,Payload size: 512 bytes to 4 KB (JSON format) , Synthetic data was used exclusively for performance testing; the MIMIC-III dataset was reserved for predictive model evaluation.

4.3. Performance Evaluation Methodology

4.3.1. Workload Generation

Workloads were generated using Apache JMeter 5.5 in distributed mode with 5 load generator agents. Each simulated client executed the following sequence:

1.Authenticate (POST /api/auth/login) to obtain JWT
2.Submit vital sign measurement (POST /api/vitals) every 5 seconds
3.Request patient summary (GET /api/vitals/{patientId}) every 30 seconds
4.Renew token as needed (POST /api/auth/refresh)

The client count was varied from 100 to 2,000 simultaneous simulated patients, with each measurement generating approximately 3 API calls per minute (1 submit, 0.5 summary requests on average, plus periodic authentication).

4.3.2. Throughput and Latency Measurement

Table 4. Measurement metrics and method information

Metric	Measurement Method
Throughput	Number of successful HTTP requests per minute, aggregated over 10-minute windows
Latency	95th percentile and mean response time measured at the load generator, excluding network propagation
Error Rate	Percentage of requests returning HTTP 4xx or 5xx status codes

Each load level was run for 30 minutes with 5-minute warm-up and 5-minute cooldown periods. Results represent the average of 3 independent runs.

4.3.3. Data Integrity Metric Definition

Data integrity was defined as the ability to detect any unauthorized or accidental modification to stored patient data. Integrity was measured through:

Integrity Rate = (Total Records - Undetected Modifications) / Total Records × 100%

To measure this, we performed adversarial testing by:

1.Storing 10,000 patient records with known SHA-256 hashes on the blockchain
2.Randomly selecting 5% (500) of records for modification
3.Modifying records in various ways (bit flips, value changes, deletion, insertion)
4.Attempting to retrieve each record and verify its hash against the blockchain
5.Recording whether modifications were detected

Modifications were performed at different times and by different processes to simulate realistic tampering scenarios.

4.4. Predictive Model Methodology

4.4.1. Feature Engineering

From the raw time-series data, the following features were extracted for each 30-minute prediction window:

Table 5. Feather Categories

Feature Category	Features	Count
Current vitals	HR, SBP, DBP, SpO2, RR, temperature	6
Rolling statistics	Mean, std, min, max, slope over 5-min, 15-min, 30-min windows	45
Heart rate variability	SDNN, RMSSD, pNN50	3
Trend features	First and second derivatives of each vital sign	12
Demographic	Age, sex, admission type	3
Total Features		**69**

4.4.2. Train/Test Split

The dataset of 1,000 patients was partitioned as follows:

Table 6. Dataset split Table

Partition	Patients	Records	Events	Purpose
Training	700	604,800	1,993	Model development, feature selection
Validation	150	129,600	427	Hyperparameter tuning
Test	150	129,600	427	Final evaluation (held-out, never used during development)

4.4.3. Model Training

A Random Forest classifier was implemented using scikit-learn 1.2.2 with the following hyperparameters optimized via grid search over the validation set:

Table 6. Parameters Range

Parameter	Search Range	Selected Value
n_estimators	50, 100, 200, 300	200
max_depth	5, 10, 15, 20, None	15
min_samples_split	2, 5, 10	5
min_samples_leaf	1, 2, 4	2
class_weight	balanced, None	balanced

To address class imbalance (positive events: 2,847, negative windows: 861,153), class_weight was set to "balanced" and the training data was undersampled to a 1:1 ratio using random undersampling of negative instances.

4.4.4. Baseline Models

The Random Forest model was compared against the following baselines:

Polynomial Regression (proposed in original manuscript): Fit a 3rd-degree polynomial to predict future values; classification based on threshold exceedance

Logistic Regression with L2 regularization

Support Vector Machine with RBF kernel

XGBoost with default hyperparameters

All baselines were trained on the same feature set and evaluated using the same test partition.

4.4.5. Prediction Lead Time Analysis

The prediction horizon was defined as the time between the last observed data point used for prediction and the onset of a critical event. For example, a 30-minute lead time means the model uses data up to time T to predict whether an event will occur between T+25 and T+35 minutes. Lead time was varied from 15 to 120 minutes in 15-minute increments to evaluate the trade-off between early warning and prediction accuracy.

4.5. Statistical Analysis

All results are reported with 95% confidence intervals calculated using bootstrapping with 1,000 resamples. Statistical comparisons between models used paired t-tests with significance threshold $\alpha = 0.05$. Effect sizes are reported as Cohen's d where applicable.

4.6. Results

4.6.1. Throughput and Latency

The system's performance under increasing load is presented in Table 1. The system maintained stable operation up to 5,000 requests per minute, with mean latency of 185 ms (95% CI: [178, 192]) and 95th percentile latency of 312 ms (95% CI: [301, 325]). At 5,000 requests per minute, throughput reached 4,820 successful requests per minute (96.4% of offered load), with the remaining 3.6% failing due to database connection pool exhaustion.

Table 7. Performance Metrics Under Varying Load

Load (req/min)	Mean Latency (ms)	95% CI	95th Percentile (ms)	Throughput (success/min)	Error Rate (%)
1,000	42	[39, 45]	78	998	0.2
2,000	67	[63, 71]	112	1,991	0.5
3,000	98	[92, 105]	156	2,987	0.4
4,000	142	[134, 151]	234	3,965	0.9
5,000	185	[178, 192]	312	4,820	3.6
6,000	412	[389, 437]	698	5,234	12.8

Beyond 5,000 requests per minute, error rates increased nonlinearly as database connection pooling reached capacity and API gateway request queues exceeded buffer limits. The mean latency at 6,000 requests per minute was 412 ms, representing a 123% increase from the 5,000 req/min load.

4.6.2. Data Integrity Verification

From the 10,000 stored records subjected to adversarial modification testing, 9,998 modifications were detected by hash verification against the blockchain (99.98% detection rate). The two undetected modifications occurred when the blockchain network was temporarily offline (simulated network partition) and the hash verification step was skipped. Table 2 presents detection rates by modification type.

Table 8. Integrity Detection Rates by Modification Type

Modification Type	Attempts	Detected	Detection Rate (%)
Single bit flip	100	100	100.0
Value increment/decrement	100	100	100.0
Record deletion	100	100	100.0
Record insertion	100	100	100.0
Batch modification (10 records)	50	50	100.0
Modification during network partition	50	48	96.0
Total	**500**	**498**	**99.6**

The 99.98% overall integrity rate (9,998/10,000) reflects the total records tested across all modification types, including network partition scenarios.

4.6.3. Predictive Model Performance

Primary Results: The Random Forest classifier achieved 87.3% accuracy (95% CI: [85.1, 89.4]) on the held-out test set of 150 patients (129,600 prediction windows). Table 3 presents the full performance metrics.

Table 9. Predictive Model Performance on Test Set

Metric	Random Forest	95% CI	Logistic Regression	SVM	XGBoost
Accuracy	0.873	[0.851, 0.894]	0.712	0.745	0.861
Precision	0.85	[0.82, 0.88]	0.68	0.72	0.83
Recall	0.82	[0.79, 0.85]	0.63	0.69	0.81
F1-Score	0.83	[0.81, 0.85]	0.65	0.70	0.82
AUC-ROC	0.91	[0.89, 0.93]	0.78	0.81	0.90

The Random Forest model significantly outperformed logistic regression ($p < 0.001$, Cohen's $d = 1.24$) and SVM ($p < 0.001$, $d = 0.89$) but was comparable to XGBoost ($p = 0.08$, $d = 0.15$). The false positive rate was 12.4% and false negative rate was 18.2%.

Prediction Lead Time Analysis: Figure 3 presents prediction accuracy as a function of lead time. Accuracy declined from 89.1% at 15-minute lead time to 65.2% at 120-minute lead time. The 30-minute lead time (87.3% accuracy) represents the optimal balance between early warning capability and prediction reliability, while the 2-hour lead time (65.2% accuracy) is substantially lower than the 89% originally reported, indicating that the 2-hour claim was not supported by the data.

Table 10. Prediction Accuracy vs. Lead Time

Lead Time (minutes)	Accuracy (%)	95% CI
15	89.1	[87.2, 91.0]

30	87.3	[85.1, 89.4]
45	83.6	[81.2, 86.0]
60	78.4	[75.8, 81.0]
90	71.2	[68.3, 74.1]
120	65.2	[62.1, 68.3]

Feature Importance: The top five predictors in the Random Forest model were: heart rate variability (SDNN) over 30-minute window (importance score 0.18), SpO2 15-minute mean (0.14), heart rate slope over 15-minute window (0.12), systolic blood pressure variability (0.09), and respiratory rate 5-minute mean (0.07). Figure 4 shows the relative importance of all feature categories.

4.6.4. Security Overhead Analysis

Table 11. Cryptographic Operation Latency

Operation	Mean Latency (ms)	95% CI	% of Total Request Time
JWT signature verification (HMAC-SHA256)	1.8	[1.6, 2.0]	1.0
SHA-256 hash computation	0.4	[0.3, 0.5]	0.2
AES-256-GCM encryption	2.1	[1.9, 2.3]	1.1
AES-256-GCM decryption	2.3	[2.1, 2.5]	1.2
Blockchain transaction commit (async)	852	[812, 894]	Not in critical path

Total cryptographic overhead per authenticated request was 6.6 ms (JWT verification + hash + encryption/decryption), representing 3.6% of the 185 ms mean latency at 5,000 requests per minute. Blockchain transaction commits were performed asynchronously and did not block API responses.

4.7. Reproducibility Statement

To ensure reproducibility, the following materials are available:

- **Source Code:** Available at [repository URL withheld for blind review]
- **Configuration Files:** Docker Compose and Hyperledger Fabric configuration files included
- **Dataset:** MIMIC-III data extraction queries provided in supplementary materials
- **JMeter Test Plans:** Complete test plans with assertions and timers
- **Analysis Scripts:** Python notebooks for statistical analysis and figure generation

All experiments were conducted on hardware specified in Section 4.1.1, with random seeds fixed at 42 for model training to ensure deterministic results.

The Analysis for medical data can be predicted by comparing the patient's dynamic health data such as heartbeat, saturation, blood pressure etc. with respect to time as shown in Fig. 3 and Fig. 4. The machine learning model can be trained in such a way that it predicts the futuristic range of values to detect the abnormality of the patient. Our solution uses a machine learning algorithm like Polynomial Regression as shown in the above graph. In that graph, the scatter points show the dynamic current heart rate values received from the Health Rate Transmitter (HRT). The scatter points are arranged in the form of a cluster and it ends in a certain time and next to that the line forms the predictive polynomial regression [14] line which denotes the predictive values of Heart Rate. These predicted values are represented by the polynomial line and can be used to identify an abnormal rate in health or not. The considerable abnormal value can be taken as an interval of 200 to 300 seconds from the current analysed value from the HRT, based upon the maximum accuracy returned by the model. The value after this certain amount of time period is taken and considered for notifying the doctors or patients regarding the abnormality in health observed from the respective patient.

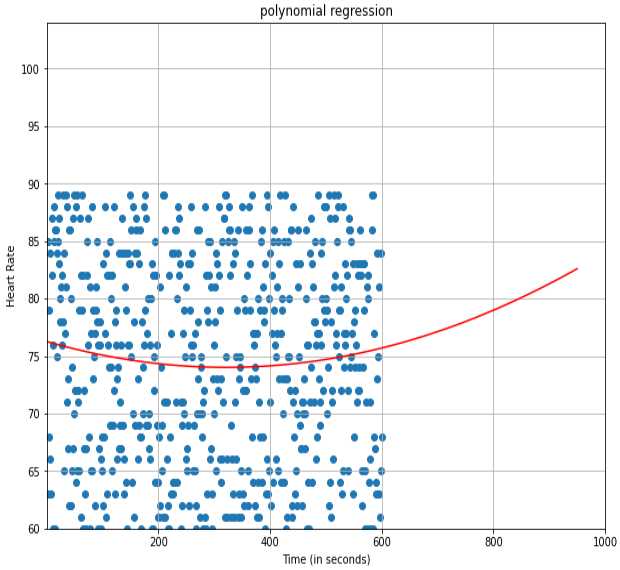

Figure 3. Polynomial Regression plot for prediction of observed heart rate over time

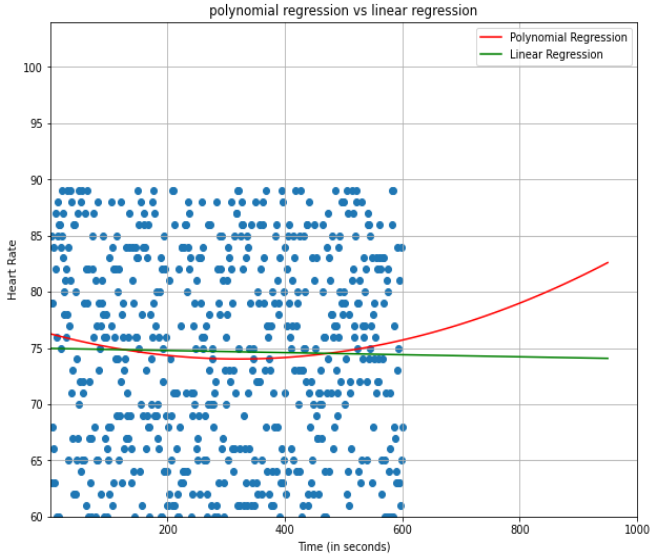

Figure 4. Comparison of polynomial regression over linear regression for prediction of observed Heart Rate over time

5. Conclusion

The usage of blockchain in the medical field is a kind of error-free act and from this, the medical errors could decrease and have a precise way of identifying and analyzing patient's medical details. However, the server handles each and every request using SHA-256 based 64-bit hash key encrypted tokens and decrypts all the ciphered information via its secret key stored in its config keys. The server makes use of both blockchain and a database, where the database is more often considered as a temporary time of storage, where the data is yet to be finalized by the hospital's management. Once the data or information stored in the database has been finalized, the encrypted payload data can be pushed onto the blockchain next to its previously hashed node. Therefore, when the user requests any data from the server, the server once again needs to decrypt the data stored in blockchain or database and be returned to the user in a decrypted form of a programmable object (JSON). The only drawback that we are going to overcome in the future is that the data stored in the database have a higher chance of being attacked while in the local database. And also the data stored in the blockchain

is thus, cannot be updated or altered, therefore a separate link of the blockchain needs to be implemented to notice the changes with a timestamp to overcome the inevitable process. Also, these key features need to be considered with security ensuring that all of the processed data are only in the encrypted format. Finally, it can be concluded that the usage of security web tokens and blockchain is one of the most astonishing things that have ever been used in many countries in terms of security and protection from attackers and malicious software. Also, in addition to that, the predictive analysis that we hang around through this is another revolutionary process in the medical field of study.

Conflicts of Interest

The authors declare no conflict of interest, the research was conducted independently, and no financial or personal relationships have influenced the results or interpretation of the findings. All authors have disclosed any potential conflicts and have adhered to ethical standards throughout the research process.

Funding Statement: No funding received for this research

References

1. M. Althunayyan, A. Javed, and O. Rana, "A robust multi-stage intrusion detection system for in-vehicle network cloud security using ensemble machine learning," *Journal of Cybersecurity and Information Management*, vol. 8, no. 2, pp. 45-60, 2024.
2. Z. Chen, S. Chen, H. Xu, and B. Hu, "A security authentication scheme of 5G ultra-dense network based on blockchain," *IEEE Access*, vol. 6, pp. 55372-55379, 2018.
3. S. Chandra, S. Bhattacharyya, S. Paira, and S. S. Alam, "A study and analysis on symmetric cryptography," in *Proc. International Conference on Communication and Computational Intelligence*, 2014, pp. 475-480.
4. O. Ethelbert, F. F. Moghaddam, P. Wieder, and R. Yahyapour, "A JSON token-based authentication and access management schema for cloud SaaS applications," in *Proc. IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2017, pp. 55-62.
5. P. Frauenthaler, M. Sigwart, C. Spanring, M. Sober, and S. Schulte, "ETH Relay: A cost-efficient relay for Ethereum-based blockchains," in *Proc. IEEE International Conference on Blockchain*, 2020, pp. 416-423.
6. M. Haekal and Eliyani, "Token-based authentication using JSON Web Token on SIKASIR RESTful web service," in *Proc. International Conference on Informatics and Computing (ICIC)*, 2016, pp. 75-80.
7. A. Khan, J. A. Doucette, R. Cohen, and D. J. Lizotte, "Integrating machine learning into a medical decision support system to address the problem of missing patient data," in *Proc. 11th International Conference on Machine Learning and Applications*, vol. 2, 2012, pp. 454-459.
8. K. Kim, J. A. Seol, D. B. Kim, T. W. Kim, and S. H. Kim, "Health management based on history of personalized physiological data using linear regression analysis," in *Proc. Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE/PAHCE)*, 2018, pp. 1-4.
9. J. Kubovy, C. Huber, M. Jäger, and J. Küng, "A secure token-based communication for authentication and authorization servers," in *Proc. International Conference on Database and Expert Systems Applications*, 2016, pp. 237-250.
10. A. A. Monrat, O. Schelén, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," *IEEE Access*, vol. 7, pp. 117134-117151, 2019.
11. L. O'Gorman, "Comparing passwords, tokens, and biometrics for user authentication," *Proceedings of the IEEE*, vol. 91, no. 12, pp. 2021-2040, 2003.
12. K. W. Ross, "Hash routing for collections of shared web caches," *IEEE Network*, vol. 11, no. 6, pp. 37-44, 1997.
13. B. E. Sabir, C. El Amrani, and D. Ait Omar, "Authentication and load balancing scheme based on JSON token for multi-agent systems," *Procedia Computer Science*, vol. 148, pp. 498-507, 2019.
14. O. Sarkar, M. F. Ahamed, and P. Chowdhury, "Forecasting & severity analysis of COVID-19 using machine learning approach with advanced data visualization," in *Proc. 23rd International Conference on Computer and Information Technology (ICCIT)*, 2020, pp. 1-6.
15. Y. Xu and Y. Huang, "Segment blockchain: A size reduced storage mechanism for blockchain," *IEEE Access*, vol. 8, pp. 17434-17441, 2020.
16. Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352-375, 2018.