Software Requirements Specification

Blockchain Based Audit Log System: Secure and Tamper-Proof Audit Logging

Document Version: 1.0

Date: February 2026

Classification: Internal - Draft

Status: Under Review

Project Team: Aashin Mohammed A Z

A S Abhishek

Ananth K

Aman Sami

1. Introduction

1.1 Purpose

The purpose of this Software Requirement Specification (SRS) is to describe the design and operational behavior of the **Blockchain-Based Audit Log System**. The document provides a clear understanding of how the system securely captures, stores and verifies audit logs with blockchain technology. This specification serves as a reference for development,design as well as testing decisions and to be updated as the project evolves throughout the phases.

1.2 Project Scope

The project focuses on developing a secure and tamper-proof audit logging system that records **aggregated representations of critical activities** performed by users and administrators within an application. These activities include authentication-related events such as login attempts and logout operations, as well as sensitive data operations like record creation, modification, and deletion. Instead of storing detailed per-action logs, the system captures **summary metrics such as counts, frequency, and time-windowed activity patterns**, ensuring that no sensitive or personally identifiable information is recorded.

To guarantee audit integrity and privacy-first security, the system adopts an **on-chain aggregated audit logging approach**. Aggregated audit data are directly stored on a blockchain network, where immutability and decentralization ensure that recorded summaries cannot be altered or deleted once committed. Each aggregation is cryptographically secured and serves as a verifiable audit evidence of system activity without revealing individual user actions.

The blockchain layer functions as a trust anchor for monitoring system behavior, enabling reliable detection of anomalies such as repeated failed authentication attempts, excessive request rates, or suspicious access patterns. Any attempt to manipulate historical audit summaries is immediately detectable due to the immutable nature of blockchain storage.

During attack scenarios, including denial-of-service or brute-force attempts, the system continues to record aggregated activity metrics on-chain, ensuring that evidence of abnormal behavior remains permanently available for post-incident analysis. Compared to traditional centralized audit logging systems—where detailed logs may be modified, erased, or lost—the proposed solution enhances accountability and transparency while maintaining strong privacy guarantees.

1.3 Definitions, Acronyms, and Abbreviations

Term/Acronym	Definition
Audit Log	A structured record of system events capturing actions performed by users or administrators.
Hash(SHA-256)	A cryptographic hash algorithm used to generate a fixed-length, unique hash for audit log data.
Blockchain Anchoring	The process of storing cryptographic hashes of audit logs on a blockchain to ensure immutability and integrity.
Smart Contract	A self-executing program deployed on a blockchain that enforces predefined rules and logic.
WebAssembly(WASM)	A portable binary instruction format that enables efficient and secure execution of smart contracts.
ink!	A Rust-based smart contract language used for developing WebAssembly smart contracts on Polkadot parachains.
Parachain	An application-specific blockchain that runs in parallel within the Polkadot ecosystem and connects to the relay chain.
Trust Anchor	A reliable reference point used to verify the authenticity and integrity of audit data.
Substrate	A modular blockchain development framework used to build custom blockchains and parachains within the Polkadot ecosystem.
Audit Pallet	A Substrate runtime module responsible for managing audit-related logic, including storage and verification of audit log hashes.
Consensus Mechanism	A protocol used by blockchain networks to achieve agreement on the validity and order of transactions.
JSON-RPC	A remote procedure call protocol that uses JSON for data exchange,commonly used for communication between blockchain nodes and external applications.

2.Overall System Description

2.1 System Overview

The Blockchain-Based Audit Log System is a **privacy-preserving, blockchain-centric audit logging solution** designed to ensure the integrity, authenticity, and reliability of audit evidence generated within an application. Rather than storing detailed per-user activity logs, the system captures **aggregated representations of critical activities** performed by users and administrators, including authentication-related events and sensitive data operations.

Audit information is summarized in the form of **non-identifiable metrics**, such as event counts, frequency of actions, and time-windowed activity patterns. These aggregated audit records are **directly stored on a blockchain network**, where immutability and decentralization guarantee that once recorded, they cannot be altered or removed without detection.

The blockchain layer functions as a **tamper-proof trust anchor**, enabling reliable detection of abnormal or unauthorized system behavior, such as repeated failed login attempts or excessive access to sensitive resources. Since only aggregated and non-sensitive data are recorded on-chain, the system maintains strong privacy guarantees while still providing verifiable accountability.

By eliminating reliance on mutable centralized log storage and leveraging blockchain's immutable ledger, the system offers a scalable and trustworthy audit logging framework that enhances transparency and accountability compared to traditional centralized logging solutions.

2.2 System Architecture

The system is designed to operate at the backend layer of the application, ensuring that all critical system operations are monitored in a centralized and non-bypassable manner. By integrating the audit mechanism directly within the backend, the system guarantees that sensitive actions—such as authentication attempts and access to protected resources—are consistently accounted for.

Instead of generating detailed per-event audit logs, the system aggregates audit information into **privacy-preserving summaries**. These summaries capture non-identifiable metrics such as the number of occurrences of specific actions, frequency of events within defined time windows, and threshold-based activity indicators. This approach avoids storing sensitive user-specific data while still enabling effective monitoring and accountability.

Aggregated audit records are **directly stored on a blockchain network** through smart contract interactions. Each on-chain record represents a cryptographically secured snapshot of system activity for a given interval, ensuring immutability and chronological consistency. Any attempt to modify or remove previously recorded audit data is inherently prevented by the blockchain's append-only nature.

The smart contract is deployed on a **Polkadot parachain** and implemented using **WebAssembly-based ink!**, providing a deterministic and secure execution environment. This architecture eliminates reliance on mutable centralized audit databases while maintaining scalability through aggregation. By combining backend-level enforcement with blockchain immutability, the system ensures trustworthy audit evidence even in adversarial or compromised system conditions.

2.3 On-Chain Aggregated Storage Model

The system adopts an **on-chain aggregated audit storage model** designed to provide accountability and integrity guarantees while minimizing privacy risks and storage overhead. Instead of storing detailed audit logs or individual user actions, the system records **summarized audit metrics** directly on the blockchain.

All aggregated audit records are written directly to the blockchain through smart contract transactions. Due to the immutable and append-only nature of blockchain storage, once an aggregation is recorded, it cannot be altered or removed without detection. This design eliminates reliance on centralized, mutable audit databases and ensures that audit evidence remains trustworthy even if the backend system is compromised.

2.4 Security and Unauthorized Activity Detection

The system continuously monitors backend-level events to identify unauthorized, suspicious, or policy-violating behavior within the application. Examples of such behavior include repeated failed authentication attempts, excessive request rates, or abnormal access to protected resources.

When such patterns are detected, the system records **aggregated security indicators** on-chain, ensuring that evidence of abnormal behavior is preserved in an immutable form. Audit aggregation continues without interruption during attack scenarios, including brute-force attempts or denial-of-service conditions, ensuring that security-relevant activity is always accounted for.

By relying on immutable on-chain storage of aggregated metrics rather than detailed logs, the system maintains strong privacy guarantees while still enabling reliable post-incident analysis. This approach strengthens accountability and ensures that security-relevant evidence cannot be silently modified or erased.

2.5 Audit Verification and Accountability

Audit verification in the proposed system is performed by examining the **on-chain aggregated audit records** stored in the blockchain ledger. Authorized auditors or security analysts can independently review these immutable records to validate system behavior over time.

Since audit data are recorded directly on-chain, integrity verification does not rely on recomputation or comparison with external databases. Any attempt to manipulate historical audit evidence would require altering blockchain records, which is infeasible due to cryptographic and consensus protections.

This verification mechanism provides **tamper-evident accountability** rather than detailed forensic reconstruction. While the system does not expose individual user actions, it enables auditors to reliably determine whether system activity remained within defined security and policy thresholds, thereby preserving trust in the audit process.

3.Session Initialization and on boarding

3.1 User Input Collection

Before a session begins, the system collects structured authentication and context data required to initialize a secure and auditable session. This step is critical, as all subsequent actions are bound to the session identity and recorded for audit integrity.

Input Field	Type	Required	Description
Wallet	Polkadot Wallet	Yes	Unique identifier for user or administrator
Wallet Signature	Cryptographic Signature	Yes	Verifies wallet ownership during session initialization
Access Context	System-generated	Yes	IP address, timestamp, device metadata

All input events (successful or failed attempts) are recorded as audit logs and prepared for blockchain anchoring.

3.2 Session Initialization Module

The Session Initialization Module is responsible for verifying user credentials and establishing a secure session environment. It serves as the controlled entry point to the system and ensures that only authenticated entities are allowed to perform actions.

Upon successful authentication, the system:

- Generates a unique and non-reusable session identifier
- Initializes session state parameters
- Triggers the Auditing software Module to record the session start

Session State Information Maintained:

- Session ID
- User identity
- Session start time
- Current session status (active, expired, or terminated)

In the event of authentication failure, the system records the attempt and forwards the information to the Attack Detection and Prevention Module for monitoring abnormal behavior patterns.

3.3 Onboarding Flow

User onboarding is a controlled process carried out exclusively by administrators to maintain system security and accountability. Self-registration is intentionally excluded to minimize unauthorized access and reduce security risks.

Onboarding Process Includes:

- Administrator-driven user registration
- Secure creation and storage of initial credentials
- Activation of audit logging from the user's first session

The completion of onboarding is recorded as a high-integrity audit event, and its cryptographic hash is anchored to the blockchain to ensure verifiability.

3.4 Security and Audit Integration

Session-related activities are considered security-critical and are fully integrated with the audit logging and blockchain mechanisms of the system.

Audited Session Events Include:

- Successful login
- Failed login attempts

- Session timeout due to inactivity
- Explicit user logout

For each event, a detailed audit record is generated, hashed, and anchored to the blockchain. Any future alteration of session records can be detected by comparing database logs with blockchain-stored hashes.

Potential Risk:

Compromised credentials or session hijacking could lead to unauthorized access.

Mitigation Measures:

Rate limiting, monitoring repeated failures, session expiration policies, and immutable blockchain-based audit verification.

4.Core Modules - Functional Requirements

Each module is responsible for a specific part of the system and collectively ensures secure, tamper-evident, and privacy-preserving audit logging. The functional requirements are specified with unique identifiers and priority levels to guide implementation and evaluation.

4.1 Audit specific Parachain

The Audit-Specific Parachain acts as the **blockchain-based trust layer** of the system. It is responsible for maintaining immutable records related to audit activities without storing sensitive log data. The parachain stores **cryptographic hashes, counters, or aggregated audit data** that represent audit events generated by external systems.

This module ensures that once an audit proof is recorded on-chain, it cannot be altered or deleted, even if the off-chain infrastructure is compromised. The parachain is designed to be lightweight and optimized for audit verification rather than general-purpose computation.

Req Id	Requirement	Priority
4.1.1	The parachain shall accept audit proof data (hashes or counters) from authorized backend services/SDKs	High

4.1.2	The parachain shall store audit proof data in an immutable ledger	High
4.1.3	The parachain shall prevent modification or deletion of previously stored audit proofs	High
4.1.4	The parachain shall associate each audit proof with a timestamp	Medium
4.1.5	The parachain shall support querying audit proof data for verification purposes	High
4.1.6	The parachain shall store only minimal metadata to preserve user privacy	High
4.1.7	The parachain shall operate on a local or test network suitable for academic deployment	Medium

4.2 Auditing Software Development Kit (SDK)

The Auditing Software Development Kit (SDK) is designed to be **embedded directly within the source code** of client applications that require auditing services. The SDK provides **event listeners and event emitters** that capture significant application-level events such as user actions, system changes, and administrative operations.

These event listeners operate transparently within the host application and collect audit-relevant data at runtime. Captured events are standardized, enriched with metadata, and forwarded to the auditing software for further processing. This approach allows organizations to integrate auditing capabilities **without redesigning their existing systems**.

The SDK ensures that data collection is **consistent, configurable, and secure**, while avoiding direct exposure of sensitive business logic or user information.

Req Id	**Requirement**	**Priority**
4.2.1	The SDK shall provide event listeners to capture application-level audit events	High

4.2.2	The SDK shall allow integration within the source code of client applications	High
4.2.3	The SDK shall support event emitters for generating standardized audit events	HIgh
4.2.4	The SDK shall collect audit data using configurable event triggers	High
4.2.5	The SDK shall generate cryptographic hashes of captured audit events	High
4.2.6	The SDK shall securely transmit audit data to the auditing software backend	High
4.2.7	The SDK shall minimize performance overhead on the host application	Medium
4.2.8	The SDK shall prevent direct access to sensitive application data	High
4.2.9	The SDK shall support reuse across multiple organizations and applications	Medium

4.3 Auditing Software

The Auditing Software is the core processing component responsible for handling audit data generated by the Auditing SDK and preparing it for blockchain-based verification. In this system, the auditing software operates on a **privacy-preserving, aggregate-only audit model** and does not rely on a traditional database for persistent storage.

The auditing software processes **quantitative audit metrics**, such as the number of user activities, frequency of actions, and aggregated interaction counts. It explicitly avoids collecting or storing content-level information, user identities, message details, or exact timestamps. This design ensures that sensitive user data is never persisted in mutable storage.

Upon receiving aggregated audit metrics from the SDK, the auditing software generates **cryptographic hashes** that represent the current state of system activity. These hashes act as compact, irreversible proofs of observed behavior. The generated hashes are then submitted to the blockchain through the audit-specific parachain, where they are permanently recorded as immutable audit proofs.

By eliminating traditional database storage, the auditing software reduces the attack surface, prevents insider manipulation, and ensures that audit integrity is enforced exclusively through blockchain immutability. The software also supports verification operations by recomputing hashes from incoming audit summaries and comparing them against blockchain records.

This approach prioritizes **integrity, accountability, and privacy**, making the auditing software suitable for environments where trust minimization and data protection are critical.

Req Id	**Requirement**	**Priority**
4.3.1	The auditing software shall receive audit events from the Auditing SDK	High
4.3.2	The auditing software shall process audit data without collecting content-level information	High
4.3.3	The auditing software shall avoid storing audit data in a traditional database	High
4.3.4	The auditing software shall generate cryptographic hashes from aggregated audit metrics	High
4.3.5	The auditing software shall submit audit hashes to the audit-specific parachain	High
4.3.6	The auditing software shall support integrity verification using blockchain records	High
4.3.7	The auditing software shall ensure that no personally identifiable information is processed or stored	High

4.3.8	The auditing software shall detect and report inconsistencies between computed hashes and blockchain entries	Medium

4.4 Attack Detection and Prevention Module

The Attack Detection and Prevention Module is responsible for identifying suspicious, unauthorized, or policy-violating activities occurring within the application. Instead of relying on detailed per-user audit logs, the module operates on **aggregated backend activity metrics** to preserve user privacy while ensuring system accountability.

The module continuously monitors security-relevant events such as authentication attempts, access to protected resources, and abnormal request patterns. These events are analyzed against predefined thresholds and policies to detect potential attack scenarios, including brute-force login attempts, excessive request generation, and unauthorized access behavior.

When suspicious activity is detected, the system records **aggregated security indicators directly on the blockchain**, ensuring that evidence of abnormal behavior is immutably preserved. This design guarantees that even during active attack scenarios—such as denial-of-service or credential-stuffing attacks—security-related evidence remains tamper-proof and verifiable.

4.4.1 Attack Detection Strategy

The system employs a **rule-based detection approach** based on threshold evaluation and time-window analysis. Rather than identifying individual users, the system evaluates **patterns of activity at the system level**, such as unusually high frequencies of failed authentication attempts or abnormal spikes in sensitive operations.

This approach balances security monitoring with privacy preservation, ensuring that detection mechanisms do not expose personally identifiable information.

4.4.1 Attack Detection Scenarios

Attack Type	**Detection Indicator**	**Detection Method**
Brute-force authentication	Excessive failed login attempts within a short time window	Threshold-based aggregation
Denial-of-Service (DoS)	Sudden spike in request	Rate-based aggregation

	volume	
Unauthorized access	Repeated access attempts to protected resources	Pattern frequency analysis
Abuse of system functions	Excessive execution of sensitive operations	Time-windowed count analysis

4.5 Frontend Interface

The Frontend Interface provides a **read-only, visualization-focused view** of the auditing system for authorized users such as administrators and auditors. Its primary purpose is to display **aggregated audit metrics** and **verification results** without exposing any sensitive or content-level user information.

The interface retrieves audit summaries and integrity proofs directly from the blockchain and the auditing software, presenting them in an understandable and structured manner. It does not allow creation, modification, or deletion of audit records, thereby preserving the integrity of the auditing process.

The frontend emphasizes **transparency and verification**, allowing users to observe system activity patterns, verify the authenticity of audit data, and identify abnormal behavior trends. All displayed information is limited to quantitative metrics, such as activity counts and verification status, in accordance with the system's privacy-preserving design.

Role-based access control ensures that only authorized users can access audit visualizations, while preventing misuse or unauthorized data exposure.

Req Id	**Requirement**	**Priority**
4.5.1	The frontend shall provide a dashboard for viewing aggregated audit metrics	High
4.5.2	The frontend shall display audit verification status based on blockchain records	High
4.5.3	The frontend shall retrieve audit data without exposing	High

	content-level or user-identifying information	
4.5.4	The frontend shall operate in read-only mode for audit records	High
4.5.5	The frontend shall support role-based access control for authorized users	High
4.5.6	The frontend shall visualize trends and abnormal activity patterns using aggregated data	Medium
4.5.7	The frontend shall prevent any modification or deletion of audit data	High
4.5.8	The frontend shall provide a user-friendly and responsive interface	Medium

5. Audit Chain: Parachain Design and Implementation Overview

AuditChain is a purpose-built, audit-specific blockchain implemented as a Polkadot parachain using the Substrate framework. It serves as the trust anchor of the system by providing immutable, verifiable storage for audit proofs generated by external applications. Unlike general-purpose blockchains, AuditChain is optimized exclusively for audit verification and integrity assurance rather than transactional or financial operations.

The primary objective of the AuditChain parachain is to ensure that once an audit proof is recorded, it cannot be altered, deleted, or forged—even in the presence of insider threats or compromised off-chain infrastructure. To achieve this, the parachain stores only cryptographic representations of audit activity, such as hashes, counters, and minimal metadata, instead of raw audit logs or sensitive user information. This design preserves user privacy while still enabling strong integrity guarantees.

AuditChain is built using Substrate due to its modular architecture, which allows the creation of a custom runtime tailored specifically for auditing requirements. The runtime consists of dedicated logic responsible for validating incoming audit proofs, associating them with timestamps, and committing them to the blockchain ledger. No business logic, application data, or personally identifiable information is executed or stored on-chain.

The parachain interacts with the auditing software through well-defined interfaces. Aggregated audit metrics and cryptographic hashes generated off-chain are submitted to the parachain, where they are permanently recorded. These on-chain records act as tamper-evident proofs that can later be used to verify system behavior and detect inconsistencies or manipulation attempts.

From an operational standpoint, AuditChain is deployed in a controlled academic environment using a local or test network configuration. This ensures feasibility within the scope of a mini-project while still demonstrating real-world blockchain principles such as immutability, decentralization, and trust minimization. The design intentionally avoids complex features such as smart contract execution, token economics, or cross-chain communication to maintain focus on audit integrity.

Overall, AuditChain represents a lightweight yet robust blockchain layer that enhances accountability and security in audit logging systems. By separating audit proof verification from data collection and avoiding mutable storage, the parachain establishes a reliable foundation for secure, privacy-preserving auditing.

5.1 AuditChain Parachain Components

Component	**Description**	**Implementation Approach**	**Scope**
Parachain Framework	Base framework used to build the audit blockchain	Substrate-based custom runtime	Core
Runtime Logic	Defines on-chain behavior for audit proof	Custom audit pallet	Core
Audit Pallet	Handles submission and storage of audit proofs	Stores hashes, counters, minimal metadata	Core
Data Storage Model	On-chain storage structure	Key–value storage (hash → metadata)	Core
Timestamping	Associates audit proofs with time	Block timestamp at submission	Core
Access Control	Restricts who can submit audit proofs	Authorized accounts / whitelisted origins	Core
Transaction Validation	Ensures correctness of submitted proofs	Runtime-level validation checks	Core
Immutability Guarantee	Prevents modification or deletion	Blockchain append-only ledger	Core
Query Interface	Enables retrieval of stored audit proofs	Runtime storage queries via RPC	Core

Privacy Preservation	Prevents exposure of sensitive data	No raw logs or PII stored	Core
Consensus Mechanism	Block production and finality	Default Substrate test configuration	Out of Scope

5.2 Supported Libraries and Frameworks

The AuditChain parachain is implemented using selected components from the Polkadot and Substrate ecosystem. These libraries provide the necessary infrastructure for building a secure, modular, and audit-focused blockchain while keeping the implementation feasible within an academic mini-project scope. Only core runtime and networking dependencies required for parachain functionality are included.

The selected libraries support custom runtime logic, secure storage, transaction validation, and interaction with external systems. Advanced features such as cross-chain messaging, token economics, and production-grade consensus mechanisms are intentionally excluded.

Category	**Library/Tool**	**Purpose**
Blockchain Framework	Substrate	Base framework for building the custom audit-specific parachain
Runtime Development	FRAME (Framework for Runtime Aggregation of Modularized Entities)	Enables creation of custom runtime pallets for audit logic
Audit Logic	Custom Audit Pallet	Implements storage and validation of audit proof hashes
Cryptography	sp-core/sp-io	Provides cryptographic primitives and hashing functions
Runtime Storage	sp-storage	Manages on-chain key–value storage for audit proofs
Timestamp	Pallet-timestamp	Associates audit proofs with blockchain timestamps
Node Template	Substrate Node Template	Provides a base node for local and test network deployment
RPC Interface	JSON-RPC (Substrate)	Allows external querying and

		submission of transactions
Networking	libp2p (via Substrate)	Peer-to-peer networking between blockchain nodes

5.3 Fallback and Error Handling

The AuditChain parachain incorporates structured error handling and fault tolerance mechanisms at the runtime level to ensure that only valid and authorized audit proofs are recorded on the blockchain. Since the parachain functions as an immutable, append-only ledger, error handling focuses on prevention rather than recovery, ensuring that incorrect or malicious data never enters the chain.

All incoming audit proof submissions are validated before block inclusion. Submissions that fail validation—such as malformed hashes, unauthorized origins, or incorrect data formats—are rejected by the runtime and do not result in state changes. This prevents invalid audit records from being permanently committed.

The parachain leverages Substrate's dispatch error mechanism to provide deterministic error responses. When a transaction fails validation, a clear runtime error is returned to the submitting entity, allowing the auditing software to detect the failure and take appropriate corrective action off-chain. No partial writes or inconsistent states are possible due to the atomic execution model of Substrate runtime calls.

Fault tolerance in the AuditChain parachain is achieved through blockchain-level redundancy and deterministic execution rather than traditional rollback or backup mechanisms. Because audit proofs are immutable once stored, there is no requirement for data modification or deletion recovery. In the event of temporary network failures or node downtime, unsubmitted audit proofs can be safely re-submitted by the auditing software without risking duplication or corruption of existing records.

The system intentionally avoids complex recovery logic such as state rollback, data migration, or on-chain correction to maintain simplicity and verifiability. This design ensures that audit integrity is preserved even under adverse conditions, while keeping the implementation feasible within an academic mini-project scope.

Error Scenario	**Handling Mechanism**	**Outcome**
Invalid audit hash format	Runtime validation failure	Transaction rejected
Unauthorized submission	Origin verification failure	Transaction rejected
Duplicate or malformed data	Validation checks	No state change
Network interruption	Off-chain retry mechanism	Safe resubmission

Node failure	Blockchain redundancy	No data loss

6. Auditing Software Module

6.1 Purpose

The Auditing Software Module is responsible for ensuring accountability, transparency, and traceability of all critical actions performed within the system. It provides a reliable mechanism to generate, store, and verify audit records in a tamper-evident manner.

6.2 Audit Event Capture Module

This sub-module is responsible for monitoring and capturing all critical system, user, and administrator activities. It ensures that every significant action performed within the system is detected and prepared for auditing.

Captured events include:

- User and administrator login/logout
- Failed authentication attempts
- Data creation, modification, and deletion
- Permission and role changes
- Session termination

All captured events are forwarded to the audit log generation process without modification.

6.3 Audit Log Generation Module

This sub-module converts captured events into structured and standardized audit records. Each audit log entry is enriched with metadata to ensure traceability and accountability.

Audit log attributes include:

- Unique event ID
- Event type and description

- Associated user or wallet identifier
- Timestamp
- Source context (IP address, device metadata)

Once generated, audit logs are treated as immutable records.

6.4 Audit Storage and Blockchain Anchoring Module

This sub-module handles secure storage and integrity protection of audit logs.audit records are aggregated which are to be stored on-chain in AuditChain with cryptographic hashes of each record which will be verified in cases of anomaly detections as well as to check whether the audits has been changed.

Responsibilities:

- Securely store audit logs on-chain.
- verify cryptographic hashes for each audit entry in cases of anomalies.
- Anchor hashes with timestamps on the blockchain

6.5 Audit Verification and Review Module

This sub-module enables verification, inspection, and analysis of audit records.

Functions include:

- Retrieval of audit logs for review
- Integrity verification using hash comparison
- Detection of tampering or unauthorized modifications
- Support for forensic analysis and compliance audits

7. Non-Functional Requirements

7.1 Performance

Req ID	Requirement	Target
7.1.1	Log ingestion latency for single events (API to DB commit).	≤ 500 ms under normal load
7.1.2	Anchoring latency from log write to on-chain confirmation (configurable).	≤ 5 minutes (default)
7.1.3	Integrity verification time for a typical session or day range.	≤ 60 seconds

7.2 Reliability & Fault Tolerance

- The system must never expose unhandled errors to client applications; all failures are handled with appropriate status codes and logging.
- Log ingestion must tolerate transient database or blockchain outages by queueing and retrying where possible.
- System components should support horizontal scaling to handle increased log volume.

7.3 Security & Compliance

- All communication channels must use TLS and enforce authentication and authorization controls.
- Administrative and verification operations must be fully auditable.
- The system should be designed to support common compliance regimes (e.g., ISO 27001, SOC 2) regarding logging and data retention.

7.4 Scalability

- The architecture must support increasing log volume (e.g., from thousands to millions of events per day) without redesign.
- Hashing and anchoring components must be horizontally scalable and able to batch work efficiently.

7.5 Usability

- The onboarding configuration for a new application (API key, schema registration) should be completable within 15 minutes.
- Dashboards must present clear indicators of integrity status, anchoring health, and alert summaries.

8.Constraints and Non-goals

ID	Constraint / Non-Goal	Rationale
8.1	Must not store full audit logs directly on the blockchain.	Cost and privacy; only aggregated audit log data are anchored.
8.2	Must not rely on AI outputs for any security decision or integrity verdict.	AI is non-authoritative and may hallucinate.
8.3	Must not allow arbitrary deletion of audit logs via standard APIs.	Protects accountability and forensic value.
8.4	Must not expose raw blockchain private keys via any UI or API.	Prevents key compromise.
8.5	No cross-organization log sharing or multi-tenant data mixing in v1.0.	Limits complexity and regulatory scope.

9.Success Criteria and Evaluation Metrics

ID	Metric	Target	Measurement Method

9.1	Tamper Detection	100% detection of any modified anchored log (no false negatives).	Controlled tests with modified logs.
9.2	Integrity Adoption	≥ 90% of critical events anchored within configured latency.	Anchoring and log coverage statistics.
9.3	Operational Overhead	≤ 10% overhead on existing application response times due to logging.	Performance benchmarking.
9.4	Analyst Productivity	≥ 30% reduction in time to reconstruct an incident vs. baseline tools.	User studies and time-to-resolution metrics.

10. Risks & Mitigations

ID	Risk	Severity	Mitigation
10.1	Compromised administrators attempt to alter database records.	Critical	Blockchain anchoring ensures changes are detectable via hash mismatch; strong RBAC and MFA.
10.2	Blockchain network congestion leads to delayed anchoring.	High	Batching, multiple nodes, configurable anchoring intervals, and local backlog with clear status indicators.
10.3	High-volume attacks overload log ingestion.	High	Rate limiting, backpressure mechanisms, priority for critical events, and attack-aware pause/resume logic.
10.4	Hash algorithm becomes vulnerable over time.	Medium	Support pluggable hash algorithms and migration strategies.

11. Appendices

Appendix A: Audit Log JSON Schema

```
{
  "aggregation_window_start": "ISO-8601",
  "aggregation_window_end": "ISO-8601",
  "event_type": "AUTH|ACCESS|SYSTEM",
  "event_count": "integer",
  "severity_level": "LOW|MEDIUM|HIGH",
  "threshold_exceeded": "boolean",
  "record_timestamp": "ISO-8601"
}
```

Appendix B: Agent Interaction Diagram

Source Agent	**Target Agent**	**Payload**	**Trigger**
User / Admin	Activity Monitoring Module	Event Signal (Non-identifiable)	User Action
Activity Monitoring Module	Aggregation Engine	Incremented Counters / Metrics	Event Detected
Aggregation Engine	Attack Detection Module	Aggregated Activity Metrics	Time Window Expiry / Threshold Check
Attack Detection Module	Blockchain Module	Aggregated Audit Record (Counts,	Threshold Met or Periodic Commit

		Severity, Timestamp)	
Blockchain Module	Blockchain Ledger	Immutable On-Chain Record	Successful Transaction
Auditor	Blockchain Ledger	Read-Only Query (Aggregated Data)	Manual Audit Request
Blockchain Ledger	Auditor	Immutable Audit Evidence	Verification Request

Appendix C: Attack Detection Threshold Configuration

Metric	**Threshold**	**Time Window**	**Action**
Failed login attempts	>50	5 minutes	Record HIGH severity aggregate
Request rate	>1000	1 minute	Record MEDIUM severity aggregate
Sensitive access attempts	>20	10 minutes	Record HIGH severity aggregate

Appendix D:Smart Contract Storage Structure

```
struct AuditAggregate {
  uint256 windowStart;
  uint256 windowEnd;
  uint256 eventCount;
  uint8 severityLevel;
  bool thresholdExceeded;
}
```

End of Document - SRS v1.0
Blockchain-based Audit Log System