CISSAN

Collective intelligence supported by security aware nodes

D4.2 Network logging system

Editor: Alberto Doval (Councilbox Technology)

Contact: alberto.doval@councilbox.com

Abstract

This report presents the architecture and operational framework of a dual-blockchain logging system specifically engineered for IoT environments. This system achieves a significant advancement in securing and managing IoT event data. It leverages blockchain's immutable ledger capabilities to ensure the authenticity, integrity, and verifiability of event histories generated by IoT devices. The system includes a master node for centralized processing and hub nodes for distributed data handling, ensuring scalability, verifiability, and resilience. Each event is timestamped on established blockchains like Bitcoin, adding an extra layer of security through external validation. The document provides insights into the system's design principles, operational workflow, and the strategic application of blockchain technology to resolving key challenges in IoT data management.

.

Participants in project CISSAN are:

- University of Jyväskylä
- Bittium Wireless Ltd
- Bittium Biosignals Ltd
- Geodata ZT GmbH
- Mattersoft
- Mint Security Ltd
- Netox Ltd
- Nodeon Ltd
- Scopesensor Ltd
- Wirepas Ltd
- Councilbox Ltd
- Affärsverken Karlskrona AB
- Arctos Labs
- Clavister AB
- Blekinge Tekniska Högskolan
- Blue Science Park
- Savantic AB
- Techinova AB

CISSAN – Collective intelligence supported by security aware nodes

D4.2 Network logging system

Editor: Alberto Doval (Councilbox Technology)

Project coordinator: Alexey Kirichenko (University of Jyväskylä)

CELTIC published project result

Executive Summary

This report describes the dual-blockchain logging system which is a CISSAN solution representing a pioneering approach to securing and managing data in IoT ecosystems. This system addresses critical challenges faced by current IoT deployments, including data integrity, security, and efficient data management, through a robust blockchain-based architecture.

- **Core Components**: The system integrates IoT devices, hub nodes for localized processing, and a master node for global blockchain management. It leverages blockchain's immutable nature for data logging.
- **Security Enhancements**: By utilizing blockchain, the system ensures that once an event is logged, it cannot be altered, providing a tamper-proof record. Secure communication protocols and device authentication mechanisms further fortify the network against breaches.
- **Operational Efficiency**: The architecture allows for scalable data handling, with hubs processing data locally before syncing with a central ledger. This distributed approach reduces latency and enhances system resilience.
- **Data Integrity and Verification**: Each block of IoT events is timestamped onto external, widely trusted blockchains, providing an indisputable timestamp and additional layer of security.
- **Strategic Implications**: This system positions organizations to leverage IoT data with confidence, reducing risks associated with data tampering and ensuring compliance with regulatory standards. It fosters trust among stakeholders by offering transparent yet secure data access.

The report particularly describes the objective, architecture, data and users of the system, and provides an example workflow and the relevant implementation details.

The implementation of this system is not just a technological advancement but a strategic move towards enhancing operational security, improving data-driven decision making, and compliance in IoT applications across various industries.

The system combines blockchain-based techniques with event anomaly detection methods. Given that several CISSAN partners use anomaly detection in their security solutions and mechanisms, this opens collaboration and integration opportunities to be explored further in the project.

It is important to note that the plan is to make the complete source code and deployment instructions for the system available on GitHub with the first production release.

List of Authors

In alphabetic order by partner name:

- Alberto Doval, Councilbox Technology, Spain
- Rodrigo Martinez, Councilbox Technology, Spain
- Martin Rubio, Councilbox Technology, Spain

Table of Contents

List of Figures

1 Introduction

This document outlines the design of a dual-blockchain system tailored for logging events in IoT settings. It leverages blockchain technology to ensure consensus on the sequence and validity of IoT events, thereby creating an indisputable ledger.

Key to this system is its use of advanced timestamping methods, where events are logged onto well-established blockchains like Bitcoin, enhancing both the security and the credibility of the event records. This method leverages the inherent immutability of blockchain, coupled with the high-security standards of leading blockchain networks, to safeguard data.

The system transcends traditional data collection by actively ensuring the integrity, authenticity, and security of IoT event data. This not only improves the management and security of data in IoT networks but also marks a progressive step in how data is processed and secured in IoT environments.

1.1 Objective of this document

The primary objective of this document is to provide a comprehensive overview of the dual-blockchain logging system designed for IoT environments. This document aims to:

- **Detail the Architecture**: Outline the structural components, including the Device Registry, Hub Nodes, Master Node, and their interactions, highlighting how they collectively ensure data integrity and security through blockchain technology.
- **Explain the Functionality**: Describe how the system processes, logs, and secures IoT event data, focusing on blockchain consensus mechanisms and timestamping on established blockchains like Bitcoin.
- **Highlight Security Measures**: Illustrate the encryption and secure transmission protocols that safeguard data from device to blockchain.
- **Define Data Handling**: Specify the standardized format for event data and how it is managed, transformed, and validated through the network, ensuring consistency and reliability.
- **Provide Workflow Examples**: Offer practical examples of how events are logged, processed, and verified, to aid in understanding the system's workflow.

1.2 Content and structure of this document

To achieve the above objective, the document is structured to facilitate a thorough understanding of the dual-blockchain logging system, guiding readers through its conceptual framework, technical architecture, operational workflows, and security measures. Here's an overview of its content and structure:

- **Introduction**
  - Objective of this document
  - Content and structure of this document
- **Network Logging System**
  - Overview of the system's purpose and benefits within IoT ecosystems.
  - Detailed explanation of the system's objectives in managing and securing IoT event data.
- **System Architecture**
  - **Device Registry**: Description of the central repository for device management and authentication.
  - **Hub Node**: Role in data processing, and blockchain integration.
  - **Master Node**: Central processing and block minting.

  - **Blockchain Timestamping**: Enhancing security by timestamping on external blockchains.

- **System Data**
  - Description of how IoT devices convert sensor data into events.
  - Standardized event format for data consistency and processing efficiency.
  - Explanation of how the system outputs blockchain data.
- **System Users**
  - Roles and responsibilities of different user types within the system.
- **Example Workflow**
  - Walkthroughs and visual aids to illustrate typical and complex system operations.

Each section is designed to progressively build upon the reader's understanding, from high-level concepts down to specific technical implementations, ensuring clarity on how the dual-blockchain logging system operates to secure and manage IoT data effectively.

2 Network logging system

The dual-blockchain logging system is designed to enhance data logging within IoT ecosystems. Its primary purpose is to provide a secure, verifiable, and immutable record of events generated by IoT devices. By integrating blockchain technology, this system establishes a consensus mechanism that ensures all recorded data is agreed upon by multiple nodes, thereby enhancing the trustworthiness and integrity of the data.

This network logging system not only addresses the current challenges in IoT data management but also positions organizations to leverage IoT data fully, securely, and efficiently, driving innovation and operational excellence across various industries.

2.1 System objective

The primary objectives of a network logging system within an IoT ecosystem revolve around ensuring data integrity, enhancing security, and providing robust data management capabilities. Here's a detailed look into these objectives:

Data Integrity and Verifiability:

- **Immutable Logs**: By integrating blockchain technology, the system ensures that once data is logged, it cannot be altered. This immutability is crucial for maintaining trust and verifiability, essential for applications where data authenticity is paramount.
- **Long-term Data Preservation**: Blockchain's inherent properties make it an ideal solution for long-term data preservation. The data logged today can be reliably accessed and verified in the future, important for long-term monitoring or legal purposes.
- **Consensus Mechanisms**: Employing consensus protocols ensures that all participants in the network agree on the logged data, reducing the risk of fraud or data tampering.

Enhanced Security Measures:

- **Secure Transmission**: The system employs common encryption and secure communication protocols like WPA2-Enterprise (for Wi-Fi connections) and MQTT over TLS to protect data during transmission from IoT devices to the logging system.
- **Device Authentication**: Through the Device Registry, only authorized devices can log data, preventing unauthorized access or data injection from rogue nodes.

Data Management:

- **Event Standardization**: Implementing a standardized format for event data ensures consistency across different IoT devices, facilitating easier analysis and integration with external systems.
- **Data Handling**: The system is optimized for handling large volumes of IoT data efficiently, from collection at the device level to aggregation and storage on the blockchain.

Real-time Monitoring and Compliance:

- **Real-time Insights**: The system provides real-time visibility into network health and event occurrences, which is crucial for operational efficiency and immediate response to issues.
- **Audit Trails**: Immutable logs serve as perfect audit trails, helping in compliance with various regulatory frameworks by offering transparent, verifiable records of all operations.

Reliability and Integration:

- **Redundancy**: With multiple Hub nodes, the system reduces single points of failure, enhancing reliability.
- **Reduction in Data Discrepancies**: With a unified logging system, discrepancies in data across different systems or devices are minimized, ensuring that data analysis and decision-making processes are based on consistent and accurate information.
- **External Blockchain Integration**: Utilizing existing blockchains for timestamping adds an additional layer of security and trust, leveraging established infrastructures.

User Accessibility and Transparency:

- **Blockchain Explorer**: Allows stakeholders to verify transactions without exposing sensitive data, maintaining privacy while ensuring transparency.
- **Stakeholder Engagement**: By providing clear, verifiable logs, the system fosters trust among users, regulators, and partners, crucial for broader adoption and cooperation.

This network logging system, therefore, not only addresses the immediate security and operational needs of IoT deployments but also sets a foundation for a secure data management across diverse IoT applications, aligning with both strategic business goals and regulatory compliance standards.

2.2 System architecture

The architecture of the dual-blockchain logging system within the CISSAN framework is designed to manage IoT events with high security, precision, and integrity. Each component plays a critical role in ensuring data is captured, processed, and stored effectively. The system architecture is composed of the following components:

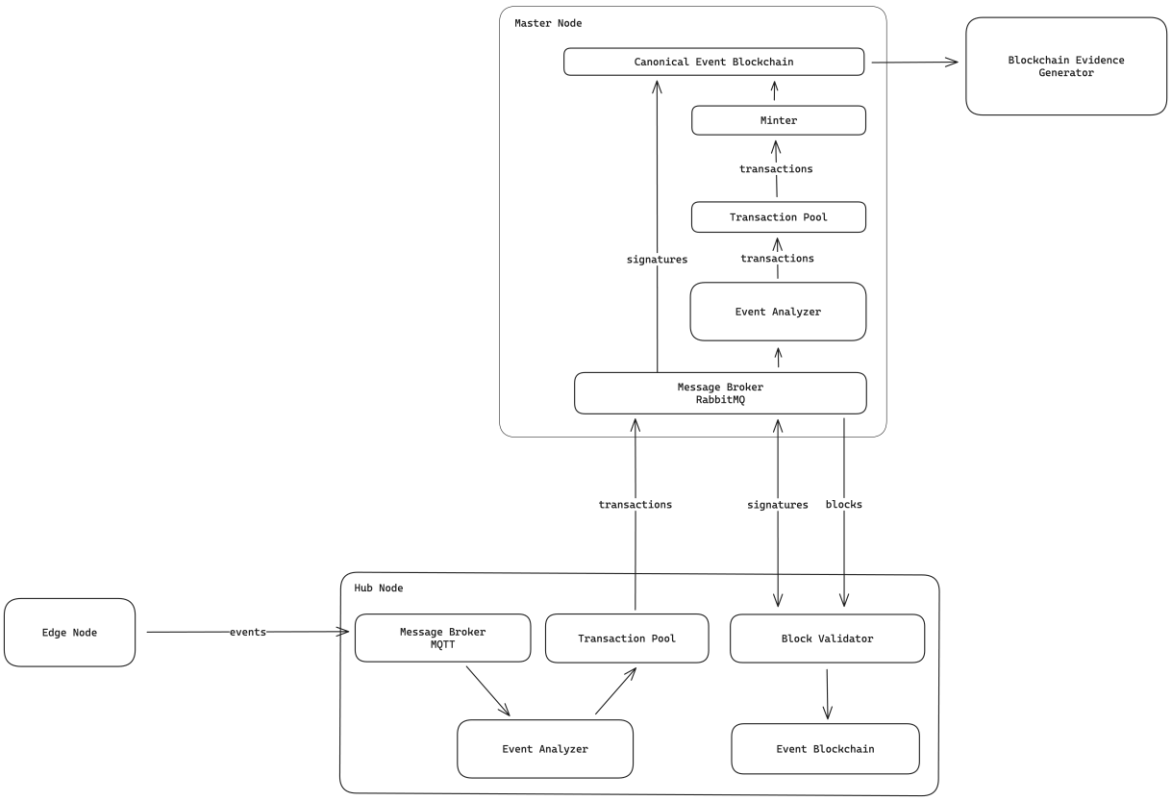

Figure 1. System architecture of the network logging system.

2.2.1 Device Registry

The Device Registry serves as the central repository where all IoT devices are registered before they can participate in the network. It's crucial for authentication and device management. It assigns unique identifiers (device_uid) and device tokens. This ensures that only authorized devices can connect and communicate within the network.

2.2.2 Hub Node

Every hub node acts as a central processing unit for data from connected IoT devices. It processes incoming events and integrates data into the blockchain framework.

2.2.2.1 MQTT Broker

The MQTT broker Facilitates secure and efficient communication between IoT devices and hub nodes using the MQTT protocol. It ensures that only authenticated devices can communicate, enhancing network security.

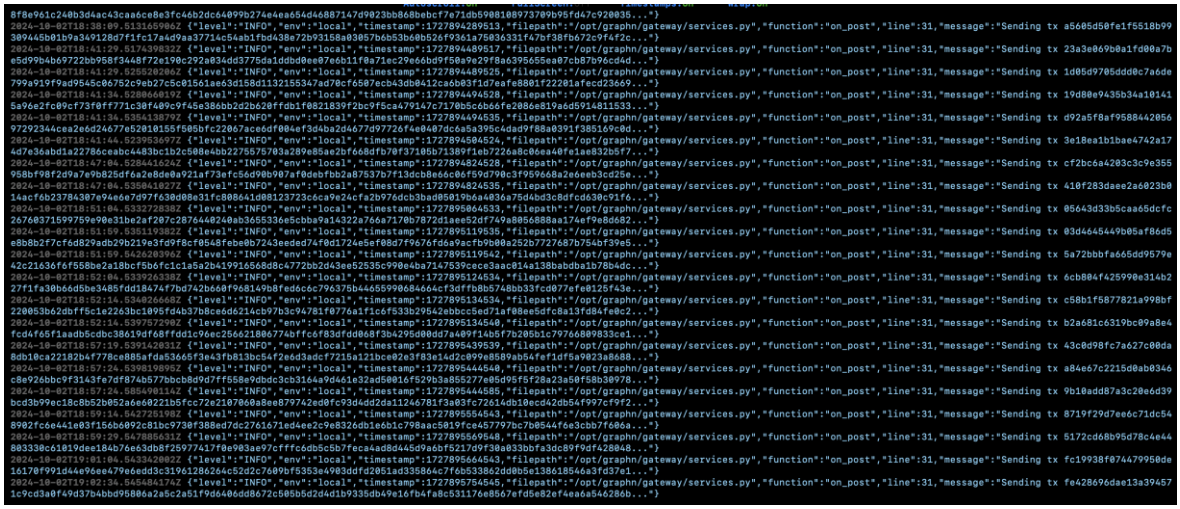

Figure 2. Logs of the Hub MQTT Broker.

2.2.2.2 Transaction Pool

The Transaction Pool acts as a crucial intermediary layer, tasked with collecting processed events before they are sent to the Master node via its message broker. In this stage, events are transformed into signed transactions.

2.2.2.3 Block Validator

The Block Validator evaluates transactions by confirming that blocks received from the Master Node accurately reflect the data the hub node submitted. This ensures the blockchain's integrity at the hub level.

2.2.2.4 Event Blockchain

The Hub Event Blockchain represents a localized version of the event ledger, meticulously reconstructed and maintained by each individual hub node. This decentralized approach ensures that every hub possesses an up-to-date and accurate copy of the blockchain, reflecting all transactions as they are validated. By maintaining this independent yet synchronized version, the system enhances its resilience and data integrity, allowing for robust verification processes and a unified view of the event history across the network.

2.2.3 Master Node

The Master Node acts as the central hub for processing event transactions received from all connected hub nodes, meticulously validating each transaction for consistency and accuracy. Following validation, these transactions are methodically grouped and incorporated into new blocks at regular intervals. These newly formed blocks are then distributed to the hub nodes for further validation. This cyclical process ensures that the integrity and reliability of the blockchain are upheld across the network, facilitating a cohesive and secure data ecosystem.

2.2.3.1 Message Broker

The Master Node employs a RabbitMQ message broker to facilitate the seamless exchange of data between itself and the hub nodes. This broker serves a dual purpose: it efficiently handles incoming transactions from the hub nodes, ensuring their timely receipt and processing. Concurrently, it is responsible for dispatching newly minted blocks back to the hub nodes. It also gathers and distributes block validations among the hub nodes. This bi-directional communication system is crucial for maintaining the synchronization of blockchain data across the network, ensuring that all nodes operate with the latest and most accurate information.

2.2.3.2 Transaction Pool

The Master Transaction Pool functions as a provisional storage, aggregating transactions until they are ready to be compiled into a new block. This is crucial for organizing and sequencing the transactions efficiently, ensuring that the data is systematically processed and prepared for the next phase of block creation. By managing transactions in this manner, the system ensures a smooth transition from transaction receipt to block formation, maintaining the orderly flow of data through the blockchain minting process.

2.2.3.3 Blockchain Minter

The Blockchain Minter operates on a predetermined schedule to gather transactions from the Master Transaction Pool and incorporate them into new blocks. Each block, once assembled, is authenticated with the digital signature of the Master Node, signifying its validity and readiness for distribution. These signed blocks are then relayed to the hub nodes via the Master Node's message broker, ensuring that the updated blockchain information is consistently propagated throughout the network. This process not only secures the integrity of the blockchain but also facilitates the timely update of the ledger across all nodes, maintaining the system's overall coherence and reliability.

2.2.3.4 Event Blockchain

The Master Event Blockchain stands as the authoritative ledger within the system, embodying the canonical sequence of validated transactions. This blockchain is meticulously reconstructed and verified by each hub node, ensuring that every participant in the network maintains an identical and accurate copy of the ledger. This process of continuous synchronization and validation across the hub nodes safeguards the blockchain's integrity, guaranteeing that the data it contains is universally recognized and trusted within the system. The uniformity of the Master Event Blockchain across all nodes is essential for upholding the security and consistency of the network's transaction history.

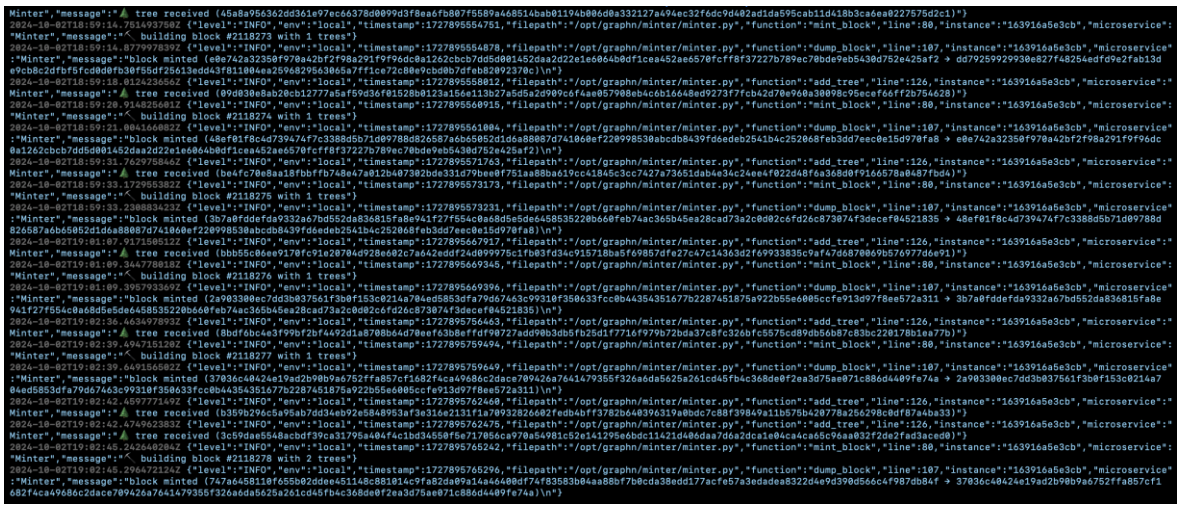

Figure 3. Logs of the Blockchain Minter.

2.2.4 Event Blockchain Timestamping:

To enhance the security and verifiability of the event blockchain, each block is timestamped on a reputable blockchain, such as Bitcoin. This timestamping process serves as an additional layer of validation, embedding an immutable record of the block's creation time within a globally recognized and secure blockchain infrastructure. This method not only provides a tamper-evident seal on the event data but also leverages the widespread trust and robustness of established blockchain networks. Timestamping on such a platform ensures that the event blockchain's integrity is beyond reproach, offering an indisputable chronology of events that can be relied upon for accuracy and authenticity.

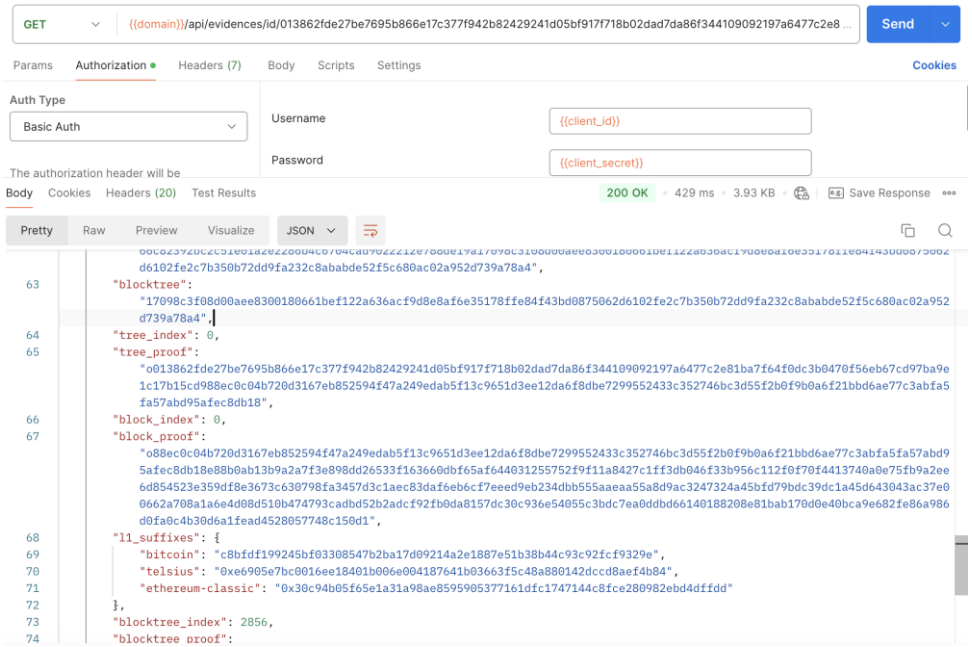

Figure 4. Blockchain evidence of an event.

2.3 System data

2.3.1 Input data

IoT devices are equipped with various sensors designed to measure different environmental or operational parameters such as temperature, humidity, light intensity, motion, or any other relevant physical quantities. Here's how the conversion process unfolds:

1. **Sensor Data Collection**: Each IoT device continuously collects raw sensor data. This data varies based on the sensor type but is typically in the form of electrical signals or digital values representing the physical state or event captured.
2. **Data Processing**: On-device microcontrollers or processors interpret these raw sensor readings. This might involve converting analog signals to digital, filtering noise, or applying calibration to ensure accuracy.
3. **Event Formation**: The processed data is then structured into an "event." An event encapsulates this processed sensor data within a specific format that includes metadata like timestamps and device identifiers. This step converts raw sensor data into meaningful, actionable information.
4. **Event Transmission**: Once formatted, these events are transmitted to the nearest hub node through the network. The transmission is governed by protocols ensuring security and reliability.

To ensure interoperability, streamline processing, and maintain consistency across the network, a standardized event format is crucial. Here's the format used:

```
{
    "uid": "unique_event_identifier",
    "timestamp": 1696248400000,
    "items": [
        {
            "key": "temperature",
            "value": 23.5
        },
        {
            "key": "humidity",
            "value": 55.6
        }
    ],
    "namespace": "organization/device_group",
    "device_uid": "device_001"
}
```

- **UID**: A unique identifier for the event, ensuring no two events have the same ID. This helps in tracking and logging events distinctly.
- **Timestamp**: The time when the event was generated, in Unix time format with millisecond precision.
- **Items**: An array where each object contains:
    - **Key**: A string describing the type of data, e.g., "*temperature*", "*motion_detected*".
    - **Value**: The actual data value, which could be a number, string, or even an array for complex data points.

- **Namespace**: Used to categorize events from the same type or organization.
- **Device UID**: Identifies the specific IoT device sending the event, crucial for tracking and maintenance.

2.3.2 Output data

The output data from the dual-blockchain logging system primarily comprises the blockchain data itself, along with associated verification mechanisms. Here's a detailed look at what constitutes the output:

Blockchain Data:

- **Event Transactions**: Each event from IoT devices, once processed, becomes a transaction within the blockchain. This transaction includes the event details, device identifiers, and timestamps.
- **Blocks**: Composed of multiple transactions, blocks are linked in a chain, ensuring that data is stored in a linear, verifiable sequence. Each block contains:
    - A list of transactions.
    - A timestamp when the block was created.
    - A cryptographic hash of the previous block, ensuring immutability.
    - Metadata like the block number, minter signature, etc.

External Blockchain Timestamps:

- For each block in the Master Event Blockchain, a transaction is made on an external blockchain to timestamp the block. This transaction includes:
    - The hash of the block in our system's blockchain.
    - A timestamp from the external blockchain, providing an indisputable record of when the block was created in our system.

Verification Components:

- **Merkle Trees**: Used within each block to efficiently and securely verify the integrity of all transactions within a block. Each transaction hash contributes to a Merkle root, which is stored in the block header.
- **Proof of Inclusion**: For any specific event, a Merkle proof can be generated to prove its inclusion in a block, without revealing other transactions. In the same way, for each block, a Merkle proof can be generated to prove its inclusion in an external blockchain.

2.4 System users

In the dual-blockchain logging system designed for IoT environments, various user types play distinct roles to ensure the system operates smoothly, securely, and efficiently. Here's a breakdown of the roles and responsibilities:

2.4.1 Device Registry Operator

- **Role**: Oversees the management and security of the central repository for IoT device authentication and management.
- **Responsibilities**:
    - **Device Onboarding**: Manage the registration of new IoT devices, ensuring each device is assigned unique identifiers (*device_uid*) and credentials.
    - **Maintenance**: Update device information, manage device status (active/inactive), and handle revocation of device access when necessary.

2.4.2 Master Node Operator

- **Role**: Responsible for the operation of the Master Node, which is crucial for the integrity of the entire network.
- **Responsibilities**:
  - Oversee the creation, validation, and distribution of new blockchain blocks.
  - Manage the transaction pool and ensure the timely minting of blocks.
  - Monitor and maintain the Master Event Blockchain and its timestamping process on external blockchains.
  - Ensure high availability and performance of the Master Node to prevent network disruptions.

2.4.3 Hub Node Operator

- **Role**: Manages individual Hub Nodes, which are the gateways for IoT devices.
- **Responsibilities**:
  - Validate and relay transactions to the Master Node.
  - Maintain local blockchain copies and validate received blocks.
  - Ensure the security and connectivity of IoT devices within their jurisdiction

3 Example work flow

Typical Workflow:

1. **Event Generation:**
    - An IoT device detects a change in sensor data (e.g., temperature rise).
    - The device formats this into an event according to the standardized format.

```
{
    "uid": "7488f8aa-97c7-4c4f-a546-f06a97bb0c63",
    "timestamp": 1696248400000,
    "items": [{"key": "temperature", "value": 25.3}],
    "namespace": "councilbox/weather-stations",
    "device_uid": "weather_station_001"
}
```

2. **Event Transmission:**
    - The event is transmitted securely to a Hub Node via the MQTT broker.

3. **Hub Node Processing:**
    - **Transaction Pool**: The event is processed and added to the transaction pool.

4. **Master Node Interaction:**
    - The event is sent to the Master Node via the RabbitMQ broker.

5. **Master Node Processing:**
    - **Transaction Pool**: The event is validated and added to the transaction pool.
    - **Blockchain Minter**: At regular intervals, new blocks are created.

6. **Block Timestamping:**
    - The new block is timestamped in an external blockchain network.

7. **Block Validation:**
    - The block is retrieved via the RabbitMQ broker.
    - The Hub Node validates the known events included in the block and adds it to the local blockchain.
    - The block is validated with a digital signature and the validations are exchanged through the RabbitMQ broker.

4 Implementation details

This chapter describes the technical implementation of our blockchain-based IoT system. It covers the key software components, technologies, and design decisions that form the foundation of the platform.

4.1 Software components

The system is implemented using various software components and technologies, each chosen for their reliability, security, and performance characteristics:

4.1.1 Programming Language

Python 12 serves as the primary programming language, representing the latest stable version. It offers enhanced security features, improved performance characteristics, and comprehensive type hinting support that enables better code quality and maintainability.

4.1.2 Host

The system is intended to run on Ubuntu 24.04 LTS, a long-term support release that provides the necessary stability foundation. This version ensures consistent security updates and patches while maintaining compatibility with our technology stack over an extended support period. However, the system is distributed as a set of Docker containers, so it is likely run well on any other GNU/Linux distribution. Docker provides the containerization layer, following industry standards for application packaging and deployment. It enables consistent environments across development and production, while ensuring proper isolation between system components.

4.1.3 Databases

RocksDB serves as the primary storage solution for blockchain data, optimized specifically for high-performance block storage and retrieval operations. It implements an LSM tree-based storage engine that excels at write-heavy workloads, while providing efficient compression to minimize storage footprint. The database supports atomic batch operations ensuring data consistency, and offers point-in-time snapshots for backup and recovery purposes. Its selection is justified by superior write performance and optimizations for sequential access patterns that are typical in blockchain operations.

SQLite complements the primary database by providing robust querying and data analytics capabilities. It features advanced indexing mechanisms for fast data retrieval, comprehensive full-text search functionality for event content analysis, and native JSON support enabling flexible event data storage. The rich querying capabilities make it ideal for analytics workloads. This database choice efficiently handles complex data relationships and analytical queries that would be inefficient to implement in a key-value store like RocksDB.

4.1.4 Communication

Message Broker (RabbitMQ)

Each node (Hub and Master) utilizes an internal RabbitMQ message broker for inter-container communication within the node. The Master Node additionally operates a second RabbitMQ broker dedicated to external communication, specifically for broadcasting blocks and block validations across the network. The internal brokers provide reliable message delivery through robust queueing mechanisms and support the publisher/subscriber pattern for flexible communication between containers. The external broker on the Master Node ensures reliable block distribution and validation message propagation across all nodes in the network.

IoT Communication (MQTT)

For IoT device communication, we utilize the MQTT protocol implemented through the Mosquitto broker. This lightweight and efficient message handling system is used through the Paho-MQTT Python client.

Application Server (uWSGI)

The uWSGI application server manages our Python web applications with sophisticated process management capabilities and internal load balancing. It optimizes memory usage through various strategies and operates in Emperor mode to efficiently manage multiple applications. This provides robust process supervision and automatic application recovery in case of failures.

4.1.5 Libraries and Frameworks

REST API Framework - Falcon

Falcon is our chosen REST API framework, selected for its exceptional high-performance characteristics, minimalist design philosophy, excellent request handling capabilities, and seamless middleware integration options that allow for flexible customization of the request/response cycle.

MQTT Client Library - Paho-MQTT

Paho-MQTT serves as our MQTT client library, delivering both asynchronous and synchronous APIs for flexible implementation approaches. It provides comprehensive Quality of Service level support, robust SSL/TLS encryption for secure communications, and intelligent automatic reconnection handling to maintain stable device connections.

RabbitMQ Client Library - Pika

Pika is implemented as our RabbitMQ client library, featuring robust connection management capabilities, efficient channel multiplexing for optimal resource utilization, reliable publisher confirms to ensure message delivery, and consumer acknowledgments for guaranteed message processing.

Data Serialization Library - MessagePack

MessagePack handles our data serialization needs with its compact binary format that minimizes network bandwidth and storage requirements. It provides extremely fast serialization and deserialization operations, schema-less encoding for flexibility, and broad language interoperability for diverse system integration.

RocksDB Interface - python-rocksdb

Python-rocksdb provides native Python bindings to RocksDB, delivering native performance for database operations, support for column families to organize related data, efficient batch operations for bulk processing, and comprehensive iterators and snapshots functionality for consistent data access.

SQLite Interface - sqlite3

The sqlite3 library serves as our SQLite database interface, enabling standard SQL database operations, reliable transaction support for data integrity, connection pooling for optimal resource management, and parameterized queries to prevent SQL injection vulnerabilities.

Bitcoin Integration - Bit

Bit handles our Bitcoin network integration requirements by providing comprehensive transaction creation and signing capabilities, reliable network communication functions, thorough block verification mechanisms, and complete wallet management features.

Cryptography Library

The Cryptography library manages our digital signatures and encryption needs, offering robust RSA and ECC support for asymmetric cryptography, AES encryption for symmetric operations, essential hash functions, and complete certificate handling capabilities.

Machine Learning Library - Scikit-learn

Scikit-learn powers our anomaly detection system for IoT data through different implementations such as *IsolationForest*. This unsupervised learning algorithm efficiently detects anomalies by isolating outliers in the feature space. The isolation forest approach is particularly well-suited for real-time IoT data analysis due to its low computational complexity and ability to handle high-dimensional data without requiring density estimation.

4.1.6 Component Architecture

The system is deployed as a set of interconnected Docker containers, with all components implemented in Python. This containerized architecture ensures isolation, scalability, and ease of deployment across different environments. Each container is carefully designed to handle specific aspects of the system while maintaining loose coupling through well-defined interfaces.

The Hub Node consists of several specialized containers that form the foundation of local event processing. The MQTT Message Broker container, implemented using Mosquitto broker and Paho-MQTT Python client, provides a lightweight and efficient communication protocol perfectly suited for IoT device connections. This broker can handle millions of messages per day while maintaining low latency and high reliability. The internal RabbitMQ Message Broker container facilitates communication between the node's microservices.

The Event Analyzer enriches all the incoming events with crucial metadata. It features a pluggable architecture that allows integration of analysis components such as anomaly detection, which can tag events with relevant markers based on their analysis results.

The Transaction Pool container efficiently converts validated events into blockchain transactions. The Block Validator container is a cornerstone of the system's security - every Hub Node independently validates all blocks by verifying cryptographic signatures, block content, and metadata generated by event analyzers. This distributed validation ensures the integrity and authenticity of the entire blockchain.

The Event Blockchain container provides complete node functionality with efficient data persistence using RocksDB for blockchain storage and SQLite for indexing, handling complex operations like Merkle tree management, block header processing, and chain reorganization scenarios that might occur during network partitions.

The Master Node implements the most sophisticated components through its container architecture. The internal RabbitMQ Message Broker container facilitates communication between the node's microservices, while the external RabbitMQ Message Broker container handles network-wide communication for block broadcasting and validation messages.

The Event Analyzer container utilizes the same pluggable architecture to support various analysis components that can tag events with additional context that must be consensused by the hub nodes of the network. The Transaction Pool container manages the global transaction state. The Minter container represents the heart of the blockchain system, creating the canonical blockchain of events.

The Canonical Event Blockchain container serves as the authoritative data store for the entire network using RocksDB for storing the blockchain data and SQLite for efficient indexing and querying, maintaining the complete blockchain history.

Two additional containers provide critical supporting functions for the entire system. The Device Registry container delivers enterprise-grade security through a comprehensive service that maintains a secure registry of all authorized devices, implements and manages the complete lifecycle of access tokens.

The Bitcoin Timestamping container handles the crucial task of anchoring event chain blocks to the Bitcoin network, implementing sophisticated protocols for transaction creation, network

communication, and proof verification to provide an indisputable timestamp and additional layer of security for the entire event chain.

4.2 System requirements

Minimum Requirements:

- CPU: 4 cores
- RAM: 8GB
- Storage:
  - Initial: 512GB SSD
  - Compatible with seamless volume growth
- Network: 100Mbps stable connection
- Docker Engine: 24.0 or higher
- Docker Compose: 2.0 or higher

Recommended Requirements:

- CPU: 8 cores
- RAM: 16GB
- Storage:
  - Initial: 1TB SSD
  - Compatible with seamless volume growth
- Network: 1Gbps stable connection
- Docker Engine: 24.0 or higher
- Docker Compose: 2.0 or higher

4.3 Installation and deployment

The system is distributed as a Docker-based solution, making deployment straightforward across different environments. The complete source code and deployment instructions will be available on GitHub with the first release.