Article

A Secure Rubber Supply Chain Management System Based on Hyperledger Fabric Blockchain: A Use Case in Cambodia

Ratanak Keo , Muhammad Firdaus , Kyung-hyune Rhee *

Posted Date: 28 May 2024

doi: 10.20944/preprints202405.1793.v1

Keywords: blockchain; rubber supply chain management; traceability; sustainability; security; privacy-preserving

Article

A Secure Rubber Supply Chain Management System Based on Hyperledger Fabric Blockchain: A Use Case in Cambodia

Ratanak Keo [1], **Muhammad Firdaus** [2] **and Kyung-hyune Rhee** [3,*]

[1] Department of IT Convergence and Application Engine, Pukyong National University, Busan, Republic of Korea; keoratanak@pukyong.ac.kr
[2] Department of Information Security, Pukyong National University, Busan, Republic of Korea; mfirdaus@pukyong.ac.kr
[3] Division of Computer Engineering and AI, Pukyong National University, Busan, Republic of Korea; khrhee@pknu.ac.kr
* Correspondence: khrhee@pknu.ac.kr

Abstract: Conventional traceability systems in Cambodian agriculture, such as Rubber Supply Chain Management (RSCM), often struggle to provide timely data and fair market prices, leading to mistrust among stakeholders, exploitation by intermediaries, and hindering progress towards ethical sourcing and sustainable practices. This research aims to showcase the potential of integrating blockchain technology, the Internet of Things (IoT), the InterPlanetary File System (IPFS), and Differential Privacy (DP) methods to empower Cambodian rubber farmers. By enhancing transparency, facilitating fair pricing, and promoting sustainable methods in the rubber industry, this integrated approach leverages Hyperledger Fabric (HLF) to create a decentralized and secure system for tracking rubber from its source to the end consumer. Integration with IPFS ensures secure and easily accessible data storage, further enhancing the system's reliability. Additionally, the implementation of differential privacy techniques safeguards confidential data during information sharing among farmers and supply chain stakeholders, minimizing potential privacy concerns. This comprehensive system encourages consumers to make ethical and sustainable alternative options by providing accessible information on harvesting methods, processing procedures, labor conditions, and environmental impacts. The potential benefits extend beyond Cambodia, serving as a model for other developing countries aiming to modernize and enhance the agricultural distribution networks. Our results show that transaction send rates (TPS) and throughput in systems with and without differential privacy grow with increased transaction volumes. However, the system without differential privacy achieves greater scalability and efficiency above a certain threshold. However, it offers a slightly reduced latency, reflecting a trade-off between privacy and performance efficiency in blockchain-based agricultural traceability systems.

Keywords: blockchain; rubber supply chain management; traceability; sustainability; security; privacy-preserving;

---

1. Introduction

Cambodia's Natural Rubber (NR) sector began with small-scale cultivation in the early 1900s and saw industrialization start in 1921 [1]. By the 1970s, NR covered about 60,000 hectares (ha). The sector expanded significantly to 436,682 ha across 22 provinces by 2018, with a production capacity of 220,000 tons projected to grow from 2019 to 2023. The majority of NR processing and export is composed of 78% Technically Specified Rubber (TSR) and 21% Ribbed Smoked Sheets (RSS), with the remainder being Concentrated Latex (CL) [2]. The Cambodia RSCM sector also involves 284 participants, including estate agencies, manufacturers, and cooperatives [3]. Despite lacking a rubber futures market, Cambodia relies on MRB pricing, influenced by international markets such as TOCOM, SICOM, and Shanghai. Cambodia's NR export markets include Vietnam, Malaysia, China, Singapore, South Korea, and Europe [2].

The challenges faced by Cambodia's rubber industry, including labor issues, unstable markets, and limited traceability, emphasize the need for innovative solutions. Blockchain technology, with its potential for transparency, security, and efficiency, can play a transformative role in the rubber supply chain. Hyperledger Fabric offers a scalable, flexible framework for secure transactions, while IoT can provide real-time data to improve quality control and reduce fraud. The InterPlanetary File

System (IPFS) ensures secure, decentralized data storage. However, data privacy remains a concern. Differential privacy offers a solution by protecting sensitive data while ensuring data sharing remains informative.

This study aims to develop and evaluate a blockchain-based Cambodian rubber supply chain management system using Hyperledger Fabric and differential privacy. The primary objectives of this system are to enhance traceability and visibility, promote sustainable practices, and secure sensitive business data. By achieving these objectives, the system aims to increase the competitiveness and viability of the NR industry, thereby contributing to the overall economic growth and sustainability of Cambodia.

The motivation for this research stems from the urgent need for a decentralized data-sharing and storage system to enhance sustainability and traceability in Cambodia's rubber supply chain management. Agriculture, an important part of Cambodia's economy, contributed 22% to the GDP in 2022 and employed 2.6 million people [6]. Despite its significance, the rubber industry faces challenges like worker exploitation, price unpredictability, and insufficient traceability. These issues hinder the sector's competitiveness and sustainability. Blockchain technology can address these problems by providing a transparent and secure method for tracking rubber throughout the supply chain, thus enhancing efficiency, fairness, and stakeholder confidence. We summarize the contributions of this paper as follows:

1. We leverage blockchain to enhance pricing transparency and supply chain traceability with a decentralized approach. Moreover, by establishing a secure and tamper-proof record of transactions, our framework effectively reduces fraud and corruption, fostering trust among stakeholders and facilitating ethical sourcing practices.
2. We utilize IPFS to facilitate a peer-to-peer distributed file system and provide effective off-chain storage. By leveraging IPFS, the paper ensures the protection of sensitive information and maintains data integrity, thereby enhancing the overall security framework of the system.
3. Additionally, we employ DP with the Laplace mechanism to safeguard sensitive data during the data-sharing process. This innovative approach guarantees the rigorous protection of individual privacy and data confidentiality while facilitating the sharing of crucial information among stakeholders. As a result, it significantly enhances the overall security and integrity of blockchain-based RSCM systems."

2. Background

2.1. Supply Chain Management

Supply chain management (SCM) focuses on the strategic synchronization of product information and data movements across organizations, aiming to deliver products or services to consumers efficiently. This includes coordinating operations such as raw material purchasing, manufacturing planning, logistics, inventory management, and forecasting consumer demand. Effective SCM is essential for cost savings, market responsiveness, and consumer satisfaction [9]. Organizations must balance trade-offs, like cost reduction and maintaining sufficient supply to meet demand. The complexity of global supply chains necessitates advanced SCM practices utilizing technologies like blockchain, data analytics, AI, and IoT for real-time visibility and agility [10]. Traditional SCM also emphasizes sustainability, reducing environmental impact, and ensuring transparency in purchasing processes, reflecting rising consumer awareness and regulatory demands [11].

2.1.1. Cambodia Rubber Value Chain

In Cambodia's rubber value chain, latex is processed into rubber using RSS or TSR methods. Smallholder farmers sell latex or coagulum based on factors like purchase price, processing unit proximity, and transportation availability. Numerous RSS processing facilities exist near smallholder farms, accessible within a day by motorcycle. Large landholders may sell directly to RSS facilities, exporting processed rubber to destinations such as Korea, Vietnam, China, India, and Malaysia [3].

TSR processing units also operate, sourcing latex and coagulum, with primary exports to Eastern Europe, Vietnam, China, Korea, Malaysia, and India. Smallholder farmers often coagulate latex on the farm and sell to local collectors, who transport it directly to Vietnam and Thailand, bypassing RSS or TSR processing. Supply chain management (SCM) focuses on the comprehensive synchronization of product information and data movements throughout an organization of businesses, including the primary objective of delivering products or services to the consumer or end user. Subsequently, it represents strategic coordination of operations such as raw material purchasing, manufacturing planning, logistics, inventory management, and forecasting consumer demand. Effective SCM is fundamental to achieving cost savings, increasing market responsiveness, and improving consumer endorsement [9].

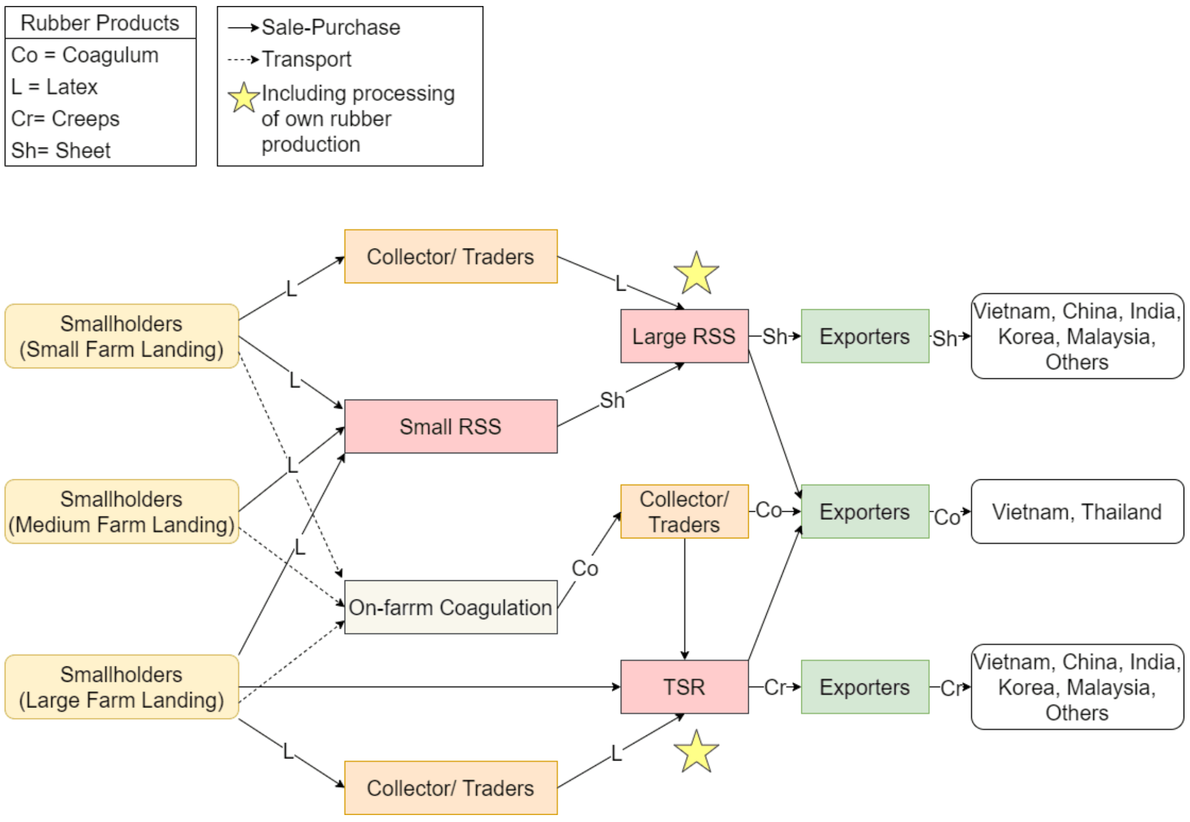

Rubber Products
Co = Coagulum
L = Latex
Cr= Creeps
Sh= Sheet
Sale-Purchase
Transport
Including processing of own rubber production
Smallholders (Small Farm Landing)
Smallholders (Medium Farm Landing)
Smallholders (Large Farm Landing)
Collector/ Traders
Small RSS
On-farrm Coagulation
Collector/ Traders
Large RSS
Collector/ Traders
TSR
Exporters
Exporters
Exporters
Vietnam, China, India, Korea, Malaysia, Others
Vietnam, Thailand
Vietnam, China, India, Korea, Malaysia, Others
L
Sh
Co
Cr

Figure 1. The general structure of Cambodia's Rubber Supply Chain. (Source: Author, adapted from [3])

2.2. Blockchain

The concept of Blockchain (BC) has significantly changed our perception of reliability and cooperation in distributed applications (DApps). It is based on a shared, indestructible ledger made of cryptographically attributed components. Satoshi Nakamoto's 2008 Bitcoin whitepaper mentioned this revolutionary invention [12]. In Figure 2, each record has a header with versioning, timestamps, and a Merkle root for data confidentiality [13], and an object containing transaction details. Blockchains achieve immutability through hash pointers, with each block referencing its predecessor for secure traceability. These ledgers operate on decentralized networks, relying on consensus mechanisms like Proof-of-Work (PoW) to validate updates [14] and cryptography to protect data during transfer [15]. This design fosters decentralization, immutability, traceability, collaborative maintenance, and customizable transparency [16]. However, Bitcoin's pioneering protocol exposes constraints in scalability and its scripting language's adaptability for complex applications [17].

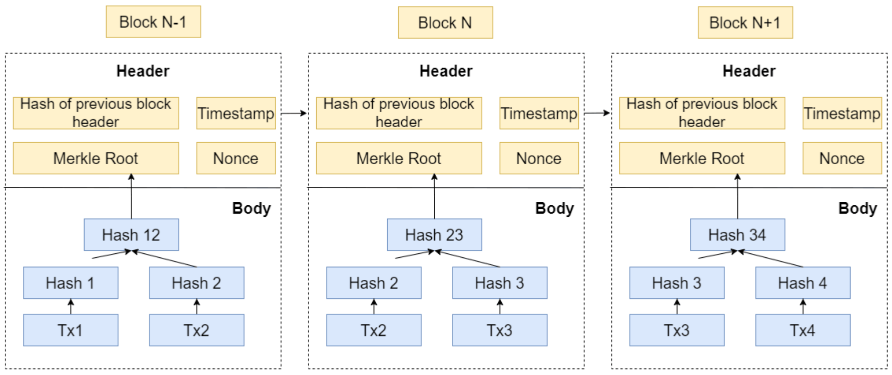


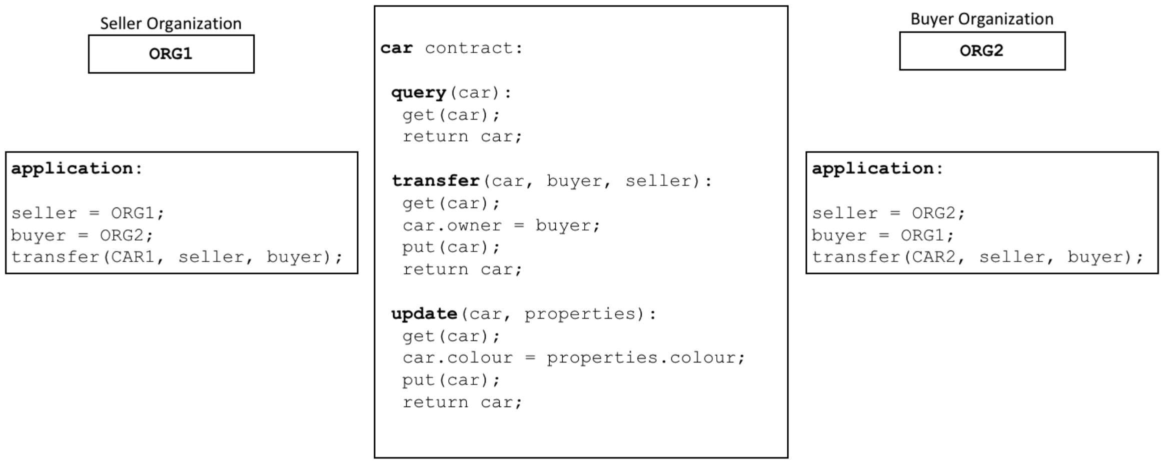
Figure 2. Structure of the blocks.[19]

2.2.1. Blockchain Platform

1. **Ethereum:** Ethereum is a decentralized system that allows smart contracts to function. Ethereum creates machine code from smart contracts, which are then run by the Ethereum virtual machine (EVM) [21]. Ethereum smart contracts use an account-based data structure where each user is identified by their digital wallet [22]. Ethereum employs the computationally costly PoW consensus process, just like Bitcoin [12]. However, Proof-of-Stake (PoS) will soon replace PoW on Ethereum, expanding Ethereum through Eth2 enhancements [18]. Gas functions as an internal charge for executing a transaction to offset ETH's unstable value [21]. Ether, the native cryptocurrency token of Ethereum, provides an engine for transaction execution and ecosystem-wide interaction with DApps [22]. Efforts to address scalability challenges are motivated by modest transaction latency and current scalability restrictions, assessed at 15 transactions per second (TPS) [23]. The large Ethereum development community continuously optimizes the protocol and contributes to regular upgrades [18].
2. **Hyperledger Fabric:** Hyperledger Fabric, a distributed ledger technology (DLT) within the Hyperledger project overseen by the Linux Foundation [20], offers a distinct approach to smart contract execution. Unlike Ethereum's reliance on a Virtual Machine (EVM), Fabric leverages Docker containers for smart contract code deployment. This strategy provides enhanced isolation and resource efficiency compared to VMs [24]. While initially receiving substantial investment from IBM, Fabric's open-source nature promotes collaboration and prevents single-entity dominance. Fabric supports traditional high-level languages like Java and Go, contrasting with Ethereum's domain-specific languages [25]. Fabric maintains Turing completeness, ensuring the network's ability to perform general-purpose computations. Fabric employs a key-value data model for state representation. Designed for enterprise deployment, Fabric implements a permissioned blockchain model, requiring network participation authorization from Certificate Authorities (CAs) [4]. Multiple CA types serve distinct roles within the network. Fabric's permissioned structure streamlines consensus mechanisms, optimizing for efficiency within controlled environments [26].
3. **R3 Corda:** Corda focuses on use cases addressing digital currencies and assets, providing a framework for managing and documenting digital asset ownership [27]. High-level programming languages like Java and Kotlin are used to write Corda's smart contracts, running in the Java Virtual Machine (JVM). Corda confirms transaction validity only among immediate parties involved, not throughout the network [27]. With a focus on asset statuses and their changes, Corda uses a transaction-based data architecture, often resulting in its deployment in private, permissioned blockchains [28]. Corda's permissioned structure and use of the Raft consensus method speed up consensus achievement [29]. Raft operates as a "leader and follower" ordering service, with decisions replicated by a predetermined leader node. It uses a crash-fault-tolerant (CFT) architecture, guaranteeing consensus finality and protocol protection even if some network components fail [14].

2.2.2. Smart Contract in Hyperledger Fabric

Smart contract (SC) are the core foundation behind Hyperledger Fabric's efficiency and performance. In this permissioned blockchain architecture, SC operates as self-executing contracts, including agreed-upon business logic directly throughout the source code [4]. Data preservation, in addition to an immutable record of state change transactions, provides remarkable transparency and automation to operations previously limited by complications, inefficiencies, and possible conflicts [31]. In Hyperledger Fabric, SC is arranged into components known as Chain Code (CC) for administrative ease [4]. These CCs, primarily created in Go but also support Node.js and Java, are run by network peers [30]. This peer execution ensures that all authorized parties agree on transaction results before modifying the ledger state [25] as shown in Figure 3.



```
application:

seller = ORG1;
buyer = ORG2;
transfer(CAR1, seller, buyer);
```

```
car contract:

 query(car):
  get(car);
  return car;

 transfer(car, buyer, seller):
  get(car);
  car.owner = buyer;
  put(car);
  return car;

 update(car, properties):
  get(car);
  car.colour = properties.colour;
  put(car);
  return car;
```

```
application:

seller = ORG2;
buyer = ORG1;
transfer(CAR2, seller, buyer);
```

Figure 3. Smart Contracts in HLF (Source: HLF Smart Contracts Docs[30])

2.2.3. IPFS with Hyperledger Fabric

The InterPlanetary File System (IPFS) offers an attractive solution for blockchain technology's data storage limitations. IPFS, a peer-to-peer (P2P) distributed file system, uses content-addressing to generate unique content identifiers (CIDs) through file hashing. By storing these CIDs on the blockchain instead of raw data, IPFS significantly reduces storage pressure and costs [5]. This relationship between IPFS and blockchain allows effective off-chain storage of significant or sensitive data. Sensitive information benefits from IPFS's distributed nature, enhancing privacy, while the blockchain provides a tamper-proof record for references and metadata. Furthermore, IPFS incentivizes persistent data storage via Filecoin, its native asset, ensuring ongoing accessibility [32]. Additionally, the content-addressable aspect of IPFS presents various benefits. Immutability is achieved by identifying content via a unique hash; any changes to the file's contents generate a new CID. This improves version management, maintains verifiable data connections, and develops trust in data consistency. Therefore, integrating IPFS with blockchain systems allows decentralized data management, reducing dependency on centralized storage providers while offering a more secure and flexible structure.

3. Proposed Framework

3.1. System Overview

In this chapter, we present our proposed system to address the rubber supply chain in Cambodia, which faces obstacles like limited traceability, poor visibility, and potential security risks and is worried about its long-term viability. To tackle these issues, we propose a resolution utilizing HLF blockchain technology to enhance the rubber supply chain by increasing traceability, visibility, and security from start to finish and promoting sustainable practices. The proposed solution utilizes the inherent features of HLF, such as its flexibility and suitability for complex supply chains. Due to its decentralized nature, utilizing IPFS as an off-chain storage solution is ideal for securely storing sensitive information like rubber prices, origin details, and processing methods. IoT devices are crucial for live monitoring and use communication protocols like Bluetooth, WiFi, and the Internet. IoT devices will significantly impact live monitoring, using communication protocols like Bluetooth, WiFi, and the Internet. Protocols like MQTT, HTTPS, and Application Programming Interface (API) will enable efficient linking of IoT devices, the blockchain, and user interfaces, ensuring a continuous data flow for effective supply chain management. Last but not least, we utilize differential privacy based on the Laplace algorithm for secure data sharing, which is written in the smart contract's code.

3.1.1. Use Case

Our proposed system includes seven organizations, including Farmers, Distributor, Manufacturers, Exporters, Retailers, Consumers, and IPFS, as described in the following paragraphs and Figure 4 and Figure 5.

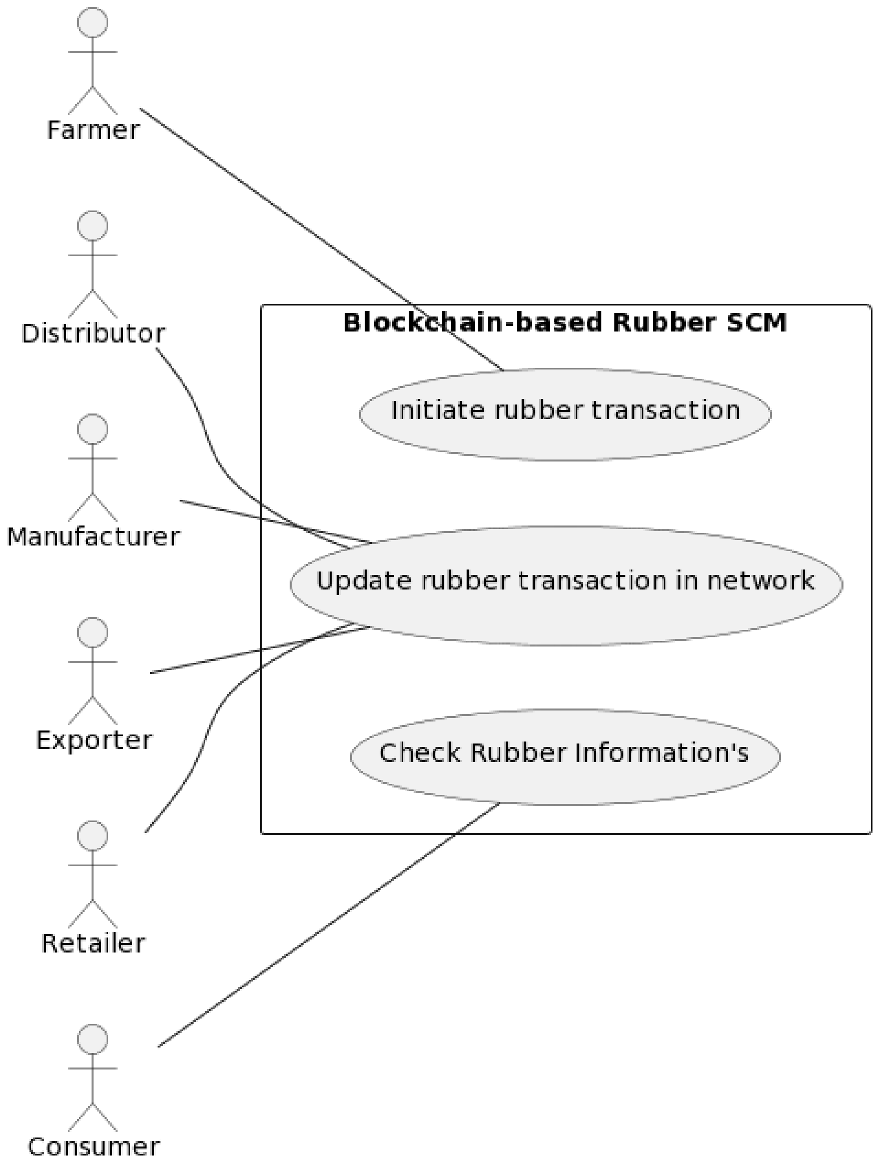

Farmer
Distributor
Manufacturer
Exporter
Retailer
Consumer
Blockchain-based Rubber SCM
Initiate rubber transaction
Update rubber transaction in network
Check Rubber Information's

Figure 4. Rubber SCM based on Blockchain

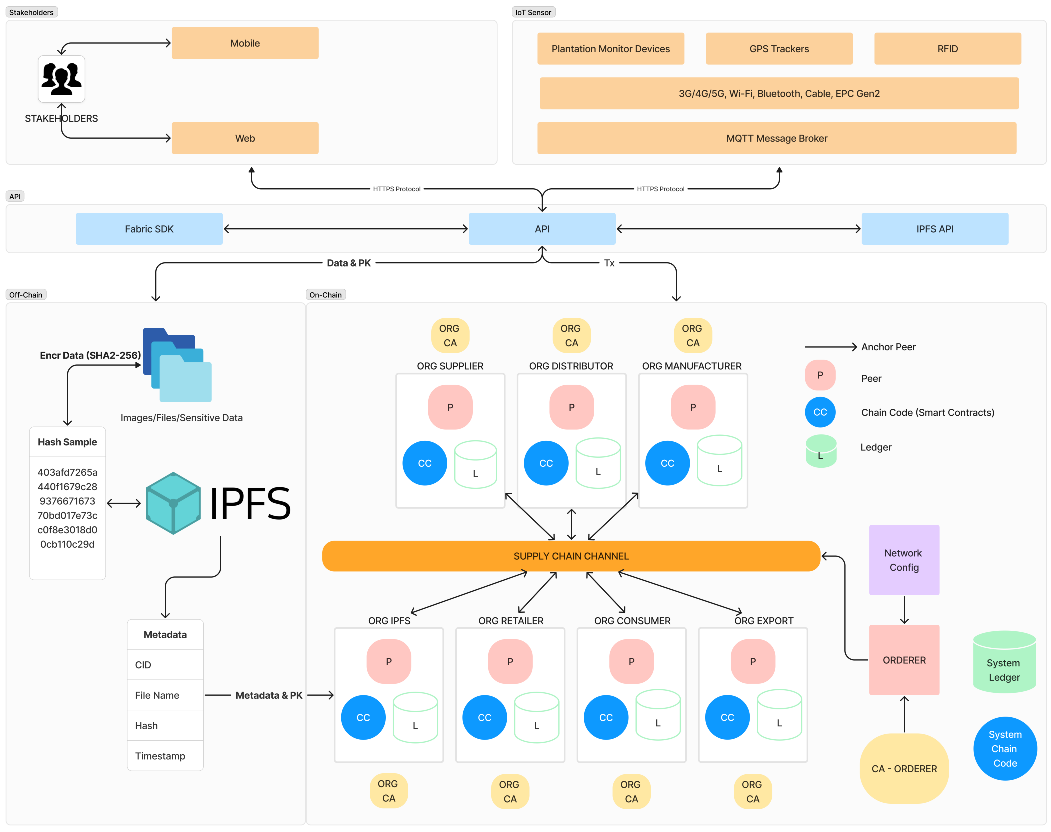


Figure 5. Details proposed system network configurations based on HLF-BC and IFPS for Cambodia RSCM

- **Farmer Organization:** Farmer organizations provide the basis for cultivating rubber trees, harvesting latex, and processing it into useful forms such as bales or sheets. They start the rubber's journey by connecting it to the supply chain network.
- **Distributor Organization:** Distributors act as intermediaries between manufacturers and Farmers. They assure a continual supply of materials throughout the production process by purchasing raw rubber from Farmers, overseeing its storage, and distributing it to manufacturers according to their demands.
- **Manufacturer Organization:** To make a wide variety of rubber goods, such as tires, hoses, and seals, they buy raw rubber from distributor organizations. They provide value through manufacturing processes by converting raw resources into completed commodities.
- **Exporter Organization:** Exporters enable commerce between countries. By acquiring completed rubber goods from producers, managing export paperwork and shipping, and guaranteeing that the goods get to global consumers, they help the rubber sector grow internationally.
- **Retailer Organization:** Retailers are the last port of call. Completed rubber goods are acquired from exporters or distributors and sold to customers via a variety of channels, including retail locations and internet retailers. They bring the finished product to the people who use rubber items on a daily basis, serving as the last point of sale.
- **Consumer Organization:** Retailers are the last port of call. Completed rubber goods are acquired from exporters or distributors and sold to customers via a variety of channels, including retail locations and internet retailers. They bring the finished product to the people who use rubber items on a daily basis, serving as the last point of sale.
- **IPFS Organization:** The IPFS, a decentralized and impenetrable network platform, is run by the IPFS group. This system promotes openness and confidence across the supply chain by enabling the safe storage and exchange of critical rubber-related data, including contracts, certifications, and sensor data.

3.1.2. System Architecture

- **Stakeholder Engagement:** Initially, stakeholders, including Farmers, manufacturers, and retailers, interact with the system via mobile and web interfaces. These interfaces are designed for intuitive use, allowing stakeholders to input data, retrieve information, monitor processes, and make informed decisions efficiently. Moreover, this user-friendly approach facilitates widespread accessibility and constant communication, crucial for real-time supply chain management.
- **IoT Sensor Integration:** Subsequently, the system incorporates IoT devices such as plantation monitor devices, GPS trackers, and RFID sensors. These devices are instrumental in collecting real-time data and tracking the movement of goods across the supply chain. They connect using various protocols, including 3G/4G/5G, Wi-Fi, and Bluetooth, ensuring comprehensive coverage and connectivity. Data collected from these devices are typically transmitted via an MQTT message broker, which is a standard protocol for IoT communications, enhancing the reliability and timeliness of data transmission.
- **Blockchain Interaction:** Moreover, the core of the system utilizes Hyperledger Fabric, a permissioned blockchain framework known for its robust security features and performance efficiency. Interaction between the users and IoT devices with the blockchain is facilitated through API interfaces using the Fabric SDK. All transactions are conducted over HTTPS to ensure secure and reliable data transmission. This setup ensures that all interactions within the supply chain are immutable and verifiable, enhancing trust among all participants.
- **Off-Chain Data Storage:** For managing sensitive data such as images and confidential documents, the system employs SHA-256 encryption and stores this data off-chain in the IPFS. This approach prevents the blockchain from being overwhelmed by large volumes of data while ensuring that the data remains accessible and secure. Metadata for these files, along with content identifiers (CIDs), are anchored to the blockchain, maintaining the integrity and traceability of off-chain stored data.
- **On-Chain Process Flow:** Within the blockchain, various organizational entities interact through dedicated channels termed 'SUPPLY CHAIN CHANNEL'. These interactions involve peers (P), chaincode (CC), and ledgers (L) for different entities such as Farmers, distributors, and consumers. This delineation facilitates efficient and secure transaction processing and information flow within the network.
- **Network Configuration:** Lastly, the network configuration of the system is designed to be modular. Components such as the ORDERER play a crucial role in the consensus and ordering of transactions into blocks. The System Channel Code manages network-wide settings, and the System Ledger records the state of the network, with an anchor peer serving as a synchronization point for organizational data.

3.2. Transaction Flow

Initially, the proposed Hyperledger Fabric-based rubber supply chain management system in Cambodia incorporates a sophisticated architecture designed to enhance the integrity, efficiency, and transparency of the supply chain. It leverages Internet of Things (IoT) technology to collect essential data points, and employs blockchain technology for secure data processing and storage.

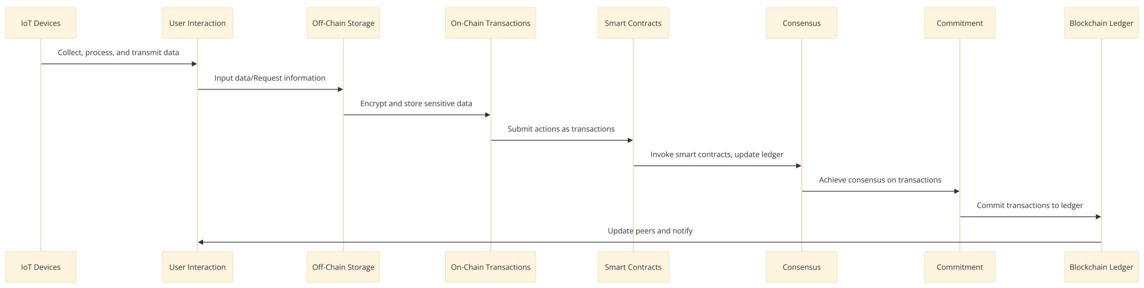

IoT Devices
User Interaction
Off-Chain Storage
On-Chain Transactions
Smart Contracts
Consensus
Commitment
Blockchain Ledger
Collect, process, and transmit data
Input data/Request information
Encrypt and store sensitive data
Submit actions as transactions
Invoke smart contracts, update ledger
Achieve consensus on transactions
Commit transactions to ledger
Update peers and notify

Figure 6. Transaction flow in Rubber BC-based SCM

Subsequently, the initial stage of the system architecture involves extensive data collection via IoT sensors strategically placed throughout the rubber supply chain. These sensors are tasked with gathering critical real-time data such as temperature readings from storage facilities, humidity levels critical for rubber preservation, and GPS tracking information from shipping containers to monitor logistical movements. This collected data is crucial for proactive supply chain management, allowing for immediate response to potential issues such as temperature deviations that could affect product quality.

Moreover, the data collected by these IoT devices is then transmitted securely using MQTT (Message Queuing Telemetry Transport), a machine-to-machine (M2M)/"Internet of Things" connectivity protocol designed for lightweight messaging between devices in low-bandwidth environments. MQTT is crucial for this architecture as it ensures that data transmission between IoT devices and the central system is efficient and secure, minimizing exposure to unauthorized access and data breaches.

Furthermore, following data collection and processing, the architecture supports active stakeholder interaction through both mobile and web interfaces. These interfaces serve as the primary means for both data input and retrieval, facilitating tasks such as entering shipment details, checking inventory levels, and monitoring supply chain operations in real time. The design of these interfaces focuses on user-friendliness and accessibility to ensure that stakeholders, regardless of their technical proficiency, can interact with the system effectively.

Additionally, to enhance security and scalability, sensitive data such as personal information and detailed logistical data are encrypted using SHA-256, a cryptographic hash function that provides a high level of security. This encrypted data is stored off-chain on the IPFS, a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS is selected for its ability to handle large volumes of data without compromising the speed and performance of the blockchain, as it stores only metadata and hashes on-chain. This approach significantly reduces the load on the blockchain while ensuring data immutability and availability.

Consequently, central to the transaction flow of the system is the creation and management of on-chain transactions. Actions taken by stakeholders, such as placing orders or updating shipment statuses, are captured as transactions within the Hyperledger Fabric network. These transactions are managed via APIs that interface with the Hyperledger Fabric SDK, facilitating a secure and standardized method for transaction handling.

Upon initiation, these transactions trigger specific chaincode (smart contracts), which are programmed with the business logic of the supply chain operations. The chaincode executes these transactions by validating them against predefined rules and protocols, ensuring that all actions are consistent with the business's operational standards and regulatory compliance requirements.

Finally, for transaction validation and ledger updates, the system employs a consensus mechanism such as RAFT. In this mechanism, a leader is elected among the nodes to propose the order and details of transactions. This leadership and consensus approach is vital for maintaining the integrity and trustworthiness of the system, ensuring that all nodes agree on the transaction data before it is committed to the ledger.

After achieving consensus, the transactions are committed to the ledger, thereby updating the blockchain with new and validated data. This step is critical for maintaining a chronological record of all transactions, supporting traceability and auditability.

To ensure the system remains up-to-date and synchronized across all participants, notification and synchronization mechanisms are implemented. Once a transaction is committed, all relevant parties are notified, and their respective ledgers are automatically updated to reflect the new state. This process not only ensures data integrity across the network but also enhances the visibility and reliability of the entire supply chain management system.

Through this detailed architecture and workflow, the proposed system harnesses the combined strengths of IoT and Hyperledger Fabric to revolutionize rubber supply chain management in Cambodia, providing a robust solution that addresses traditional challenges of transparency, efficiency, and security in supply chain operations.

3.3. Consensus Protocol

The RAFT consensus mechanism is implemented in our proposed system to manage the ordering of transactions within our network. Distributed nodes use a leader-based consensus mechanism known as RAFT, or the Reliable, Replicated, Redundant, and Fault-Tolerant protocol, to ensure data consistency throughout the network. All client requests are processed by a leader, who monitors the log replication procedure and provides the data committed to the replicated log before execution. A new leader is chosen when the current one collapses, reducing the time needed to achieve consistency and efficiently handle redundancy. RAFT is the most commonly utilized consensus mechanism in HLF implementations. It is very flexible and supports versions 2.0.0 and higher, which makes it suitable for our system, which is now running v2.5.0.

3.4. Secured Data Sharing Protocol based on Differential Privacy Using Laplace Algorithms

3.4.1. Vulnerable Scenario in the HLF-based Network

Figure 7 illustrates a possible vulnerable scenario in our proposed system for data exchange between the user, API, HLF, and the IPFS. This scenario describes a sequence of actions an attacker could employ to compromise system security.

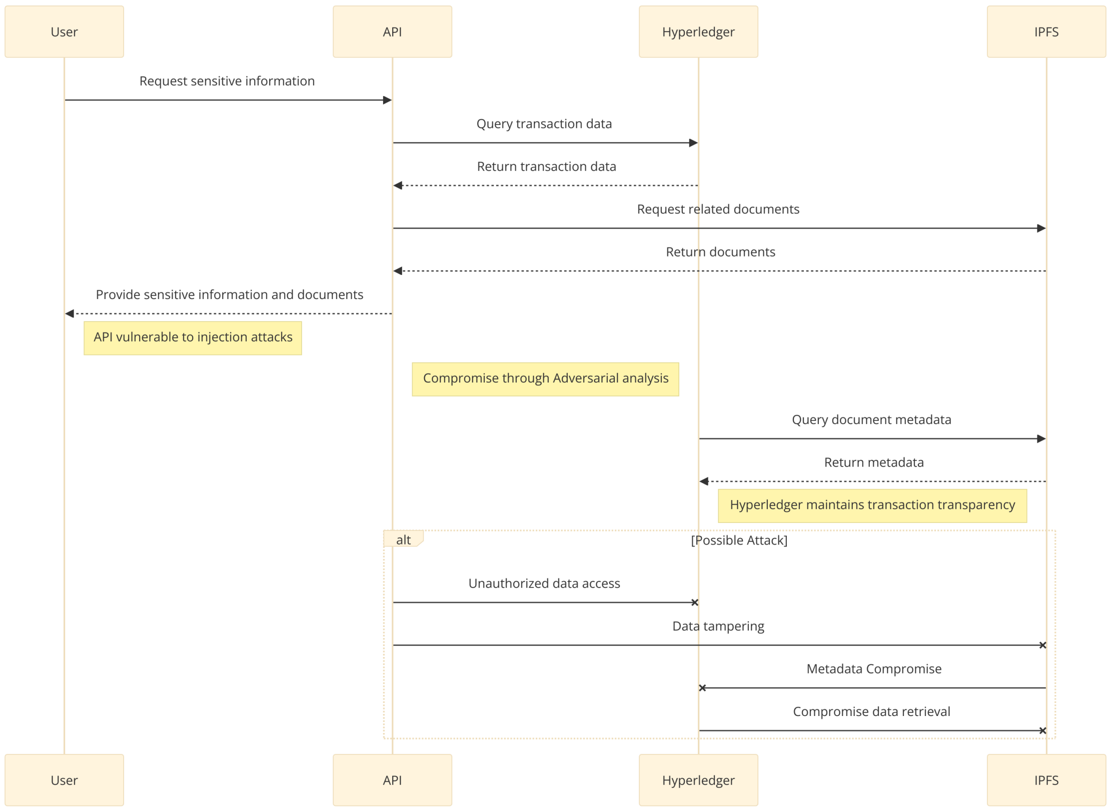

User
API
Hyperledger
IPFS
Request sensitive information
Query transaction data
Return transaction data
Request related documents
Return documents
Provide sensitive information and documents
API vulnerable to injection attacks
Compromise through Adversarial analysis
Query document metadata
Return metadata
Hyperledger maintains transaction transparency
alt
[Possible Attack]
Unauthorized data access
Data tampering
Metadata Compromise
Compromise data retrieval
User
API
Hyperledger
IPFS

Figure 7. The scenario of Compromise sensitive information in Rubber SCM Blockchain

The initial vulnerability was identified at the API level, where the user enters sensitive information. The API is vulnerable to injection attacks, a type of manipulation in which an attacker injects malicious code into the system, resulting in unauthorized access or data tampering. This breach might compromise sensitive user information and allow the attacker to gain further capabilities within the network.

Additionally, the adversarial assessment indicates that Hyperledger, regardless of its transaction transparency, is vulnerable to possible security breaches. The scenario exposes illegal data access, data manipulation, and metadata breaches. Each of these attack routes presents a critical risk to the system's security. Unauthorized access could expose confidential transaction data; data tampering could alter transaction records, compromising the ledger's reliability; and metadata compromise could reveal more details that could be used in future attacks or to cause current compromises [33].

Furthermore, utilizing IPFS to query document information might offer an attack surface. However, rather than directly shown in the illustration, the metadata transmission could be intercepted or manipulated, resulting in a compromised retrieval of the information system.

3.4.2. Solution for the Data Sharing in HLF-Based Network

Figure 8 illustrates a privacy-focused system that employs differential privacy mechanisms within a blockchain network to handle sensitive user queries. The procedure begins when the User queries an API that includes intentionally generated noise, protecting the individual's privacy. The API then sends this noisy query to blockchain-based Smart Contracts, which apply Laplace noise to raw information while keeping up with differential privacy standards. This guarantees that the User's sensitive information, such as pricing, location, and processing methods, remains secret. The BC network evaluates and solves the query while hidden by noise. Nevertheless, it contains valuable information while adhering to strict privacy criteria. This approach provides an improved process for protecting personal information in our proposed system of differential privacy theoretical protections for private information. The equation for differential privacy using the Laplace mechanism, as introduced by [34], is expressed as:

$$\Pr[K(D) \in S] \leq e^{\epsilon} \Pr[K(D') \in S] \tag{1}$$

Where:

- Pr is the probability function, denoting a specific event.
- $K(D)$ represents the randomized function applied to dataset D.
- S is the set of possible outputs of K.
- D' is a dataset differing in at most one element from D.
- ϵ is the privacy parameter which controls the privacy guarantee.

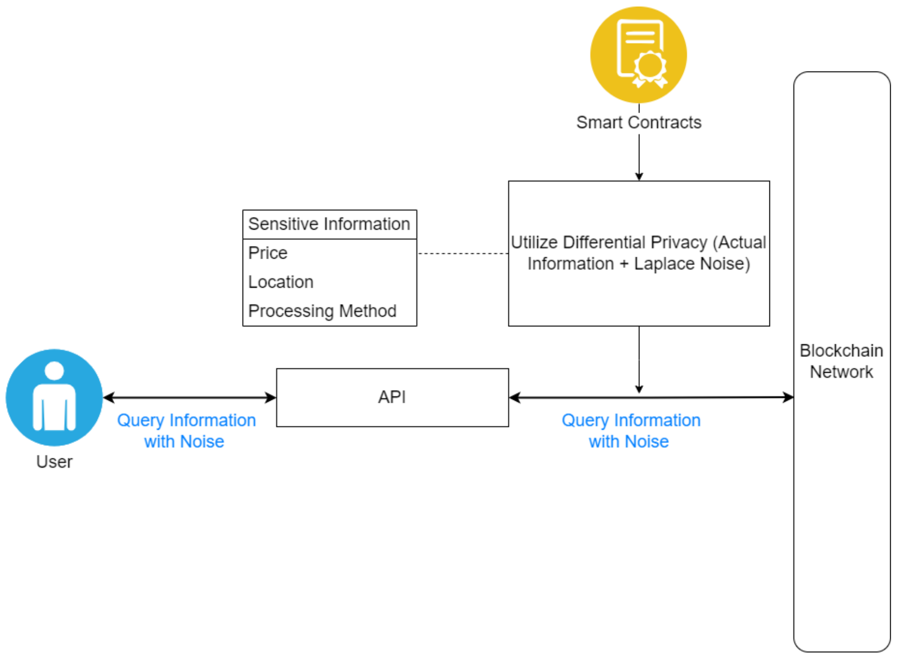

Smart Contracts
Sensitive Information
Price
Location
Processing Method
Utilize Differential Privacy (Actual Information + Laplace Noise)
Blockchain Network
API
Query Information with Noise
Query Information with Noise
User

Figure 8. The scenario of Compromise sensitive information in Rubber SCM Blockchain

The Laplace distribution adds noise to the query results, effectively masking the contributions of individual data points. The probability density function of the Laplace distribution used in differential privacy is:

$$f(y|\mu, b) = \frac{1}{2b} e^{-\frac{|y-\mu|}{b}} \tag{2}$$

Where:

- f represents the query function applied to a dataset.
- y represents the noise added.
- μ is typically 0 (centering the distribution at 0).
- b is the scale parameter.

The scale b is determined by the sensitivity of the query Δf and the privacy parameter ϵ, as given by:

$$b = \frac{\Delta f}{\epsilon} \tag{3}$$

Where:

- Δf represents the sensitivity of function f, measuring the maximum change in its output from a single data alteration.

This equation ensures that the output remains private according to the defined privacy parameter ϵ, providing mathematical security of privacy under specific assumptions about adversarial knowledge.

4. Implementation and Evaluation

In this part, we get into the essential prerequisites for successfully setting up the system as proposed. These prerequisites include the required hardware and software components, as well as the installation and configuration process. These factors are fundamental to assessing the performance of blockchain network implementations throughout the implementation phase and collecting relevant data such as transaction per second(TPS), transaction latency(TL), and transaction throughput(TT).

Table 1. Simulation Parameters Settings

Parameter	Value
Operating System	Ubuntu 20.04.1 LTS 64-bit
CPU	Intel® Core™ i7-7700 8 CPU @ 3.60GHz
RAM	8 GB
Minifab for Hyperledger Fabric	2.5 (Latest)
IPFS	12
Hyperledger Caliper	2.0
Docker-compose	1.29.2
Docker	25.0.3
Go	1.20.2
Google Differential Privacy Library (Go)	1.1.2

In the proposed system, the implementation environment is carefully chosen for optimal performance and compatibility. The system runs on Ubuntu 20.04.1 LTS 64-bit, ensuring security and reliability. It uses an Intel® Core™ i7-7700 processor with 8 CPUs at 3.60GHz and 8 GB of RAM, balancing high processing power with cost-effectiveness.

The environment uses Minifab for Hyperledger Fabric 2.5 for up-to-date blockchain features and security. IPFS version 12 supports decentralized storage, enhancing data availability. Hyperledger Caliper 2.0 benchmarks blockchain performance, providing insights into transaction throughput and latency. Docker and Docker-compose (versions 25.0.3 and 1.29.2) facilitate containerization for scalable and isolated service execution. Go version 1.20.2 is chosen for efficient system-level programming. This configuration supports robust, scalable, and efficient blockchain application testing and deployment.

4.1. Implementation

4.1.1. Ledger Initialization

Algorithm 1 describes the implementation of the **Initialize Ledger (InitLedger)** smart contract algorithm on the HLF network. The algorithm accepts a single input, **ctx**, an instance of **contractapi.TransactionContextInterface**. This algorithm aims to initialize the ledger by calling the **initCounter function**, which maintains the counter's initial state in the ledger. If the initialization is successful, the algorithm returns **nil**, indicating that the ledger was properly initialized. If an error occurs during the initialization procedure, the algorithm will output an error message for the type of `error init counter:%s`. This architecture guarantees that any errors during ledger startup are identified and reported.

Algorithm 1 Initialize Ledger (InitLedger)

```
procedure INITLEDGER(contractapi.TransactionContextInterface)
    $error \leftarrow$ INITCOUNTER($ctx$)
    if error $error$
        return fmt.Errorf("error init counter: %s", error.Error())
    return nil
end procedure
```

4.2. Batch Transaction

4.2.1. Growth Batch Transaction

Algorithm 2 presented the outlines of the CreateBatch method within a Farmer SC tailored for blockchain applications. Initially, it fetches the transaction ID from the context and ifs if the provided batch ID is empty. An empty ID triggers a return of false along with an error message about the missing batch ID. If the ID is valid, it proceeds to fetch batch data from a transient state—not permanently recorded on the blockchain. Failure at this stage also results in a return of false due to the error encountered. Successful data retrieval leads to an attempt to deserialize the JSON-formatted string into a FarmerBatch object; a failure in deserialization returns false with the error. The next step involves uploading the batch data to the IPFS, a distributed file system, and storing the resulting IPFS hash. Any issues during this upload result in a return of false. Subsequently, the FarmerBatch object is updated with the IPFS hash and transaction ID, followed by an attempt to record this updated object in the blockchain's state database using the batch ID as the key. If this operation fails, it results in a false return with an error. If all processes execute without errors, the function successfully concludes by returning true, signifying the successful creation of the batch. This method encapsulates typical blockchain operations involving data retrieval, validation, and storage, efficiently managing errors at each step to ensure data integrity and accuracy in batch recording.

Algorithm 2 Create a batch in Farmer Contract

```
procedure CREATEBATCH($ctx, batchID$)
    $txnID \leftarrow$ GetTxID from $ctx$
    if $batchID =$ ""
        return false, "Batch ID is empty"
    $batchStr, err \leftarrow$ GetBatchFromTransient($ctx$)
    if error $err$
        return false, $err$
    $batch \leftarrow$ new FarmerBatch
    $err \leftarrow$ json.Unmarshal($batchStr$, &$batch$)
    if error $err$
        return false, $err$
    $ipfsHash, err \leftarrow$ UploadDataToIPFS($batch$)
    if error $err$
        return false, $err$
    $batch.IPFSHashes \leftarrow$ {"batchData" : $ipfsHash$}
    $batch.TxnID \leftarrow txnID$
    $err \leftarrow$ PutState($ctx, batchID, batch$)
    if error $err$
        return false, $err$
    return true, nil
end procedure
```

4.2.2. Query Grown Batch Transaction

Algorithm 3 presented represents the operation of the **QueryGrownBatchById** function within a smart contract, used to retrieve and process a batch of data from a blockchain network. Initially, the function attempts to fetch the batch data from the blockchain's state database using the **ctx.GetStub().GetState** method. If an error occurs, such as the absence of data or a network issue, the function returns an error formatted to indicate a failure in data retrieval. Assuming successful data fetch, the next step involves deserializing the string format data into a **FarmerBatch** object for programmatic manipulation. If errors arise during this unmarshalling, such as from data corruption or incorrect formatting, the function returns an error that includes the specific **batch ID** and a detailed failure message. If unmarshalling is successful, the function returns the structured **FarmerBatch** object with a nil error, indicating successful execution.

Algorithm 3 Query Grown Batch by ID

```
procedure QueryGrownBatchById($ctx, batchId$)
    batchStr, err ← $ctx.GetStub().GetState(batchId)$
    if error err
        return nil, fmt.Errorf("error while getting state, %w", err)
    grownBatch ← new models.FarmerBatch
    err ← json.Unmarshal(batchStr, &grownBatch)
    if error err
        return nil, fmt.Errorf("error unmarshalling batch id:%s, Error:%w", batchId, err)
    return grownBatch, nil
end procedure
```

4.3. IPFS Integration

4.3.1. Add Information and File to IPFS

Algorithm 4 for the **UploadDataToIPFS** function illustrates the process of uploading data to the IPFS. The procedure initiates by establishing a connection to an **IPFS** node using a specified **IPFS Shell address (localhost:5001)**. The input data is then marshalled into **JSON** format to prepare it for IPFS, which handles raw bytes. If this conversion fails, an error message is promptly returned, indicating an issue with data marshaling. Once successfully converted, the **JSON** bytes are uploaded to **IPFS** by invoking the Add method on the shell instance with the marshalled bytes. If any error occurs during the upload, it returns an error message highlighting the problem with the **IPFS** upload process. Otherwise, the successful upload returns a **unique hash(CID)**, provided by **IPFS** and stored in **if PS Org.** in the **HFL network**, which can be used to retrieve the uploaded data at any point, ensuring its persistent availability across the decentralized network.

Algorithm 4 Upload data to IPFS

```
procedure UploadDataToIPFS($data$)
    $sh$ ← New IPFS Shell on localhost:5001
    $dataBytes, err$ ← Marshal data into JSON
    if error $err$
        return "", error marshalling data to JSON
    $hash, err$ ← $sh$.Add($dataBytes$)
    if error $err$
        return "", error uploading data to IPFS
    return $hash$, nil
end procedure
```

4.3.2. Retrieve Data from IPFS

Algorithm 5 The RetrieveDataFromIPFS function illustrate the process of retrieving data from the IPFS with a given hash number. Initially, the method creates a connection to an IPFS node by starting a new IPFS shell instance at localhost:5001. It then uses the Cat technique to extract the data stream corresponding with the hash supplied. If this retrieval fails, which might be due to an incorrect hash or network troubles, the function will instantly provide an error message. Considering that the data stream was successfully retrieved, the method reads all the data into a byte array, managing faults such as I/O difficulties by issuing an error if the read fails. After reading, the data stream is closed to free resources, which is enforced by a defer statement that ensures execution at the function's end

regardless of previous results. If all actions are successful, the method returns the byte array, allowing the caller to access the retrieved data.

Algorithm 5 Retrieve Data from IPFS

```
procedure RetrieveDataFromIPFS($hash$)
    $sh \leftarrow$ New IPFS Shell on localhost:5001
    $dataReader, err \leftarrow sh$.Cat($hash$)
    if error $err$
        return nil, "error retrieving data from IPFS using hash: "$hash$
    defer $dataReader$.Close()
    $dataBytes, err \leftarrow$ Read all data from $dataReader$
    if error $err$
        return nil, "error reading data from IPFS"
    return $dataBytes$, nil
end procedure
```

4.4. *Differential Privacy for Data Sharing*

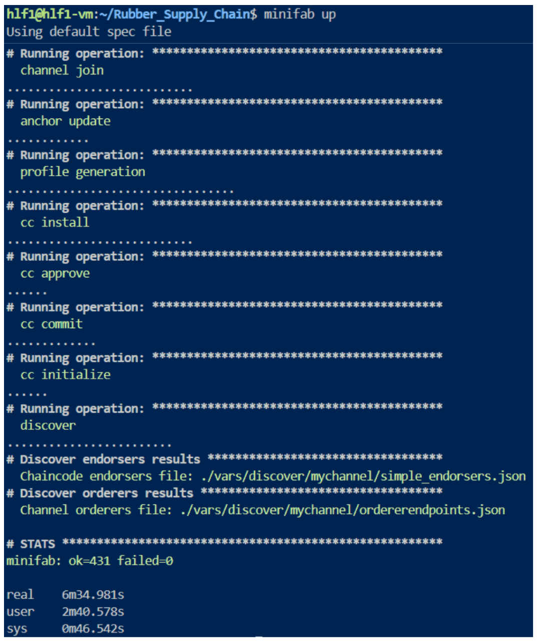
The algorithm 6 **Query Grown Batch With Differential Privacy** includes two primary functions for securely querying and processing rubber batch information while preserving privacy. The first procedure, **QueryGrownBatchById**, obtains a batch of data identified by **batchId** from a state ledger in the HLF network. Essentially, it converts data from a JSON string into a FarmerBatch model. If there are any issues in retrieving or decoding the data, it will immediately return the error. Additionally, it pulls related information from IPFS, ensuring the data is safe and verifiable. The following procedure, Differential Privacy, is designed to improve data privacy. This calculates noise injected into the data utilizing the Laplace approach, with a privacy budget set by **epsilon** and the query sensitivity **Delta f**. The disturbance, which is noise, is then injected into the data to guarantee privacy, which is a procedure for ensuring that the query's result does not compromise the privacy of the individuals in the dataset. Our technique provides secure and confidential querying of sensitive rubber information while maintaining the information's confidentiality and reliability.

Algorithm 6 Query Grown Batch With Differential Privacy

```
procedure QueryGrownBatchById($ctx, batchId$)
    batchStr, err $\leftarrow ctx.GetStub().GetState(batchId)$
    if error err
        return nil, fmt.Errorf("error while getting state, %w", err)
    grownBatch $\leftarrow$ new models.FarmerBatch
    err $\leftarrow$ json.Unmarshal(batchStr, &grownBatch)
    if error err
        return nil, fmt.Errorf("error unmarshalling batch id:%s, Error:%w", batchId, err)
    dataBytes, err $\leftarrow$ RetrieveDataFromIPFS($hash$)
    if error err
        return nil, fmt.Errorf("error retrieving data from IPFS using hash: %s,", hash) "Error:%w", err
    return grownBatch, nil
end procedure
procedure DifferentialPrivacy($dataBytes, \epsilon, \Delta f$)
    b $\leftarrow \frac{\Delta f}{\epsilon}$
    noise $\leftarrow$ GenerateLaplaceNoise($0, b$)
    dataBytes $\leftarrow$ dataBytes + noise
    return dataBytes, nil
end procedure
```

4.5. *Starting Fabric Network*

Initially, in our implementation, the primary tool for setting up and maintaining the Hyperledger Fabric network is the **minifab up** command, shown in Figure 9. This command generates the cryptographic certificates, downloads the required Docker images, and conDefinitions/Figures the peer and orderer nodes, all of which smoothly establish the whole blockchain architecture. This application indicates that all nodes and chaincodes are appropriately installed and functional, making the network deployment process less complicated.

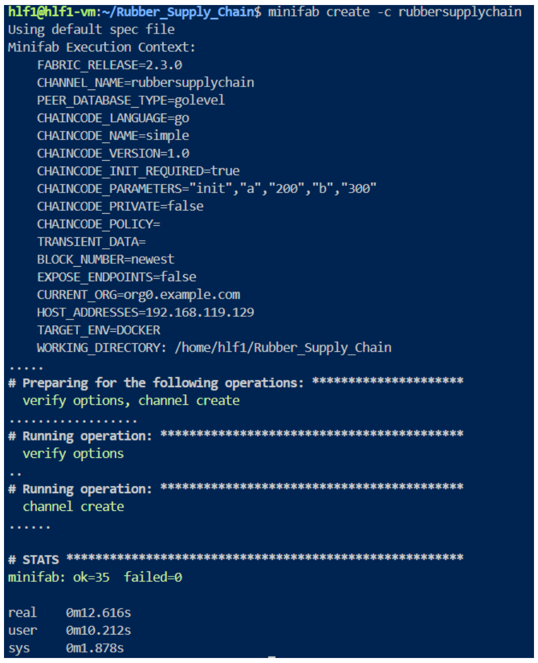


```
hlf1@hlf1-vm:~/Rubber_Supply_Chain$ minifab up
Using default spec file
# Running operation: ******************************************
  channel join
..........................
# Running operation: ******************************************
  anchor update
...........
# Running operation: ******************************************
  profile generation
................................
# Running operation: ******************************************
  cc install
..........................
# Running operation: ******************************************
  cc approve
......
# Running operation: ******************************************
  cc commit
............
# Running operation: ******************************************
  cc initialize
......
# Running operation: ******************************************
  discover
.......................
# Discover endorsers results **********************************
  Chaincode endorsers file: ./vars/discover/mychannel/simple_endorsers.json
# Discover orderers results ***********************************
  Channel orderers file: ./vars/discover/mychannel/ordererendpoints.json

# STATS *******************************************************
minifab: ok=431 failed=0

real    6m34.981s
user    2m40.578s
sys     0m46.542s
```

Figure 9. Network Initialization Logs

4.6. *Setting Up Channel*

Figure 10 present minifab build -c rubbersupplychain command represents a step in establishing the blockchain network for our proposed system. By running this command, the system creates a new blockchain channel called "rubbersupplychain," which is required for isolating and maintaining transaction data unique to our system. The tool utilizes a default specification file for setting several parameters, including the channel name, chaincode (Go), and initial chaincode parameters, which set the ledger's starting state with specified assets. This approach consists of evaluating the included settings before constructing the channel, ensuring that the blockchain environment is appropriately customized to support the operations of our proposed system.

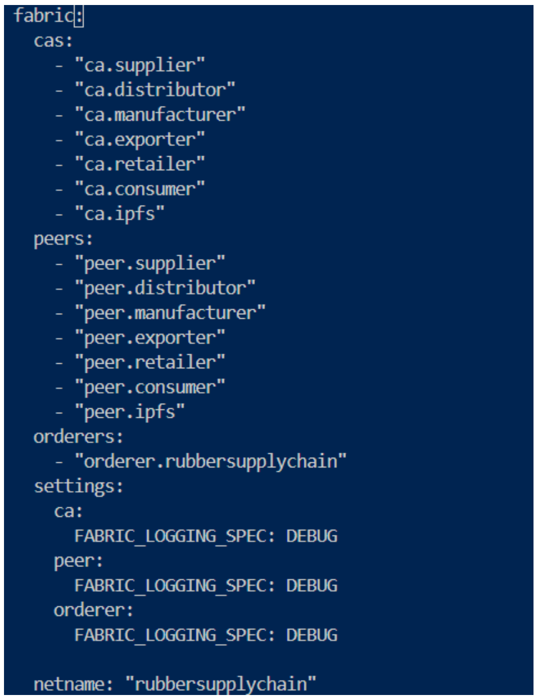


```
hlf1@hlf1-vm:~/Rubber_Supply_Chain$ minifab create -c rubbersupplychain
Using default spec file
Minifab Execution Context:
    FABRIC_RELEASE=2.3.0
    CHANNEL_NAME=rubbersupplychain
    PEER_DATABASE_TYPE=golevel
    CHAINCODE_LANGUAGE=go
    CHAINCODE_NAME=simple
    CHAINCODE_VERSION=1.0
    CHAINCODE_INIT_REQUIRED=true
    CHAINCODE_PARAMETERS="init","a","200","b","300"
    CHAINCODE_PRIVATE=false
    CHAINCODE_POLICY=
    TRANSIENT_DATA=
    BLOCK_NUMBER=newest
    EXPOSE_ENDPOINTS=false
    CURRENT_ORG=org0.example.com
    HOST_ADDRESSES=192.168.119.129
    TARGET_ENV=DOCKER
    WORKING_DIRECTORY: /home/hlf1/Rubber_Supply_Chain
.....
# Preparing for the following operations: *********************
  verify options, channel create
..................
# Running operation: ******************************************
  verify options
..
# Running operation: ******************************************
  channel create
......

# STATS *******************************************************
minifab: ok=35  failed=0

real    0m12.616s
user    0m10.212s
sys     0m1.878s
```

Figure 10. Channel Configuration Logs

4.7. *Set Up Organization Configurations*

Figure 11 presents the organization configuration in our HLF network; the architecture contains several Certificate Authorities (CAs) for each organization in the SC, from suppliers to consumers, illustrating a decentralized approach to authenticating and authorizing participants. Peers corresponding to each position validate transactions through nodes representing each supply chain stage. A single 'orderer' indicates a streamlined consensus process required for transaction ordering. All components have their logging level set to DEBUG, allowing complete monitoring and troubleshooting. The name "rubbersupplychain" refers to the network's particular use of tracking the lifetime of rubber products to improve supply chain traceability and efficiency.



```
fabric:
  cas:
    - "ca.supplier"
    - "ca.distributor"
    - "ca.manufacturer"
    - "ca.exporter"
    - "ca.retailer"
    - "ca.consumer"
    - "ca.ipfs"
  peers:
    - "peer.supplier"
    - "peer.distributor"
    - "peer.manufacturer"
    - "peer.exporter"
    - "peer.retailer"
    - "peer.consumer"
    - "peer.ipfs"
  orderers:
    - "orderer.rubbersupplychain"
  settings:
    ca:
      FABRIC_LOGGING_SPEC: DEBUG
    peer:
      FABRIC_LOGGING_SPEC: DEBUG
    orderer:
      FABRIC_LOGGING_SPEC: DEBUG

  netname: "rubbersupplychain"
```

Figure 11. Organization Configuration Logs

5. HLF Performance Evaluation

This section presents a comprehensive analysis of the benchmark equations used in Hyperledger Caliper. These equations are crucial for evaluating the performance of the system under test.

5.1. *Evaluation Metrics*

5.1.1. Success Rate (SR)

The Success Rate (SR) is calculated using the equation:

$$SR = \frac{T_s}{T_t} \times 100\% \tag{4}$$

where SR represents the Success Rate (percentage), T_s represents the number of Successful Transactions, and T_t represents the Total number of Transactions (Successful + Failed). The Success Rate measures the percentage of transactions that were successfully completed within a test cycle.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 May 2024 doi:10.20944/preprints202405.1793.v1

5.1.2. Transaction & Read Latency (TL)

Transaction & Read Latency includes three key metrics:

$$\begin{aligned} TL_{avg} &= \frac{\sum_{i=1}^{n} TL_i}{n} \\ TL_{max} &= \text{maximum value of } TL_i \text{ (for all transactions } i\text{)} \\ TL_{min} &= \text{minimum value of } TL_i \text{ (for all transactions } i\text{)} \end{aligned} \tag{5}$$

where TL_{avg} represents the Average Transaction/Read Latency, TL_i represents the Latency of individual transaction i, n is the total number of transactions, TL_{max} is the Maximum Transaction/Read Latency, and TL_{min} is the Minimum Transaction/Read Latency. These metrics provide insights into the time taken for individual transactions to complete.

5.1.3. Transaction & Read Throughput (TT)

The Transaction & Read Throughput (TT) is calculated as:

$$TT = \frac{T_t}{T_d} \tag{6}$$

where TT represents the Transaction Throughput (transactions per second), T_t represents the Total number of Transactions, and T_d represents the Duration of the test cycle (seconds). Transaction & Read Throughput measures the rate at which transactions are processed by the system.

5.2. Performance Evaluation

This section presents our evaluation of the proposed system performance, encompassing metrics such as transaction send rate (TPS), transaction throughput (TPS), minimum latency (s), average latency (s), and maximum latency (s). We compare these metrics in scenarios without differential privacy and with differential privacy implemented, where the provided ϵ value is set to 1.

5.2.1. Transaction Send Rate

In Figure 12 evaluating the impact of differential privacy on transaction send rates, the analysis presents a comparative performance between systems configuration with and without differential privacy measures. The result encapsulates transaction send rates (TPS) across increasing transaction volumes, ranging from 50 to 300 transactions. The plot reveals a consistent increase in TPS for both conditions. However, the system without differential privacy starts at a lower rate of 84.0 TPS but surpasses the with-privacy system around the 200 transaction mark, eventually reaching a peak of 139.3 TPS at 300 transactions. This suggests that without using differential privacy may initially impose overhead, it potentially offers higher scalability and efficiency as transaction volumes increase.

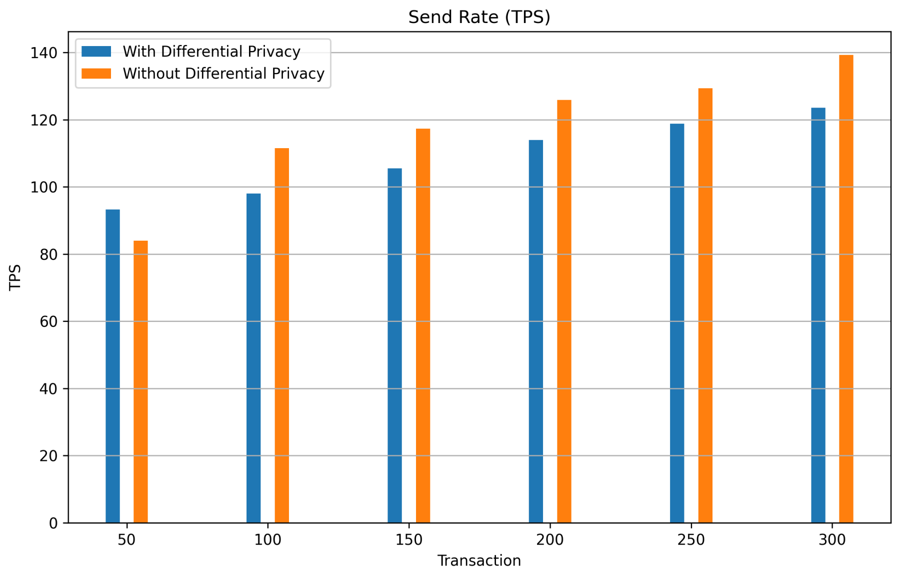


Figure 12. The Transaction Send Rate

5.2.2. Transaction Throughput

The illustration of a performance comparison between transaction systems configuration with and without differential privacy is present in Figure 13, focusing on throughput measured in transactions per second (TPS). Both configurations demonstrate a trend of increasing throughput as the number of transactions increases from 50 to 300. However, the system non-employing differential privacy shows a notable improvement in performance beyond the 100 transaction mark, achieving higher throughput rates from 107.9 to 137.3 TPS. In contrast, the system with differential privacy exhibits a more gradual increase in throughput, plateauing at 117.2 TPS. This suggests that without differential privacy, despite its initial lower performance, may enhance throughput efficiency under higher transaction loads, possibly due to optimized handling of the system at scale.

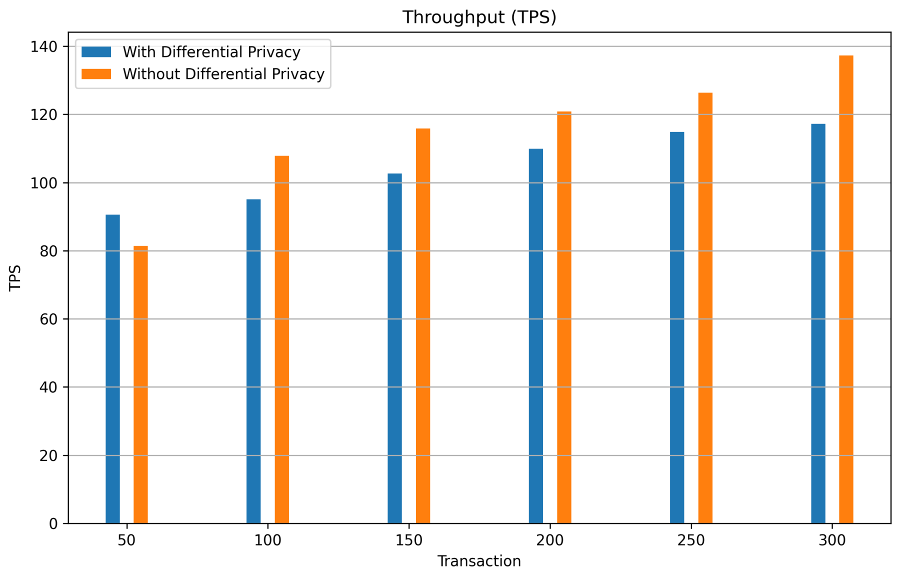


Figure 13. The Transaction Throughput

5.2.3. Transaction Latency

Figure 14 presents results on minimum latency in seconds and offers a foundational view of how differential privacy influences system performance at the lower latency bounds. Both system's configurations exhibit relatively low and stable latencies as transaction volumes increase from 50 to 300. The system without differential privacy maintains a minimum latency within a tight range of 0.10 to 0.13 seconds, indicating consistent performance. In contrast, the system with differential privacy begins at an even lower latency of 0.07 seconds but shows slightly more variability, peaking at 0.15 seconds at the 250 transaction mark. This observation highlights the minimal impact of differential privacy on the lower latency bounds, setting the stage for a deeper exploration of latency impacts at average and maximum levels.

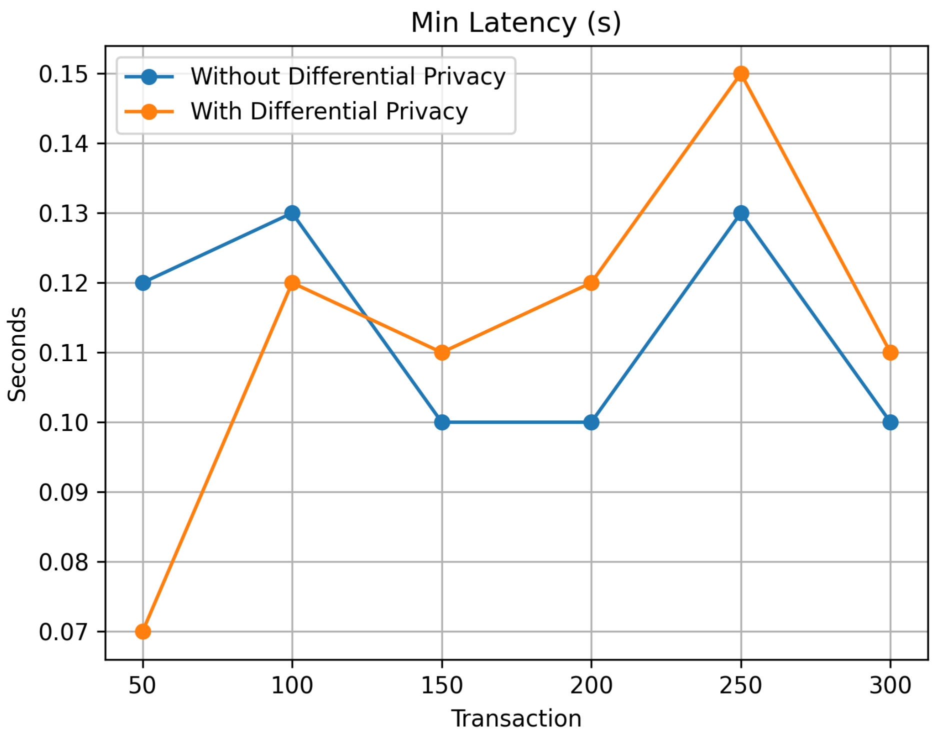


Figure 14. The Minimum Transaction Latency

However, when we consider the average latency in Figure 15, the plot summarizes the latency (in seconds) for transaction systems configuration operating with and without differential privacy across different transaction volumes. Both curves indicate a balanced increase in latency as transaction volumes rise. The system without differential privacy starts at a latency of 0.29 seconds and raises to 1.46 seconds. On the other hand, the system with differential privacy initiates at a slightly higher latency of 0.34 seconds and concludes at 1.49 seconds, indicating a marginal increase in delay due to privacy-preserving mechanisms. This effectively illustrates the balance between maintaining privacy and managing average latency in transaction processing systems configuration, indicating how differential privacy slightly increases latency but potentially enhances privacy.

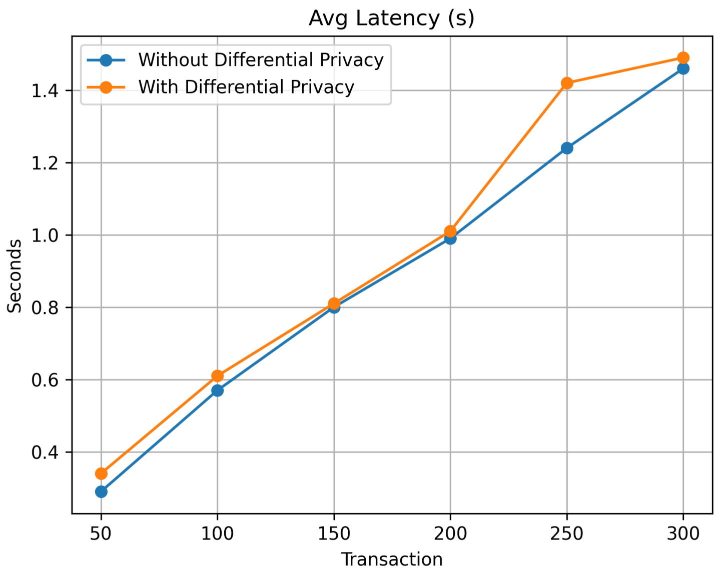


Figure 15. The Average Transaction Latency

Finally, Figure 16, which contrasts the maximum delay in seconds, allows comprehending how different system configurations with and without differential privacy affect performance. The differential privacy system indicates a strong start with a lower maximum latency of 2.07 seconds, peaking at 2.94 seconds and maintaining a continuous increase compared to the non-private system, which starts at 2.18 seconds and rises more sharply to 3.53 seconds. This indicates that differential privacy implementations might be more resilient in managing latency spikes under increasing transaction loads, potentially offering better performance stability; therefore, while minimum and average latencies provide a transparent view of consistent performance, the maximum latency underscores the stability of differential privacy in handling peak transaction demands, infusing confidence in the system's performance capabilities.

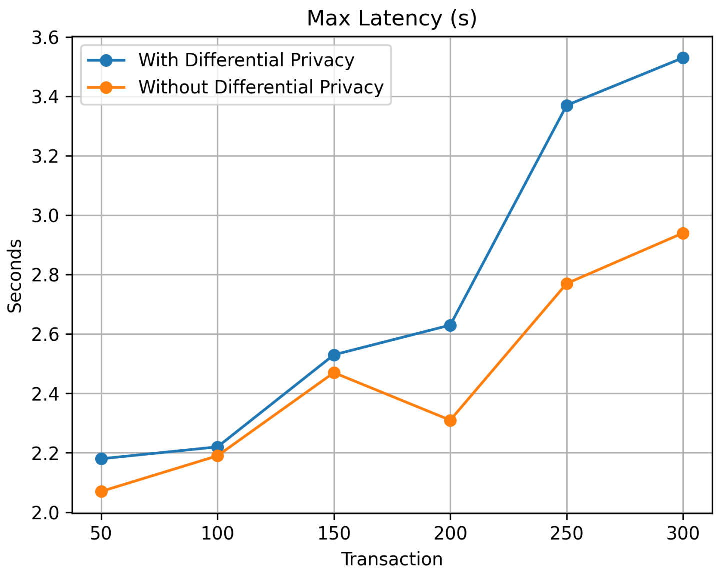


Figure 16. The Maximum Transaction Latency

6. Research Comparison

The comparative analysis table 2 provides a structured comparison between the capabilities of our proposed system and three other notable studies within the realm of transaction systems across various industries. Our framework, detailed in the table, stands out, particularly in terms of Accessibility, Data Security, and Traceability, where it matches or surpasses the cited works. Unlike the research by [8] and [35], which show limitations in data security and transparency, our system ensures both robust security and full traceability, aligning with [36] emphasis on blockchain benefits.

Table 2. Comparing the proposed framework with other related works

Key Features	[8]	[35]	[36]	Our work
Accessibility	×	✓	✓	✓
Data Security	×	×	✓	✓
Fairer prices	✓	×	×	✓
Platform	N/A	N/A	ETH	HLF
Cost Reduction	×	×	✓	✓
Sustainability	✓	✓	✓	✓
Traceability	×	×	✓	✓
Transparency	×	×	✓	✓
Use case	Rice	Bio-energy	Coffee	Rubber

A noteworthy aspect of the table is the universal Sustainability, and Fairer Prices feature across all compared studies, except where specific works have particular deficiencies. For instance, only our framework and the one developed by [8] promote fairer pricing strategies. This feature is critical in markets like rubber and rice, where price volatility can significantly impact producers. Additionally, our work incorporates Cost Reduction and Transparency, which are not uniformly addressed in other studies, especially highlighted by their absence in [35] approach to bio-energy.

Lastly, the primary use cases and platforms highlight the variety and adaptability of transaction systems to different sectors. Our framework is built on HLF-based, which contrasts with [36] use of ETH-based, suggesting a adapted approach to meeting specific industry needs, such as rubber supply chain, compared to the other studies focus on rice, bio-energy, and coffee. This differentiation emphasizes our framework's flexibility and enhances its applicability to a broader range of scenarios where such technologies can provide significant economic and operational benefits.

7. Conclusions

In conclusion, the integration of blockchain technology like Hyperledger Fabric, IPFS for data storage, and differential privacy for privacy data sharing protections in the Cambodian Rubber Supply Chain is a significant step towards transforming the sector's operational landscape. These technologies offer a practical and immutable framework, guaranteeing transaction security and fair pricing for farmers. Additionally, this approach addresses corruption and fraud, promoting sustainable practices within the Cambodian rubber sector, which aims to improve lives, support economic stability, and encourage environmentally friendly manufacturing in Cambodia. Socio-economic impact and overall development are further benefits of intentionally incorporating blockchain, IoT, and advanced technology into agricultural supply chains. However, future research should explore the applications of IoT sensors, blockchain technology, and differential privacy methods to unlock Cambodia's agricultural supply chain's potential fully. Prioritizing innovation will lead to greater resilience, global competitiveness, and sustainable development within the rubber sector. Furthermore, through these technologies, Cambodia's rubber sector can establish new standards in supply chain management and create a collaborative environment. By utilizing blockchain, IoT sensors, and differential privacy protections, the sector has the potential to optimize its operations significantly, ensuring increased efficiency, ethical standards, and sustainability. This path towards a more accountable and sustainable agricultural environment in Cambodia has the potential to benefit all stakeholders. It represents a tangible opportunity to modernize the country's agricultural landscape, emphasizing the positive impact of technological adoption.

Author Contributions: Conceptualization, K.R.; methodology, K.R.; software, K.R.; validation, K.R., M.F. and K.-H.R.; formal analysis, K.R.; investigation, K.R., M.F. and K.-H.R.; resources, K.-H.R.; data curation, K.R.; writing—original draft preparation, K.R.; writing—review and editing, M.F. and K.-H.R.; visualization, K.R.; supervision, M.F. and K.-H.R.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the Special R&D Zone Development Project(R&D) - Development of R&D Innovation Valley support program(2023-DD-RD0152) supervised by the Innovation Foundation, ITRC(Information Technology Research Center) support program(IITP-2022-2020-0-01797) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation), and also supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (No. 2021R1I1A3046590).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
BC	Blockchain
CA	Certificate Authorities
CC	Chain Code
CFT	Crash-Fault-Tolerant
CIDs	Content Identifiers
DApps	Distributed Applications
DLT	Distributed Ledger Technology
ETH	Ethereum
GDP	Gross Domestic Product
HLF	Hyperledger Fabric
IPFS	InterPlanetary File System
JVM	Java Virtual Machine
L	Ledger
M2M	Machine-to-Machine
MQTT	Message Queuing Telemetry Transport
NR	Natural Rubber
P	Peer
P2P	Peer to Peer
PoS	Proof-of-Stake
PoW	Proof-of-Work
RAFT	Reliable, Replicated, Redundant, and Fault-Tolerant
RC	Resource Consumption
RSS	Ribbed Smoked Sheets
SC	Smart Contract
SCM	Supply Chain Management
SDK	Software Development Kit
TL	Transaction Latency
TPS	Transactions Per Second
TSR	Technically Specified Rubber
TT	Transaction Throughput

References

1. Saing, C.H. Export competitiveness of the Cambodian rubber sector relative to other Greater Mekong Subregion suppliers: A simple descriptive analysis. *Unpublished Work*, 2009.

2. Official Website of the General Directorate of Rubber - Cambodia. Available online: https://gdr1.maff.gov.kh/?lang=en (accessed on 1 April 2024).
3. Diepart, J.-C.; Kong, R.; Kou, P.; Woods, K.; De Alban, J.D.; Jamaludin, J. Cambodian Smallholder Rubber Sector, 2000 to 2021: Trajectories of Change. Forest Trends: Washington DC, United States, 2023.
4. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; others. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
5. Zheng, Q.; Li, Y.; Chen, P.; Dong, X. An innovative IPFS-based storage model for blockchain. In Proceedings of the 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Santiago, Chile, 3–6 December 2018; pp. 704–708.
6. FAO. Cambodia - fao.org: Cambodia's Investment Opportunities in Agriculture. Available online: https://www.fao.org/hand-in-hand/hih-IF-2023/cambodia/en (accessed on 1 April 2024).
7. World Bank. Cambodia: Share of economic sectors in the gross domestic product (GDP) from 2012 to 2022. World Bank, 2023. Available online: https://www.worldbank.org (accessed on 1 April 2024).
8. Molyvann, B. Contract-Based Agriculture Production: A Case Study of Organic Rice Production in Cambodia. *Unpublished Work*, 2023.
9. Chopra, S.; Meindl, P. *Supply Chain Management. Strategy, Planning & Operation*; Springer: New York, NY, USA, 2007.
10. Ivanov, D.; Dolgui, A.; Sokolov, B. The impact of digital technology and Industry 4.0 on the ripple effect and supply chain risk analytics. *International Journal of Production Research* **2019**, *57*, 829–846.
11. Seuring, S.; Müller, M. From a literature review to a conceptual framework for sustainable supply chain management. *Journal of Cleaner Production* **2008**, *16*, 1699–1710.
12. Nakamoto, S.; Bitcoin, A. A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf* **2008**, *4*, 15.
13. Merkle, R.C. A digital signature based on a conventional encryption function. In Proceedings of the Conference on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, USA, 16–20 August 1987; pp. 369–378.
14. Cachin, C.; Vukolić, M. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873* **2017**.
15. Katz, J.; Lindell, Y. *Introduction to Modern Cryptography: Principles and Protocols*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2007.
16. Pilkington, M. Blockchain technology: principles and applications. In *Research Handbook on Digital Transformations*; Edward Elgar Publishing: Cheltenham, UK, 2016; pp. 225–253.
17. Croman, K.; Decker, C.; Eyal, I.; Gencer, A.E.; Juels, A.; Kosba, A.; Miller, A.; Saxena, P.; Shi, E.; Gün Sirer, E.; others. On Scaling Decentralized Blockchains: (A Position Paper). In Proceedings of the International Conference on Financial Cryptography and Data Security, Christ Church, Barbados, 22–26 February 2016; pp. 106–125.
18. Buterin, V.; others. A next-generation smart contract and decentralized application platform. *White Paper* **2014**, *3*, 37, 2–1.
19. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress), Boston, MA, USA, 11–14 December 2017; pp. 557–564.
20. Hyperledger Fabric. A Blockchain Platform for the Enterprise | Hyperledger Fabric Docs main documentation. Available online: https://hyperledger-fabric.readthedocs.io/en/release-2.5/ (accessed on 1 April 2024).
21. Wood, G.; others. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* **2014**, *151*, 1–32.
22. Antonopoulos, A.M.; Wood, G. *Mastering Ethereum: Building Smart Contracts and Dapps*; O'Reilly Media: Sebastopol, CA, USA, 2018.
23. Zheng, P.; Zheng, Z.; Luo, X.; Chen, X.; Liu, X. A detailed and real-time performance monitoring framework for blockchain systems. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, Gothenburg, Sweden, 27 May–3 June 2018; pp. 134–143.

24. Shalaby, S.; Abdellatif, A.A.; Al-Ali, A.; Mohamed, A.; Erbad, A.; Guizani, M. Performance evaluation of hyperledger fabric. In Proceedings of the 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), Doha, Qatar, 2–5 February 2020; pp. 608–613.
25. Vukolić, M. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In Proceedings of the Open Problems in Network Security: IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, 29 October 2015; pp. 112–125.
26. Sousa, J.; Bessani, A.; Vukolić, M. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In Proceedings of the 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg City, Luxembourg, 25–28 June 2018; pp. 51–58.
27. R3. Corda Technical Whitepaper. Available online: https://www.r3.com/white-papers/corda-technical-whitepaper/ (accessed on 1 April 2024).
28. Belchior, R.; Vasconcelos, A.; Guerreiro, S.; Correia, M. A survey on blockchain interoperability: Past, present, and future trends. *ACM Computing Surveys (CSUR)* **2021**, *54*, 1–41.
29. Ongaro, D.; Ousterhout, J. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.
30. Hyperledger Fabric. Smart Contracts and Chaincode | Hyperledger Fabric Docs main documentation. Available online: https://hyperledger-fabric.readthedocs.io/en/latest/smartcontract/smartcontract.html (accessed on 2 April 2024).
31. Christidis, K.; Devetsikiotis, M. Blockchains and smart contracts for the internet of things. *IEEE Access* **2016**, *4*, 2292–2303.
32. Xu, R.; Chen, Y.; Blasch, E.; Chen, G. Blendcac: A smart contract enabled decentralized capability-based access control mechanism for the IoT. *Computers* **2018**, *7*, 39.
33. Islam, M.; Rehmani, M.H.; Chen, J. Differentially private enhanced permissioned blockchain for private data sharing in industrial IoT. *Information Sciences* **2024**, *658*, 119997. doi: https://doi.org/10.1016/j.ins.2023.119997. Available online: https://www.sciencedirect.com/science/article/pii/S0020025523015827 (accessed on 1 April 2024).
34. Dwork, C.; McSherry, F.; Nissim, K.; Smith, A. Calibrating noise to sensitivity in private data analysis. In Proceedings of the Theory of Cryptography Conference, New York, NY, USA, 4–7 March 2006; pp. 265–284.
35. Bruckman, V.J.; Haruthaithanasan, M.; Miller, R.O.; Terada, T.; Brenner, A.-K.; Kraxner, F.; Flaspohler, D. Sustainable forest bioenergy development strategies in Indochina: Collaborative effort to establish regional policies. *Forests* **2018**, *9*, 223.
36. Pradana, I.G.M.T.; Djatna, T.; Hermadi, I. Blockchain modeling for traceability information system in supply chain of coffee agroindustry. In Proceedings of the 2020 International Conference on Advanced Computer Science and Information Systems (ICACSIS), Bali, Indonesia, 17–18 October 2020; pp. 217–224.