BLOCKCHAIN TECHNOLOGY

Assignment – 4

Implementation and Security Analysis of a Blockchain-Based Supply Chain Using Hyperledger Fabric

Abstract

Blockchain offers an immutable, auditable, and decentralized substrate for data integrity and transparent coordination across multiple stakeholders. This report details the design, deployment, and evaluation of a **permissioned** supply-chain network using **Hyperledger Fabric**. The implementation models three organizations—**Manufacturer**, **Distributor**, and **Retailer**—connected through both **public** and **private** channels. Core processes (product creation, shipment, and receipt) are governed by chaincode with **MSP-based access control**. We additionally present a structured security analysis across blockchain layers and outline concrete mitigations. Results indicate that a properly configured Fabric network preserves **confidentiality, integrity, and availability** of supply-chain data and reduces fraud and unauthorized manipulation.

Introduction

Modern supply chains struggle with opacity, counterfeits, and data tampering. By design, **Hyperledger Fabric** brings **traceability, provenance**, and **immutability** to inter-organizational workflows while preserving enterprise privacy through **channels** and **policies**. In this project, Fabric is used to connect manufacturers, distributors, and retailers. Participants interact via **chaincode (smart contracts)** to issue shipments, transfer custody, and verify deliveries. The report also examines typical security pitfalls and defenses to ensure a resilient, realistic deployment.

Task 1: Blockchain-Based Supply Chain Implementation

Goal. Build a permissioned Fabric network enabling decentralized supply-chain tracking across three orgs: **Org1 (Manufacturer)**, **Org2 (Distributor)**, **Org3 (Retailer)**. We use two channels:

- **Public channel**: `main-supply` — visible to all orgs.
- **Private channel**: `manu-dist` — restricted to Org1 and Org2 for sensitive shipment operations.

Platform. The network runs (in our implementation) on Ubuntu 22.04 with **Docker Engine** and Fabric **v2.5** binaries. Containers are orchestrated with **Docker Compose**. Fabric's **test-network** serves as the baseline for orderer/peer bootstrapping.

Automation. Helper scripts streamline lifecycle tasks:

- `start_testnet.sh` – boot orderer/peers and create the public channel
- `create_manu_dist.sh` – create/join the private channel
- `deploy_cc_main.sh`, `deploy_cc_manudist.sh` – package, approve, and commit chaincode to respective channels

Table 1. Summary of main network components

Component	Description
Org1 (Manufacturer)	Creates products and initiates shipment records.
Org2 (Distributor/Wholesaler)	Receives shipments and updates product ownership.
Org3 (Retailer)	Final recipient; performs verification.

Process (high-level).

1. **Manufacturer** emits a shipment record with product metadata (product ID, batch, timestamp). The write occurs on **`manu-dist`** and is visible only to Org1/Org2.
2. **Distributor** verifies and accepts custody; chaincode updates product state and logs an immutable event.
3. **Retailer** acknowledges final receipt; queries confirm end-to-end traceability.
4. **Access control** is enforced by **MSP identities**; unauthorized parties cannot read/modify restricted data. **Endorsement + validation** ensure only properly endorsed transactions commit.

Chaincode. The JavaScript chaincode **`trackchain-js`** is deployed to both channels and implements:

- `CreateProduct()`, `CreateShipment()`, `ReceiveShipment()`, `GetProduct()`, `GetProductHistory()`

Endorsement.

- On `main-supply`: any org can endorse (course-scale policy).
- On `manu-dist`: endorsements require **Org1** and **Org2** (tight control).

Simulated transaction flow (peer CLI).

```
[Org1] → peer chaincode invoke ... Args:[CreateProduct, P200,
SKU-ALPHA, "Smart Sensor Module"]
[Org1] → peer chaincode invoke ... Args:[CreateShipment, S200, P100,
Org2MSP]
[Org2] → peer chaincode invoke ... Args:[ReceiveShipment, S100]
[Org3] → peer chaincode invoke ... Args:[ReceiveShipment, S200]  →
Authorization denied
[Query] → peer chaincode query ... Args:[GetProduct, P200]
[Query] → peer chaincode query ... Args:[GetProductHistory, P200]
```

Observed behavior. Product lifecycle state changes (create → ship → receive) were immutably recorded. Unauthorized Org3 attempts to accept a shipment were **rejected** by MSP checks, confirming correct authorization. History queries returned ordered transitions with tx IDs and timestamps.

Security defaults used.

- **TLS** on all peer–orderer links (confidentiality in transit)
- **Lifecycle approvals & endorsement policies** (prevent unapproved chaincode)
- **Fabric CA** issues identities for org roles (prevents spoofing)

Task 2: Security Threat Analysis and Mitigation Strategies

We assessed risks across **network, consensus, transaction, and application** layers using a STRIDE-inspired approach and blockchain-specific guidance.

Table 2. Identified threats and mitigations

Threat	Impact	Mitigation Strategy
Double Spending / Replay	Duplicate/inconsistent state	Fabric **MVCC** & unique **tx IDs** prevent replays from committing.
Sybil Attack	Fake nodes influence consensus	**Permissioned MSP** membership; nodes join only with valid org identities.
Smart Contract Exploits	Malicious/buggy logic	**Endorsement policies**, **code review**, **lifecycle approvals** (orgs attest to package hash).
MITM on Network	Spoofed/inspected traffic	**TLS** and mutual TLS between peers/orderers; CA-rooted trust.

We conceptually applied **OWASP** and Microsoft **Threat Modeling Tool** practices to data-flow diagrams and trust boundaries. Fabric's **channels** and **endorsement** are primary defenses at network/transaction layers.

Attack example. An attacker attempts `ReceiveShipment` for a shipment not addressed to its MSP. The chaincode checks **invoker's MSP** at runtime and rejects with: *"Only the intended receiver can accept this shipment."* This validates fine-grained authorization.

Recommended hardening.

- Multi-node **Raft** for fault tolerance
- **HSMs** for private-key protection
- **Certificate rotation** via Fabric CA
- **Private Data Collections (PDCs)** for highly confidential fields
- Continuous **chaincode audits** and runtime monitoring
- Principle of **least privilege** for admin operations

Conclusion

This work demonstrates a practical integration of **Hyperledger Fabric** into supply-chain workflows using a three-org topology and dual-channel design. **Endorsement** and **MSP-based checks** ensure that only authorized participants can mutate or view sensitive state, while on-chain history guarantees provenance. The security analysis highlights Fabric's strong baseline controls and emphasizes operational best practices for production. Overall, blockchain adds verifiable, tamper-evident, and privacy-preserving data exchange to enterprise supply chains.

References

[1] Hyperledger Foundation, "Hyperledger Fabric v2.5 Documentation," 2024.
[2] M. Androulaki et al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," *EuroSys*, 2018.
[3] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.
[4] OWASP Foundation, "Blockchain Security Threat Modeling Guide," 2023.
[5] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in Internet of Things: Challenges and Solutions," *IEEE Communications Surveys & Tutorials*, 22(3), 2020.

Name- Utkarsh Dubey
Registration No.- 220911608