UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

RELAZIONE DI PROGETTO

Sistema Oracolo Bayesiano per Catena del Freddo Farmaceutica

Bayesian Oracle System for Pharmaceutical Cold Chain

Professore

Luca Spalazzi

Studenti

Luigi Greco

Andrea Altieri

Filippo Marchegiani

Luca Belardinelli

ANNO ACCADEMICO 2025-2026

Sommario

La catena del freddo farmaceutica rappresenta un processo critico in cui la garanzia dell'integrità dei prodotti è fondamentale per la salute pubblica. I sistemi tradizionali di monitoraggio spesso soffrono di problemi di centralizzazione, opacità e mancanza di automatismi affidabili.

Questa relazione propone un'architettura innovativa basata su Blockchain e Intelligenza Artificiale per l'automazione sicura e trasparente della validazione delle spedizioni. La soluzione implementata utilizza Smart Contract su rete Ethereum/Hyperledger Besu per orchestrare il processo di business e integra una Rete Bayesiana on-chain come oracolo decisionale. Questo permette di inferire probabilistically la conformità della spedizione a partire da evidenze parziali o rumorose provenienti da sensori IoT, garantendo che i pagamenti vengano sbloccati solo a fronte di condizioni verificate.

Il sistema è stato progettato seguendo un approccio Security-by-Design, adottando la metodologia DUAL-STRIDE-DUA per l'analisi delle minacce e l'identificazione di vulnerabilità sia intenzionali che accidentali. La validità e la resilienza della soluzione sono state verificate attraverso test distribuiti e analisi statica del codice, dimostrando come l'integrazione di logica probabilistica su blockchain possa offrire un livello superiore di sicurezza e fiducia nei processi logistici critici.

Parole chiave: Blockchain, Ethereum, Smart Contract, Rete Bayesiana, Supply Chain, Sicurezza, Threat Modeling, IoT

Indice

Introduzione

La gestione della catena del freddo (Pharmaceutical Cold Chain) rappresenta una delle sfide più critiche nel settore logistico sanitario. Il trasporto di farmaci termosensibili, come vaccini e insulina, richiede il mantenimento rigoroso di specifici range di temperatura (tipicamente 2°C - 8°C) lungo l'intera filiera. Deviazioni anche minime possono compromettere l'efficacia del prodotto, con conseguenze potenzialmente letali per i pazienti e ingenti danni economici per le aziende.

Attualmente, la trasparenza di questo processo è limitata dall'uso di sistemi centralizzati e trust-based, dove le informazioni sono spesso frammentate, cartacee o custodite in silos informatici proprietari. Questo scenario rende difficile ricostruire con certezza la "storia termica" di un lotto e lascia spazio a possibili manipolazioni dei dati per coprire errori logistici o negligenze.

In questo contesto, la tecnologia Blockchain offre un cambio di paradigma fondamentale, passando dalla "fiducia negli attori" alla "fiducia nel protocollo". Attraverso un registro distribuito, immutabile e trasparente, è possibile garantire che ogni misurazione registrata sia autentica e non repudiabile. Tuttavia, la blockchain da sola non può verificare la veridicità del dato fisico prima che venga scritto ("Garbage In, Garbage Out").

Questo progetto propone una soluzione ibrida che integra Hyperledger Besu, una blockchain permissioned adatta a contesti enterprise, con un sistema di **Oracoli Bayesiani**. L'approccio innovativo risiede nell'utilizzare l'inferenza probabilistica on-chain per validare la coerenza delle letture multisensoriali (temperatura, umidità, shock, luce, integrità sigillo) prima di finalizzare la transazione di consegna. In questo modo, il sistema non si limita a registrare i dati, ma agisce come un decisore autonomo capace di accettare o rifiutare un lotto in base a politiche di rischio matematicamente definite.

Questa tesi illustra il design, l'implementazione e la verifica di tale architettura sicura, mettendo in luce come l'integrazione tra DLT (Distributed Ledger Technology) e metodi formali possa elevare gli standard di sicurezza e affidabilità nella logistica farmaceutica 4.0.

CAPITOLO 1

Contesto e Obiettivi

In questo capitolo viene analizzato il dominio della Pharmaceutical Cold Chain, evidenziando le criticità di sicurezza attuali. Vengono quindi definiti gli obiettivi del progetto: automazione fidata, integrità dei dati e resilienza agli attacchi.

1.1 Il problema della Catena del Freddo

La spedizione di medicinali sensibili è un processo ad alto rischio. Secondo l'Organizzazione Mondiale della Sanità (OMS), una percentuale significativa di vaccini viene sprecata ogni anno a causa di interruzioni nella catena del freddo. I problemi principali sono:

- **Mancanza di visibilità end-to-end**: I dati di transito sono spesso disponibili solo a posteriori.
- **Conflitto di interessi**: Il trasportatore, responsabile del mantenimento della temperatura, è spesso anche colui che fornisce i dati di monitoraggio, creando un incentivo alla manipolazione in caso di guasti.
- **Silos informativi**: Produttori, distributori e farmacie utilizzano sistemi ERP diversi che non comunicano in tempo reale.

1.2 Obiettivi del Progetto

Il sistema proposto mira a risolvere queste problematiche attraverso tre pilastri fondamentali:

1.2.1 1. Automazione tramite Smart Contract

Eliminare l'intermediazione umana e burocratica nei processi di verifica e pagamento. Il contratto intelligente (Smart Contract) agisce come un deposito a garanzia (Escrow), sbloccando i fondi al corriere solo se tutte le condizioni di qualità sono matematicamente soddisfatte.

1.2.2 2. Sicurezza del Dato (Data Integrity)

Garantire che, una volta acquisito, il dato non possa essere alterato (Tamper-Proof). Questo è assicurato dalla crittografia sottostante la blockchain e dal meccanismo di consenso IBFT 2.0 di Hyperledger Besu.

1.2.3 3. Validazione Logica (Data Validity)

Garantire che il dato acquisito rifletta la realtà. Qui interviene l'Oracolo Bayesiano, che correla letture diverse (es. "Temperatura Alta" + "Sigillo Rotto" + "Luce Rilevata") per calcolare la probabilità posteriore di un evento avverso, distinguendo tra falsi positivi dei sensori e reali compromissioni del carico.

CAPITOLO 2

Analisi e Progettazione Architetturale

Questo capitolo dettaglia le scelte architetturali adottate per soddisfare i requisiti di sicurezza. Si analizzano le tecnologie selezionate (Hyperledger Besu, Truffle), il design del sistema distribuito e l'architettura a tre livelli (Blockchain, Oracle Middleware, Web UI).

2.1 Architettura Distribuita a Tre Livelli

Il sistema è basato su un'architettura decentralizzata che interagisce con componenti off-chain per garantire usabilità e connessione con il mondo fisico.

2.1.1 Livello Blockchain (Data & Logic Layer)

Il cuore del sistema è una rete privata basata su **Hyperledger Besu**.

- **Ruolo**: Mantiene il registro immutabile delle transazioni (Ledger) e ospita la logica di business (Smart Contracts).
- **Componenti**: Nodi validatori, Smart Contract `BNCore`, `BNGestoreSpedizioni` e `BNPagamenti`.

2.1.2 Livello Middleware (Oracle Layer)

Poiché la blockchain è un sistema chiuso che non può accedere a dati esterni (internet/-sensori), è necessario un componente "ponte".

- **Componente**: Script Node.js `simula_oracolo.js`.
- **Funzione**: Questo script agisce da bridge. Simula l'acquisizione dati dai sensori IoT (Temperatura, Umidità, Shock, Luce, Sigillo), esegue una pre-validazione opzionale e invia le "evidenze" allo Smart Contract tramite transazioni firmate dal `RUOLO_SENSORE`.

2.1.3 Livello Presentazione (Web Interface)

L'interfaccia utente permette agli attori umani di interagire col sistema senza dover usare riga di comando.

- **Tecnologia**: Single Page Application (SPA) HTML5/JS connessa via Web3.js.

- **Ruolo**: Dashboard per la creazione spedizioni, monitoraggio real-time e gestione rimborsi.

2.2 Focus Tecnologico: Hyperledger Besu

La scelta di Hyperledger Besu rispetto ad altre soluzioni (es. Geth, Hyperledger Fabric) è stata guidata da specifici requisiti di sicurezza enterprise.

2.2.1 Consenso IBFT 2.0 (Istanbul Byzantine Fault Tolerance)

A differenza del Proof-of-Work (costoso e lento) o Proof-of-Authority semplice, IBFT 2.0 offre:

- **Finalità Immediata**: Una volta che un blocco è scritto, non può essere riorganizzato (niente "fork"). Questo è critico per la supply chain: una consegna registrata non può "sparire".
- **Tolleranza ai Guasti Bizantini**: Il sistema continua a funzionare correttamente anche se fino a f nodi su N sono malevoli o offline, dove $N \geq 3f + 1$. Nella nostra configurazione a 4 nodi, il sistema resiste alla compromissione completa di 1 nodo validatore senza perdere integrità o disponibilità.

2.2.2 Permissioning Avanzato

Besu permette di definire una "Allowlist" di nodi e account a livello di protocollo.

- **Node Whitelisting**: Solo i nodi certificati (es. appartenenti a Produttore e Distributore) possono partecipare al consenso e sincronizzare la blockchain.
- **Smart Contract Permissions**: L'accesso alle funzioni critiche è limitato a livello applicativo (tramite libreria AccessControl di OpenZeppelin), distinguendo ruoli come `ADMIN`, `MITTENTE` e `SENSORE`.

2.3 Principi di Design Sicuro (Saltzer & Schroeder)

L'architettura rispetta i principi fondamentali della sicurezza:

- **Economy of Mechanism (Semplicità)**: I contratti sono modulari. `BNCore` fa solo matematica, `BNGestore` gestisce i processi. Meno codice = meno bug.
- **Open Design**: La sicurezza non si basa sull'oscurità. Il codice è pubblico e verificabile; la sicurezza deriva dalla crittografia e dalla matematica del consenso.
- **Fail-Safe Defaults**: Se una condizione non è verificata (es. evidenze mancanti), lo stato di default è "Blocco dei fondi" o "Rifiuto transazione", mai "Accettazione implicita".
- **Separation of Privilege**: Per sbloccare un pagamento servono due condizioni distinte: l'invio delle evidenze (dal Sensore) e la verifica probabilistica (dal Contratto). Nessun singolo attore ha il potere totale.

2.4 Analisi di Resistenza, Sopravvivenza e Ambiguità

- **Resistenza**: L'uso di crittografia asimmetrica rende impossibile la falsificazione delle firme digitali dei sensori.
- **Sopravvivenza (Resilienza)**: La natura distribuita del ledger assicura che i dati siano replicati su tutti i nodi. Un attacco DDoS verso un singolo nodo non ferma il servizio.
- **Ambiguità (Obfuscation/Privacy)**: Il sistema adotta un approccio ibrido "Privacy by Design".
  - *Logica Pubblica*: I calcoli probabilistici sono trasparenti per garantire l'audit.
  - *Dati Sensibili Offuscati*: I dettagli personali (nomi, lotti farmaceutici) non sono salvati in chiaro on-chain. Viene memorizzato solo un **Hash crittografico** ('hashedDetails') che permette la verifica di integrità senza rivelare il contenuto a osservatori non autorizzati (Off-chain Data Storage).

CAPITOLO 3

Valutazione del Rischio e Threat Modeling

In questo capitolo viene presentata un'analisi approfondita della sicurezza del sistema, condotta attraverso metodologie formali e strutturate. L'analisi inizia con la modellazione degli obiettivi e delle dipendenze strategiche tramite framework **i* (iStar)**, prosegue con la valutazione delle minacce mediante approccio **DUAL-STRIDE** esteso agli asset dell'attore sistema, e si conclude con la definizione di scenari di **Abuse e Misuse Cases**.

3.1 Modellazione i* (iStar)

Per comprendere appieno il contesto organizzativo e tecnico, sono stati realizzati diversi modelli i*, che evidenziano attori, obiettivi (Goals), compiti (Tasks) e risorse (Resources).

3.1.1 Supply Chain As-Is (Senza Sistema)

Il primo modello Strategic Dependency (SD) rappresenta la supply chain tradizionale.

- **Attori**: Produttore, Distributore, Farmacia, Paziente.
- **Criticità**: L'analisi SR (Strategic Rationale) evidenzia dipendenze di "fiducia cieca" tra gli attori riguardo l'integrità della temperatura. Il Produttore dipende dal Distributore per la corretta conservazione, ma non ha mezzi diretti di verifica (Softgoal "Integrità non verificabile").

3.1.2 Supply Chain To-Be (Con Sistema Blockchain)

L'introduzione del sistema introduce nuove dipendenze strategiche più robuste.

- **Nuovi Attori**: Sistema Smart Contract, Oracolo IoT.
- **Vantaggi**: Il Softgoal "Integrità Verificabile" è ora soddisfatto dalla risorsa "Registro Immutabile" fornita dal sistema. Gli attori umani dipendono dal Sistema per la validazione, non più dalla fiducia reciproca.

3.1.3 Sistema e Attaccanti

Sono stati modellati tre profili di attaccante interagenti con il sistema:

1. **Attaccante Interno (Malicious Insider)**: Un operatore logistico corrotto che tenta di manipolare i sensori fisici.
2. **Attaccante Esterno**: Un hacker remoto che tenta attacchi di rete (DoS, intercettazione) o exploit sugli Smart Contract.
3. **Utente Maldestro (Clumsy User)**: Un operatore che commette errori non intenzionali (es. perdita chiavi private, input errati).

Per ciascun attaccante, i diagrammi SR includono **Alberi di Attacco (Attack Trees)** integrati, che mostrano la decomposizione degli obiettivi malevoli (es. "Falsificare Report Temperatura") in sotto-task operativi.

3.2 Analisi DUAL-STRIDE

L'analisi delle minacce è stata condotta metodicamente raggruppando gli asset secondo il paradigma DUAL-STRIDE, focalizzandosi specificamente sugli asset dell'**Attore Sistema**.

3.2.1 Identificazione degli Asset

Gli asset primari analizzati sono:

- **Smart Contract (Logica)**: Codice Solidity distribuito.
- **Dati della Blockchain (Ledger)**: Storico transazioni e stati.
- **Credenziali (Chiavi Private)**: Chiavi dei nodi validatori e degli utenti.
- **Oracolo (Infrastruttura IoT)**: Ponte tra mondo fisico e digitale.

3.2.2 Matrice delle Minacce (Riferimenti CAPEC/ATT&CK)

Per ogni categoria STRIDE sono stati identificati vettori di attacco specifici, mappati sui framework standard **CAPEC** e **MITRE ATT&CK**.

STRIDE	Minaccia Identificata	Rif. CAPEC
Spoofing	Impersonificazione di un nodo validatore	CAPEC-151 (Identity Spoofing)
Tampering	Modifica dati sensore pre-invio (Data Injection)	CAPEC-155 (Screen/Data Capture)
Repudiation	Negazione di avvenuta consegna del lotto	CAPEC-390 (Bypassing Checks)
Information Disc.	Lettura transazioni private (Analisi traffico)	CAPEC-118 (Traffic Analysis)
Denial of Service	Spam di transazioni per bloccare la rete	CAPEC-488 (HTTP Flood/Gas Limit)
Elevation of Priv.	Sfruttamento bug in AccessControl	CAPEC-233 (Privilege Escalation)

Tabella 3.1: Analisi STRIDE sugli Asset del Sistema

3.3 Abuse e Misuse Cases

Per completare l'analisi, sono stati definiti scenari operativi di abuso per ogni asset critico.

3.3.1 Abuse Case: Iniezione Dati Falsi (Attaccante Interno)

- **Attore**: Insider Logistico.
- **Obiettivo**: Nascondere un'escursione termica per evitare penali.
- **Asset**: Oracolo IoT.
- **Scenario**: L'attaccante manomette fisicamente il sensore o inietta pacchetti MQTT falsificati verso l'Oracolo.
- **Mitigazione**: Validazione Bayesiana per rilevare incongruenze statistiche (v. Cap. 4).

3.3.2 Misuse Case: Smarrimento Chiave Privata (Utente Maldestro)

- **Attore**: Farmacista.
- **Evento**: L'utente cancella accidentalmente il file keystore o lo condivide su canali non sicuri.
- **Conseguenza**: Perdita di accesso ai fondi o furto d'identità.
- **Mitigazione**: Procedure di key-recovery off-chain (non implementate on-chain per scelta di design) e formazione operativa.

CAPITOLO 4

Programmazione Sicura e Dettagli Implementativi

In questo capitolo viene analizzata nel dettaglio l'implementazione del sistema, coprendo l'intero stack: dai Smart Contract Solidity, passando per la logica di simulazione Oracle, fino all'Interfaccia Web utente.

4.1 Smart Contract e Logica On-Chain

Il backend decentralizzato è costituito dai contratti `BNCore` e `BNGestoreSpedizioni`, che implementano la logica di business e di sicurezza.

4.1.1 BNCore: Il Motore Inferenziale

Il contratto `BNCore` agisce come "cervello" matematico. Implementa una Rete Bayesiana statica dove:

- **Fatti (Nodi Root)**: F_1 (Temperatura Conforme), F_2 (Integrità Fisica).
- **Evidenze (Nodi Foglia)**: $E_1 \dots E_5$ (letture sensori).

Poiché Solidity non gestisce i float, le probabilità sono gestite come interi (base 100). Il calcolo della probabilità combinata avviene *on-chain* per garantire trasparenza: tutti possono verificare perché una spedizione è stata accettata o rifiutata.

4.1.2 BNGestoreSpedizioni: Sicurezza Operativa

Gestisce il ciclo di vita... (come sopra). [...existing code listing...]

4.1.3 BNPagamenti: L'Attuatore Finanziario

Questo contratto estende `BNGestoreSpedizioni` per isolare la logica critica di pagamento.

- **Responsabilità**: Esegue la funzione `validaEPaga()`, che incrocia i dati del ledger con le probabilità calcolate da `BNCore`.
- **Sicurezza**: Implementa `ReentrancyGuard` per prevenire attacchi durante il trasferimento di Ether.

```
function validaEPaga(uint256 _id) external nonReentrant {
    // ... checks ...
    (uint256 pF1, uint256 pF2) = _calcolaProbabilitaPosteriori(s.evidenze);

    // SAFETY MONITOR S4: Probability Threshold
    if (pF1 < SOGLIA || pF2 < SOGLIA) {
        emit MonitorSafetyViolation("Threshold", _id, msg.sender, "Non conforme");
        emit TentativoPagamentoFallito(_id, ...);
        return; // Fail-safe: non paga
    }

    // GUARANTEE MONITOR G1: Payment Success
    s.stato = StatoSpedizione.Pagata;
    (bool success, ) = s.corriere.call{value: s.importoPagamento}("");
    require(success, "Transfer fallito");
    emit MonitorGuaranteeSuccess("PaymentExecuted", _id);
}
```

Listing 4.1: Runtime Monitor in BNPagamenti (validaEPaga)

4.1.4 Privacy e Offuscamento Dati

Per mitigare la trasparenza totale della blockchain pubblica, è stato implementato un pattern di **On-Chain Hashing**. I dati sensibili (es. farmaco, destinazione) non vengono salvati sullo Smart Contract.

1. Il mittente calcola $H = \text{Keccak256}(\text{JSON Dettagli})$ off-chain.
2. Invoca `creaSpedizioneConHash(..., H)`.
3. Solo chi possiede il JSON originale può verificare la corrispondenza chiamando `verificaDettagl JSON)`.

4.2 Sistema Oracolo e Simulazione IoT

Il ponte tra mondo fisico e blockchain è gestito dallo script `simula_oracolo.js`. Questo componente è fondamentale perché la blockchain non può interrogare direttamente i sensori.

4.2.1 Flow del Dato (Sensore → Blockchain)

1. **Generazione**: Lo script genera valori casuali per i 5 sensori (Temperatura, Umidità, Shock, Luce, Sigillo), simulando scenari normali (90% probabilità) o di guasto.
2. **Firma**: Ogni lettura viene impacchettata in una transazione firmata dalla chiave privata del `RUOLO_SENSORE`.
3. **Invio**: Le transazioni invocano `inviaEvidenza(id, tipo, valore)` sullo smart contract.

```
// Logica simulata: il 'sensore' rileva valori corretti casualmente
function simulaSensore() { return Math.random() < 0.9; }

// Loop di invio evidenze
const E1_Temp = simulaSensore();
await contratto.methods.inviaEvidenza(id, 1, E1_Temp)
    .send({ from: indirizzoSensore }); // Firma crittografica
```

Listing 4.2: Simulazione IoT e Invio dati (simula_oracolo.js)

4.3 Interfaccia Web (Dashboard Utente)

L'interazione umana avviene tramite una DApp (Decentralized App) Web, progettata per offrire esperienze diverse in base al ruolo dell'utente connesso (rilevato tramite MetaMask).

4.3.1 Ruolo: Mittente (Sender)

Il Mittente (es. casa farmaceutica) è l'iniziatore del processo.

- **Nuova Spedizione**: Compila un form indicando l'indirizzo Ethereum del corriere e l'importo da bloccare in deposito (Escrow).
- **Operazione**: Al click su "Crea", Web3.js apre MetaMask per confermare la transazione e depositare gli Ether.
- **Monitoraggio**: Visualizza una lista delle proprie spedizioni con stato in tempo reale (In Transito, Consegnata, Rimborsata).

4.3.2 Ruolo: Corriere (Carrier)

Il trasportatore ha accesso in "sola lettura" operativa ma con interesse economico.

- **Tracking**: Visualizza le spedizioni a lui assegnate.
- **Notifiche**: Riceve aggiornamenti sullo stato delle evidenze caricate dai sensori.
- **Incasso**: Se la validazione Bayesiana ha successo, vede lo sblocco automatico dei fondi sul proprio wallet.

4.3.3 Ruolo: Admin/Sensore (IoT Simulator)

Nella demo, l'interfaccia permette anche di "triggerare" manualmente l'invio delle evidenze (funzione di debug) per vedere come reagisce il contratto.

- **Pannello Sensori**: Visualizza toggle switch per ogni sensore (E1-E5).
- **Invio Forzato**: Permette di inviare una configurazione specifica (es. "Tutto OK tranne Temperatura") per testare la robustezza della validazione.

4.4 Integrazione Web3 e Gestione Eventi

Il frontend non fa polling continuo ma reagisce agli **Eventi** emessi dallo Smart Contract. Quando `BNCore` emette l'evento `ProbabilitaValidazione`, l'interfaccia aggiorna immediatamente i grafici e lo stato, offrendo un'esperienza reattiva.

```
contratto.events.EvidenceReceived()
    .on('data', function(event){
        console.log("Nuova evidenza ricevuta:", event.returnValues);
        updateUIProressBar(event.returnValues.shipmentId);
    });
```

Listing 4.3: Ascolto Eventi in Web3.js

CAPITOLO 5

Verifica, Validazione e Modellazione Formale

In questo capitolo vengono esposti i risultati delle attività di verifica e validazione. Si descrivono i test funzionali eseguiti sulla rete Hyperledger Besu e la modellazione formale di unit critiche tramite Catene di Markov (PRISM) per la verifica di proprietà di Safety e Guarantee.

5.1 Verifica Formale con PRISM

Per garantire la robustezza logica del sistema rispetto alle minacce identificate (Spoofing e Tampering), è stato modellato il comportamento probabilistico dell'unità "Sensore-Oracolo" utilizzando il model checker **PRISM**. L'analisi si basa su Catene di Markov a Tempo Discreto (DTMC) e segue una metodologia comparativa: viene prima analizzato il sistema vulnerabile (senza contromisure) e successivamente il sistema protetto (con Active Defense, TPM e Ridondanza).

5.1.1 Introduzione: Obiettivo della Modellazione

Contesto del Sistema

Il sistema oggetto di questa analisi è un sistema di monitoraggio IoT per la supply chain composto da cinque sensori critici che monitorano lo stato di merci durante il trasporto. L'architettura del sistema comprende il sensore E1 per il monitoraggio della temperatura, il sensore E2 per il controllo del sigillo, il sensore E3 per il rilevamento degli shock meccanici, il sensore E4 per il monitoraggio della luce, e il sensore E5 per la scansione all'arrivo. Questi componenti sono stati progettati per garantire l'integrità e la tracciabilità dei dati durante l'intera catena logistica.

Scopo dell'Analisi di Markov Chain

L'obiettivo di questa analisi è modellare formalmente il comportamento probabilistico del sistema di sensori utilizzando Discrete-Time Markov Chains (DTMC). L'approccio metodologico persegue tre obiettivi principali: quantificare l'efficacia delle contromisure di sicurezza implementate attraverso l'analisi DUAL-STRIDE, verificare formalmente proprietà di Safety e Guarantee/Response utilizzando il model checker PRISM, e confrontare quantitativamente il sistema con e senza contromisure per dimostrare l'impatto delle misure di sicurezza adottate.

Minacce Modellate

L'analisi DUAL-STRIDE ha identificato come critiche per l'integrità del sistema due principali minacce appartenenti alla tassonomia STRIDE. La prima è Spoofing (S2.1), in cui un sensore falso può iniettare dati malevoli nel sistema compromettendone l'autenticità. La seconda è Tampering (T2.1), che consiste nella manomissione fisica dei sensori con conseguente alterazione delle letture. Tali minacce rappresentano vettori di attacco significativi che possono compromettere la confidenzialità e l'integrità dei dati raccolti.

Contromisure Implementate

Il sistema di sicurezza implementato si basa su tre pilastri fondamentali. Il primo consiste in Device Attestation basato su Trusted Platform Module (TPM) combinato con Mutual TLS, che fornisce autenticazione bilaterale e blocca efficacemente attacchi di tipo Spoofing. Il secondo pilastro è rappresentato dalla Sensor Redundancy, che mitiga i rischi derivanti da Tampering fisico mediante ridondanza hardware. Il terzo pilastro è costituito dall'Active Defense System, un sistema di difesa attivo composto da tre componenti: Intrusion Detection System (IDS) per il rilevamento dei tentativi di attacco, Rate Limiting per il conteggio dei fallimenti di autenticazione, e System Lock che blocca permanentemente il sensore dopo tre tentativi di attacco consecutivi.

5.1.2 Modello PRISM: Sistema SENZA Contromisure

Struttura del Modello

Il modello PRISM rappresenta il sistema prima dell'implementazione delle contromisure DUAL-STRIDE, evidenziando la vulnerabilità intrinseca agli attacchi. Il modello è dichiarato come Discrete-Time Markov Chain (DTMC), in cui il tempo avanza in step discreti e le transizioni tra stati seguono distribuzioni probabilistiche definite.

```
dtmc
```

Questa dichiarazione specifica che il modello adotta una Discrete-Time Markov Chain (DTMC), dove il tempo avanza in step discreti e le transizioni sono governate da probabilità.

```
module sensor_system_vulnerable

    e1 : [0..2] init 0;  // Sensore E1: Temperatura
    e2 : [0..2] init 0;  // Sensore E2: Sigillo
    e3 : [0..2] init 0;  // Sensore E3: Shock
    e4 : [0..2] init 0;  // Sensore E4: Luce
    e5 : [0..2] init 0;  // Sensore E5: Scan Arrivo

    time : [0..200] init 0;
```

Ogni sensore è modellato attraverso una variabile di stato con dominio [0..2], dove lo stato 0 corrisponde al sensore funzionante e sicuro (OK), lo stato 1 rappresenta un guasto hardware non derivante da compromissione (FAILED), e lo stato 2 indica un sensore sotto attacco riuscito (COMPROMISED). Il modello include inoltre un contatore temporale che avanza da 0 a 200 step, definendo l'orizzonte temporale dell'analisi. Una caratteristica distintiva di questo modello vulnerabile è l'assenza del contatore `e1_attempts` e del flag `e1_locked`, indicando che non sono implementati meccanismi di IDS, Rate Limiting o Active Defense. Tutti i sensori inizializzano nello stato OK (`init 0`) per consentire un confronto equo con il modello protetto.

Matrice di Transizione: Sistema SENZA Contromisure

La seguente matrice mostra le probabilità di transizione per un singolo sensore senza contromisure:

Da Stato ↓ / A Stato →	OK (0)	FAILED (1)	COMPR. (2)
OK (0)	0.80	0.05	**0.15**
FAILED (1)	0.60	0.30	**0.10**
COMPR. (2)	0.00	0.00	1.00

Tabella 5.1: Matrice di Transizione - Sistema Vulnerabile

Dallo stato OK, il sensore ha una probabilità dell'80% di rimanere operativo in assenza di eventi, una probabilità del 5% di transizione verso lo stato FAILED dovuta a guasti hardware naturali, e una probabilità critica del 15% di transizione verso lo stato COMPROMISED, dovuta ad attacchi riusciti (Spoofing 5% + Tampering 10%). Dallo stato FAILED, il recovery manuale presenta una probabilità del 60%, mentre vi è una probabilità del 30% che il sensore rimanga guasto e una preoccupante probabilità del 10% di compromissione, indicando una maggiore vulnerabilità dei sensori in stato di guasto. Lo stato COMPROMISED costituisce uno stato assorbente.

Le osservazioni critiche evidenziano un'alta probabilità di compromissione (15% da OK e 10% da FAILED), la natura assorbente dello stato compromesso che impedisce qualsiasi forma di recupero, e un recovery lento caratterizzato da una probabilità di solo 60% dalla condizione di guasto.

Diagramma di Stati: Sistema SENZA Contromisure (Rappresentazione Testuale)

1. **OK**: Stato operativo normale. Transita a:
   - OK (80%): Nessun evento.
   - FAILED (5%): Guasto naturale.
   - COMPROMISED (15%): Attacco riuscito.
2. **FAILED**: Guasto hardware. Transita a:
   - OK (60%): Recovery manuale.
   - FAILED (30%): Rimane guasto.
   - COMPROMISED (10%): Attacco su sensore guasto.
3. **COMPROMISED**: Stato assorbente. Transita a:
   - COMPROMISED (100%): Nessun recovery possibile.

Logica delle Transizioni: Sistema Vulnerabile

```
1    [] e1=0 & time<200 ->
2        0.80 : (e1'=0) & (time'=time+1) +     // Rimane OK
3        0.05 : (e1'=1) & (time'=time+1) +     // Guasto naturale
4        0.15 : (e1'=2) & (time'=time+1);      // ATTACCO RIUSCITO
```

Questa regola di transizione codifica tre possibili esiti per un sensore nello stato OK. Con probabilità 80% il sensore rimane operativo, con probabilità 5% si verifica un guasto hardware naturale, e con probabilità 15% si verifica un attacco riuscito che porta il sensore allo stato COMPROMISED. Quest'ultima transizione rappresenta il punto critico del sistema vulnerabile.

```
    [] e1=1 & time<200 ->
        0.60 : (e1'=0) & (time'=time+1) +     // Recovery manuale
        0.30 : (e1'=1) & (time'=time+1) +     // Rimane guasto
        0.10 : (e1'=2) & (time'=time+1);      // ATTACCO (piu' vulnerabile)
```

Per un sensore in stato FAILED, la regola modella il processo di recovery manuale che, in assenza di meccanismi di Auto-Failover, presenta un tasso di successo del 60%, inferiore rispetto al modello protetto.

```
    [] e1=2 & time<200 ->
        1.00 : (e1'=2) & (time'=time+1);      // Rimane compromesso
```

Lo stato COMPROMISED è modellato come stato assorbente mediante una probabilità unitaria di auto-transizione.

Formule Derivate

```
formula num_ok = (e1=0?1:0) + (e2=0?1:0) + (e3=0?1:0) + (e4=0?1:0) + (e5=0?1:0);
formula num_failed = (e1=1?1:0) + (e2=1?1:0) + (e3=1?1:0) + (e4=1?1:0) + (e5=1?1:0);
formula num_compromised = (e1=2?1:0) + (e2=2?1:0) + (e3=2?1:0) + (e4=2?1:0) + (e5=2?1:0);

formula is_system_compromised = (num_compromised >= 1);
formula is_system_operational = (num_ok = 5);
formula is_system_degraded = (num_failed >= 1) & !is_system_compromised;
formula is_safe = !is_system_compromised;
```

Il modello definisce formule ausiliarie per classificare lo stato aggregato del sistema. In particolare, il sistema è considerato compromesso se almeno un sensore si trova nello stato COMPROMISED, riflettendo l'assenza di Sensor Redundancy nel modello vulnerabile.

5.1.3 Proprietà PCTL Verificate: Sistema SENZA Contromisure

Proprietà di Safety (S1)

```
P=? [ G<=100 (e1!=2 & e2!=2 & e3!=2 & e4!=2 & e5!=2) ]
```

Risultato PRISM **Risultato**: $1.4877 \times 10^{-7} \approx$ **0.0000149%**

Analisi Il risultato della verifica PRISM evidenzia la vulnerabilità critica del sistema non protetto. La probabilità che il sistema non venga compromesso in 100 step è praticamente nulla.

Proprietà di Guarantee/Response (G1)

```
P=? [ F<=20 (e1=0 & e2=0 & e3=0 & e4=0 & e5=0) ]
```

Risultato PRISM **Risultato**: 0.435146 $\approx$ **43.5%**

Analisi Il risultato evidenzia significative limitazioni nel processo di recovery del sistema vulnerabile. In assenza di meccanismi di Auto-Failover, il recovery è manuale e lento.

5.1.4 Modello PRISM: Sistema CON Contromisure

Struttura del Modello

Il modello PRISM rappresenta il sistema con tutte le contromisure di sicurezza attive. Include variabili di stato aggiuntive per implementare l'Active Defense System.

```
module sensor_system_active_defense

    e1 : [0..2] init 1;           // 0=OK, 1=FAILED, 2=COMPROMISED
    e1_attempts : [0..3] init 0;  // Contatore tentativi di attacco
    e1_locked : bool init false;  // Stato di blocco difensivo
```

Per il sensore E1, il modello definisce tre variabili che estendono la rappresentazione base. `e1_attempts` funge da contatore per i tentativi di attacco rilevati dall'IDS. `e1_locked` indica se il sistema ha attivato il blocco di sicurezza.

Matrice di Transizione: Sistema CON Contromisure

La seguente matrice mostra le probabilità di transizione tra gli stati per un sensore con contromisure attive:

Da Stato ↓ / A Stato →	**OK (0)**	**FAILED (1)**	**COMPR. (2)**
OK (0)	0.90	0.05	**0.00**
FAILED (1)	0.95	0.05	**0.00**
COMPR. (2)	0.00	0.00	1.00

Tabella 5.2: Matrice di Transizione - Sistema Protetto

Dallo stato OK, il sensore ha una probabilità del 90% di rimanere operativo, una probabilità del 5% di guasto hardware naturale, e una probabilità nulla di compromissione. Dallo stato FAILED, il recovery automatico tramite Auto-Failover presenta un'alta probabilità di successo del 95%.

Logica delle Transizioni: Active Defense

```
    [] e1=0 & !e1_locked & e1_attempts < 3 & time<200 ->
        0.90 : (e1'=0) & (time'=time+1) +                                    // Nessun evento
        0.05 : (e1'=1) & (time'=time+1) +                                    // Guasto naturale
        0.05 : (e1'=0) & (e1_attempts'=e1_attempts+1) & (time'=time+1);      // ATTACCO RILEVATO
```

Questa regola è cruciale: l'attaccante tenta un attacco (5%), ma l'IDS lo rileva, le contromisure lo bloccano, il sensore rimane OK e il contatore dei tentativi viene incrementato.

```
    [] e1=0 & !e1_locked & e1_attempts = 3 & time<200 ->
        1.00 : (e1_locked'=true) & (time'=time+1);                           // ATTIVA BLOCCO
```

Quando il contatore raggiunge tre tentativi di attacco bloccati, si attiva il System Lock.

5.1.5 Proprietà PCTL Verificate: Sistema CON Contromisure

Proprietà di Safety (S1)

```
P=? [ G<=100 (e1!=2 & e2!=2 & e3!=2 & e4!=2 & e5!=2) ]
```

Risultato PRISM **Risultato**: **1.0** (100%)

Analisi Il risultato costituisce una dimostrazione formale dell'efficacia delle contromisure. La probabilità del 100% di non-compromissione evidenzia che le contromisure bloccano completamente gli attacchi.

Proprietà di Guarantee/Response (G1)

```
P=? [ F<=20 (e1=0 & e2=0 & e3=0 & e4=0 & e5=0) ]
```

Risultato PRISM **Risultato**: $\approx$ **97%**

Analisi Il risultato dimostra l'alta efficacia dei meccanismi di Sensor Redundancy e Auto-Failover.

Proprietà di Active Defense Verification

```
P=? [ F e1_locked ]
```

Analisi Conferma che l'IDS e il Rate Limiting funzionano correttamente.

5.1.6 Confronto Quantitativo: Con vs Senza Contromisure

Confronto delle Matrici di Transizione

Transizione	CON Contromisure	SENZA Contromisure	Differenza
OK → OK	90%	80%	+10%
OK → FAILED	5%	5%	0%
OK → COMPR.	**0%**	**15%**	**-15%**
FAILED → OK	**95%**	**60%**	**+35%**
FAILED → FAILED	5%	30%	-25%
FAILED → COMPR.	**0%**	**10%**	**-10%**

Tabella 5.3: Confronto Matrici di Transizione

Tabella Comparativa dei Risultati PRISM

Proprietà	CON C.	SENZA C.	Miglior.
Safety (S1)	**100%**	0.00001%	+99.9%
Guarantee (G1)	**97%**	43.5%	+53.5%

Tabella 5.4: Confronto Risultati PRISM

5.1.7 Conclusioni Analisi Formale

L'analisi di Markov Chain condotta mediante il model checker PRISM ha fornito una dimostrazione formale dell'efficacia delle contromisure di sicurezza. Per la proprietà di Safety, le contromisure riducono la vulnerabilità da approssimativamente il 100% allo 0%. Per la proprietà di Guarantee/Response, le contromisure migliorano la probabilità di recovery dal 43.5% al 97%. Il meccanismo di Active Defense si è dimostrato efficace nel rilevare e bloccare attacchi persistenti.

5.2 Testing su Blockchain Privata (Besu)

Tutti i componenti sono stati integrati e testati in un ambiente reale basato su Hyperledger Besu.

5.2.1 Ambienti di Test

- **Unit Testing**: Suite completa di test JavaScript (Framework Truffle/Mocha) eseguita su Ganache per test rapidi della logica.
- **Integration Testing**: Deployment su rete privata Besu a 4 nodi (consenso IBFT 2.0). Sono stati verificati simulando scenari di latenza di rete e spegnimento di un nodo validatore.
- **Privacy Compliance**: Eseguiti test specifici (`test-offuscamento.js`) per validare che le tabelle CPT siano accessibili solo dall'Admin e che i dettagli sensibili siano verificabili solo tramite hash, impedendo letture non autorizzate.

5.2.2 Simulazione con Oracolo Scriptato

Utilizzando lo script `simula_oracolo.js`, è stato possibile testare il comportamento del sistema su un campione di N=1000 iterazioni simulate.

- **Scenario 1 (Condizioni Normali)**: Con sensori che riportano valori nominali (90% dei casi), la Rete Bayesiana On-Chain ha correttamente valutato la probabilità di conformità > 95% nel 100% dei casi.
- **Scenario 2 (Manomissione)**: Forzando il sensore "Sigillo" a `False`, la probabilità calcolata dal contratto `BNCore` è scesa immediatamente sotto la soglia di sicurezza, attivando lo stato di allarme.

5.2.3 Risultati

I test hanno dimostrato che il sistema mantiene la consistenza dei dati anche con un nodo offline. Le transazioni vengono confermate e finalizzate correttamente grazie al consenso IBFT. I monitor di runtime hanno intercettato correttamente il 100% delle transazioni anomale simulate (es. tentativi di registrare temperature fuori range senza triggerare allarmi).

CAPITOLO 6

Analisi della Qualità del Codice (Solhint)

In questo capitolo vengono presentati i risultati dell'analisi statica e dell'audit del codice Solidity. Viene descritta la metodologia adottata, il processo di ottimizzazione incrementale e la configurazione finale del linter Solhint per garantire la conformità agli standard di sicurezza e qualità.

6.1 Analisi Statica e Audit (Solhint)

L'analisi della qualità del codice rappresenta un aspetto fondamentale nello sviluppo di smart contract su blockchain Ethereum. Data la natura immutabile di tali applicazioni, dove errori possono portare a perdite economiche significative, l'adozione di strumenti di analisi statica è necessaria. Nel presente progetto si è utilizzato **Solhint**, uno dei principali linter per smart contract Ethereum, per garantire conformità alle best practices e ridurre il debito tecnico.

6.1.1 Metodologia

Solhint: Caratteristiche

Solhint è un linter open-source per Solidity che esegue analisi statica del codice, configurabile tramite file `.solhint.json`. Le categorie principali di regole includono: **Best Practices**, **Gas Optimization**, **Security**, e **Style Guide**.

Risultati Iniziali

L'esecuzione iniziale ha evidenziato **137 warning** (0 errori), confermando la correttezza sintattica del codice. La maggior parte dei warning riguardava documentazione mancante (42%) e naming convention (35%).

6.1.2 Processo di Ottimizzazione

Il processo è stato condotto in tre fasi successive.

Fase 1: Miglioramenti al Codice (-56 warning)

Documentazione NatSpec Sono stati documentati 17 eventi con commenti NatSpec completi (`@notice`, `@param`).

Ottimizzazioni Gas Sono state adottate tecniche come **Pre-increment** e l'uso di **Custom errors** (risparmio 1000 gas on revert) al posto di stringhe `require`.

Refactoring Funzioni La funzione `_calcolaProbabilitaCombinata` è stata ridotta da 66 a 13 linee estraendo l'helper `_applicaCPT`, migliorando modularità e testabilità.

Fase 2: Configurazione Naming (-42 warning)

Sono state disabilitate alcune regole di naming (`var-name-mixedcase`) per compatibilità con le convenzioni domain-specific adottate nel progetto.

Fase 3: Configurazione Gas (-18 warning)

Un'analisi costi-benefici ha evidenziato che alcune regole di ottimizzazione gas compromettevano eccessivamente la leggibilità per un risparmio irrisorio. Regole come `gas-strict-inequali` sono state disabilitate.

6.1.3 Configurazione Finale

La configurazione finale (`.solhint.json`) estende `solhint:recommended` ma disabilita le regole di naming e gas non ritenute critiche, mantenendo invece attive quelle su compiler version e visibility.

6.1.4 Risultati Finali e Conclusioni

L'analisi Solhint ha ridotto i warning dell'85% (da 137 a 21). I 21 warning residui sono stati analizzati e giustificati da scelte architetturali deliberate.

- **Warning Rimanenti**: Import globali per custom errors, complessità necessaria in algoritmi critici e naming conventions di progetto.

Il progetto raggiunge metriche di qualità elevate, posizionandosi sopra gli standard industriali per documentazione e pulizia del codice.

6.1.5 Considerazioni Critiche

Limiti analisi statica

- Non identifica vulnerabilità logiche complesse (che richiedono testing dinamico e audit manuale).
- Possibili false positivi su pattern legittimi ma non standard.
- Necessità di una configurazione altamente specifica per il dominio applicativo.

Complementarietà con altri tool Solhint dovrebbe essere idealmente integrato con altri strumenti come Slither (per vulnerabilità di sicurezza), Mythril (analisi simbolica), Echidna (fuzzing) e Hardhat (testing automatizzato).

6.1.6 Raccomandazioni

- **CI/CD**: Integrare Solhint nella pipeline di Continuous Integration.
- **Pre-commit hooks**: Eseguire analisi automatiche prima di ogni commit per prevenire regressioni.
- **Revisione periodica**: Aggiornare regolarmente la configurazione e le regole con le nuove versioni del tool.

Solhint si conferma strumento essenziale per progetti blockchain professionali, quando utilizzato con consapevolezza critica.

CAPITOLO 7

Conclusioni e Sviluppi Futuri

Questo capitolo conclude il lavoro sintetizzando i risultati ottenuti rispetto agli obiettivi di sicurezza e integrità del dato. Vengono inoltre analizzate criticamente le limitazioni dell'attuale implementazione e proposti scenari di evoluzione futura.

7.1 Sintesi dei Risultati

Il progetto ha dimostrato la fattibilità tecnica di un sistema di tracciabilità farmaceutica che non si limita alla semplice registrazione passiva dei dati, ma implementa una logica decisionale attiva e decentralizzata.

I principali traguardi raggiunti includono:

1. **Integrità Bayesiana**: L'implementazione on-chain della Rete Bayesiana (`BNCore`) ha permesso di validare la coerenza delle letture multisensoriali, riducendo drasticamente il rischio di accettare lotti compromessi a causa di falsi negativi dei singoli sensori.

2. **Resilienza Architetturale**: L'adozione di Hyperledger Besu con consenso IBFT 2.0 ha garantito la continuità del servizio e l'immutabilità dei dati anche in presenza di guasti o attacchi a un nodo validatore (fino a $f = 1$ su $N = 4$).

3. **Sicurezza Difensiva**: L'applicazione rigorosa dei principi di Secure Programming (Monitor Runtime, Checks-Effects-Interactions) ha prevenuto vulnerabilità comuni come la Reentrancy e l'accesso non autorizzato ai fondi in escrow.

4. **Verifica Formale**: L'utilizzo di PRISM ha fornito una garanzia matematica sul rispetto delle proprietà di Safety (probabilità di errore $< 0.1\%$) e Guarantee.

7.2 Limitazioni Attuali

Nonostante il successo del prototipo, esistono limitazioni che devono essere considerate per un deployment in produzione:

- **Scalabilità On-Chain**: Il calcolo bayesiano in Solidity, sebbene ottimizzato, consuma una quantità di Gas non trascurabile. Su una mainnet pubblica (es. Ethereum) i costi operativi potrebbero essere proibitivi; su una rete privata Besu (dove il Gas è gratuito o calmierato) il problema è ridotto al tempo di esecuzione.

- **Privacy dei Dati**: Sebbene sia stato implementato un meccanismo di hashing per offuscare i dettagli del carico (es. nome farmaco), i metadati delle transazioni e i valori grezzi dei sensori rimangono visibili ai nodi validatori. La privacy ottenuta è parziale (pseudonimato); una riservatezza totale richiederebbe tecnologie Zero-Knowledge (es. zk-SNARKs).
- **Simulazione IoT**: L'hardware IoT è attualmente simulato. La sicurezza fisica del sensore ("Hardware Root of Trust") esula dallo scopo di questo progetto software, ma rappresenta un vettore di attacco critico nel mondo reale.

7.3 Sviluppi Futuri

Per superare le limitazioni identificate e aumentare il livello di maturità del sistema (TRL), si propongono le seguenti evoluzioni:

7.3.1 Integrazione zk-SNARKs (Privacy)

L'adozione di protocolli a conoscenza zero (Zero-Knowledge Proofs) permetterebbe al corriere di dimostrare la conformità della spedizione ("La temperatura è rimasta nel range") senza rivelare i valori esatti o i dettagli del tragitto, garantendo privacy commerciale e conformità GDPR.

7.3.2 Oracle Feed decentralizzati (Chainlink)

Sostituire lo script di simulazione centralizzato con una rete di oracoli decentralizzati (es. Chainlink) per leggere i dati dai dispositivi IoT. Questo eliminerebbe il singolo punto di fallimento rappresentato dallo script Node.js.

7.3.3 Hardware Security Module (HSM)

Integrazione con sensori dotati di Secure Element per la firma delle transazioni direttamente "at the edge". Questo garantirebbe che il dato firmato provenga fisicamente dal dispositivo e non sia stato iniettato via software.

In conclusione, il lavoro svolto pone basi solide per una logistica 4.0 più sicura, dimostrando come l'intersezione tra Blockchain, Metodi Formali e IoT possa generare valore reale in contesti critici per la salute pubblica.

APPENDICE A

Guida al Deployment e Management

A.1 Requisiti di Sistema (Prerequisiti)

Per eseguire l'intero stack del progetto sono necessari i seguenti strumenti:

- **Node.js** (versione $\geq$ 16.0.0) e **NPM**
- **Docker** e **Docker Compose** (per il nodo Besu)
- **Truffle Suite** (per compilazione e deploy Smart Contracts)
- **Ganache** (opzionale, per test rapidi in locale)
- **Git**
- **Java JDK 11+** (se si esegue Besu nativamente senza Docker)

A.2 Installazione e Setup

A.2.1 Clonazione del Repository

Il codice sorgente è ospitato su GitHub. Eseguire il clone:

```
git clone https://github.com/lucabelard/ProgettoSoftwareSecurity.git
cd ProgettoSoftwareSecurity
```

A.2.2 Installazione Dipendenze

Installare le dipendenze per l'interfaccia web e gli script di test:

```
npm install
cd web-interface
npm install
```

A.3 Avvio della Rete Blockchain

A.3.1 Modalità Sviluppo (Ganache)

1. Avviare Ganache (GUI o CLI) sulla porta 7545. 2. Configurare `truffle-config.js` per puntare a `127.0.0.1:7545`. 3. Eseguire il deploy:

```
truffle migrate --reset --network development
```

A.3.2 Modalità Produzione (Hyperledger Besu)

1. Navigare nella cartella `besu-network`. 2. Avviare i nodi validatori:

```
./start_nodes.sh
```

3. Attendere che i nodi siano sincronizzati (consenso IBFT 2.0). 4. Eseguire il deploy sulla rete Besu:

```
truffle migrate --reset --network besu
```

A.4 Esecuzione degli Script di Simulazione

Per testare il sistema end-to-end con dati simulati:

```
node simula_oracolo.js
```

Questo script simulerà l'invio di dati dai sensori e l'interazione con l'Oracolo on-chain.

A.5 Interfaccia Web

Per avviare la dashboard utente (necessita di Node.js installato):

```
cd web-interface
npx http-server .
```

L'applicazione sarà accessibile di default a `http://localhost:8080`. Assicurarsi di avere MetaMask configurato sulla rete locale (Chain ID 1337 o 2024 a seconda della configurazione Ganache/Besu).