# TrustIot - A Blockchain-Based Architecture to Ensure Data Integrity in IoT Environments

Nicolas Carregosa de Jesus – Prof. Claudia Barenco

University of Brasília - UnB - Transversal Project of Communication Networks

# Summary

# 1. Abstract

The rapid expansion of the Internet of Things (IoT) has introduced numerous efficiencies in industries ranging from healthcare to logistics. However, this proliferation has brought significant security challenges, with data integrity being the primary concern. Compromised or falsified sensor data can lead to erroneous decisions, system failures, and security risks. This paper proposes TrustIot, an architecture that uses blockchain technology to ensure the immutability and traceability of data generated by IoT devices. The solution focuses on recording data *hashes* in a *smart contract* on the Ethereum network (Sepolia Testnet), providing a low-cost, auditable proof of existence and integrity. The feasibility of the architecture is demonstrated through a simulation environment (Python/Hardhat) and validated by a monitoring dashboard (React).

# 2. Introduction

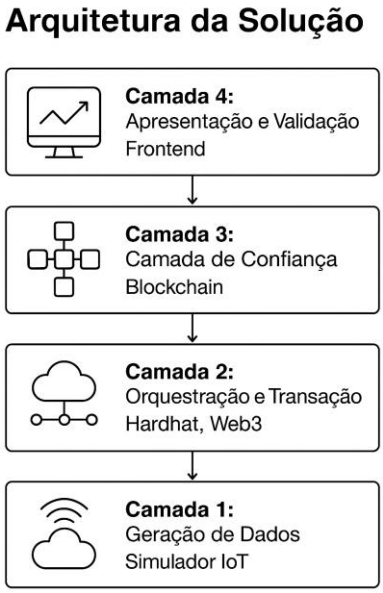
The Internet of Things (IoT) is characterized by a network of physical devices that collect and exchange data. While the value of this data is immense, its centralization and the vulnerability of *edge* devices make it easy targets for *Man-in-the-Middle* (MITM) attacks and data *spoofing*. In a telemedicine scenario, an altered patient sensor data can be fatal; In a supply chain, a false temperature record can invalidate a batch of vaccines.

Trust in these systems is therefore key. Traditional solutions based on centralized servers and SSL/TLS certificates are necessary but often insufficient as they do not guarantee the historical *immutability* of the data after it has been received by the server.

TrustIot addresses this problem by proposing a decentralized architecture. The central premise is that while raw IoT data may be too bulky to be stored *on-chain*, its cryptographic "fingerprints" (SHA-256 hashes) are small and efficient. By recording these *hashes* on a public blockchain, we create a permanent, chronological, tamper-proof record, allowing any interested party to audit and verify the integrity of the data.

# 3. Solution Architecture

TrustIot's architecture is divided into four logical layers, designed to decouple data generation from its validation, ensuring efficiency and scalability.




## Layer 1: Data Generation (IoT Simulator)

Due to the current phase of the project, this layer is represented by a simulator.

- **Technology:** Python scripts.
- **Function:** The script simulates an IoT device (e.g., temperature sensor) by generating data readings at regular intervals. For each read, it calculates a cryptographic hash (e.g., SHA-256) of the data. This *hash* serves as a unique, tamper-proof representation of the original data.

## Layer 2: Orchestration and Transaction

This layer acts as the bridge between the *off-chain* (device) world and the *on-chain* (blockchain) world.

- **Technology:** Hardhat (for testing and *deployment*) and Web3 libraries (such as web3.py or ethers.js).
- **Function:** The orchestrator is responsible for picking up the deviceId and hashData generated by Layer 1. It uses a digital wallet (with funds from the Sepolia testnet) to format and send a transaction to the *smart contract* on the blockchain, invoking the registerDevice function.

## Layer 3: Trust Layer (Blockchain)

The core of the solution, where trust is established.

- **Technology:** Solidity and Ethereum (Sepolia Testnet).
- **Function:** The *TrustIoT.sol* smart contract (detailed in Section 5) receives the transaction. It stores the *hash* and block.timestamp (the exact time of the record in the block) in a mapping associated with the deviceId. The Ethereum network guarantees that once registered, this data cannot be changed or removed.

### Layer 4: Presentation and Validation (Frontend)

The end-user interface, which allows auditing of the data.

- **Technology:** React (with ethers.js) and Vercel (for *hosting*).
- **Function:** The dashboard (available in trust-io-t-site.vercel.app) connects directly to a node in the Sepolia network. It allows the user to query a deviceId and calls the smart contract's *getRecord function* to retrieve the latest *hash* and *timestamp*, proving the integrity of the simulated data.

## 4. Methodology: Validation via Simulation

To validate the architecture without the initial dependence on specific hardware and to mitigate costs, a simulation approach was chosen. This methodology allows you to focus on the robustness of *on-chain* logic and the integration of software layers.

The simulation flow occurs as follows:

1. **Generation:** The Python script (Layer 1) generates a value (e.g. {"temp": 25.5, "unit": "C"}).
2. **Hashing:** The script calculates the SHA-256 of that *payload*.
3. **Logging:** The orchestrator (Layer 2) sends this *hash* to Layer 3.
4. **Validation:** The Dashboard (Layer 4) is used to query the *hash* that has just been recorded, confirming that the cycle has been successfully completed.

This approach proves that the architecture is functionally feasible and that the smart *contract* logic meets the health logging requirements.

## 5. Implementation: The TrustIoT Smart Contract

The core component of the platform is the *TrustIoT.sol* smart contract, written in Solidity v0.8.20 and deployed on the Sepolia testnet.

The contract is designed to be minimalist and efficient in terms of *gas* (cost of execution).

Solidity

```
/*
 * SPDX-License-Identifier: MIT
 */
pragma solidity ^0.8.20;

/**
 * @title TrustIoT
 * @dev This contract records data hashes from IoT devices
 * to ensure integrity and proof of existence.
 */
contract TrustIoT {

    Framework for storing a device's record
    struct DeviceRecord {
        string deviceId;   Device ID
        string hashData;   The SHA-256 hash of the sensor data
        uint256 timestamp; Ethereum block timestamp
    }

    Mapping the deviceId to its last record
    mapping(string => DeviceRecord) public records;

    Event emitted when a new registration is made
    event DeviceRegistered(
        string deviceId,
        string hashData,
        uint256 timestamp
    );

    /**
     * @dev Records or updates the hash of data from a device.
     * @param deviceId The unique identifier of the device.
     * @param hashData The hash of the data to be recorded.
     */
    function registerDevice(
        string memory deviceId,
        string memory hashData
```

```
    ) public {
        Stores the record using the block timestamp
        records[deviceId] = DeviceRecord(
            deviceId,
            hashData,
            block.timestamp
        );

        Emits an event so that off-chain applications can listen
        emit DeviceRegistered(deviceId, hashData, block.timestamp);
    }

    /**
     * @dev Retrieves the last record of a specific device.
     * @param deviceId The identifier of the device to query.
     * @return DeviceRecord The complete device record.
     */
    function getRecord(
        string memory deviceId
    ) public view returns (DeviceRecord memory) {
        return records[deviceId];
    }
}
```

**Design Rationale:**

- **registerDevice:** It is the main writing function. It updates the latest device registration. The use of block.timestamp is crucial as it provides a secure and immutable source of time, guaranteed by the network's validators.
- **getRecord:** A view function (that doesn't waste *gas*) for reading, allowing the dashboard and other auditing systems to validate the data publicly.
- **DeviceRegistered (Event):** Allows *frontend* applications to listen for new registrations in real time without the need to query (*poll)* the contract repeatedly.

# 6. Results: Validation Dashboard

The functional dashboard serves as the visual Proof of Concept (PoC) of the architecture.


The *frontend* implements the following functionalities:

1. **Blockchain Connection:** Uses the ethers.js library to connect to an RPC provider of the Sepolia network, allowing direct interaction with the blockchain through the browser.
2. **Log Query:** The user can enter a deviceId (used in the simulation) and the dashboard calls the getRecord(deviceId) function of the *smart contract*.
3. **Proof View:** The dashboard displays the hashData and timestamp (converted to readable date and time) stored *on-chain*.

This result demonstrates the complete flow: a simulated data (Layer 1) has its *hash* recorded (Layer 3) and can be publicly validated by a user interface (Layer 4).

# 7. Completion and Future Work

This paper presented the architecture and preliminary implementation of TrustIot, a system for ensuring the integrity of IoT data using blockchain. Through a simulation environment and a *smart contract* on the Ethereum network (Sepolia), we demonstrate the feasibility of creating an immutable audit trail for sensor data.

The use of simulation allowed us to validate the business logic and the integration of the software layers quickly and cost-effectively.

The main "Future Work" is the transition from Layer 1 (Simulation) to physical hardware. Next steps include:

1. **Industrial/Academic Partnership:** Seek partnerships with local companies (in the Brasilia ecosystem) or university departments to obtain IoT devices (such as ESP32, Raspberry Pi) for a pilot project.
2. **Hardware Deployment:** Integrate Layer 2 (Orchestration) directly into the embedded device or an edge *gateway*.
3. **Cost Analysis (Gas):** Conduct a detailed analysis of *gas* costs on the mainnet (Ethereum Mainnet) or Layer 2 (L2s) solutions, such as Polygon or Optimism, to assess large-scale economic viability.