TrustIot - Uma Arquitetura Baseada em Blockchain para Garantir a Integridade de Dados em Ambientes IoT

Nicolas Carregosa de Jesus – Profª. Claudia Barenco

Universidade de Brasília - UnB - Projeto Transversal de Redes de Comunicação

Sumário

1. Resumo (Abstract)

A rápida expansão da Internet das Coisas (IoT) introduziu inúmeras eficiências em setores que vão da saúde à logística. Contudo, essa proliferação trouxe desafios de segurança significativos, sendo a integridade dos dados a principal preocupação. Dados de sensores comprometidos ou falsificados podem levar a decisões errôneas, falhas de sistema e riscos de segurança. Este documento propõe o TrustIot, uma arquitetura que utiliza a tecnologia blockchain para garantir a imutabilidade e a rastreabilidade dos dados gerados por dispositivos IoT. A solução foca no registro de *hashes* de dados em um *smart contract* na rede Ethereum (Sepolia Testnet), provendo uma prova de existência e integridade auditável e de baixo custo. A viabilidade da arquitetura é demonstrada através de um ambiente de simulação (Python/Hardhat) e validada por um dashboard de monitoramento (React).

2. Introdução

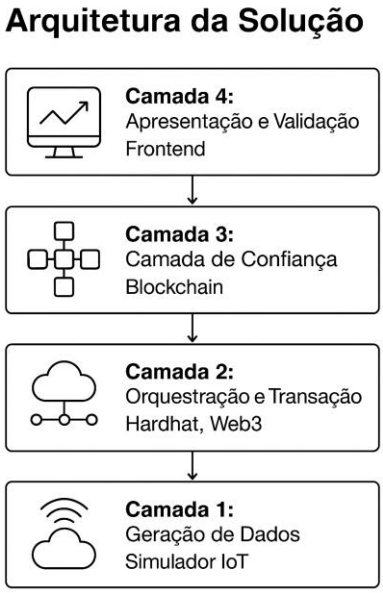
A Internet das Coisas (IoT) é caracterizada por uma rede de dispositivos físicos que coletam e trocam dados. Embora o valor desses dados seja imenso, sua centralização e a vulnerabilidade dos dispositivos *edge* os tornam alvos fáceis para ataques *Man-in-the-Middle* (MITM) e falsificação de dados (*data spoofing*). Em um cenário de telemedicina, um dado de sensor de paciente alterado pode ser fatal; em uma cadeia de suprimentos, um registro de temperatura falso pode invalidar um lote de vacinas.

A confiança nesses sistemas é, portanto, fundamental. Soluções tradicionais baseadas em servidores centralizados e certificados SSL/TLS são necessárias, mas muitas vezes insuficientes, pois não garantem a *imutabilidade* histórica do dado após ele ter sido recebido pelo servidor.

O TrustIot aborda esse problema propondo uma arquitetura descentralizada. A premissa central é que, embora os dados brutos de IoT possam ser volumosos demais para serem armazenados *on-chain*, suas "impressões digitais" criptográficas (*hashes* SHA-256) são pequenas e eficientes. Ao registrar esses *hashes* em uma blockchain pública, criamos um registro permanente, cronológico e à prova de adulteração, permitindo que qualquer parte interessada possa auditar e verificar a integridade dos dados.

3. Arquitetura da Solução

A arquitetura do TrustIot é dividida em quatro camadas lógicas, projetadas para desacoplar a geração de dados da sua validação, garantindo eficiência e escalabilidade.



Camada 1: Geração de Dados (Simulador IoT)

Devido à fase atual do projeto, esta camada é representada por um simulador.

- **Tecnologia:** Scripts em Python.
- **Função:** O script simula um dispositivo IoT (ex: sensor de temperatura) gerando leituras de dados em intervalos regulares. Para cada leitura, ele calcula um *hash* criptográfico (ex: SHA-256) do dado. Este *hash* serve como uma representação única e inviolável do dado original.

Camada 2: Orquestração e Transação

Esta camada atua como a ponte entre o mundo *off-chain* (dispositivo) e o mundo *on-chain* (blockchain).

- **Tecnologia:** Hardhat (para testes e *deployment*) e bibliotecas Web3 (como web3.py ou ethers.js).
- **Função:** O orquestrador é responsável por pegar o deviceId e o hashData gerados pela Camada 1. Ele utiliza uma carteira digital (com fundos da testnet Sepolia) para formatar

e enviar uma transação para o *smart contract* na blockchain, invocando a função registerDevice.

Camada 3: Camada de Confiança (Blockchain)

O núcleo da solução, onde a confiança é estabelecida.

- **Tecnologia:** Solidity e Ethereum (Sepolia Testnet).
- **Função:** O *smart contract* TrustIoT.sol (detalhado na Seção 5) recebe a transação. Ele armazena o *hash* e o block.timestamp (o momento exato do registro no bloco) em um mapeamento associado ao deviceId. A rede Ethereum garante que, uma vez registrado, este dado não pode ser alterado ou removido.

Camada 4: Apresentação e Validação (Frontend)

A interface com o usuário final, que permite a auditoria dos dados.

- **Tecnologia:** React (com ethers.js) e Vercel (para *hosting*).
- **Função:** O dashboard (disponível em trust-io-t-site.vercel.app) se conecta diretamente a um nó da rede Sepolia. Ele permite que o usuário consulte um deviceId e chama a função getRecord do *smart contract* para recuperar o *hash* e o *timestamp* mais recentes, provando a integridade dos dados simulados.

4. Metodologia: Validação via Simulação

Para validar a arquitetura sem a dependência inicial de hardware específico e mitigar custos, optou-se por uma abordagem de simulação. Esta metodologia permite focar na robustez da lógica *on-chain* e na integração das camadas de software.

O fluxo de simulação ocorre da seguinte forma:

1. **Geração:** O script Python (Camada 1) gera um valor (ex: {"temp": 25.5, "unit": "C"}).
2. **Hashing:** O script calcula o SHA-256 desse *payload*.
3. **Registro:** O orquestrador (Camada 2) envia este *hash* para a Camada 3.
4. **Validação:** O Dashboard (Camada 4) é usado para consultar o *hash* que acabou de ser registrado, confirmando que o ciclo foi completado com sucesso.

Esta abordagem prova que a arquitetura é funcionalmente viável e que a lógica do *smart contract* atende aos requisitos de registro de integridade.

5. Implementação: O Smart Contract TrustIoT

O componente central da plataforma é o *smart contract* TrustIoT.sol, escrito em Solidity v0.8.20 e implantado na rede de testes Sepolia.

O contrato foi projetado para ser minimalista e eficiente em termos de *gas* (custo de execução).

Solidity

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

/**
 * @title TrustIoT
 * @dev Este contrato registra hashes de dados de dispositivos IoT
 * para garantir a integridade e a prova de existência.
 */
contract TrustIoT {

    // Estrutura para armazenar o registro de um dispositivo
    struct DeviceRecord {
        string deviceId;   // ID do dispositivo
        string hashData;   // O hash SHA-256 dos dados do sensor
        uint256 timestamp; // Timestamp do bloco Ethereum
    }

    // Mapeamento do deviceId para seu último registro
    mapping(string => DeviceRecord) public records;

    // Evento emitido quando um novo registro é feito
    event DeviceRegistered(
        string deviceId,
        string hashData,
        uint256 timestamp
    );

    /**
     * @dev Registra ou atualiza o hash de dados de um dispositivo.
     * @param deviceId O identificador único do dispositivo.
     * @param hashData O hash dos dados a serem registrados.
```

```
     */
    function registerDevice(
        string memory deviceId,
        string memory hashData
    ) public {
        // Armazena o registro usando o timestamp do bloco
        records[deviceId] = DeviceRecord(
            deviceId,
            hashData,
            block.timestamp
        );

        // Emite um evento para que aplicações off-chain possam escutar
        emit DeviceRegistered(deviceId, hashData, block.timestamp);
    }

    /**
     * @dev Recupera o último registro de um dispositivo específico.
     * @param deviceId O identificador do dispositivo a ser consultado.
     * @return DeviceRecord O registro completo do dispositivo.
     */
    function getRecord(
        string memory deviceId
    ) public view returns (DeviceRecord memory) {
        return records[deviceId];
    }
}
```

Justificativa de Design:

- **registerDevice:** É a função principal de escrita. Ela atualiza o registro mais recente do dispositivo. O uso de block.timestamp é crucial, pois fornece uma fonte de tempo segura e imutável, garantida pelos validadores da rede.
- **getRecord:** Uma função view (que não gasta *gas*) para leitura, permitindo que o dashboard e outros sistemas de auditoria validem os dados publicamente.
- **DeviceRegistered (Evento):** Permite que aplicações *frontend* escutem por novos registros em tempo real sem a necessidade de consultar (fazer *polling*) o contrato repetidamente.

6. Resultados: Dashboard de Validação

O dashboard funcional serve como a Prova de Conceito (PoC) visual da arquitetura.

O *frontend* implementa as seguintes funcionalidades:

1. **Conexão com a Blockchain:** Utiliza a biblioteca `ethers.js` para se conectar a um provedor RPC da rede Sepolia, permitindo interação direta com a blockchain pelo navegador.
2. **Consulta de Registros:** O usuário pode inserir um `deviceId` (usado na simulação) e o dashboard chama a função `getRecord(deviceId)` do *smart contract*.
3. **Exibição de Prova:** O dashboard exibe o `hashData` e o `timestamp` (convertido para data e hora legíveis) armazenados *on-chain*.

Este resultado demonstra o fluxo completo: um dado simulado (Camada 1) tem seu *hash* registrado (Camada 3) e pode ser publicamente validado por uma interface de usuário (Camada 4).

7. Conclusão e Trabalhos Futuros

Este documento apresentou a arquitetura e a implementação preliminar do TrustIot, um sistema para garantir a integridade de dados IoT usando blockchain. Através de um ambiente de simulação e um *smart contract* na rede Ethereum (Sepolia), demonstramos a viabilidade de criar um registro de auditoria imutável para dados de sensores.

O uso de simulação permitiu validar a lógica de negócios e a integração das camadas de software de forma rápida e econômica.

O principal "Trabalho Futuro" é a transição da Camada 1 (Simulação) para hardware físico. Os próximos passos incluem:

1. **Parceria Industrial/Acadêmica:** Buscar parcerias com empresas locais (no ecossistema de Brasília) ou departamentos da universidade para obter dispositivos IoT (como ESP32, Raspberry Pi) para um projeto piloto.
2. **Implementação em Hardware:** Integrar a Camada 2 (Orquestração) diretamente no dispositivo embarcado ou em um *gateway* de borda.
3. **Análise de Custo (Gas):** Realizar uma análise detalhada dos custos de *gas* na rede principal (Ethereum Mainnet) ou em soluções de Camada 2 (L2s), como Polygon ou Optimism, para avaliar a viabilidade econômica em larga escala.