**Blockchain-Based Embedded Device Integrity Verification Scheme**

1. **Introduction:** The purpose of this project is to develop a blockchain-based embedded device integrity verification scheme. The scheme involves computing a digital DNA (Ref\_DNA) for a device using selected static and dynamic parameters. By comparing the computed DNA at runtime with the reference DNA, the integrity of the device can be verified. In case of a mismatch, an alert notification will be generated to inform the user about the integrity check result, along with the correct device ID.
2. **Objectives:**

* Develop a blockchain-based scheme for device integrity verification.
* Select appropriate static and dynamic parameters for computing the device's DNA.
* Implement a smart contract to store and verify the DNA on the blockchain.
* Research ARM Trustzone secure hardware architectures and analyze their security guarantees in relation to blockchain and smart contracts.
* Propose a cost-effective secure ARM device integrated with blockchain for ensuring security in device integrity verification.
* Analyze the performance of the proposed model, including transaction confirmation time.
* Develop a user-friendly web interface for registering devices, computing and storing Ref\_DNA, selecting devices for sending sensor readings, computing runtime DNA, and displaying verification results.

1. **Methodology:** The methodology involves the following steps:

* Select appropriate static and dynamic parameters for the device's DNA computation.
* Develop a working smart contract for storing and verifying the DNA on the blockchain.
* Implement the computation of static parameters (SP) and dynamic parameters (DP) for each device.
* Create a transaction to submit the computed SP, DP, and the device ID.
* Research various ARM Trustzone secure hardware architectures and compare their security features with those of blockchain and smart contracts.
* Propose a cost-effective secure ARM device integrated with blockchain, ensuring the security of the verification code, device authentication, and cryptographic key protection.
* Analyze the performance of the proposed model, including transaction confirmation time.
* Develop a web user interface for device registration, Ref\_DNA computation, sensor reading submission, runtime DNA computation, and result display.

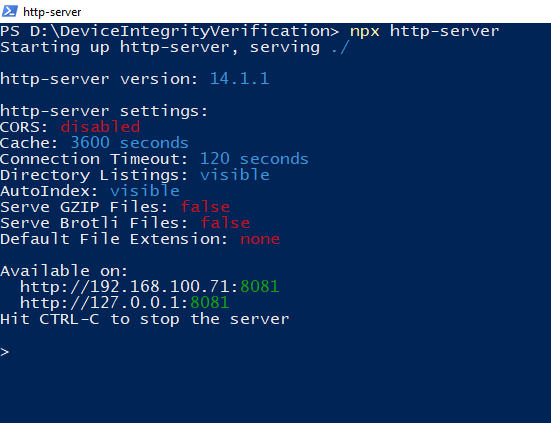
1. **Digital Forensics Parameters**: Static Parameters and Dynamic Parameters numerical value is chosen which will first be registered and then verified.
2. **System Architecture:** The system architecture includes an embedded device that computes its DNA using static and dynamic parameters. The computed DNA is stored on the blockchain through a smart contract. During runtime, the device's DNA is computed again and compared with the Ref\_DNA stored on the blockchain. If there is a match, the device integrity is verified, and if there is a mismatch, an alert notification is generated.
3. **Implementation:** The implementation includes developing a smart contract for storing and verifying the DNA on the blockchain. The static and dynamic parameters are computed using the selected values. Transactions are created and submitted to the blockchain with the computed SP, DP, and device ID. ARM Trustzone secure hardware architectures are researched to propose a cost-effective secure ARM device integrated with blockchain. Performance analysis is conducted, and a web user interface is developed for device registration, Ref\_DNA computation, sensor reading submission, runtime DNA computation, and result display.
4. **Demonstration of the Implementation**

**Pre-Requisites**

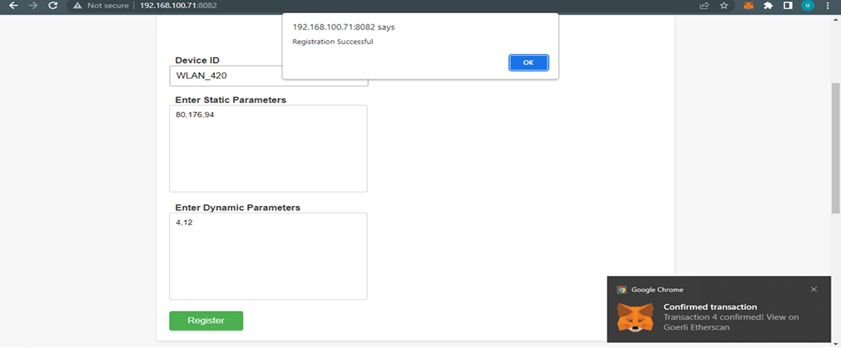
1. Create an account on the **MetaMask wallet.** MetaMask is a browser extension that allows you to interact with Ethereum networks.
2. Set up the **Georli test** network on MetaMask. The G network is a test network that allows you to test your smart contracts without using real Ether.
3. Once you have set up the Georli test network, you can request test Ether from a Georli faucet. This will provide you with the necessary test Ether to perform transactions on the Georli network.
4. After acquiring the test Ether, you can perform the device integration by executing the necessary transactions in the Device Integrity Verification contract.

* **Step- I** Choose Device ID **(WiFi Router)**
  + - * Device ID: **WLAN\_420**
* **Step-II** Choose Static and Dynamic Parameters for **registration.**
  + ***Static Parameters Dynamic Parameters***
  + MAC Address (Device Specific) : 80 Upload Speed : 4
  + MAC Address (Vendor Specific) **:** 176 Download Speed : 12
  + Operating Frequency : 94
* ***Note : The Numerical values assigned to above mentioned parameters are chosen randomly to help understand the Proof of Concept***
* **Step III** Open the GUI through **PowerShell**

**(Note:** To access Web 3.0 functionalities and **ensure the correct display of the IP address** instead of the folder path in the HTML file for the GUI, **the installation of Node.js is required.** Node.js enables server-side JavaScript execution and provides a runtime environment that allows running JavaScript code outside of a web browser. By utilizing Node.js, the HTML file can accurately retrieve and display the IP address, enhancing the technical functionality and user experience of the GUI**.)**



* **STEP- IV** Now **Register the Device ID along with its Static and Dynamic parameters**. Here we can see that after adding the adding the required details, the device got registered successfully.

****

* **STEP- V** Now **Verify the device** by adding the device ID and its static and dynamic parameters respectively.

A screenshot of a computer

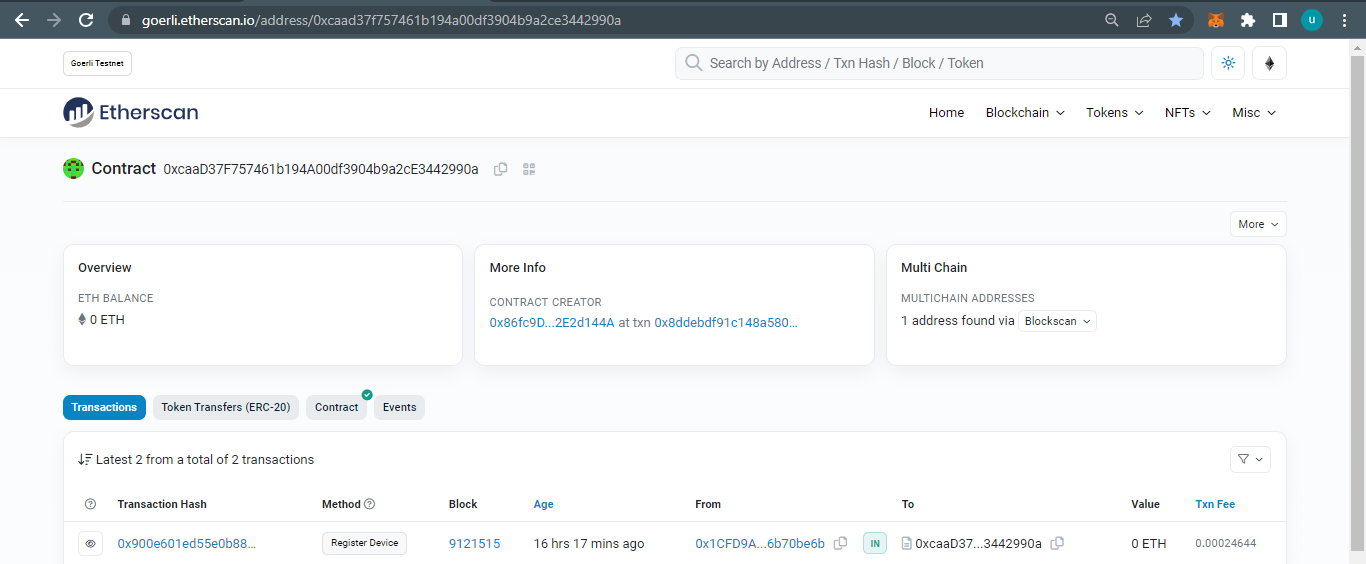
Description automatically generated

As in the screenshot it is visible that the **verification has been successful**.

* **STEP-VI** Now go to link

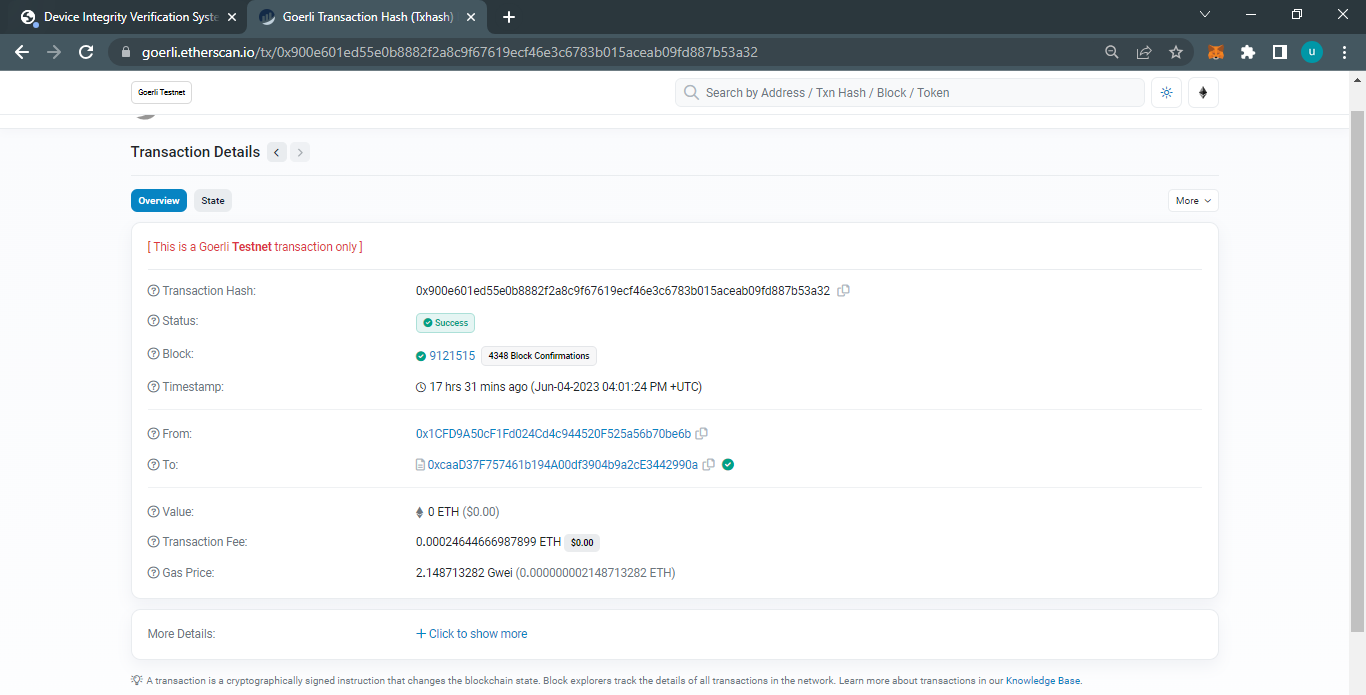
<https://goerli.etherscan.io/address/0xcaad37f757461b194a00df3904b9a2ce3442990a>

Once MetaMask confirms that the transaction has been successfully completed, you can visit etherscan.io. Etherscan is a blockchain explorer that allows you to view detailed information about transactions on the Ethereum network.

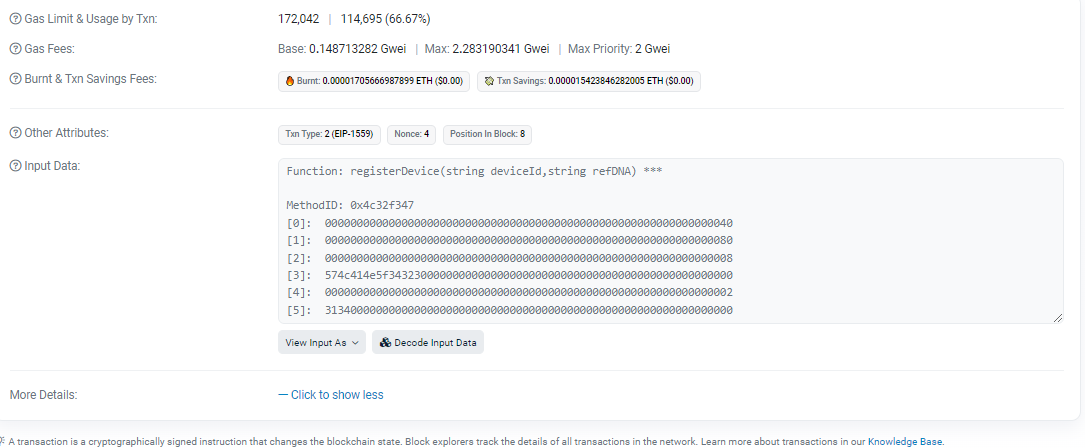


* **STEP-VII**

Now click on the transaction (Register Device) to see the further details of the transaction.



Now after clicking on show more details (at the bottom of the interface) the following will be shown.



**Now after decoding the input data, following detail will be shown**

A screenshot of a computer

Description automatically generated with low confidence

* **Results and Evaluation**: The results include the successful implementation of the blockchain-based embedded device integrity verification scheme. The smart contract effectively stores and verifies the device's DNA on the blockchain. The proposed secure ARM device ensures the security of the verification code, device authentication, and cryptographic key protection. The performance analysis demonstrates the efficiency of the model, including transaction confirmation time.

1. **Smart Contract Code:**

pragma solidity ^0.8.0;

// SPDX-License-Identifier: MIT

contract DeviceIntegrityVerification {

struct Device {

string refDNA;

bool registered;

}

string[] public registered\_ids;

mapping(string => Device) public devices;

function registerDevice(string memory device\_id, string memory refDNA) public {

require(!devices[device\_id].registered, "Device Id is already registered");

Device memory d = Device (refDNA, true);

devices[device\_id] = d;

registered\_ids.push(device\_id);

}

function verify(string memory device\_id, string memory DNA) public view returns(bool) {

return keccak256(abi.encodePacked(devices[device\_id].refDNA)) == keccak256(abi.encodePacked(DNA));

}

function get\_registered\_ids() public view returns(string[] memory){

return registered\_ids;

}

}

1. **WEB UI CODE**

<!DOCTYPE html>

<html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Device Integrity Verification System</title>

<link rel="stylesheet" href="styles.css">

<style>

body {

font-family: Arial, sans-serif;

background-color: #f7f7f7;

color: #333;

margin: 0;

padding: 0;

}

.container {

max-width: 800px;

margin: 0 auto;

padding: 20px;

background-color: #fff;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

align-items: center;

}

h1 {

font-size: 36px;

text-align: center;

margin-bottom: 20px;

font-family: 'Helvetica', sans-serif;

color: #2c3e50;

text-shadow: 2px 2px 4px #000000;

}

p {

font-size: 18px;

line-height: 1.5;

}

.transition-heading {

position: relative;

transition: color 0.3s ease-in-out;

}

.transition-heading::before {

content: "";

position: absolute;

bottom: 0;

left: 0;

width: 100%;

height: 2px;

background-color: #000;

transform: scaleX(0);

transform-origin: left center;

transition: transform 0.3s ease-in-out;

}

.transition-heading:hover {

color: #FF0000; /\* Change the color to your desired hover color \*/

}

.transition-heading:hover::before {

transform: scaleX(1);

}

.button {

display: inline-block;

background-color: #4CAF50;

border: none;

color: white;

text-align: center;

font-size: 16px;

padding: 10px;

width: 120px;

transition: all 0.5s;

cursor: pointer;

border-radius: 4px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);

}

.button:hover {

background-color: #3367D6;

}

.custom-label {

color: #333;

font-size: 16px;

font-weight: bold;

margin-bottom: 10px;

padding: 10px;

}

ul {

list-style-type: none;

margin: 0;

padding: 0;

}

li {

background-color: #f4f4f4;

padding: 10px;

margin-bottom: 5px;

border-radius: 5px;

}

.input-field {

border: 2px solid #ccc;

border-radius: 4px;

padding: 10px;

width: 300px;

font-size: 16px;

font-family: Arial, sans-serif;

outline: none;

}

.input-field:focus {

border-color: #007bff;

box-shadow: 0 0 5px #007bff;

}

.text-area {

width: 300px;

height: 150px;

padding: 10px;

border: 1px solid #ccc;

border-radius: 4px;

font-family: Arial, sans-serif;

font-size: 14px;

resize: none;

}

</style>

</head>

<body>

<h1>Device Integrity Verification </h1>

<h2 style="padding: 10px;"><strong>Project Team</strong></h2>

<ul>

<li>Usama Mustafa</li>

<li>Junaid Aslam</li>

<li>Aftab Ali</li>

</ul>

<div id="registration-form" class="container">

<h2 class="transition-heading" style="color: black; font-family: 'Helvetica Neue', sans-serif; text-align: center; font-size: 28px; font-weight: bold; text-transform: uppercase;">Register New Device</h2>

<label for="device-id" class="custom-label">Device ID</label><br>

<input type="text" id="device-id" class="input-field" name="name" placeholder="Device ID"/>

<br />

<br />

<label for="device-id" class="custom-label">Enter Static Parameters</label>

<br />

<textarea class="text-area" placeholder="Numbers Only & Separated By Comma" cols="31" id="static-params"></textarea>

</label>

<br />

<br />

<label class="custom-label" for="dynamic-params">Enter Dynamic Parameters</label>

<br />

<textarea class="text-area" placeholder="Numbers Only & Separated By Comma" cols="31" id="dynamic-params"></textarea>

<br />

<br />

<button class="button" onclick="registerDevice()">Register</button>

</div>

<br>

<div id="verification-form" class="container">

<h2 class="transition-heading" style="color: black; font-family: 'Helvetica Neue', sans-serif; text-align: center; font-size: 28px; font-weight: bold; text-transform: uppercase;">Device Verification</h2>

<label class="custom-label" for="device-select">Select Device</label>

<select id="device-select"></select>

<br />

<br />

<label class="custom-label" for="static-params-verification">

Enter Static Parameters</label><br>

<textarea class="text-area" placeholder="Numbers Only & Separated By Comma" cols="31" id="static-params-verification"></textarea>

<br />

<br />

<label class="custom-label" for="static-params-verification">

Enter Dynamic Parameters</label><br>

<textarea class="text-area" placeholder="Numbers Only & Separated By Comma" cols="31" id="dynamic-params-verification"></textarea>

<br />

<br />

<button class="button" onclick="computeRuntimeDNA()">Verify Integrity</button>

</div>

<script

src="https://cdnjs.cloudflare.com/ajax/libs/web3/1.2.7/web3.min.js"

integrity="sha512-yAmmVkOXqtDZRaueLdmLwMB67JtaRtjFzQGiAiJKvLiFlDvq4Tzm5R6uLcOMyvPRQlXpYHdYxc5IbiCwuquhUw=="

crossorigin="anonymous"

referrerpolicy="no-referrer"

></script>

<script src="script.js"></script>

</body>

</html>

**10. Comparison of ARM TrustZone Secure Hardware Architectures and Blockchain Security Features**

* **Introduction**

ARM TrustZone is a hardware security architecture that provides a secure execution environment for critical applications. It is used in a wide range of devices, including smartphones, tablets, and servers. TrustZone achieves security by partitioning the hardware and software resources of a device into two worlds: the secure world and the non-secure world. The secure world is used to run critical applications, such as the operating system and the bootloader. The non-secure world is used to run all other applications.

TrustZone provides a number of security features, including:

* **Isolation**: The secure world and the non-secure world are isolated from each other. This means that code running in the secure world cannot access code or data in the non-secure world, and vice versa.
* **Trusted boot:** TrustZone ensures that only trusted code can be loaded into the secure world. This helps to prevent malicious code from gaining control of the system.
* **Secure storage:** TrustZone provides secure storage for sensitive data, such as passwords and encryption keys. This helps to protect this data from unauthorized access.

**Minimal Security Hardware**

TrustZone can be implemented with a variety of different security hardware components. The minimal security hardware required for TrustZone is a TrustZone-enabled CPU, a TrustZone-aware operating system, and a TrustZone-enabled secure boot ROM.

In addition to the minimal security hardware, TrustZone can also be enhanced with additional security hardware components, such as a trusted execution environment (TEE), a memory protection unit (MPU), and a physical unclonable function (PUF).

* Trusted execution environment (TEE): A TEE is a secure environment that can be used to run trusted applications. TEEs are typically implemented in hardware, but they can also be implemented in software.
* Memory protection unit (MPU): An MPU is a hardware device that can be used to protect memory from unauthorized access. MPUs are typically used to protect the secure world from the non-secure world.
* Physical unclonable function (PUF): A PUF is a hardware device that can be used to generate a unique identifier for a device. PUFs can be used to implement security features, such as device authentication and secure boot.
* **Security Guarantees**

TrustZone provides a number of security guarantees. These guarantees are dependent on the specific security hardware components that are used.

* Isolation: TrustZone ensures that the secure world and the non-secure world are isolated from each other. This means that code running in the secure world cannot access code or data in the non-secure world, and vice versa. This helps to protect the secure world from attacks that target the non-secure world.
* Trusted boot: TrustZone ensures that only trusted code can be loaded into the secure world. This helps to prevent malicious code from gaining control of the system.
* Secure storage: TrustZone provides secure storage for sensitive data, such as passwords and encryption keys. This helps to protect this data from unauthorized access.

**Security Features of Blockchain (Smart Contracts)**

Blockchain is a distributed ledger technology that is used to record transactions in a secure and transparent manner. Smart contracts are self-executing contracts that are stored on the blockchain. They can be used to automate a wide range of tasks, such as financial transactions, supply chain management, and voting.

Blockchain and smart contracts provide several security features, including:

* Transparency: All transactions on the blockchain are public and can be verified by anyone. This makes it difficult to tamper with counterfeit transactions.
* Immutability: Once a transaction is added to the blockchain, it cannot be changed. This makes it difficult to hack or modify the blockchain.
* Security: The blockchain is secured by cryptography. This makes it very difficult to attack the blockchain or steal data from it.

**Comparison of Security Guarantees**

The security guarantees provided by TrustZone and blockchain are similar. Both technologies provide isolation, trusted boot, and secure storage. However, there are some key differences between the two technologies.

* TrustZone is a hardware-based security technology, while blockchain is a software-based security technology. This means that TrustZone is typically more secure than blockchain. However, blockchain is more flexible and scalable than TrustZone.
* TrustZone is typically used to protect the operating system and other critical applications, while blockchain is typically used to protect data and transactions. This means that TrustZone is typically used to protect the system from attack, while blockchain is typically used to protect data from theft or modification.