**Application of Blockchain Technology for Integrated Cyber Physical Systems.**

The coming decades may see the large-scale deployment of networked cyber–physical systems (CPSs) to address global needs in areas such as energy, water, health care, and transportation. CPSs are intelligent systems that integrate control, communication, and computing. CPSs are also classical complex networks. Based on intelligent perception and cyber communication, CPSs can realize the deep integration and real-time interaction of calculation, communication, and control through the mutual influence of cyber computing and physical process. It can also detect and control the physical system in a safe, reliable, efficient, and real-time way, and realize self-governance and cooperation of the whole system. However, the physical system and cyber system in CPSs are highly integrated. Meanwhile, their cyber acquisition, communication, calculation, and control are deeply coupled, which brings profound challenges to related fields, such as cyber attacks, high dimensional data disasters, communication network jam, etc. Other setbacks in the implementation of CPS are largely due to its centralised approach and poor implementation of security/privacy mechanism,

In a centralize system, when a node is attacked; the system becomes vulnerable, and the entire process is disrupted. One good way to solve this problem is setting up a decentralized system like blockchain technology.

**Case Study:**

**Fire Detection and Suppression System**

**Description:** The system is designed to provide optimum safety against fire and smoke in a given compartment by detecting the type of fire and providing a specific fire suppression solution.

**Blockchain Application**

**Objectives:**

The relevant applications of blockchain technology to the case study involves the development of a time-based ledger system that stores the information of the different nodes (sensor values) in real time and also to create a time stamp when any action is performed. This eliminates ambiguities in record and causes of actions.

In principle when flame is present, the system determines the cause of the fire and create a ledger to record the actions with time stamp. This record is made for every action.

The advantages of this application in terms of the business model of operation is to create transparency, immutability, visibility and traceability, decentralized structure with improved security and privacy. This is significant to situations such as insurance and econometrics cases.

Diagram

Description automatically generated with low confidence

**Flowchart**

**Key Points:**

* Contract is created using Solidity version >=0.5.0 and 0.8.0
* The values of the different nodes (sensors) are received and communicated to ESP8266 Arduino Board, but in this case, it is required to input the values manually.
* This values are solved in keccak256 hash function of abi.encodePacked(\_str). This is used to generate a pseudo-random hexadecimal.
* The registry call function calls/display the array of the node, user ID and the time stamp for every action step.
* //SPDX-License-Identifier: GPL-3.0

Solidity Version

* pragma solidity >=0.5.0 <0.8.0;
* contract AlarmRegistry {
* uint idDigits = 16;
* uint idModulus = 10 \*\* idDigits;
* struct FireMen {
* string \_UserID;
* uint \_Flame;

Registry Structure

* uint \_Electric;
* uint \_Gas;
* uint \_timestmp;

Hash Function

* }

Array

* FireMen[] public Registry;
* function \_generateId(string memory \_str) private view returns (uint) {
* uint rand = uint(keccak256(abi.encodePacked(\_str)));
* return rand% idModulus;
* }
* function  ClassofFire(string memory \_UserID, uint \_Flame, uint \_Electric, uint \_Gas)public{
* uint randId = \_generateId(\_UserID);
* \_registerAction(\_UserID,\_Flame,\_Electric,\_Gas);
* SensorValue1[randId] = \_Flame;
* SensorValue2[randId] = \_Electric;
* SensorValue3[randId] = \_Gas;
* CheckingSensorValues(randId);
* }

Function to input the Node values

* function CheckingSensorValues(uint \_values) private view

Initial values of the Nodes before contract can be executed

* SensorValues(1,0,0,\_values)  {}
* mapping (uint => uint) private SensorValue1;

Function to store the nodes value at every action

* mapping (uint => uint) private SensorValue2;
* mapping (uint => uint) private SensorValue3;
* modifier SensorValues (uint \_Flame, uint \_Electric, uint \_Gas, uint \_values) {
* require(SensorValue1[\_values]  == 1,"Flame Must be First Detected");
* require (SensorValue2[\_values] <= 1,"Digital Output of Electrical Connectivity");
* require (SensorValue3[\_values] <= 1,"Digital Output of Gas Sensor");

Condition for the Contract to be executed

* // NOTE:
* /\*if (SensorValue1[\_values] == 1){
* if (SensorValue2[\_values] == 1) {
* revert("Electric Fire - Class\_C Fire");
* }
* else if (SensorValue3[\_values]  == 1) {
* revert("Gas Fire - Class\_B Fire");
* }
* }
* else
* {

Create an event and communicatee to the front end

* revert("Other Fire - Class\_A Fire");
* }\*/
* \_;
* }
* event registerAction(uint SensorValue\_1, uint SensorValue\_2, uint SensorValue\_3,uint Time);
* function \_registerAction(string memory \_UserID, uint \_Flame, uint \_Electric, uint \_Gas) private {

Registry Call function

* Registry.push(FireMen(\_UserID, \_Flame, \_Electric, \_Gas, block.timestamp));
* emit registerAction(msg.value, msg.value, msg.value, block.timestamp);
* }

Emit changes to the front end in case the node values are modified

* }

**Deployment and Testing**A screenshot of a computer

Description automatically generated with medium confidence

* Text

  Description automatically generatedUser to input node values and User ID that acknowledges the alarm
* Text

  Description automatically generatedFirst set of node values are imputed. From the input, it means flame is present and the cause of the flame is due to Class-A type of fire. Check //NOTE for explanations.
* Second set of node values are imputed. From the input, it means flame is present and the

cause of the flame is due to Class-C type of fire. Check //NOTE for explanations Text

Description automatically generated

* Third set of node values are imputed. From the input, it means flame is present and the

cause of the flame is due to Class-B type of fire. Check //NOTE for explanations

Text

Description automatically generated

* In order to display the ledger for the different Action steps, we need to input the level of the action steps.

Where 0 corresponds to the values of the initial stored values with time stamp in UNIX.

Where 1 corresponds to the values of the second stored values with time stamp in UNIX.

This continues up to infinity action level.

Text

Description automatically generated

Call Function

Initial level

Unix Time of Action

Initial Stored Values

Text

Description automatically generated

Unix Time of Action

2nd level

2nd Stored Values

Call Function

Text

Description automatically generated

Call Function

Unix Time of Action

3rd Stored Values

2nd level

**Conclusions.**

The program was developed in line with the prescribe objective to create a time-based ledger system from the application of blockchain technology. It stores different values of the nodes with a time stamp for every step of action. This program is highly relevant and creates transparency, immutability, visibility and traceability, decentralized structure with improved security and privacy to the system.

This program can be adopted for various process such as insurance and econometrics purposes, time-based action reporting system and other field of operations.

Recommendations.

* Further enhancement can be performed to provide proper security to the ecosystem against phishing attacks.
* It is also important to integrate the program into the CPS through a secured communication and reliable monitoring interphase.