Done – Add node, add block, get block by inputting unique ID

To do – add an array of accepted block hashes in the struct Person

/\*\*

   \* @title ContractName

   \* @dev ContractDescription

   \* @custom:dev-run-script file\_path

   \*/

// SPDX-License-Identifier: MIT

pragma solidity 0.8.7;

contract MyContract {

    uint256 public peopleCount = 0;

    uint \_flag;

    mapping(string => Person) public people;

    mapping(address => Block) public merkletree;

    address owner;   // owner is recognized by his address

    modifier onlyOwner() {

        require(msg.sender == owner);

        \_;

    }

    struct Person {

        uint id;

        string \_name;

        uint \_reputation;

        string \_uniqueID;

    }

    struct Block {

        string \_message;

        uint \_nonce;

        //uint \_reputation;

        string \_uniqueID;

        uint \_timestamp;

        address \_blockAddress;

    }

//    struct RetrieveBlock {

//        string \_uniqueID;

//        BlockHash[] \_blockhash;

//    }

//    struct BlockHash {

//        address \_Address;

//    }

    constructor() public {

        owner = msg.sender;

    }

    function addPerson (string memory \_name, uint \_reputation, string memory \_uniqueID, address \_blockhash) public onlyOwner {

        // here \_blockhash = \_blockAddress of addBlock - represents hash of the latest block added by node

        string memory message = "abc";

        string memory uniqueID = "ac1";

        uint256 nonce = 123;

        uint reputation = 6;

        uint timestamp = 40; //block.timestamp

        bytes32 hash = 0x7465737400000000000000000000000000000000000000000000000000000000;

        string memory secret = 'aa';

        //

        // decrypt \_h and give values

        verifyNode(hash, message, uniqueID, nonce, reputation, timestamp);

        if ((\_flag == 1) && (people[\_uniqueID].\_reputation > 5)) {

            people[\_uniqueID] = Person(peopleCount, \_name, \_reputation, \_uniqueID);

            //incrementRep(people[\_uniqueID].\_reputation);

            people[\_uniqueID].\_reputation = people[\_uniqueID].\_reputation + 1;

            incrementCount();

        }

        else {

            //decrementRep(people[\_uniqueID].\_reputation);

            people[\_uniqueID].\_reputation = people[\_uniqueID].\_reputation - 1;

        }

    }

    // add block with zpk

    function addBlock (string memory \_message, uint \_nonce, string memory \_uniqueID, uint \_timestamp, address \_blockAddress) public onlyOwner {

        uint \_FLAG;

        //string memory secret = 'aa';

        string memory pass;

        //

        guessPassword(pass);

        //if (\_FLAG == 1) {

        merkletree[\_blockAddress] = Block(\_message, \_nonce, \_uniqueID, \_timestamp, \_blockAddress);

        //}

    }

//    function viewBlock (string memory \_uniqueID, address \_blockAddress) public onlyOwner {

//

//    }

    function incrementCount() internal {

        peopleCount += 1;

    }

//    function computeZPK(string memory \_message, string memory \_uniqueID, uint \_nonce, uint \_reputation, uint \_timestamp) internal {

//        bytes32 h = keccak256(\_message, \_uniqueID, \_nonce, \_timestamp);

//        //encrypt E(h, message, uniqueID, nonce, timestamp)

//    }

    function verifyNode(bytes32 h, string memory \_message, string memory \_uniqueID, uint \_nonce, uint \_reputation, uint \_timestamp) internal onlyOwner returns(uint, bytes32) {

        //uint \_flag;

        //decrypt D(h, message, uniqueID, nonce, timestamp)

        // assume message is decrypted by base station. So, declare all the values here

        //uint flag = 1;

        string memory message = "abc";

        string memory uniqueID = "ac1";

        uint256 nonce = 123;

        uint reputation = 6;

        uint timestamp = 40;

        bytes32 h = 0x7465737400000000000000000000000000000000000000000000000000000000;

        //

        //bytes32 \_h = keccak256(abi.encodePacked(\_message, \_uniqueID, \_nonce, \_timestamp));

        bytes32 \_h = 0x7465737400000000000000000000000000000000000000000000000000000000;

        if (h == \_h) {

            \_flag = 1;

        }

        else {

            \_flag = 0;

        }

        return (\_flag, \_h);

    }

    // guess password for authentication using ZPK

    function guessPassword(string memory \_guessPassword) public onlyOwner returns(uint) {

        string memory \_secret = 'aa';

        uint \_FLAG;

        bytes32 \_h = keccak256(abi.encodePacked(\_secret));

        bytes32 PASS = keccak256(abi.encodePacked(\_guessPassword));

        if (PASS == \_h) {

            \_FLAG = 1;

        }

        else {

            \_FLAG = 0;

        }

        return (\_FLAG);

    }

//    function incrementRep(uint \_reputation) internal onlyOwner returns(uint) {

//        \_reputation += 1;  // reputation of only particular device should be changed

//        return \_reputation;

//    }

//    function decrementRep(uint \_reputation) internal onlyOwner returns(uint) {

//        \_reputation -= 1;  // reputation of only particular device should be changed

//    }

}