Public Blockchain-based Data Integrity Verification for Low-power IoT Devices

Jing Huey Khor, Michail Sidorov, Ming Tze Ong, Shen Yik Chua

***Abstract* — The integration of sensor nodes with public blockchains is possible with the help of low-power communication networks that use Bluetooth Low Energy and LoRa. However, power-consuming Wi-Fi is still the main means of communication for the existing sensor nodes, especially in urban environments. Typically high power consumption, private key disclosure, and high transaction fees are the issues that prevent battery-powered sensor nodes from being integrated with a public blockchain. Therefore, this paper proposes a data protection protocol that is able to secure the data integrity of the stored sensor data, help to reduce transaction fees, and prolong battery life for IoT devices that are used with public blockchain networks. A proof of concept is presented using an ESP32S2 device to evaluate and verify the performance of the proposed data storage protocol. A smart contract is designed and analyzed using a formal smart contract analysis tool. A decentralized web application is designed to display and verify the sensor data extracted from the public blockchain. The power consumption, memory usage, and security of the proposed solution are evaluated. The evaluation results show that data integrity can be achieved even for low-power sensor nodes that connect to public blockchains via Wi-Fi network.**

***Index Terms*—Blockchain, Data Integrity, IoT**

I. Introduction

Data associated with decentralized applications can be stored in several ways - either on the blockchain itself, a method which is called on-chain storage, or using external databases, making this an off-chain solution. Since on-chain storage becomes expensive for large amounts of data, off-chain solutions such as CouchDB, StateDB, etc., can be used to store vast amounts of sensitive data in a cost-efficient manner. However, raw data stored using off-chain storage is susceptible to modification or deletion by attackers. Hence, on-chain storage becomes essential for certain IoT applications, such as Structural Health Monitoring (SHM) [1, 2], where an immutable record and transparent public access is required. IoT applications like this store only small amounts of data collected by sensor nodes, e.g., temperature, humidity, preload, etc., in which case on-chain storage becomes a viable solution compared to the off-chain one. Depending on the application, this sensory data can then be utilized for various smart city services. In case of SHM, for example, citizens in close proximity to a certain structure can be notified of a developed fault to stay away. This would be made possible by smart city services reading the on-chain data continuously stored by sensor nodes monitoring the abovementioned structures.

Using on-chain storage is not without challenges. High power consumption, vulnerability to security attacks, and high transaction fees are among the typical issues associated. Most small IoT devices are battery-powered; therefore, using Wi-Fi to connect to a blockchain is a power-hungry approach. However, Wi-Fi is the dominant wireless protocol, especially in a smart city environment [3], even though power-efficient communication protocols such as Bluetooth Low Energy (BLE) or Long-Range (LoRa) are available for IoT applications. Existing IoT devices typically support older IEEE 802.11b/g/n/ac standards, although a few devices support the newer power-saving IEEE 802.11ax (Wi-Fi 6) standard, e.g., ESP32-C6 [4]. Therefore, there is a need to address the power consumption issue found among the low-end IoT devices that use older Wi-Fi protocols and not the newer Wi-Fi 6 or other aforementioned communication protocols. In addition, IoT devices placed in remote areas are susceptible to security attacks where the stored sensory data and the program executed by the IoT device can be modified if the device is accessed locally. Thus, sensor data is usually sent to a local server, cloud server, or blockchain directly for storage instead of being retained on the IoT device itself.

Storing one data packet on the blockchain constitutes one transaction, and transactions are not free. Different blockchains have different pricing for this, e.g., at the time of writing, the average Ethereum transaction cost is 4 USD [5]; however, it fluctuates greatly depending on the network usage. Transaction fees should be reduced once the upgrade to Ethereum 2.0 is finalized. However, this is not scheduled for several years from now. Therefore, a second layer scaling solution, e.g., Polygon blockchain [6], where the transaction fee is 0.00004 USD can be used instead. Although the cost is lower, IoT applications normally use a large number of connected devices, and these devices typically send data frequently. Hence, a large number of transactions is going to be made, which can result in high running costs. Therefore, to solve the issues this paper aims to provide cost-efficient and secure data protection for public

[T]his paragraph of the first footnote will contain the date on which you submitted your paper for review. This work was supported by the International Scholar Exchange Fellowship program at the Chey Institute for Advanced Studies. This work was further carried out during the tenure of an ERCIM 'Alain Bensoussan' Fellowship Programme.

J. H. Khor is with University of Southampton Malaysia, Eko Botani 3, Taman Eko Botani, 79100 Iskandar Puteri, Malaysia and with the Barun ICT Research Center, Yonsei University, 50 Yonsei-ro, Seodaemun-gu, Seoul, South Korea (e-mail: J.Khor@soton.ac.uk)

M. Sidorov is with Norwegian University of Science and Technology, Høgskoleringen 1, Trondheim, Norway. (e-mail: michail.sidorov@ntnu.no).

M. T. Ong is with Digipen-The One Academy, Block B4, Leisure Commerce Square, No.9, Jln PJS 8/9, 46150 Petaling Jaya, Selangor. (e-mail: mingtze.ong@digipen.edu)

S. Y. Chua is with University of Southampton, University Road, Southampton SO17 1BJ, United Kingdom (email: syc2e18@soton.ac.uk).

blockchain-enabled low-power IoT applications in urban environments where Wi-Fi connections dominate. In order to support low-power IoT devices, the sensor data will be stored on the IoT device itself for a certain period of time and then will be grouped into a single transaction to reduce power consumption and lower transaction fees. In addition, a lightweight data protection protocol is designed to guarantee the integrity of sensor data while it is stored on the IoT device itself.

The main contributions of this paper are as follows:

- A new lightweight protocol is proposed using chain hashing for data integrity protection.
- A proof of concept is presented based on the ESP32S2 low-power, single-core Wi-Fi System-on-Chip from Espressif to verify the performance of the proposed protocol.
- A smart contract is designed to send a batch of data in a single transaction to save transaction costs and reduce power consumption.
- A decentralized web application is developed to display and verify the IoT sensor node data.
- Power consumption savings are achieved using an intermittent Wi-Fi connection that prolongs battery life.
- An emergency real-time monitoring mechanism is used to write the stored data to the blockchain once an anomaly is detected.

II. Related Work

Internet of Things (IoT) devices have been widely integrated with various blockchain applications. However, most of those devices were not used as blockchain nodes due to low power and low computational capabilities. Instead, powerful IoT devices, such as proxy servers [7], computers [8-10], and gateways [1, 11, 12] were used as blockchain nodes to make transactions using received data from resource-limited IoT devices. In order to use low-power devices as blockchain nodes, customized lightweight consensus algorithms are needed to add transactions to the blockchain [13, 14]. These consensus algorithms include customized Practical Byzantine Fault Tolerance, Proof of Authentication, etc. It is possible for low-end IoT devices to directly transact on the Ethereum blockchain via Infura without the need of deploying a full node [15]. However, high power budget is required to send transactions via Wi-Fi or cellular networks, which is an obstacle preventing the integration of resource-constrained IoT devices with the Ethereum blockchain network.

Several solutions exist that enable the integration of low-cost IoT devices with blockchain networks, such as Nodle and Helium. Nodle is a decentralized wireless network that is built on top of the Polkadot blockchain, and it utilizes BLE to establish a connection between IoT devices and smartphones [16]. The Rendez-Vous Protocol is used to secure the communication network between two endpoints. In addition, a security stack is provided to enable IoT devices to send signed and encrypted data anonymously [17]. However, the Nodle network is only suitable for those who do not require real-time monitoring. Data collected by a sensor device will only be sent to smartphones when they are in close proximity. In addition, the Nodle network relies highly on the number of smartphones on the network. The owners of smartphones are incentivized with Nodle Cash rewards whenever their phones transmit a packet to the network, therefore attracting more users to expand the network [16]. However, this incentive might not be enough as at this point in time as the monetary value of 1 Nodl stands at 0.006 USD. At the time of writing this paper, there were approximately 4,600,000 active nodes on the network, which is around 0.065% of the total mobile devices used around the world [18].

Helium is another blockchain aimed at IoT and connected devices. Similar to Nodle, the Helium blockchain relies on Helium Hotspots. There is a number of manufacturers that produce these hotspots and currently, a large number of additional ones are joining. Helium calls its communication protocol LongFi; however, the underlying technology is nevertheless LoRaWAN. In order to encourage the use and distribution of Helium Hotspots, a cryptocurrency called HNT is rewarded to the owners of Helium Hotspots [19]. Hotspot owners get rewards for providing coverage for the Helium network by participating in the Proof of Coverage consensus algorithm and by transferring data received from the connected sensor nodes. Helium hotspots devices cost in the range of USD250-700 [20], which is quite expensive, as the device mainly acts as a packet forwarder with almost zero configuration possible from the user side which typically can be found in commercial LoRaWAN gateways. Currently, Helium Hotspot devices mainly concentrate in United States, Europe, and East China [21]. This means other parts of the world will not have proper Helium network coverage to communicate with IoT devices yet and this is mainly due to the regulatory approvals. Since Helium hotspots are new, and network is incredibly immature, manufacturers need to go through a lengthy process of getting their devices certified to comply with local radio laws for the hotspots to be safely used in the respective country. However, the demand to join the Helium network is incredibly high, therefore the adoption will grow faster than The Things Network, or any alternative LoRaWAN network.

The aforementioned shows that there is a degree of research done and that some solutions that integrate IoT devices with public blockchains do exist. In addition, there are several research works that guarantee sensor data integrity when used with cloud storage [22-26] or local storage [14]. However, at this time of writing, no research has focused on data protection for low-power IoT devices used in public blockchain networks. Thus, this paper fills the research gap.

III. Data Protection Protocol

The following section describes the proposed protocol. The purpose of the node is to collect sensory data. This data, together with other data (sensor ID, *startTime*, *deltaTime*, and hash value) is stored in the Real Time Clock (RTC) memory. Since the sensor data that is stored in the RTC memory is not encrypted, there is a possibility that attackers can manipulate it. Therefore, a protocol is designed based on chain hashing to ensure data integrity for the RTC memory. This chain hashing increases the complexity by using different sensor data for each session, a secret key, and a previous hash value as its input data. The

proposed protocol consists of two phases, namely data storage and verification. Initially, a 256-bit secret key and an initial hash value are generated and are stored in the sensor node using the methods described in section IV.A. The same secret key and initial hash value will also be stored at a more powerful machine for data validation purposes, hereafter referred to as a Validation Machine (VM).

A. Data Storage and Data Verification Phases

The data storage phase of the proposed protocol is designed, as shown in Figure 1. The sequence is further described below:

1. An IoT node wakes up from a deep sleep mode, collects the data (i.e., temperature and humidity), and stores it in the RTC memory, together with its sensor ID, *startTime*, and *deltaTime*.
2. The IoT node hashes the concatenated stored data, secret key, and the initial hash string value to generate a 256-bit hash value using the SHA256 hash function.
3. The IoT node goes back to deep sleep mode.
4. The IoT node switches to an ON mode from a deep sleep mode for the next data collection session to sense data and add the data to RTC memory. The IoT node then hashes the stored data with the secret key, and the previous session generated hash value to obtain a new hash value.
5. Steps 3-4 then repeat a further 14 times, upon which at the end of the 16th session, each of the temperature, humidity, and deltaTime arrays contain 256 bits of data.
6. The IoT node then calls the *setInput* function from the smart contract to generate a signed transaction. The *setInput* function stores data obtained from the 16 sessions (256-bit each), including *startTime*, sensor ID, temperature, humidity, *deltaTime*, as well as the last hash value on the blockchain as input data.
7. The IoT node connects to Wi-Fi and broadcasts the signed transaction to the Ethereum public blockchain.
8. After sending the transaction, the IoT node is set to deep sleep mode.

During the data verification phase, a decentralized web application is used to verify the validity of the collected data as described in section IV. This application uses the data from transactions to generate a sequence of hash values. Data integrity and authentication scheme, which is described in the next section, only requires comparing the last hash value of the hash sequence with the one stored on the blockchain. If they are the same, the web application reflects *PASS* in the Verification column to show that the data is original and has not been modified. Otherwise, a *FAIL* will be shown in the web application to indicate the data has been modified.

B. Data Integrity and Authentication Scheme

As mentioned in section III.A, at the end of every 16th session, the IoT node sends data that it has collected in a single transaction to the blockchain. Since existing resource-constrained IoT nodes by themselves are not secure, a data integrity scheme for a VM to determine if the data that the IoT node has included in transactions to the blockchain is valid and authentic, needs to be designed and implemented. A very simple, efficient, and reliable data authentication scheme using the secure cryptographic hash function SHA256 combined with chain hashing is used. Every IoT node has a secret *key* which the VM knows. As data is being collected at every i^{th} session, $1 \leq i \leq 16$, the IoT node calculates, based on collected data described in section IV.A, the chained hash sequence: $h_{SHA256,i}(startTime_i|deltaTime_i|SensorID_i|Temperature_i|Humidity_i|h_{SHA256,i-1}|key)$, where $h_{SHA256,0}$ = 000...0 and | denotes concatenation. The only hash value that the IoT node writes to the blockchain at the end of the transaction period (i.e. the 16th session) is $h_{SHA256,16}$. To detect if an adversary has maliciously altered any data that the IoT node writes to the blockchain, the VM itself calculates its own sequence of hash values :

$VM_{SHA256,i}(startTime_i|deltaTime_i|SensorID_i|Temperature_i|Humidity_i|VM_{SHA256,i-1}|key)$, $1 \leq i \leq 16$, where $VM_{SHA256,0}$, = 000...0 with $startTime_i, deltaTime_i, Temperature_i$ and $Humidity_i$ here being data values read from the blockchain. If $VM_{SHA256,16}$ and

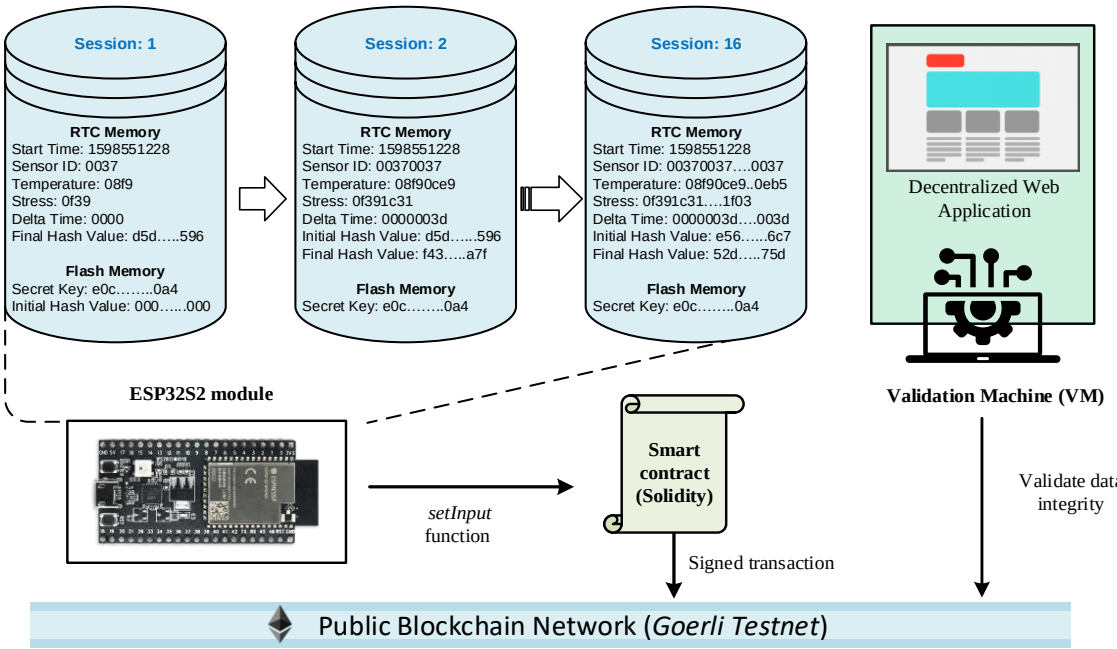

Fig. 1. Data Storage Phase

$h_{SHA256,16}$ match, the VM declares the data on the blockchain to be authentic via the decentralized web application mentioned in section IV.B.

IV. Proof Of Concept

A. Data Collection and Storage

A system configuration shown in Table I is used to verify the validity of the proposed scheme. According to [27], IoT devices can be categorized into 4 categories based on their processing capability and power consumption. A Lenovo T14s laptop which belongs to Class IV is used to simulate a powerful machine used for verification of transaction data on the Ethereum public blockchain. For the proof of concept, the Ethereum Goerli testnet is used instead of the Ethereum mainnet to transact without the use of real Ether. While other Ethereum testnets are available such as Kovan, Rinkeby, or Ropsten, these testnets are currently being deprecated, i.e. they will not receive any new network updates in the future and support for them is scheduled to end. Hence, they are not recommended for new smart contract deployment. A remote node, called Infura, is used to access the Goerli testnet through Application Programming Interface (API) for convenience. This approach eliminates the need of deploying a full Ethereum node on-site. The low-power sensor node was developed using the ESP32S2 module [28], which belongs to Class II of IoT devices. IoT sensor nodes are typically battery-powered and sometimes used for monitoring in places where easy access is not available. While a USB-powered ESP32-S2-WROVER version of the board was used for prototyping purposes, attaching a battery or using a similar battery-powered board, e.g., ESP32-S2-WROVER-DevKit-Lipo [29] is an easy transition. The primary application of the designed sensor node was to monitor the temperature and humidity in a building. Since this paper focuses on data storage protection for IoT devices, a built-in temperature sensor inside the ESP32S2 was used for simplicity. An SMTHS08A humidity sensor was used as an external sensor and was connected to the EPS32S2 input port. To provide real-time data monitoring, the ESP32S2 system synchronizes its clock with the internet time, as shown in [30]. Each sensor data is stored in a *uint16_t* array in the RTC memory, transformed into a 4-letter hexadecimal string, as shown in Figure 2.

B. Decentralized Web Application

A smart contract named *Iot.sol*, was designed using Solidity to store the data on the blockchain. This smart contract has one

Table I: System configuration

Description	Powerful Machine	Low-power IoT device
IoT Device Class	IV	II
Name	Lenovo T14s	ESP32-S2 SoC
Operating System	Windows	NA
Processing unit type	AMD Ryzen™ 5 Pro 4650U 64-bit CPU, clock speed up to 4 GHz	Xtensa® low power single-core 32-bit LX7 MCU, clock speed up to 240 MHz
RAM (KB)	16,000,000	320
ROM (KB)	NA	128
NVRAM (GB)	475	-

```
Data Count 16
Start Time 6142a6c6
Sensor ID 0037003700370037003700370037003700370037003700370037003700370037
Sensor Temperature  0a550a0f0b380b260ae00acf0a890a77081c080b0b490b370af10ae00885083f
Sensor Humidity  0f950ed311ef112d106c0faa0ee80e26114210800fbe0efc0e3a11c410940fd2
Delta Time 0000000b000a000b000b000b000b000b000a000b000b000b000b000b000a000b
```

Fig. 2. Sensor data stored in the RTC memory

function, named *setInput()*. This function is used to set data as input in a transaction. This smart contract was compiled on a powerful machine and deployed to the blockchain. In addition, a web application was designed using HTML to interact with the smart contract via web3j to display the data, as shown in Figure 3. Figure 4 shows the *setInput()* function in the *Iot.sol* smart contract. There are 3 sensor related parameters passed to the smart contracts *setInput()* function, namely sensor ID, temperature, and humidity. As each sensor data is stored in a *uint16_t* array, the maximum value of each data is capped to 2^{16}. However, any data type in a smart contract will take up at least 256-bits in a transaction. Thus, each transaction can store data from 16 data collection sessions. The data is linked together as a *c* style character string (cstring), representing a *uint256* hexadecimal string. Data for each session can then be passed to the 256-bit parameter of *setInput()* function. A character string instead of an array is used as input to the smart contract method because the resultant hexadecimal string is much longer. Each element in the array will occupy 64 characters and cause the input data to become unnecessarily long [31].

There are two time-related parameters passed into the set *input()* function, namely *startTime* and *deltaTime*. The *startTime* is stored as Unix timestamp format (time_t), which is 32-bit. However, it will be padded with 224-bit zeros when being passed to the parameter of the smart contract. In contrast, the *deltaTime* is stored as a uint16_t. The *startTime* is a reference time where the ESP32S2 starts logging data for these 16 sessions, whereas the *deltaTime* is the interval between two session reading times. Every time sensor data is read, the difference between previous and current data in seconds is stored in the *deltaTime* array. The exact time corresponding to the current sensor ID is calculated by adding the *startTime* with the sum of *deltaTime* from the first reading to the current reading, as shown in Table II. By having these two time-related parameters, the data from 16 sessions can be grouped in a transaction with the minimum parameters used for the *setInput()* function. The last parameter of the smart contract is to store the last session of the hash value on the blockchain for data verification purposes. The details of this hash value can be found in section III.B. Table III shows the data size for each parameter for one session and the 16th session. The data in the 16th session are the arguments that will be passed on to the *setInput()* function.

The average transaction fee to store data on a blockchain

```
function setInput (uint _sensorIds, uint _temperatures, uint _humidities,
 uint _startTime, uint _deltaTimes, uint _hash) external onlyOwner{
            sensors[_sensorIds].sensorId=_sensorIds;
            sensors[_sensorIds].temperature.push(_temperatures);
            sensors[_sensorIds].humidity.push(_humidities);
            sensors[_sensorIds].startTime.push(_startTime);
            sensors[_sensorIds].deltaTime.push(_deltaTimes);
            sensors[_sensorIds].hash=_hash;
}
```

Fig. 4. A snippet of *Iot.sol* smart contract

Timestamp	Sensor ID	Temperature	Humidity	Data Hash	Transaction Hash	Verification
Aug-28-2020 2:15:45	0037	37.65	79.39	0x52d312e5b57a947b019afb87884f52f7471383d783d64ac9a7f05e0f4372875d	0xfc25908368122e8e4e0cc431ab71415950369f5fe5a8ec16eeabbfd0df1778ed	Pass
Aug-28-2020 2:14:44	0037	29.13	99.94	0x52d312e5b57a947b019afb87884f52f7471383d783d64ac9a7f05e0f4372875d	0xfc25908368122e8e4e0cc431ab71415950369f5fe5a8ec16eeabbfd0df1778ed	Checked
Aug-28-2020 2:13:42	0037	38.46	16.87	0x52d312e5b57a947b019afb87884f52f7471383d783d64ac9a7f05e0f4372875d	0xfc25908368122e8e4e0cc431ab71415950369f5fe5a8ec16eeabbfd0df1778ed	Checked
Aug-28-2020 2:12:41	0037	45.58	83.92	0x52d312e5b57a947b019afb87884f52f7471383d783d64ac9a7f05e0f4372875d	0xfc25908368122e8e4e0cc431ab71415950369f5fe5a8ec16eeabbfd0df1778ed	Checked
Aug-28-2020 2:11:40	0037	45.75	24.961	0x52d312e5b57a947b019afb87884f52f7471383d783d64ac9a7f05e0f4372875d	0xfc25908368122e8e4e0cc431ab71415950369f5fe5a8ec16eeabbfd0df1778ed	Checked

Fig 3. Decentralized web application excerpt

Table II: Sensor startTime and deltaTime values in a hexadecimal format

Description	Value		
Start Time	0x6142a6c6		
Delta Time	0x0000000b000a		
Reading	1st	2nd	3rd
Delta Time	0x0000	0x000b	0x000a
Total Delta	0x0000	0x000b	0x0015
Sensor Time	0x6142a6c6	0x6142a6d1	0x6142a6db

Table III: Simulation Parameters

Description	Data Type (ESP32)	Data Type Conversion for smart contract	Data Type (smart contract)	Data size for 1 session (bit)	Data size for the 16th session (bit)
startTime	time_t	char[64]	Uint	256	256
deltaTime	uint16_t	char*	Uint	16	256
Sensor ID	uint16_t	char*	Uint	16	256
Temperature	uint16_t	char*	Uint	16	256
Humidity	uint16_t	char*	Uint	16	256
Data Hash	char[64]	char*	Uint	256	256

collected during a single session is about 0.000163 ETH, while the transaction fee to store data from 16 sessions in a transaction is about 0.0019 ETH (approximately 0.000119 ETH per session). The details of the transaction can be found on the Goerli testnet [32]. Once this smart contract is deployed on the Ethereum mainnet, the transaction fee can be reduced due to integration with the Polygon Layer 2 blockchain. Since the 16 session data set is grouped into a single transaction, the transaction fee for a single data set costs 0.0000025 USD using Polygon.

In order to verify a transaction, the Recursive Length Prefix (RLP) encoded transaction needs to be hashed using Keccak256, which is a SHA3 hash function. The hashed string is then signed with the senders private key using Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 curve. ECDSA signature has two 32 bytes integers, named *r* and *s*. Ethereum uses an additional one-bit *v* variable as a recovery identifier. The ECDSA function in [33] is lightweight and is meant for embedded applications. However, some modifications are needed because the ECDSA function lacks *v* parameter and has logic errors. In addition, the ECDSA in [33] produces a different signature each time due to a random *k* value used. This random *k* value is claimed to be able to prevent private key disclosure. However, Ethereum uses a deterministic signature based on the RFC 6979 standard, which generates a secure *k* value from the private key and the hashed string. Therefore, a modification was made in the *uECC_sign_with_k()* function, where the randomization of *k* was removed and replaced with a securely calculated *k* value. Infura API provides an instant access over HTTPS to the Ethereum network. The ESP-TLS is used together with mbed TLS in the backend in the *https_post_func()* function to create a secure communication between ESP32S2 and Infura full nodes [34]. The *SendSignedTransaction()* method is called from Web3 class in [35] to transmit the transaction to the Ethereum network. This function wraps the input data into JSON-RPC format using *generateJson()* function. The *exec()* function takes in the JSON-RPC data and makes a HTTP POST request to Infura. After the transaction is broadcasted, the JSON response from Infura is then parsed before returning the transaction hash to the caller function.

C. Emergency Real-time Monitoring for Anomaly Detection

Any abnormality in the monitored building will result in sensor reading deviation from normality, and ESP32S2 should immediately broadcast this event. The ESP32S2 device was programmed to receive an interrupt from sensors if the sensed value exceeded a certain threshold. In order to do so, an external wake-up (EXT1) was added to wake ESP32S2 up from the deep sleep mode and service this event. Once the ESP32S2 is awake, it signs and sends all of the stored data. After the data is transmitted, the data collection process restarts.

V. Security Protection Using ESP32S2

A. Flash Encryption

As IoT devices are normally placed in a remote environment, they are susceptible to a variety of security attacks, including key disclosure attacks. Thus, sensitive data such as program code, Ethereum account's private key, and Wi-Fi configuration should be encrypted. The ESP32S2 devices feature a Flash Encryption, where the flash content can be encrypted using AES256-XTS cryptographic algorithm with a key tweaking method based on the flash offset [36]. The Flash Encryption key can be obtained from eFuse, and the access is only possible using a hardware approach. There are two Flash Encryption modes available – Development and Release modes. This

project uses the Development mode, where the encrypted flash can be re-flashed through UART. However, Release mode is recommended for real applications as it provides a higher security protection by preventing physical readout of encrypted flash content, i.e., modification through UART by burning the eFuse of EFUSE_DIS_DOWNLOAD_MANUAL _ENCRYPT. The Release mode can prevent changes made to disable Flash Encryption using the EFUSE_SPI _BOOT_CRYPT_CNT eFuse. In the Release mode, the firmware can only be updated through Over The Air (OTA) scheme.

B. *Secure Boot*

By enabling Secure Boot, the ESP32S2 will only run a program signed with a secured key. The key can be generated locally using "espsecure.py generate_signing_key" provided by the ESP tool or generated with external software like openssl. The secure key is generated randomly based on the entropy of the system. Therefore, the more random the key generating system, the better the key security. After Secure Boot is enabled, the contents of the flash cannot be changed by unauthorized sources as they cannot sign the binary file and force upload the binary code. The level of Secure Boot security depends on how the key is generated and the key storage security level. Thus, the signing key is stored in a flash and is encrypted using the Flash Encryption.

VI. Performance Evaluation

A. *Smart Contract Analysis*

The security of the *Iot.sol* smart contract was analyzed using Mythx, which is a Software-As-A-Platform that analyzes smart contracts using a static analyzer, a symbolic analyzer, and a greybox fuzzer in parallel. A total of 900 seconds were taken to perform a standard scan mode, and there were no vulnerabilities found in the designed smart contract, as shown in Figure 5. This smart contract was checked against the 36 Smart Contract Weakness Classification (SWC) registry. These SWC registries are classified based on a list of known smart contract vulnerabilities found in the Common Weakness Enumeration (CWE) database. The details of the type of vulnerabilities covered by the Mythx can be found in [37].

B. *Security Analysis of Ethereum Blockchain Network*

Security attacks on the Ethereum blockchain can be categorized into network attacks and consensus attacks. Network attacks relate to any attacks on peer-to-peer nodes in the network using a gossip protocol and include Sybil attacks, Eclipse attacks, Distributed Denial of Service (DDoS), and Routing attacks. In contrast, consensus attacks relate to any attacks on the validation process of a block, and include double-spend attacks (i.e., Finney attacks, race attacks, selfish mining), majority attacks (i.e. 51% attacks), timejacking attacks, and quantum attacks. The analysis of these attacks on Ethereum blockchain has been well covered in [27] and it was proven that Ethereum is a secure blockchain.

C. *Security Analysis of the Proposed Scheme*

Chain hashing has been used to detect malicious data modifications as shown in [38]. However, there is no research

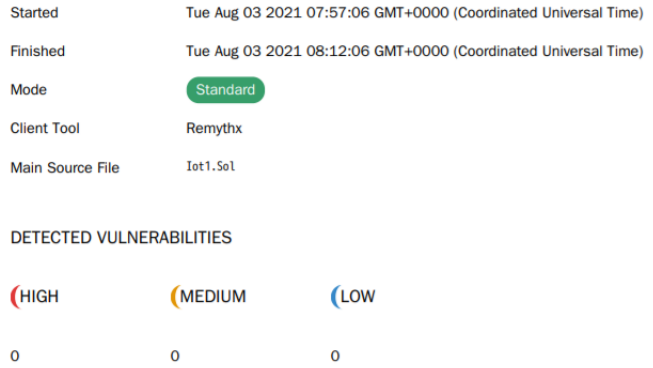

Fig 5. Mythx analysis result

on the implementation of chain hashing for data integrity protection of low-power IoT devices. Therefore, in this work, a new lightweight protocol that uses chain hashing for detecting any malicious data modifications during any communication session suitable for implementation with low-power IoT devices is introduced. Chain hashing uses cryptographic primitives for which 2nd preimage collision attacks and length extension attacks may be attempted. However, with the use of SHA256 protecting the secret key, 2nd preimage collision attacks are not feasible for at least the next ten years. It is estimated that SHA256 hash function has a 2nd preimage resistance strength between 201 to 256 bits [39, 40]. Thus, the proposed protocol allows for low-power IoT devices to be resistant to online and offline 2nd preimage collision attacks and guarantees safety from secret key discovery.

In general, adversaries might attempt to perform length extension attacks on strings that have been encrypted with a SHA256 algorithm, to alter data written to the blockchain and falsely make the VM to authenticate this data by providing a “valid hash”. Such attacks require data to be padded. Our proposed protocol, however, completely negates this since it is able to prevent length extension attacks by the integration of a smart contract, where it requires reading data of fixed lengths, as shown in section IV. This prevents adversaries from attempting to pad any data string with malicious data.

D. *Power Consumption*

The battery lifespan in [1] is benchmarked, where a lifespan of at least 5 years is targeted for this project. A Lithium Polymer Battery, with a nominal capacity of 1600 mAh was selected in this case (i.e., LP385085, LP385385, LP387062, etc) [41]. The current consumption of this device during Wi-Fi Transmit (TX) mode is higher compared to ON and Sleep modes. Deep sleep was implemented where RTC timer is the only peripheral running in the background to save the current consumption. This device wakes up regularly to read and store the data offline before going back to deep sleep, as described in Figure 6.

Since the power consumption of ESP32S2 is high when it connects to the Wi-Fi, the ESP32S2 will only connect to Wi-Fi during initialization and before sending a transaction. To further reduce the power consumption, the ESP32 will need only be connected to Wi-Fi to send a signed transaction once data has been collected 16 times. Furthermore, the boot log and ROM log are disabled to reduce startup time every time the device wakes up from the deep sleep mode. ROM log needs to be disabled by burning eFuse UART_PRINT_CONTROL to 1 and

pulling GPIO 46 high. The aforementioned power optimization is able to prolong the battery lifespan from 2.9 to 5.2 years.

Table IV shows how the proposed solution outperforms the other related solutions in terms of security protection, data integrity verification, power consumption, and transaction fees.

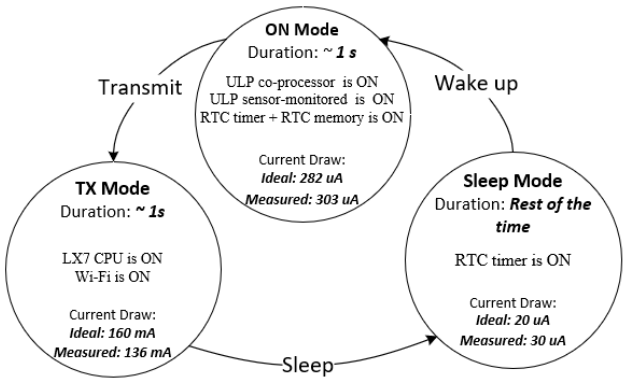

Fig 6. Current consumption of different operating modes

A computation cost and security feature comparison is not included in this table as there are no data integrity schemes implemented in the other referenced works.

Table IV: Performance comparison to existing solution

Description	Proposed Solution	Nodle [18]	Helium [21]	Okada [15]
Communication	Wi-Fi	BLE	LoRaWAN	Wi-Fi
Blockchain	Ethereum + Polygon	Polkadot	Helium	Ethereum
Security protection	Yes	Yes	Yes	No
Data integrity verification	Yes	No	No	No
Real-time anomaly monitoring	Yes	No	No	Yes
Transaction fee for one set of data (USD)	0.0000025	Undefined	0.00001 for 24 bytes	~ 4.81
Network coverage	High	Low	Low	High
Current consumption	Low	Low	Low	High

VII. Conclusion

This paper presented a blockchain-based data verification scheme for low-power IoT devices. A prototype sensor node based on an ESP32S2 device was implemented, and a firmware was written to send the stored data as transactions to the Ethereum blockchain. A smart contract was designed to set a batch of data as an input in a transaction. The smart contract was analyzed using Mythx security tool and was proven to be free from vulnerabilities. A decentralized web application was designed to display the extracted data from the blockchain. Power consumption was minimized by enabling intermittent Wi-Fi connections. The private key, secure key, and the executable program are protected using security features provided by the ESP32S2. The integrity of the stored sensor data in the RTC memory is guaranteed using the proposed data storage protocol and can be verified using the decentralized web application. An emergency system that allows for real-time monitoring was designed that sends all stored data to the blockchain once anomalous sensor values are detected, allowing for a near real-time response by the applications reading the blockchain data. This paper proves that data integrity can be achieved for low-power IoT devices that utilize Wi-Fi connections and shows that they are suitable for integration with public blockchains. The proposed data storage protection solution can be applied to any IoT devices that support hash function, flash encryption, and secure boot features. Future research will include developing a new key update system for better security protection.

Acknowledgement

We would like to thank Professor Beomsoo Kim of the Barun ICT Research Centre at Yonsei University, Seoul, South Korea for hosting Jing Huey Khor during her research visit, and Professor Jingyue Li from Norwegian University of Science and Technology for hosting Michail Sidorov during his ERCIM tenure.

References

[1] M. Sidorov, J. H. Khor, P. V. Nhut, Y. Matsumoto, and R. Ohmura, "A Public Blockchain-Enabled Wireless LoRa Sensor Node for Easy Continuous Unattended Health Monitoring of Bolted Joints: Implementation and Evaluation," *IEEE Sensors Journal*, vol. 20, no. 21, pp. 13057-13065, 2020, doi: 10.1109/JSEN.2020.3001870.

[2] G. Gürsoy, C. M. Brannon, and M. Gerstein, "Using Ethereum Blockchain to Store and Query Pharmacogenomics Data via Smart Contracts," *BMC Medical Genomics*, vol. 13, no. 1, p. 74, 2020, doi: 10.1186/s12920-020-00732-x.

[3] "Analyst: WiFi Dominates In-home Net Connectivity " Advanced-television. https://advanced-television.com/2019/08/08/analyst-wifi-dominates-in-home-net-connectivity/ (accessed 19 July, 2021).

[4] D. Zomaya. "When to Use 802.11a,b,g,b,nc: WiFi Standards." CBTNuggets. https://www.cbtnuggets.com/blog/technology/networking/when-to-use-802-11-a-b-g-b-nc-wifi-standards (accessed 10 March, 2022).

[5] TheBlock. "On-chain Metrics Ethereum." The Block Crypto. https://www.theblockcrypto.com/data/on-chain-metrics/ethereum (accessed 6 July, 2021).

[6] GraphLinq. "Polygon (Matic) Announces Their Offical Parnership With GraphLinq Protocol." GlobeNewsWire. https://www.globenewswire.com/news-release/2021/04/09/2207604/0/en/Polygon-Matic-Announces-Their-Official-Partnership-With-GraphLinq-Protocol.html (accessed 6 July 2021).

[7] K. O. B. O. Agyekum, Q. Xia, E. B. Sifah, C. N. A. Cobblah, H. Xia, and J. Gao, "A Proxy Re-Encryption Approach to Secure Data Sharing in the Internet of Things Based on Blockchain," *IEEE Systems Journal*, pp. 1-12, 2021, doi: 10.1109/JSYST.2021.3076759.

[8] W. Ren, Y. Sun, H. Luo, and M. Guizani, "SILedger: A Blockchain and ABE-based Access Control for Applications in SDN-IoT Networks," *IEEE Transactions on Network and Service Management*, pp. 4406-4419, 2021, doi: 10.1109/TNSM.2021.3093002.

[9] T. Nivan. "Arduino on Algorand Blockchain." Algorand. https://developer.algorand.org/solutions/arduino-algorand-blockchain/ (accessed 20 July, 2021).

[10] M. Sidorov, M. T. Ong, R. V. Sridharan, J. Nakamura, R. Ohmura, and J. H. Khor, "Ultralightweight Mutual Authentication RFID Protocol for Blockchain Enabled Supply Chains," *IEEE Access*, vol. 7, pp. 7273-7285, 2019, doi: 10.1109/ACCESS.2018.2890389.

[11] K. R. Özyılmaz and A. Yurdakul, "Work-in-progress: Integrating Low-Power IoT Devices to A Blockchain-based Infrastructure," in *International Conference on Embedded Software*, Seoul, South Korea, 15-20 October 2017, pp. 1-2, doi: 10.1145/3125503.3125628.

[12] L. D. C. Silva, M. Samaniego, and R. Deters, "IoT and Blockchain for Smart Locks," in *10th Annual Information Technology, Electronics and*

Mobile Communication Conference, Vancouver, Canada, 17-19 October 2019, pp. 262-269, doi: 10.1109/IEMCON.2019.8936140.
[13] K. Zhidanov *et al.*, "Blockchain Technology for Smartphones and Constrained IoT Devices: A Future Perspective and Implementation," in *IEEE 21st Conference on Business Informatics*, Moscow, Russia, 15-17 July 2019, pp. 20-27, doi: 10.1109/CBI.2019.10092.
[14] L. Hang and D.-H. Kim, "Design and Implementation of an Integrated IoT Blockchain Platform for Sensing Data Integrity," *Sensors*, vol. 19, no. 10, p. 2228, 2019, doi: 10.3390/s19102228.
[15] T. Okada. "Handle Smart Contract on Ethereum with Arduino or ESP32." Medium. https://medium.com/@takahirookada/handle-smart-contract-on-ethereum-with-arduino-or-esp32-1bb5cbaddbf4 (accessed 20 July, 2021).
[16] F. Lardinois. "Nodle Crowdsources IoT Connectivity." TechCrunch. https://techcrunch.com/2019/12/11/nodle-crowdsources-iot-connectivity/ (accessed 19 July, 2021).
[17] P. Lucsok. "IoT on Substrate: Nodle.io." Parity. https://www.parity.io/blog/iot-on-substrate-nodle-io/ (accessed 22 July, 2021).
[18] "Connecting & Securing The Next Trillion Things." Nodle. https://nodle.io/ (accessed 22 July, 2021).
[19] P. Pachal. "IoT Startup Helium Floats New Hardware Device for Mining Its HNT Crypto Tokens." CoinDesk. https://www.coindesk.com/helium-iot-hnt-cryptocurrency-miner (accessed 19 July, 2021).
[20] J. Constine. "Helium Launches $51M-funded 'LongFi' IoT Alternative to Cellular." TechCrunch. https://techcrunch.com/2019/06/12/helium-network/ (accessed 19 July, 2021).
[21] "Helium Explorer." Helium. https://explorer.helium.com/ (accessed 22 July, 2021).
[22] F. Rezaeibagha, Y. Mu, K. Huang, L. Zhang, and X. Huang, "Secure and Privacy-Preserved Data Collection for IoT Wireless Sensors," *IEEE Internet of Things Journal*, pp. 17669-17677, 2021, doi: 10.1109/JIOT.2021.3082150.
[23] Y. Lin, J. Li, S. Kimura, Y. Yang, Y. Ji, and Y. Cao, "Consortium Blockchain based Public Integrity Verification in Cloud Storage for IoT," *IEEE Internet of Things Journal*, pp. 3978-3987, 2021, doi: 10.1109/JIOT.2021.3102236.
[24] H. Wang and J. Zhang, "Blockchain Based Data Integrity Verification for Large-Scale IoT Data," *IEEE Access*, vol. 7, pp. 164996-165006, 2019, doi: 10.1109/ACCESS.2019.2952635.
[25] P. Wei, D. Wang, Y. Zhao, S. K. S. Tyagi, and N. Kumar, "Blockchain Data-based Cloud Data Integrity Protection Mechanism," *Future Generation Computer Systems*, vol. 102, pp. 902-911, 2020, doi: 10.1016/j.future.2019.09.028.
[26] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain Based Data Integrity Service Framework for IoT Data," in *IEEE International Conference on Web Services*, Honolulu, USA, 25-30 June 2017, pp. 468-475, doi: 10.1109/ICWS.2017.54.
[27] J. H. Khor, M. Sidorov, and P. Y. Woon, "Public Blockchains for Resource-constrained IoT Devices -A State of the Art Survey," *IEEE Internet of Things Journal*, pp. 11960-11982, 2021, doi: 10.1109/JIOT.2021.3069120.
[28] *ESP32-S2-WROVER/ESP32-S2-WROVER-I Datasheet*, Espressif, 2020. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s2-wrover_esp32-s2-wrover-i_datasheet_en.pdf
[29] J.-L. Aufranc. "ESP32-S2 Board Targets Battery-powered Applications With 30uA Deep Sleep Power Consumption." CNX-Software. https://www.cnx-software.com/2020/10/28/esp32-s2-board-targets-battery-powered-applications-with-30ua-deep-sleep-power-consumption/ (accessed 20 July, 2021).
[30] "Example: Using LwIP SNTP Module and Time Functions." Github. https://github.com/espressif/esp-idf/tree/master/examples/protocols/sntp (accessed 25 August, 2020).
[31] "Contract ABI Specification." Solidity. https://docs.soliditylang.org/en/v0.5.3/abi-spec.html (accessed 27 August, 2020).
[32] "Goerli Testnet Network." https://goerli.etherscan.io/address/0x62249719D71616a12Bc0E4742994542dc385D429 (accessed 20 July 2021).
[33] K. MacKay. "Micro-ECC." Github. https://github.com/kmackay/micro-ecc (accessed 8 July, 2021).
[34] "ESP-TLS." Espressif. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_tls.html (accessed 25 August, 2020).
[35] Kopanitsa. "Web3-Arduino." Github. https://github.com/kopanitsa/web3-arduino (accessed 24 August, 2020).
[36] "Flash Encryption." Espressif. https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/security/flash-encryption.html (accessed 24 August, 2020).
[37] "Mythx - Smart Contract Security Service for Ethereum." Consensys. https://mythx.io/ (accessed 31 May, 2021).
[38] "FIPS 180-4 Secure Hash Standard (SHS)," National Institute of Standards and Technology, 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
[39] Q. H. Dang, "Recommendation for Applications Using Approved Hash Algorithms," National Institute of Standards and Technology, 2012. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=911479
[40] B. Preneel, "Second Preimage Resistance," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia Eds. Boston, MA: Springer US, 2011, pp. 1093-1093.
[41] "Lithium Polymer Battery 1500mAh 1800mAh 2000mAh." LiPol Battery. https://www.lipolbattery.com/lithium%20polymer%20battery.html (accessed 4 August, 2021).

JING HUEY KHOR is an Assistant Professor at the University of Southampton Malaysia. She joined the university as a lecturer in 2014. She received her B.Eng degree in Electrical Engineering (Electronic) with First Class Honours from Universiti Malaysia Pahang in 2009. She then received her Ph.D degree at Universiti Sains Malaysia in 2013. She was a visiting researcher at Yonsei University under the International Scholar Exchange Fellowship (ISEF) Program of the Chey Institute for Advanced Studies in 2021. She has been actively designing privacy preserving protocols for communication between IoT devices and blockchain, new consensus algorithms, and decentralized application for IoT purposes. She has served as a technical committee member for several international conferences, and as a reviewer for IEEE and Elsevier journals.

MICHAIL SIDOROV is an ERCIM Postdoc at the Norwegian University of Science and Technology (NTNU), Norway working in the field of IoT. He has received his B.Sc degree in Informatics from Coventry University, UK in 2009, B.Sc Degree in Informatics Engineering from Klaipeda University, Lithuania in 2010, a joined M.Sc degree in Embedded Computing Systems (EMECS) from NTNU and University of Southampton (UoS), UK in 2013, and Ph.D in Computer Science and Engineering from Toyohashi University of Technology (TUT), Japan in 2020. He was a Teaching Fellow and Laboratory Officer for the period of 2013 – 2017 at the University of Southampton Malaysia, and worked as a Researcher after obtaining his PhD in TUT. His current research interests include blockchain integration with IoT.

MING TZE ONG is a Mathematics Lecturer in the Computer Science department at DigiPen--The One Academy. He received a B.Sc in Math and Computing from the National University of Singapore (NUS) and a M.Sc in Numerical Analysis from the University of Manchester. His interests mainly include High Performance Computing.

SHEN YIK CHUA is currently pursuing his MSc degree in Electrical and Electronic Engineering with the University of Southampton. His current interests include energy harvesting, blockchain technology in IoT, robotics, and automation.