IoT Data Storage on a Blockchain Using Smart Contracts and IPFS

João Paulo de Brito Gonçalves
Federal Institute of Espirito Santo (Ifes)
Cachoeiro de Itapemirim - ES, Brazil
jpaulo@ifes.edu.br

Gabriel Spelta, Rodolfo da Silva Villaça, Roberta Lima Gomes
Federal University of Espirito Santo (Ufes)
Vitoria - ES, Brazil
gabriel.spelta@edu.ufes.br, rodolfo.villaca@ufes.br, rgomes@inf.ufes.br

2022 IEEE International Conference on Blockchain (Blockchain) | 978-1-6654-6104-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/BLOCKCHAIN55522.2022.00078

***Abstract*—Since the creation of the cryptocurrency *Bitcoin*, the interest in blockchain technology has increased, entering areas such as IoT (*Internet of Things*) and data sharing. The main objective of this paper is to develop a system that allows the storage of data from IoT services in a decentralized network with a blockchain managing transactions through a smart contract. The project was carried out using the blockchain *Ethereum*, IPFS (*InterPlanetary File System*) for storage, *Solidity* language for contract development, NodeJS for coding the simulation of IoT devices, *web* interface and *back-end* of the solution. The MQTT protocol was used to transport data from the devices. Our main objective was achieved, as the tests carried out show use cases in which this solution has an advantage over the direct storage in the *Ethereum* blockchain.**

***Index Terms*—Blockchain, Internet of Things, Smart Contracts, IPFS.**

I. Introduction

Blockchain has as its great differential the data structure that is fully distributed and replicated in a decentralized way for all members of its network. Along with a strong consensus mechanism, blockchain ensures security, integrity and immutability of data in its ecosystem. This structure is formed by blocks that store transactions between two parties directly and without the need for an intermediary. Combining these benefits, the platform *Ethereum* [1] was created, a technology which also has its own cryptocurrency, which brought a new generation of blockchains in which scripts are stored and executed in an automated way. These codes were called smart contracts and made it possible to store other types of data, not just only those linked to cryptocurrency transactions. This has made the expansion of the use of *dApps*, decentralized applications that have their data and records stored on a public blockchain, to gain momentum.

With the increase in demand for data storage, especially when we talk about Internet of Things (IoT), in a secure, integral and available way, blockchain has proved to be a true candidate for this issue. However, due to the fact that replication of the entire chain is necessary to ensure decentralization, it is evident that there is a high cost to allocate data in the network. Therefore, the search for an ally that would guarantee all the requirements and keep costs within a feasible scenario became inevitable. Therefore, for this paper, the protocol *InterPlanetary File System* (IPFS) [2] was chosen to work together with the smart contracts in *Ethereum* smart contracts in order to serve as a storage location for IoT devices.

In this context, the goal of this paper is to design and implement a system capable of storing data in encrypted form from simulated IoT devices, using a combination of blockchain *Ethereum* and IPFS. In addition, a dApp was developed to allow users to interact with the smart contract, ensuring access controls and data visualization.

This paper is divided as follows: in Section 2, we present the works in the literature related to our proposal. Section 3 presents the development of the system initially proposed, as well as its architecture and execution flow. In Section 4 the performed tests are presented and in the last section we present the final considerations of the work, including some future work proposals.

II. Related Work

Sargilar et. al. [3] propose Hybrid-IoT, a hybrid blockchain architecture for IoT. In Hybrid-IoT, subgroups of IoT devices form separated blockchains, referred to as sub-blockchains. Sharma et al. [4] propose a novel blockchain-based distributed cloud architecture with a software defined networking (SDN) to enable controller fog nodes at the edge of the network to meet the required design principles.

Dorri et. al. [5] propose a lightweight blockchain architecture for IoT that tries to eliminate the overheads of classic blockchain, while maintaining most of its security and privacy benefits. IoT devices benefit from a private immutable ledger, that acts similar to blockchain but is managed centrally, to optimize energy consumption.

Helium [6] is a decentralized wireless network aimed at extending the Internet coverage for devices. Miners earn cryptocurrency tokens by providing wireless network coverage to devices through specialized routing hardware named Hotspot. Encrypted device data collected by the miners can then be sold to Internet applications called Routers. As a device can use several miners for coverage, a Router application can purchase data from several miners in order to fetch a device location [7].

Another blockchain platform designed for the IoT is IoTex [8]. IoTex presents DioTA [9], the decentralized ledger-based framework for data authenticity protection in IoT systems

978-1-6654-6104-7/22/$31.00 ©2022 IEEE
DOI 10.1109/Blockchain55522.2022.00078

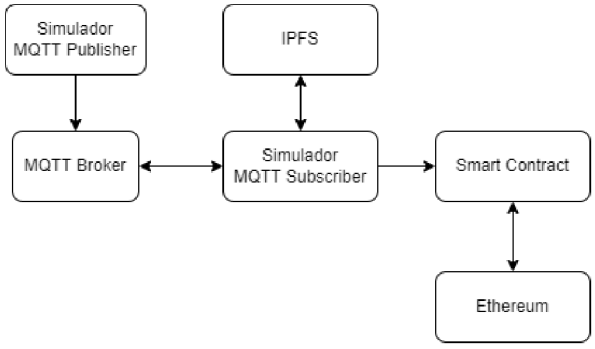

Fig. 1. Solution Back-end Diagram showing the interactions between the entities

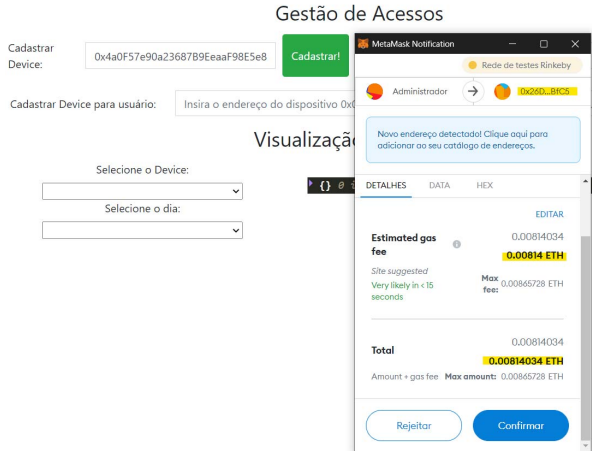

Fig. 2. Web Interface showing the Access Control and Visualization fields

which is a way to maintain data integrity, including identity related data for IoT systems. The system comprises a collection of decentralized ledgers, supports smart contracts, and can be implemented in permissioned or permissionless blockchains.

III. Design and Implementation

A. MQTT Protocol

The Message Queue Telemetry Transport (MQTT) [10] is a protocol used for communication within an IoT environment that functions on top of the Transport Control Protocol (TCP). The protocol was created by IBM as a machine-to-machine, lightweight communication method. The protocol functions on a server-client system where the server, called a broker, pushes updates to MQTT clients. The clients won't send messages directly to each other, instead relying on the *broker* for this. MQTT is an open protocol of the *Publisher/Subscriber* type. In this mode, the *Publisher* device publishes pre-established information, while the *Subscriber* device consumes this information.

Every MQTT message contains a topic, organized in a tree-like structure, to which the clients can subscribe or publish. The broker receives published messages from clients that contain a certain value or command and relays the information to every client that has subscribed to that specific topic.

B. MQTT Application

The system proposal aims to store data on a blockchain network with integration with the IPFS protocol. Separating the developed system into two modules, we have the first module described in Figure 1 representing only the back-end part of the solution, in which we do not have user interaction. For the other module, which will be explained later, we will have a front-end with a web interface for interactions with the users involved in the solution.

To facilitate the configurations, the public broker of EMQX was used [11]. The choice for this model was due to its ease of configuration and well-defined documentation and to simulate an IoT device, the MQTT protocol was used to send messages. The first function of the MQTT simulator module is to connect to the *Broker* to consume the messages sent by the *Publisher*.

Another function is to accumulate these messages and, at each parameterized time cycle, submit this data to the IPFS in an encrypted form and insert the *hash* into the blockchain through the smart contract. Queuing of transactions occurs on *Ethereum* due to the use of the PoW consensus mechanism. The average time of validation of transactions on the network can vary between 10 and 15 seconds. During the job development stage, the shortest successfully tested cycle time was 1 minute.

C. Back-End

Also, with the purpose of simplifying the configurations to be made, the Infura API [12] was used for all the interaction with the IPFS.

The blockchain used, *Ethereum*, allows the creation of smart contracts through the use of the *Solidity* [13] programming language. The entire contract was developed in this language and the coding was compiled using the *Remix* IDE. *Remix* is used in the process of developing a contract on the *Ethereum* network, having modules capable of performing tests, code analysis, contract implementation, among other features [14].

In *Ethereum*, in addition to the main network, there are four test networks: *Ropsten*, *Kovan*, *Rinkeby* and *Goerli*. For this work, the *Rinkeby* test network was chosen. Using this network all transactions on the blockchain and interaction with the contract are executed. Among the functionalities implemented in the smart contract it is worth mentioning:

- **`modifier adminsOnly` and `registeredDevicesOnly`**: this feature of the *Solidity* language allows applying conditions to functions, for example: for `adminsOnly`, every function that receives this `modifier` will perform validation if the user who performed the function is an administrator. If not, the function is not executed.
- **`registerDevice`**: to be executed exclusively by users with administrator profile. Performs the creation of a

File Size (Kb)	Average Insertion Time (ms)	Insertion Standard Deviation (ms)	Average Recovery Time (ms)	Recovery Standart Deviation (ms)
3	935	100	1201	340
5.9	895	40	1039	25
11.7	749	328	1550	226

Fig. 3. Results obtained for data insertion/retrieval in IPFS

Method	Average Validation Time (ms)	Validation Standard Deviation (ms)	Gas Used	Gas Price (Gwei)
SetDevicetoUSer	16045	452	51800	5,144
addHashIPFS	14205	8711	144126	34,102
registerDevice	10152	3275	73894	24,244

Fig. 4. Results obtained in the analyzed methods of the contract.

device and grants permission to enter data into the contract.

- **`unregisterDevice`**: function exclusive for use by users with administrator profile. Performs the removal of the data entry device's permission in the contract.
- **`addHashIPFS`**: function exclusive for use by enabled devices. Performs the insertion in the contract of the *hash* coming from the insertion of the file in of a file stored in IPFS.
- **`getDeviceHashMap`**: collects the *hash* associated to a given device and date.
- **`setDeviceToUser`**: grants a user permission to access data from a specified device.

D. Front-end

There are two types of actors that will interact with the dApp: administrator and standard user. The standard user will be able to query the data stored by the devices on which he is enabled. The administrator has all the permissions that the standard user has plus the ability to enroll and remove devices from the system.

The *dApp* was developed using the *framework* ReactJS [15]. It has the following functionalities: device registration and removal, device selection, date selection of the data to be consumed and, finally, data content visualization. The integration between the interface and the smart contract is done through *MetaMask*. *Metamask* is both a digital cryptocurrency wallet and a gateway to blockchain-based applications [16].

It can be seen that the system has three crucial parts: device simulation using MQTT, smart contract implemented on the *Ethereum* blockchain and the *web* interface that allows interaction with the user. This interface can be seen in Figure 2.

IV. Deployment and Evaluation

Every transaction needs a certain amount of *Gas* to be processed. According to [17], the amount of *Gas* to be computed will be relating to operations performed on the blockchain.

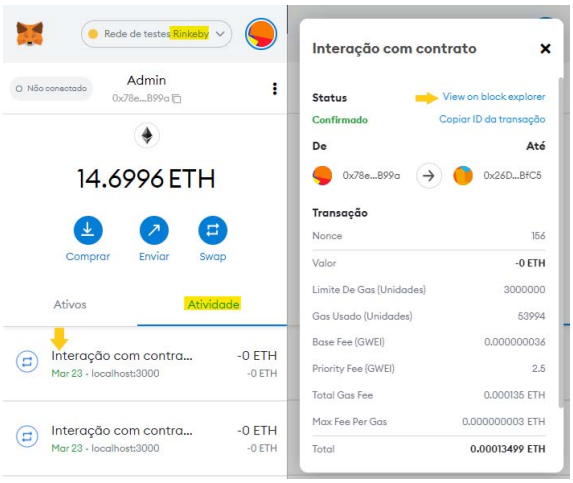

Fig. 5. Details of activities performed by *MetaMask* plugin

Besides, in transactions we have some other important variables: *GasLimit* and *GasPrice*. The *GasLimit* is the maximum value that a transaction can consume and it changes for each set of operations. If an amount greater than this limit is sent, the excess returns to the sender of the transaction. The *GasPrice* is the cost, calculated in Gwei, of each *Gas* at the time the transaction is being executed. This value can constantly vary and the higher the *GasPrice* configured, the bigger the chances of your transaction being validated by a node on the network.

A. Use Case

For our use case, we will show the device registration process. Using the user account with administrator permission, the device registration process whose hash is `0xd096...442E407d` was carried out. After confirming the transaction on *MetaMask*, it is awaited its validation, which is then evidenced by a notification from *MetaMask*. For this example, we waited about 24 seconds for the transaction to commit.

Using the *MetaMask* interface, you can check the transaction details at the `Activities` menu, as we can see in in Figure 5 that shows details about the transaction as:

- ***Transaction Hash***: *hash* transaction ID.
- ***Timestamp***: when the transaction was performed.
- ***From***: who sent the transaction.
- ***To***: to whom it was sent.
- ***Value***: transferred value.
- ***gas Used***: gas value used in the transaction.
- ***Transaction Fee***: transaction fee cost.
- ***Gas Price***: price, in *Gwei*, of *Gas* for this transaction.

B. Inserting/Retrieving IPFS Data

IoT data files with different sizes were generated and the time to get the return of the file reference *hashes* after recorded in IPFS were measured. Figure 3 shows the results obtained. We calculated the costs involved to perform three types of contract methods (previously presented) and also the time spent for the transactions to be confirmed by the *Ethereum* network. The results are shown in Figure 4.

Figure 3 shows an average time for file retrieval in IPFS of about 1201 ms for a 3 Kb size file and of 1550 ms for a 11,7 kb file. It is worth to mention that the times for operations inside the blockchain are much bigger and operations inside the blockchain take longer, with the **`addHashIPFS`** operation during 14205 ms for a 256 bits writing while the cost in gas was 144126 units. According to [17] we can see that the cost of a function of type `Gsset` is equivalent to about 20000 *Gas* units. This function is based on the `SSTORE` operation that performs the storage of 256 bits (1 *word* in EVM).

We can consider that the proposed project was successful in solving the problem of finding an option to store files inside the blockchain, because the option of storing the files in IPFS proved to be a faster and cheaper option. It is important to note that this time is not feasible for cases where the need for real-time information is crucial, because this project, in the way it was designed, is not viable for the implementation of an industrial monitoring center, in which a few seconds are essential for decision-making based on sensor information.

V. Conclusions and Future Work

In this paper it was presented a decentralized file storage to store data from IoT devices. The implementation was carried out by simulating IoT devices and generating dummy data to be stored in the IPFS network and, to store the data references we took advantage of smart contracts of the *Ethereum* network. The test stages have verified that the proposed solution, compared to direct storage in the blockchain of *Ethereum*, is fully viable and that the combination of technologies can be a great asset to the need for data consumption of current society. However, despite this positive result, the work, due to the reduced time to perform, has points to be improved in future works.

For future work, some evolutions are suggested:

- Improvement of MQTT modules to make them less susceptible to failure, creating a way to persist data out of program memory until it is stored in IPFS;
- Use of a private blockchain to avoid the costs of transaction fees existing in the public one;
- Creation of an IPFS cluster to bypass the public network and maintain greater control over data.

Acknowledgements

This research was financed in part by the following Brazilian research agencies: FAPES, FAPESP/MCTIC/CGI.br (#2020/05182-3) and IFES.

References

[1] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[2] J. Benet, "Ipfs-content addressed, versioned, p2p file system (draft 3)," *arXiv preprint arXiv:1407.3561*, 2014.

[3] G. Sagirlar, B. Carminati, E. Ferrari, J. D. Sheehan, and E. Ragnoli, "Hybrid-iot: Hybrid blockchain architecture for internet of things-pow sub-blockchains," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1007–1016.

[4] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for iot," *Ieee Access*, vol. 6, pp. 115–124, 2017.

[5] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for iot," in *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2017, pp. 173–178.

[6] A. Haleem, A. Allen, A. Thompson, M. Nijdam, and R. Garg, "A decentralized wireless network," 2018.

[7] E. Vieira, J. Ferreira, and P. C. Bartolomeu, "Blockchain technologies for iot applications: Use-cases and limitations," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1560–1567.

[8] X. Fan and Q. Chai, "Roll-dpos: a randomized delegated proof of stake scheme for scalable blockchain-based internet of things systems," in *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2018, pp. 482–484.

[9] L. Xu, L. Chen, Z. Gao, X. Fan, T. Suh, and W. Shi, "Diota: Decentralized-ledger-based framework for data authenticity protection in iot systems," *IEEE Network*, vol. 34, no. 1, pp. 38–46, 2020.

[10] D. Dinculeană and X. Cheng, "Vulnerabilities and limitations of mqtt protocol used between iot devices," *Applied Sciences*, vol. 9, no. 5, p. 848, 2019.

[11] E. Technologies, "Emqx," 2022. [Online]. Available: https://www.emqx.com/en/mqtt/public-mqtt5-broker

[12] I. Inc, "Infura webpage," 2022. [Online]. Available: https://infura.io/

[13] Ethereum, "Solidity," 2021. [Online]. Available: https://docs.soliditylang.org/en/v0.8.13/

[14] Remix, "Remix - ethereum ide," 2021. [Online]. Available: https://remix-ide.readthedocs.io/en/latest/

[15] M. Plataforms, "Reactjs webpage," 2022. [Online]. Available: https://infura.io/

[16] ConsenSys, "Metamask docs," 2021. [Online]. Available: https://docs.metamask.io/guide/#why-metamask

[17] G. Wood *et al.*, "Ethereum: a secure decentralised generalised transaction ledger (2014)," 2017.