Universität Zürich[UZH]

CAS Blockchain

Blockchain-based Keyless Locker System

Jesús Fernández-Benito

ESGroup AG.

December 8, 2021

ABSTRACT

The number of projects integrating Internet-of-Things (IoT) systems with the blockchain technology is increasing rapidly due to its benefits, such as data immutability and transparency. In this sense, a blockchain-based solution could improve the interoperability on the growing IoT locker market. However, the choice of a decentralized system based on blockchain technology is not always justifiable as it often implies a higher cost, more complexity and lower throughput, compared to a centralized solution. In this report, a blockchain-based solution for keyless IoT lockers is critically analyzed and an architectural design is proposed. Following the design, a prototype for such a locker system is presented, analyzing in depth the Decentralized Application (DApp) proposal and the Smart Contract (SC) design. This report concludes by providing an outlook for further research and improvements, as the measurements obtained during simulations advice against the usage of the prototype in its current state, given its latencies and transaction costs.

1 INTRODUCTION

Lockers in public settings are already used in everyday situations, for instance, in a gym for storing clothes during trainings or in a train station to store luggage during a sightseeing tour. In private settings, the use of lockers is increasing as companies are adopting a hybrid working model [1] and, therefore, moving towards hybrid workspaces [2], where employees share desks and alternate the days that they are working on-site. Hence, requiring the storing of personal items in a safe environment to avoid unauthorized use.

Currently, there are multiple locking options available, starting on basic ones, where the user has to bring his/her private padlock, or the coin-based locking system (where a coin is locked until the lock is released), to the so-called 'smart' locks [3] that can be opened with a badge or controlled from a smartphone.

As the technology gets more affordable, driven by *(a)* the rapid growth of the Internet of Things (IoT) market and *(b)* the infrastructure of smartphones being already available [4]; the demand for smart locks increases, expecting to have a Compound Annual Growth Rate (CAGR) of 21.4% from 2021 to 2028 [5].

However, the smart locks-based solutions present also two main disadvantages: *(a)* these solutions are proprietary systems, which present vendor lock-in problems [6], such as, difficult integration with different systems or maintenance tasks that can be completed only by highly trained technicians [7]; *(b)* security concerns, as shown in the vulnerabilities of some models [8].

Therefore, to address the vendor lock-in and security issues, the proposal on this document is to create a generic decentralized management system for lockers by relying on a blockchain [9] and using Smart Contracts (SC) [10]. Further, due to the decentralization and open aspect of the blockchain network, this management system could be used by any existing or new manufacturer, reducing the investment that these companies would need to do developing new or maintaining proprietary systems. Additionally, in the case of pay-per-use lockers, the blockchain infrastructure could ease as well the usage of this management system internationally, as it would not depend on a given fiat currency (e.g., Swiss Franc).

Moreover, for the buyers of the lockers, it would be easy to mix and match lockers from different manufactures and for third-party companies, it would enable them to develop applications that could interact with different models. Finally, for the users of the lockers, it would introduce interesting use cases such as the exchange of parcels without using the postal service. For example, a user could deposit an item in a locker, closing it using his/her blockchain address and finally transferring the 'key' of the locker to another person's blockchain address; therefore, enabling that person to open the locker and collect the item.

The rest of this document is structured as follows. Section 2 details the reasons why a blockchain is required within this locker system context. Then, Section 3 outlines the overall solution architecture, leading to a detailed analysis of the SC in Section 4 and of the user interface in Section 5. Further, Section 6 discusses the proposed solution and Section 7 concludes the report with an outlook to future work.

2 DO WE NEED A BLOCKCHAIN?

A blockchain is a decentralized, distributed, and, by design, public, digital ledger consisting of records (called blocks) that are used to record transactions across a network of computers. Such blocks are linked using hash pointers based on the information from the previous block. Hence, any block stored in the blockchain cannot be altered retroactively, without the alteration of all subsequent blocks [9].

This type of architecture provides high availability, auditability, transparency, and neutrality, it also reduces or eliminates censorship and reduces certain counterparty risks [11]. However, at the same time, comparing it to centralized solution, a system built on the blockchain is more complex to build, requires more time to process transactions, and requires more resources, such as processing, electricity, and data transfer [12]. Thus, a careful assessment whether a blockchain is necessary for a given use case must be conducted before the investment in such a paradigm.

Besides the increasing attention of blockchain technology in the corporate world and the personal interest of the author of this document to increase his knowledge in the topic, is there a justifiable reason to choose blockchain as an architecture of this particular project (i.e., a blockchain-based locker system)?

2.1 Decision Models

To help analyze whether integrating a blockchain is an appropriate technical solution, three decision models are proposed in the literature [13]:

- **IBM Model**: A blockchain model that highlights how the key features of blockchain fit into business processes and applications.
- **Birch-Brown-Parulava Model**: A model helping a user select between permissioned and permission-less implementations of a distributed ledger. It was designed by developers at Consult Hyperion.
- **Wüst-Gervais Model**: A simplified model combining the best from both the previous models. Proposed by two researchers from the Swiss Federal Institute of Technology (ETH) in Zurich.

These models share important design principles to consider before choosing a system architecture [13]:

- **Database**: Blockchains use a shared database that is visible to all the participants.
- **Writers**: Blockchain is designed to have multiple writers; i.e., multiple entities that are generating and verifying the transactions that are published to the ledger.
- **Trust**: The default setting for a blockchain is no trust between parties writing to the ledger, but some of the blockchain constructs managing trust can be safely removed.
- **Disintermediation**: Blockchains remove a centralized authority from the network, and the transactions are therefore considered decentralized. These transactions are independently verified and processed by every node on the network.
- **Transactional interdependence**: A blockchain is best for handling transactions that build on each other as they are published to the blockchain by writers and never deleted.

2.2 Wüst-Gervais System Analysis

After a research on the models described in the previous section, the Wüst-Gervais model [14] was chosen to analyze whether it would be an advantage using a blockchain as an underlying infrastructure for the locker system.

Hence, this report agrees with was is stated in [13], which is that this model could be seen as a simplified combination of the other two (i.e., IBM and Birch-Brown-Parulava).

To ease the decision making process, the Wüst-Gervais model [14] provides a flow chart depicted in **Figure 1** with questions that project stakeholders should answer and, depending on the answers, a different technical architecture in terms of blockchain type is proposed.

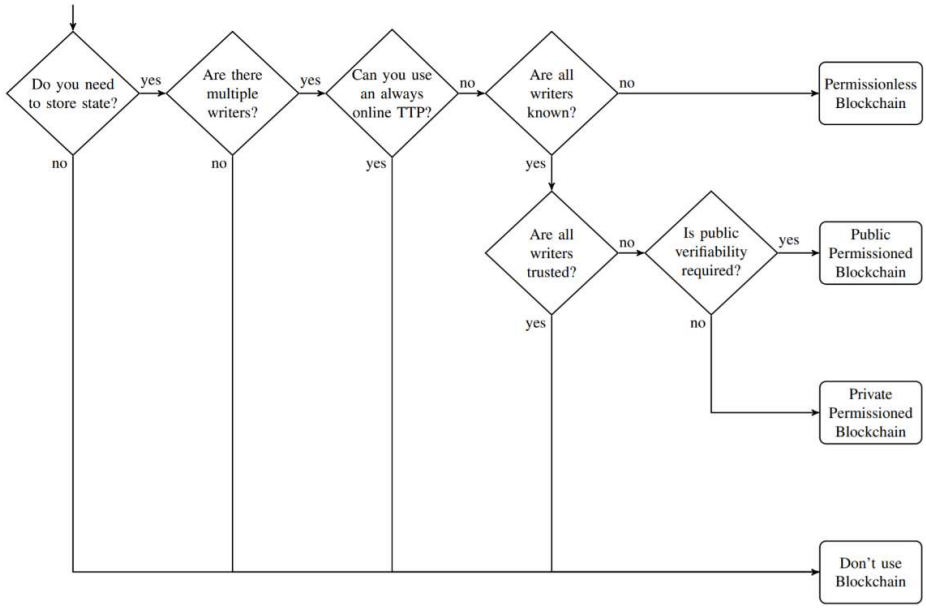

Figure 1: Wüst-Gervais flow chart [14].

Thus, such questions were answered within the scope of the proposed project as follows:

1. Do we need to store state?

Yes, each of the lockers in the system needs to have a state (i.e., open/closed). One possibility could be that this state is stored in the locker itself, which could work in a simple and isolated scenario. In contrast, maintaining the list of locker states in an independent storage, it would reduce the complexity of the locker itself, providing flexibility to the system and allowing developers to build complex use cases (e.g., statistics, payment methods, and transfer of ownership).

2. Are there multiple writers?

Yes, depending on how the concept of writer is described, one could consider that every single user should be seen as a writer as he/she triggers the locker state to change. However, this is most probably not the reality because a normal user would rather interact with a simple intermediary system that would in fact write that change on his/her behalf. In this sense, there could be multiple intermediaries that would offer different interfaces to interact with the locker store (e.g., locker manufacturers, sport centers or other software companies).

3. Can you use an always online Third Trusted Party (TTP)?

No, as different parties or, at least, different manufacturers would be involved, there is no good reason to believe that all of them would agree on one of them owning all the information.

4. Are all writers known?

No, by not having a TTP in charge of the data ownership or the decision making, anyone could join the project at any moment without needing to be admitted.

2.3 Decision Model Conclusion

The result obtained by following the flow chart it is that the project qualifies for considering a '*Permissionless Blockchain*'. After this analysis, other considerations should be taken before starting the development of such a system. For instance, the literature [14] suggest to always take into account factors such as throughput and latency, combining the results of flow chart with the values shown in **Table 1**:

	Permissionless Blockchain	Permissioned Blockchain	Central Database
Throughput	Low	High	Very High
Latency	Slow	Medium	Fast
Number of readers	High	High	High
Number of writers	High	Low	High
Number of untrusted writers	High	Low	0
Consensus mechanism	Mainly PoW, some PoS	BFT protocols (e.g. PBFT [5])	None
Centrally managed	No	Yes	Yes

Table 1: Wüst-Gervais comparison table [14].

These aspects will be further addressed in Section 6, where the feasibility of the project is reviewed. However, for the prototype, it is assumed that the throughput, latency and transactions cost are in an acceptable range to consider the development of this project within the project timeframe.

Selecting the most suitable permissionless blockchain for a project could lead to a whole new research topic on its own [15] and, therefore, the selection criteria was reduced to the capability of supporting complex SCs (e.g., Turing-complete SC).

The selected blockchain was Ethereum [16] being the most popular solution for running SCs [10] and the most market capitalized (based on [17]). Furthermore, around Ethereum a large ecosystem of development and testing tools is freely available due to its engaged community.

3 SYSTEM ARCHITECTURE

In a very high level view (see **Figure 2**), the overall blockchain-based locker system can be reduced to its two participants (i.e., the users and the lockers) communicating with a locker manager. On the core of the Locker Manager there is the SC, as it was decided in the previous section. The focus of this section is to understand the components needed in between the participants and the SC to communicate.

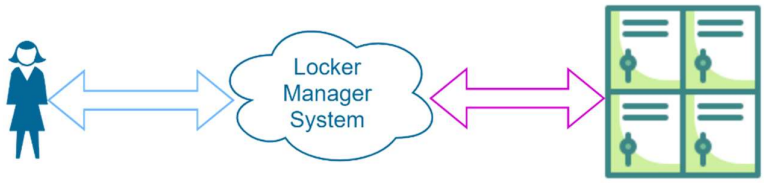

Figure 2: High level view of the solution.

3.1 SC Direct Communication

One design possibility would be to choose to communicate directly with the SC. However, there are some restrictions to consider regarding SCs:

- A SC only executes if it is called by a transaction [11].
- There is no return value in a transaction, instead, when a transaction completes (successfully or not), it produces a transaction receipt containing logs that can be observed (i.e., events), which provide information about the actions occurred [11].

To better understand how the direct communication with the SC occurs, **Figure 3** depicts the simplified flow of a transaction that triggers a SC execution (i.e., calls a function).

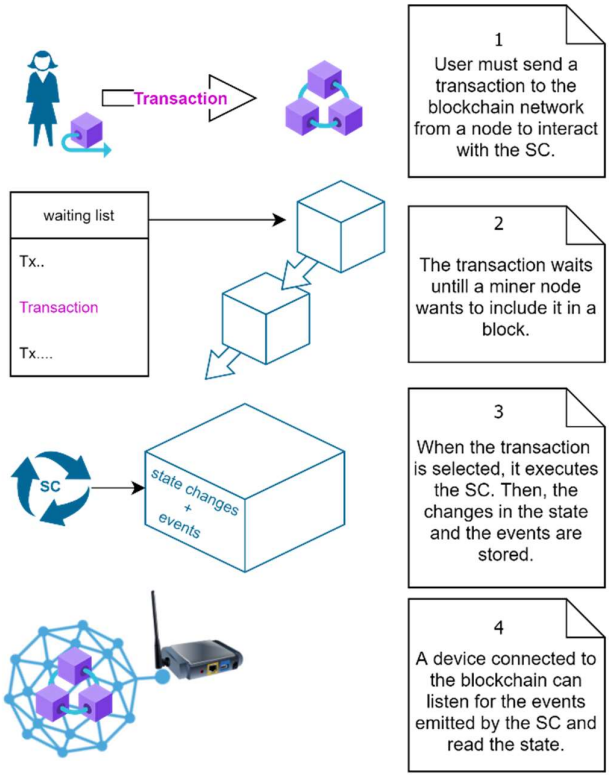

Figure 3: User triggering a SC execution.

In this scenario, a user that would want to interact with the locker system, would need to have strong knowledge of the blockchain, to know the interface of the locker manager SC and to run a node of the blockchain network. On the other side, each locker would need to be able to run a node as well. Thus, it was decided that the direct communication with the SC is not suited for this project.

3.2 User Interface

For the communication with the users, it is proposed that the interaction is handled through a Web page as interface. A website can be integrated with a user wallet and handle transactions, hiding the details of the interaction with the blockchain, relying on multiple available libraries (e.g., web3js [18]). This type of application (i.e., user interface connected to a SC) is called a decentralized application or "DApp" [19].

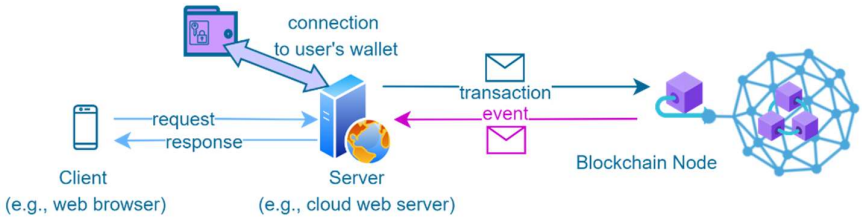

Figure 4: DApp example architecture.

A DApp does not perfectly fit the common architecture styles found in the literature [20], however, as seen in **Figure 4**, it could be considered as a distributed architecture with different layers:

- The presentation layer is the interface with the user (e.g., browser rendering a web page hosted in a server).
- The business layer is the logic of the application. In this case, the SC running in a blockchain node.
- The data layer is the blockchain.

Between the user and the presentation layer, the communication is synchronous, following the client-server architecture and giving the user direct feedback on actions (e.g., transaction successfully transmitted). However, the communication with the business layer is asynchronous (e.g., a transaction is sent, but it is not certain when or if it is going to be processed) following a prioritized pipeline architecture on one direction (i.e., the transaction execution) and an event driven architecture on the other (i.e., triggering actions on an event emitted by an executed transaction).

3.3 Locker Interface

The lockers on the blockchain-based system can be considered an IoT device [21] as they need to communicate through the internet to get the changes in their state in order to open or close their locks.

The literature (e.g., [22] [23]) shows that it is a challenge to connect an IoT device directly to the blockchain due to the constraints on storage, processing resources and power consumption that these devices present. Therefore, the proposal is to use an intermediary blockchain node that listen for specific events and then to create and to send instructions to the lockers, as shown in **Figure 5**.

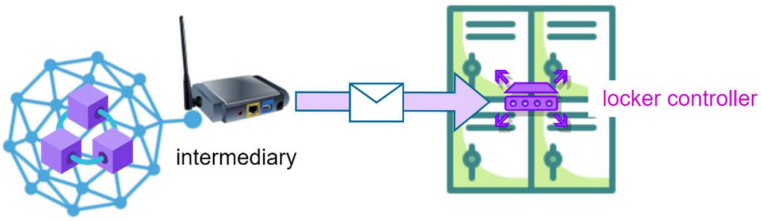

Figure 5: Locker communication.

The lockers do not need to communicate back their state to the blockchain, avoiding the heavy computation needed for signing of transactions and the extra resources that the IoT device in the locker would need. Thus, although different architectures could be researched on, having different levels of complexity in the lockers, it was decided upon the simplest locker possible to reduce the cost of the overall system. On such a system, multiple lockers could be connected to a simple controller that would manage the locks following instructions received from a remote device.

The locker controller could be designed as a board with a simple controller, different connectors for the lockers and an antenna with a SIM card for the internet connectivity. An example of such a board is the prototype produced by ESGroup [24] as seen in **Figure 6**.

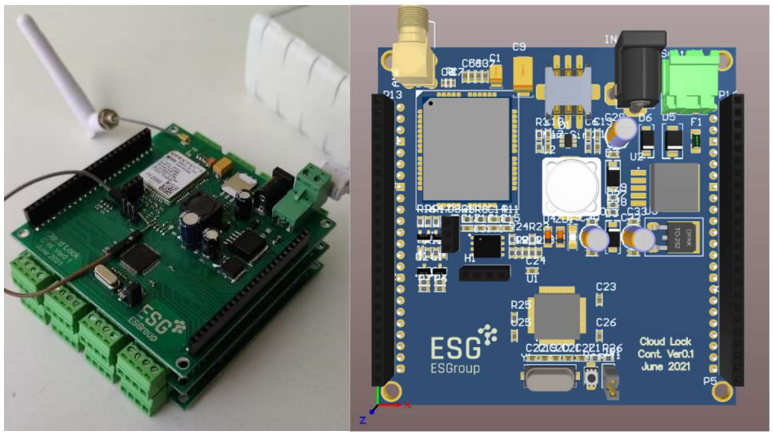

Figure 6: ESGroup Cloud Lock.

For the intermediary node, a small computer such as a Raspberry Pi [25] would be enough for running an Ethereum client [26]. From the multiple client software implementation [27] available, a light client is suggested because of the advantages it presents regarding storage needed and power consumption.

A light client or light node is a lightweight node that connects to full nodes to interact with the blockchain. Unlike their full node counterparts, light nodes do not need to keep all the information of the blockchain and, therefore, the amount of resources and storage needed is several orders of magnitude lower [28]. Thus, a small computer such as a Raspberry Pi [25] would be enough for running the light client [26].

To communicate the light client and the locker controller, a messaging protocol such as MQTT [29] could be used as it is becoming the *de facto* standard for IoT [30].

3.4 Overall Flow

The entry point to the locker system is to scan a QR code [31] that leads the user to the web interface of the locker system DApp. Using a QR code is faster and less error prone than manually typing the needed information in a web browser. Moreover, smartphone users are already familiar with QR codes (~ 87% already used it [32]) and more than 60% use them on daily basis [33].

Finally, to better understand the architecture of the whole system and all its connections, **Figure 7** shows the end-to-end flow, following the "close locker" use case and, step by step, it shows the participating elements. In this sense, it starts with a user searching for an empty locker and finishes with the user leaving, after successfully closing his/her belongings in the locker. For this simplified example, no payment is made for renting the locker and no cost is assumed for the blockchain transaction. However, this extra simplicity assumed does not have an impact on the architecture of the system itself, only on the interaction with the SC, which is not the focus of this section.

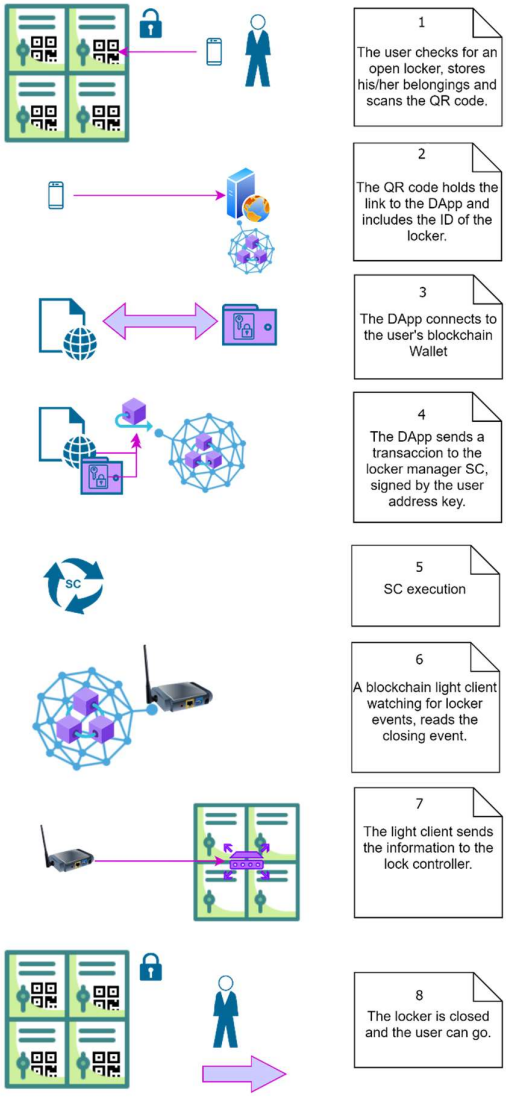

Figure 7: Simplified example for the basic closing use case.

4 THE SMART CONTRACT

This section describes the design and development of the SC in the center of the blockchain-based locker system. In this sense, it shows the design of the functionality of the SC in a progressive way, starting with the core functions and progressively adding complexity. Finally, it describes the decisions taken for the development of the SC prototype.

4.1 Core Functionality

A system designed to manage lockers (see **Figure 8**) needs to offer, at least, two functions: Open and Close.

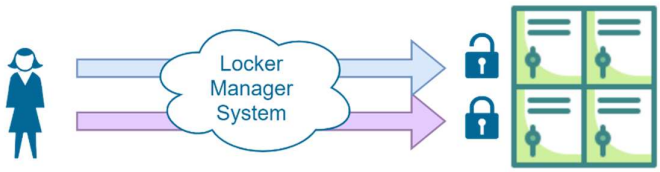

Figure 8: Core functionality of the locker manager system.

To be able to process these functions (i.e., open/close) reliably, the system would need to identify the locker in a unique way and to be aware of its current state (i.e., opened/closed). Further, in order to ensure that only the user that closed a particular locker is able to open it again, the user must be also uniquely identifiable and be stored together with the locker state.

In this sense, the user can be identified by his/her blockchain address, which is unique by the blockchain design [13] (i.e., using public and private key cryptography) and is included by default in each of the blockchain transactions. However, for the locking system, there is not such a clear identifier available. Therefore, the Universally Unique IDentifier (UUID [34]) was selected as the best option. A UUID can be written as a single sequence of characters (i.e., a string) and needs to be sent explicitly on each interaction with the SC.

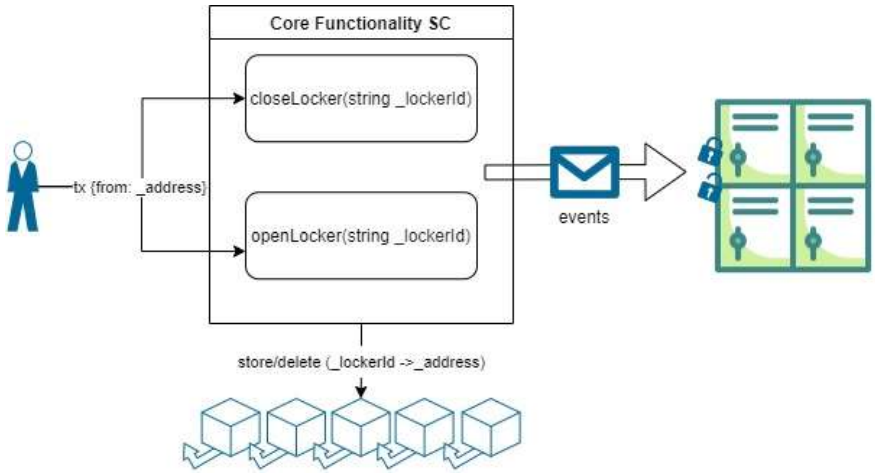

Figure 9: Core Functionality SC design.

The design of the SC (see **Figure 9**) is based on two main principles:

- **On-chain data storage:** ensuring immutability and transparency. The data storage is designed as a one-to-one map linking each *lockerId* with the user address that currently locks it. By storing the closed lockers, the system improves its storage efficiency and the logic of the SC is simplified. Further, as the map of closed *lockerId-to-address* is stored in the blockchain using the SC, it is ensured that only this SC is able to update it.
- **Events**: to communicate with off-chain applications. Events are a useful mechanism for light clients and DApps to "watch" for specific transaction-triggered events in the blockchain and react on it [11].

Further, the following security checks are performed in the SC on user requests:

- A user can only request a locker to close if it is currently free.
- A closed locker can only be opened by the user address that closed it.

Thus, avoiding unwanted situations where the users' belongings might be compromised.

4.2 Improved Core Functionality

Before adding further complexity (i.e., enabling other use cases) to the solution, changes were implemented to the existing Core Functionality SC in order to *(a)* fix issues that arose during the development of the prototype and *(b)* to improve its design to increase modularity (i.e., dividing functionality in modules that can be used to build bigger pieces). In this sense, the Core Functionality SC could be deployed and used by a locker system offering only basic functionality and, at the same time, it could be used as the foundations to build a more complex system.

As shown in **Figure 10**, only minimal changes were implemented in the functions exposed by the SC, where only the *lockerId* type was switch to a fixed length byte representation of the UUID to solve issues encountered with string manipulations in the blockchain.

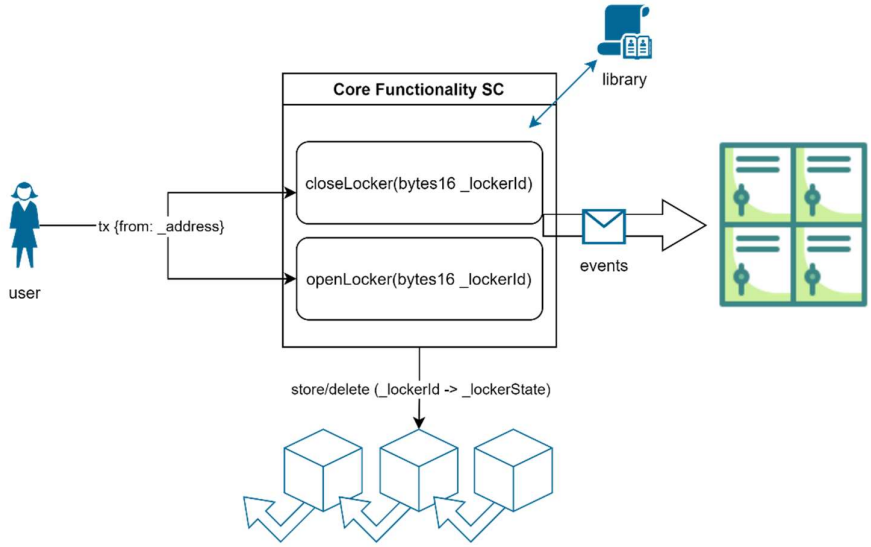

Figure 10: improved core functionality.

To improve modularity, the data storage design was improved to be able to store more information regarding the locker state. Thus, instead of storing a simple link between a *lockerId* and the address allowed to open it, the SC was enhanced to store a link between a *lockerId* an object containing its state. Further, to be able to add attributes in the future to the *lockerState* object, without altering the Core Functionality SC, the object definition, together with other reusable pieces, were extracted to a library.

Finally, the default payable function was disabled to avoid any funds to be transferred to the Core Functionality SC. Otherwise, the funds would be stuck in the SC balance forever, as there is no withdraw function implemented.

4.3 Payable Functionality

Relying on the modularity described in the previous section, a new SC was designed, as seen in **Figure 11**, where the Core Functionality SC is used as fundamentals and new features are added to it. In this sense, the withdrawal function was designed and, to avoid misuse, only a designated user is allowed to call it. In order to achieve access control, the *ownable* pattern [35] is used, where the address that deploys the SC is considered the owner and has special privileges.

Finally, with access control in place and a withdraw implemented, the default receive function was unlocked.

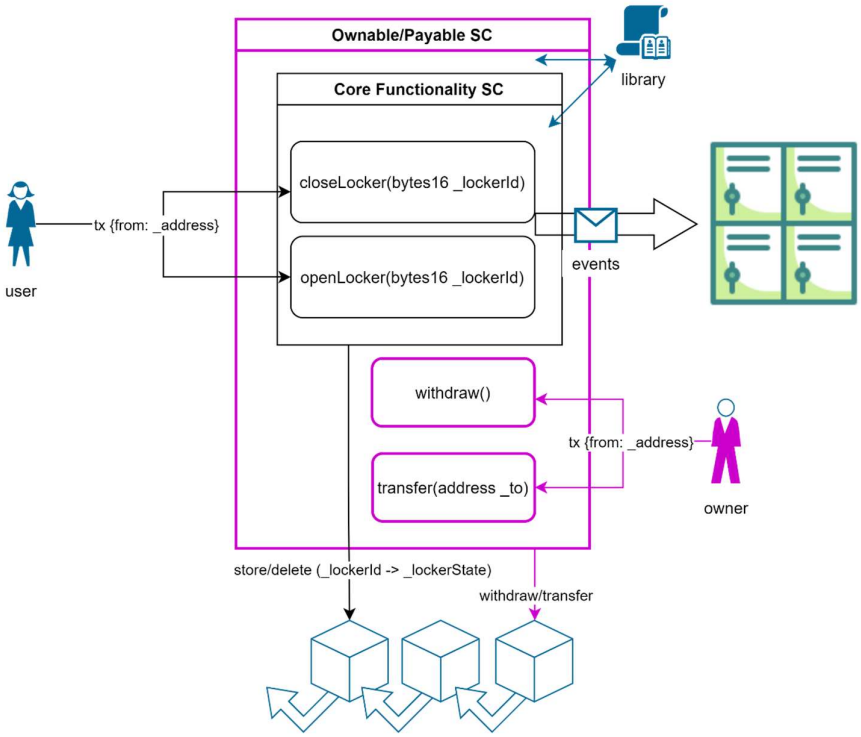

Figure 11: Ownable functionality

4.4 Opening Rights Transfer and Timeout Functionality

As the last SC functionality designed in this report, two use cases were introduced, as seen on **Figure 12**, built on top of the two previously described SCs.

The first use case is specific to IoT locker systems, where the rights for opening a locker are stored digitally and could, therefore, be transferred. In this sense, a user could transfer an item to another user by closing the item in a locker and afterwards transferring the opening rights of that locker to the other user, without never physically meeting each other.

The second use case, the rental of a locker, is a common use case for public lockers. In this case, a user pays for storing his/her belongings for a given period of time. However, as described in Section 3.1, a SC only executes if it is called by a transaction. Therefore, it would not be possible to release a locker automatically after the renting period has finished, unless *(a)* when the user calls the SC to close the locker, the SC would stay running a countdown until the remaining time is exhausted, which would be prohibitively expensive and a waste of blockchain resources or *(b)* an oracle (i.e., an entity that connect blockchains to external systems and that can trigger SC executions [36]) is used.

Although using an oracle such as the ChainLink Alarm-Clock [37] would be an accurate way to release the lockers, integrating an oracle requires additional funding and introduces extra complexity. Thus, different solutions that would not require automatic locker release were analyzed.

One solution could be to simply enable any user to open the locker, once the exclusive opening rights have expired. However, in this case, the currently existing "*openLocker*" method form the Core Functionality SC would need to be overridden. To avoid overriding functionality, the "*transferLocker*" was used instead, enabling any user to transfer the opening rights to himself/herself, once the renting period is finished.

Regarding the closing function, changes in the SC were unavoidable and the original function of the Core Functionality SC was overridden and disabled. Therefore, a new function "*closeLockerFor*" was introduced where the desired interval for renting was added as a parameter and the payable functionality for this function was enabled. Moreover, another function "*increaseRemaining*" was introduced to allow the user with the locker rights to increase the remaining time. These functions (i.e., "*closeLockerFor*" and "*increaseRemaining*") accept only the exact amount, to avoid extra complexity in the SC.

As using time in a SC is not a best practice [38], the amount of blocks was chosen as a measure of the rental interval in the functions exposed by the SC. To improve usability, the time could be approximately converted to blocks by a user interface in front of the SC (e.g., each block in Ethereum is mined after 14 seconds; hence, 256 blocks would mean approximately 1 hour).

Further, the initial price per block is set up on the constructor of the SC and can be updated at any time by the SC owner.

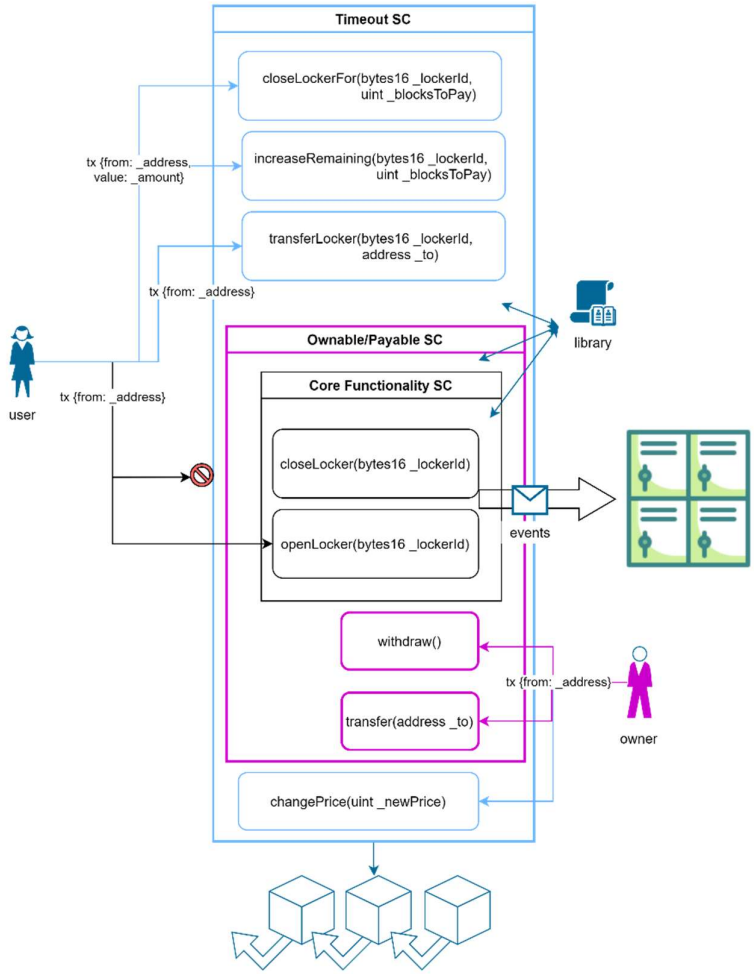

Figure 12: Timeout functionality.

4.5 Development Decisions

The two most active and maintained languages for developing SCs in Ethereum are Solidity and Vyper [39]. Although there are no comparison figures in the literature, a quick query in Udemy [40] (a platform with online courses) showed more learning materials available for Solidity compared to Vyper. Therefore, Solidity [41] was chosen as the programming language for the SC and a Solidity online bootcamp course was chosen for the author of the report.

For the local development environment, Truffle Suite [42] was chosen, because it offers a variety of mature tools to compile and deploy SCs (i.e., Truffle) that integrate easily with a local blockchain simulator (i.e., Ganache). Using these tools gives the opportunity to run tests on the SCs on each iteration proving its functionality. In this sense, **Figure 13** shows an example of the results obtained during a test execution of the prototype.

```
Terminal:  Local ×   Local (2) ×   +
      √ can be opened from the allowedOpener  (855ms)
    V2: price
      √ can be changed by owner  (596ms)
      √ cannot be changed by others  (427ms)

  Contract: LockerManagerV2Core
    V2Core: the locker is free
      √ can be locked (602ms)
      √ can lock multiple compartments (807ms)
    V2Core: the locker is closed
      √ cannot close twice the same compartment (309ms)
      √ cannot take a compartment currently taken  (426ms)
      √ cannot be opened from another account (347ms)
      √ cannot open another compartment  (358ms)
      √ can be opened from the closing account  (691ms)

  Contract: LockerManagerV2Payable
    payable
      √ can accept money (552ms)
      √ somebody cannot transfer (347ms)
      √ somebody cannot withdraw (325ms)
      √ owner can transfer (664ms)
      √ owner can withdraw (522ms)


  34 passing (19s)
```

Figure 13: Example of test execution.

Further, the locker manager SC uses inheritance from standard SCs provided by OpenZeppelin [43], a company that provides open-source, tested and reliable SC implementations for common use cases (e.g., an *ownable* SC).

The source code for the different versions of the SC is available at: https://github.com/soysus/cas-blockchain

5 USER INTERFACE

This section describes the decisions taken for the development of the prototype user interface of the blockchain-based locker system and describes its functionality when connected to the SC as a DApp.

5.1 Web Vs. Native

In order to reach the maximum amount of users, the use of a JavaScript-based website application was chosen instead of a native mobile app, since these (i.e., web apps) are more compatible, have a broader reach and are simpler to maintain [44]. Furthermore, JavaScript is among the most popular languages in the Ethereum ecosystem [45] and multiple mature libraries already exist [46]. Moreover, by using a web application, the connection to the user's blockchain wallet is simplified, as there are JavaScript libraries (e.g., web3js [18]) which offer connectivity with the most common wallets (e.g., MetaMask [47]). Finally, a Web/JavaScript application provides the possibility to be hosted in a decentralized way (e.g., using Swarm [48] and the Ethereum Naming Service [49]).

To develop the prototype web application, React [50] was chosen, because it is one of the most used and supported JavaScript frameworks [51]. For convenience, the development was done using libraries and packages (also known as "boxes") from the same tool provider (i.e., Truffle Suite [42]) used for SCs development. These boxes offer a lean integration of both SC and User Interface in the same project which helps to speed up the development.

5.2 Blockchain Connection

As mentioned previously, there are mature JavaScript libraries (e.g., web3js) that could be included in a web application to connect with the blockchain. These libraries can connect directly to a known blockchain node or to a hot wallet (i.e., wallet connected to internet). Although a blockchain node could be run in parallel to the webserver, the direct connection was discarded because, in order to sign user transactions, the node would need to hold the users' private keys. Thus, the selected approach was to connect to the user's hot wallet.

The leading non-custodian (i.e., the user has sole control of his/her private keys) wallet at the time of writing is MetaMask [47] with more than 10 million active users per month [52]. This type of wallets are considered secure, because they hold the private key of the user in the user' device and never it leaves it [53]. In this sense, the transactions are signed in the user's device and they are forwarded later to a blockchain node for processing.

In the case of MetaMask, it relies by default on Infura [54] as a blockchain node (both products are developed by ConsenSys [55]) but it could be configured to connect to any other node, including an own node or a local development blockchain.

For the prototype, MetaMask was used as a wallet, connected to the previously introduced Ganache's blockchain simulator for local development. This wallet has a browser extension enabling it to connect to whichever web application that would require it.

To implement the connection from the React application, instead of using directly a library such as web3js, it was decided to use another product from the already mentioned Truffle Suite, called Drizzle [56], which can be easily integrated with React. However, although Drizzle's core and components are used in the prototype, the extensions to handle events were cumbersome to use and not much information was found when problems arose. Therefore, manually created React components were used for event handling, instead of Drizzle's components.

5.3 The Connected DApp

To better understand the blockchain connection described in the previous section, a prototype DApp was implemented to interact with the Core Functionality SC described in Section 4.1 (i.e., only open/close functions without payment) and deployed in the local blockchain simulator (i.e., Ganache).

In this sense, **Figure 14** shows how the application is waiting until the wallet is connected before showing the SC available functions.

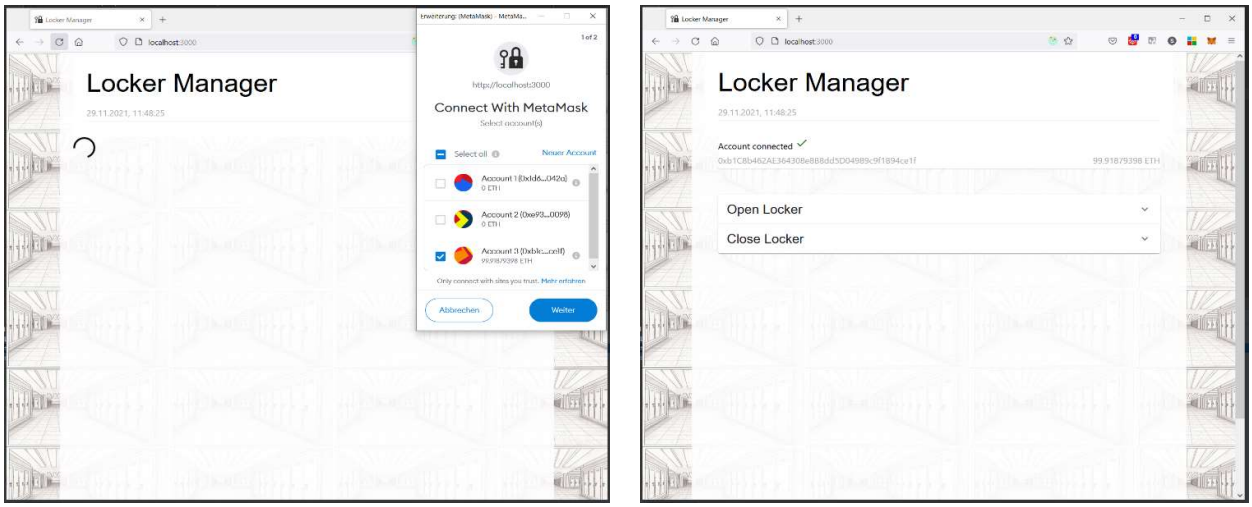

Figure 14: Connecting to the wallet.

Further, **Figure 15** shows how a test *lockerId* is closed and the corresponding event is shown in the interface, while **Figure 16** shows how the *lockerId* can be opened again.

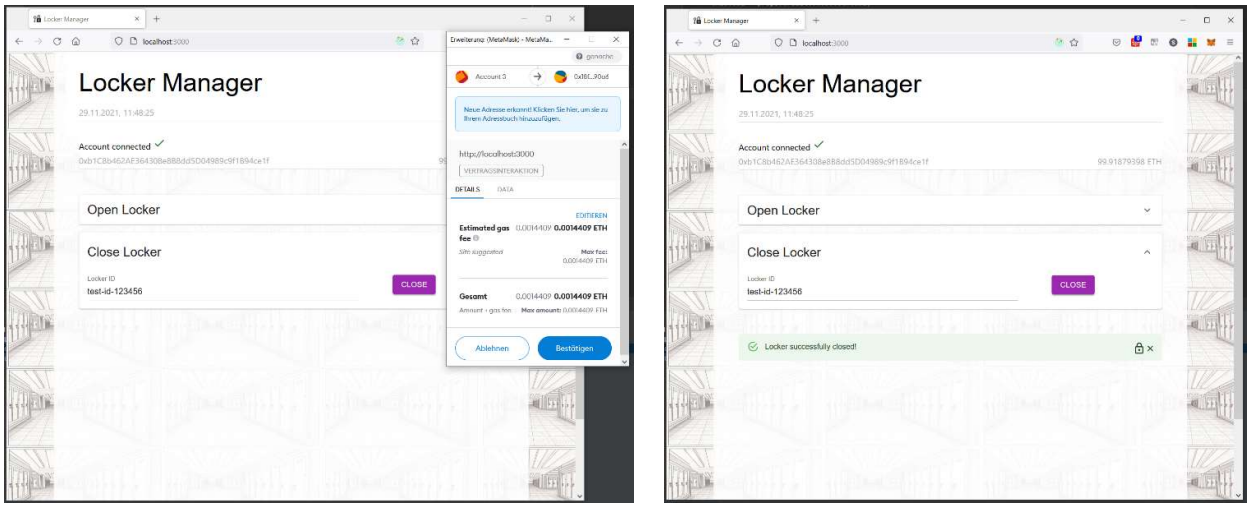

Figure 15: Closing locker + Closed event received.

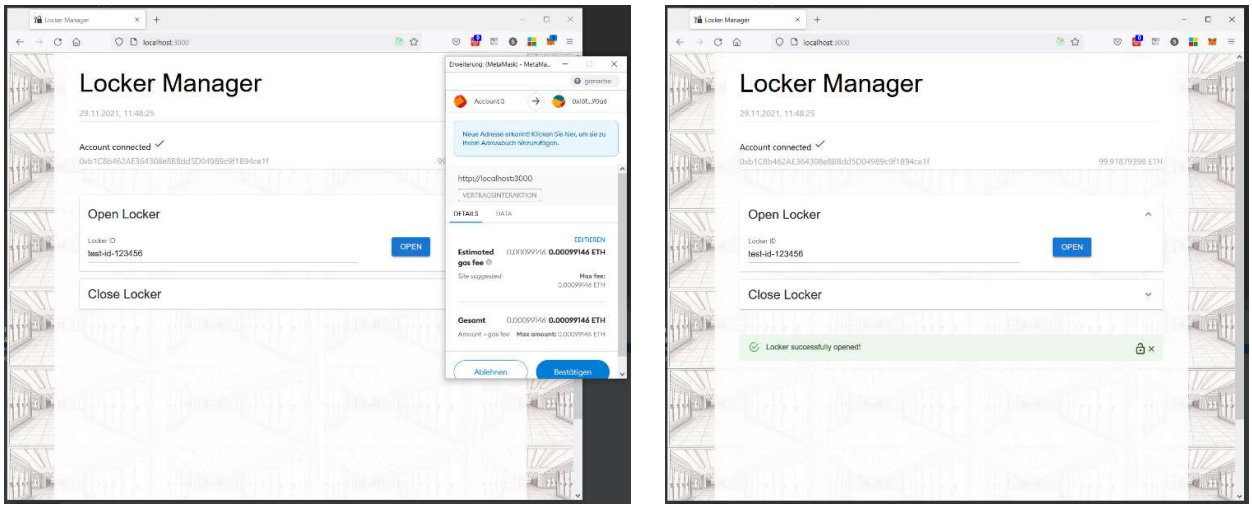

Figure 16: Opening locker + Opened event received.

Finally, **Figure 17** shows the transactions in the local simulation of the Ethereum blockchain.

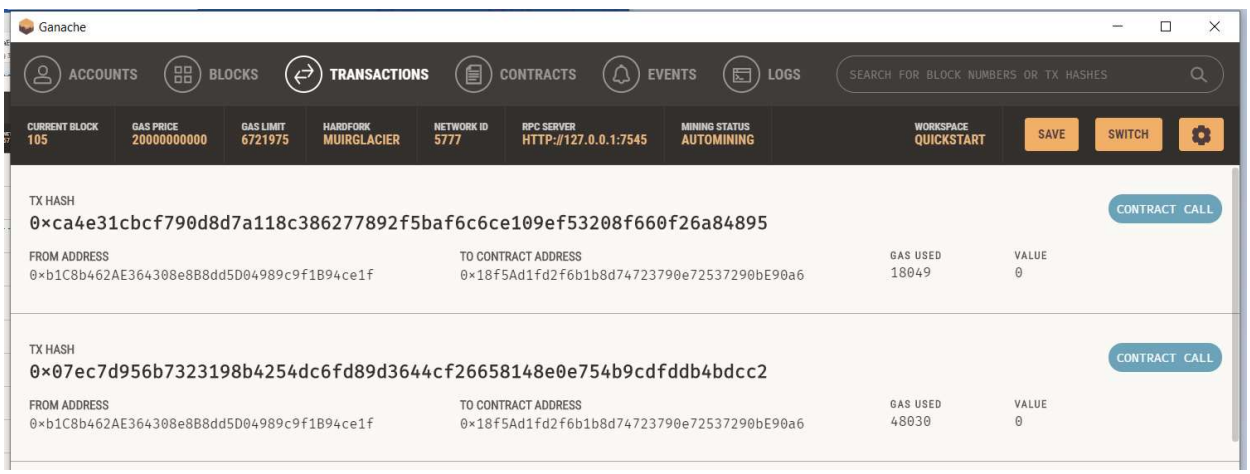

Figure 17: Transactions mined in the local blockchain.

The source code for the user interface is also available at: https://github.com/soysus/cas-blockchain

6 DISCUSSION

In this section, several aspects of the system are discussed such as the decentralization aspect, the transaction cost or other issues such as the synchronization of the locker state and the issues encountered during development.

6.1 Decentralization

The term DApp if often applied any SC with a web frontend, even if they are highly centralized [11]. Hence, one discussion question arise: Is the Locker Manager DApp really decentralized?

Following the literature [11], to achieve a fully decentralized application all possible aspects of it should be decentralized, including hosting the frontend and name resolution.

To achieve hosting decentralization, a solution such as Swarm [48] or IPFS (Inter-Planetary File System) [57] should be chosen to host the frontend application. These are Peer-to-Peer (P2P) [58] networks where the contents

are distributed among peers and can be retrieved from each node. However, these solutions bring challenges, similar to blockchain data, when a change in the application is required [59] (e.g., to fix a bug).

The files stored in these decentralized storages (e.g., a web application) are accessible via a hash that can reach 4 kilobytes [60], which can be difficult to remember. The Ethereum Naming Service [49] proposes an easier way to access the files by creating a link between the hash and a common Domain Name System (DNS) [61] structure under the ".eth" domain.

Therefore, it would be possible to further decentralize the locker system by hosting its frontend in a P2P storage and using ENS. Further analyses should be made comparing the possible advantages and disadvantages, cost and complexity of self-hosted, cloud-hosted and P2P-hosted solutions.

6.2 Transaction Cost

Gas is Ethereum's unit for measuring the computational and storage resources required to perform actions on the blockchain and serves a dual role: as a buffer between the (volatile) price of Ethereum and the reward to miners for the work they do, and as a defense against denial-of-service attacks [11].

At the time of writing, with the data available on [62], the prize of gas varies from 160 to 402 Wei. The minimum price represents an estimated value that is large enough to be of consideration for a miner when choosing transactions. However, the estimated waiting time using this price is approximately 30 minutes.

Following the data obtained during the DApp simulation in Section 5, the price paid on gas for closing a locker would still be over 30 USD. Given that the close function requires 48030 gas (as seen in **Figure 17**), a gas price of 160 Wei per gas (i.e., minimum gas price at the time of writing), and a Wei price of 0.0000044380700 USD. Hence:

$$CloseFunctionCost = 48030 \times 160 \times 0.0000044380700$$
$$CloseFunctionCost \cong 34\ USD$$

Although opening a locker is cheaper (i.e., 18049 gas needed, as seen in **Figure 17**), due to removal of state data, it seems very difficult to find a use case where these prices would be justifiable.

In the prototype described in this report, there were some considerations regarding gas, such as the choice of removing the opened lockers from the storage. However, further analyses should be made on how to improve the cost. Further, research could also be made on other emerging blockchains (e.g., Polygon [63]) with large communities, where prices are cheaper and latency smaller. Analyzing the inter-compatibility between blockchains and the reusability of the knowledge learn in this project.

6.3 Blockchain Status vs. Physical Locker Status

With the current system design, there is the possibility of having different information in the blockchain and in the locker itself. The locker actions are driven by events triggered on the blockchain, but there is no insurance that this status is transmitted to the locker. For instance, during the closing of a locker, after successfully updating the status in the blockchain, a failure in the communication with the locker might occur and this might stay open, making the blockchain state unrealistic.

Further work should be conducted in ensuring that the blockchain state and the lock state are always in sync. For that, mechanisms could be introduced to ask the user to confirm the status or to report an error, enabling the locker to communicate back to the blockchain or a combination of them.

6.4 Challenges Encountered

Although the author of this report has extensive experience designing and developing IT systems, to design and develop this prototype was a bigger challenge than expected and the scope had to be reduced. The restrictions imposed by the blockchain technology and its programming language, the complexity of the architecture and the considerations that must be taken, for instance, regarding gas cost, made this project very interesting but at the same time, very demanding. Further, the use of the selected libraries and frameworks, which were meant to speed the development were sometimes rather the opposite, when information was not widely available.

7 CONCLUSION AND FURTHER WORK

In a growing market of IoT lockers, where manufacturers try to differentiate each other by offering proprietary solutions, a blockchain-based and therefore decentralized standard could offer an opportunity for interoperability. In this sense, the design of such a solution (i.e., a blockchain-based locker system) was analyzed, an architecture was outlined and a prototype was developed. Although the architecture decision is justified and several advantages were described (e.g., easy integration of an international payment capability), the results obtained during simulations regarding transaction cost and latency discourage the usage of the current prototype on Ethereum.

Therefore, the next steps proposed are *(a)* to research different blockchains comparing transaction cost, latencies and inter-compatibility with other blockchains; *(b)* to develop a prototype for the locker interface with a light client installed on a real device (e.g., a Raspberry Pi), so that end-to-end tests could be performed using the blockchain test networks, in order to obtain realistic measures; and *(c)* further improve the SC prototype focusing on cost control.

8 REFERENCES

[1] Tony Uphoff (2021) "How The Hybrid Work Model Is Transforming Business, Impacting Industry": https://www.forbes.com/sites/tonyuphoff/2021/07/28/how-the-hybrid-work-model-is-transforming-business-impacting-industry/?sh=3a2d44882a0c, Forbes.

[2] Service Works Global (2021) "How Lockers Are The Key To Hybrid Workplaces": https://www.swg.com/can/blog/how-lockers-are-the-key-to-hybrid-workplaces/.

[3] Gartner (2021) "Smart locks for your lockers": https://www.gantner.com/solutions/smart-locks/.

[4] dotmagazine (2019) "SMART LOCK MARKET GROWTH BOOSTED BY THE RISING POPULARITY OF SMARTPHONES": https://www.dotmagazine.online/issues/digital-identities/identity-in-digital-space-doteditorial/smart-lock-market.

[5] Grand View Research (2021) "Smart Lock Market Demand To Reach 41.9 Million Units By 2028": https://www.grandviewresearch.com/press-release/global-smart-lock-market https://www.grandviewresearch.com/industry-analysis/smart-lock-market.

[6] Mitchell Grant, and Robert C. Kelly (2020) "Switching Costs": https://www.investopedia.com/terms/s/switchingcosts.asp, Investopedia.

[7] asmag.com (2014) "Smart lock: Unlocking a smarter, more secure home": https://www.asmag.com/showpost/17578.aspx.

[8] cnet (2019) "Smart lock has a security vulnerability that leaves homes open for attacks": https://www.cnet.com/home/security/smart-lock-has-a-security-vulnerability-that-leaves-homes-open-for-attacks/.

[9] Wikipedia (2021) "Blockchain": https://en.wikipedia.org/wiki/Blockchain.

[10] Wikipedia (2021) "Smart Contract": https://en.wikipedia.org/wiki/Smart_contract.

[11] Andreas M. Antonopoulos, and Gavin Wood (2018) "Mastering Ethereum", O'Reilly Media, Inc.

[12] Mahesh Chand (2021) "Do You Need A Blockchain": https://www.c-sharpcorner.com/article/do-you-need-a-blockchain2.

[13] Vikram Dhillon, David Metcalf, and Max Hooper (2021) "Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make it Work for You", O'Reilly Media, Inc.

[14] Karl Wüst, and Arthur Gervais (2018) "Do you Need a Blockchain?", Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE.

[15] E. J. Scheid, B. Rodrigues and B. Stiller (2021) "Policy-Based Blockchain Selection", IEEE Communications Magazine, vol. 59, no. 10, pp. 48-54.

[16] Gavin Wood (2014) "Ethereum: A secure decentralised generalised transaction ledger." Ethereum Project Yellow Paper.

[17] CoinMarketCap: https://coinmarketcap.com/.

[18] Ethereum JavaScript API: https://web3js.readthedocs.io.

[19] Ethereum.org (2021) "Introduction to dapps": https://ethereum.org/en/developers/docs/dapps/.

[20] Mark Richards, Neal Ford (2020) "Fundamentals of Software Architecture"., O'Reilly Media, Inc..

[21] Wikipedia (2021) "Internet Of Things": https://en.wikipedia.org/wiki/Internet_of_things.

[22] Scheid, E. J., Knecht, A., Strasser, T., Killer C., Franco, M., Rodrigues B., Stiller, B. (2021) "Edge2BC: a Practical Approach for Edge-to-Blockchain IoT Transactions", IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2021), Darlinghurst, Australia.: https://ieeexplore.ieee.org/document/9461150.

[23] Riya Thakore, Rajkumar Vaghashiy, Chintan Patel, Nishant Doshi (2019) "Blockchain - based IoT: A Survey": https://www.sciencedirect.com/science/article/pii/S1877050919310178.

[24] ESGroup: https://esgroup.ch/.

[25] Raspberry pi foundation: https://www.raspberrypi.org/.

[26] DLTLabs (2019) "How To Run an Ethereum Node on a Raspberry Pi": https://betterprogramming.pub/running-ethereum-mainnet-on-raspberry-pi-6aca5cae32aa.

[27] Ethereum.org (2021) "NODES AND CLIENTS": https://ethereum.org/en/developers/docs/nodes-and-clients/#clients.

[28] Parity (2018) "What is a light client and why you should care?": https://www.parity.io/blog/what-is-a-light-client/.

[29] MQTT: https://mqtt.org/.

[30] Steve Cope (2020) "Guide to IOT Networking and Messaging Protocols": http://www.steves-internet-guide.com/iot-messaging-protocols/.

[31] Wikipedia (2021) "QR code". https://en.wikipedia.org/wiki/QR_code.

[32] Scanova Blog (2021) "QR Code Statistics 2021: Up-To-Date Numbers On Global QR Code Usage": https://scanova.io/blog/qr-code-statistics/.

[33] Kerigan Marketing Associates (2021) "Why is 2021 the year of QR codes?": https://keriganmarketing.com/why-is-2021-the-year-of-qr-codes/.

[34] Wikipedia (2021) "Universally unique identifier": https://en.wikipedia.org/wiki/Universally_unique_identifier.

[35] GeeksForGeeks (2021) "Creating Ownable Contracts in Solidity": https://www.geeksforgeeks.org/creating-ownable-contracts-in-solidity/.

[36] Chainlink (2021) "What Is a Blockchain Oracle?": https://chain.link/education/blockchain-oracles.

[37] ChainLink Alarm Clock: https://docs.chain.link/docs/chainlink-alarm-clock/.

[38] Phillip Goldberg (2018) "Smart Contract Best Practices Revisited: Block Number vs. Timestamp". Medium: https://medium.com/@phillipgoldberg/smart-contract-best-practices-revisited-block-number-vs-timestamp-648905104323.

[39] Ethereum.org (2021) "SMART CONTRACT LANGUAGES": https://ethereum.org/en/developers/docs/smart-contracts/languages/.

[40] Udemy: https://www.udemy.com/.

[41] Solidity: https://soliditylang.org/.

[42] Truffle Suite: https://www.trufflesuite.com/.

[43] OpenZeppelin: https://openzeppelin.com/.

[44] Tania H. (2019) "Pros and Cons of Mobile Websites and Mobile Apps": https://rubygarage.org/blog/mobile-app-vs-mobile-website.

[45] Ethereum.org (2021) "Ethereum for JavaScript developers": https://ethereum.org/en/developers/docs/programming-languages/javascript/.

[46] Ethereum.org (2021) "JavaScript API libraries": https://ethereum.org/en/developers/docs/apis/javascript/.

[47] MetaMask: https://metamask.io/.

[48] Swarm: https://www.ethswarm.org/.

[49] Ethereum Naming Service: https://ens.domains.

[50] React: https://reactjs.org/.

[51] Paula Clapon (2020) "React.js Statistics 2020": https://www.thinslices.com/blog/infographic-react.js-statistics.

[52] ConsenSys (2021) "MetaMask Surpasses 10 Million MAUs, Making It The World's Leading Non-Custodial Crypto Wallet": https://consensys.net/blog/press-release/metamask-surpasses-10-million-maus-making-it-the-worlds-leading-non-custodial-crypto-wallet/.

[53] Peiyu Wang (2020) "How MetaMask 🦊 stores your wallet secret?": https://www.wispwisp.com/index.php/2020/12/25/how-metamask-stores-your-wallet-secret/.

[54] Infura FAQ: https://infura.io/faq.

[55] ConsenSys: https://consensys.net/.

[56] Drizzle: https://www.trufflesuite.com/drizzle.

[57] IPFS: https://ipfs.io/.

[58] Wikipedia (2021) "Peer-to-peer": https://en.wikipedia.org/wiki/Peer-to-peer.

[59] IPFS discussions (2019) "How can i delete or edit a file?": https://discuss.ipfs.io/t/how-can-i-delete-or-edit-file/5107.

[60] Ethereum Wiki (2021) "Swarm Hash": https://eth.wiki/concepts/swarm-hash.

[61] Wikipedia (2021) "Domain Name System": https://en.wikipedia.org/wiki/Domain_Name_System.

[62] ETH Gas Station: https://ethgasstation.info/.

[63] Poligon: https://polygon.technology/.