Cloud Computing and Data Science
http://ojs.wiserpub.com/index.php/CCDS/

Research Article

Smart Homes and Blockchains: A Smart Door Lock Operated by a Smart Contract

Riccardo Sottini, **Vitor Jesus**[*]

School of Computer Science and Digital Technologies, Aston University, Birmingham, United Kingdom
E-mail: v.jesus@aston.ac.uk

Received: 6 November 2024; **Revised:** 10 January 2025; **Accepted:** 15 January 2025

Abstract: The integration of blockchain technology into numerous domains has demonstrated its ability to address transparency, security, and cost-effectiveness. Furthermore, with its association with cryptocurrencies, it allows seamless integration with payment systems. We report on a project that investigates the practical elements of using blockchains and smart contracts in Cyber-Physical Systems (CPS) in multi-party scenarios involving the delegation of (digital) keys being handed over temporarily. Specifically, we implemented a smart lock system for a conventional front door whose digital "keys" were stored and managed (e.g., digitally handed over) via smart contracts over a public blockchain (Polygon, specifically). Through our evaluation, we found that modifying permissions takes around 5 seconds, primarily due to the time required to update the blockchain, whereas checking for access is instantaneous.

***Keywords*:** smart-homes, blockchains, smart contracts, cyber-physical systems

1. Introduction

The advent of blockchain opened the door for innovative solutions to real-world challenges. Whereas much more research and practical implementations are needed to assess the true value and proposition of smart contracts, it became evident how they can support the operations of complex decentralised systems while ensuring transparency by maintaining immutable records and enhanced security through decentralised validation [1]. The unique properties of blockchains, specifically public ones, sparked interest across industries, resulting in their integration into different domains [1-2] -among them, their applications to the Internet of Things (IoT). Another important relationship is with Cyber-Physical Systems (CPS), a domain parallel to IoT, which focuses on the functionalities of the system's hardware, employing sensors and actuators to perform actions in a physical space [3]. The combination of IoT and CPS methodologies allows the creation of efficient systems while adding blockchain to this equation is promising in multiple instances [4-8].

In this paper, a Cyber-Physical System is integrated with a blockchain where a smart contract implements physical Access Control with a granular permission set. To this end, we designed a smart door lock, both hardware and software. In a nutshell, guests request permission to open the door through a web or mobile application, and the landlord approves the request. The challenge we undertook consisted of implementing, from hardware to software, a physical access control system (as a front door) based on Smart Contracts. From this work, we aimed to learn lessons that can not only be shared with the wider community but also guide a future formalisation of a functional and cyber threat model, similar to the developments in [9].

As we adopted a public blockchain, its inherent transparency allows for an easy review of door accesses, while also enabling the design of various detection and event monitoring mechanisms. Notably, similar systems rely on centralised servers, which are prone to outages and data tampering, whereas our approach ensures availability via blockchain, though it comes with trade-offs that will be discussed later.

A key principle of our project was the use of off-the-shelf hardware. Solidity was the language chosen to implement the smart contract, which serves as the "brain" and decision-maker of the system, while the Polygon network was used as the public blockchain. For hardware, we used off-the-shelf components such as the popular Raspberry Pi with additional common electronics such as a 12 V solenoid electric lock. The interactive user interface was a web application developed in Next.js.

In the remainder of the paper, Section II overviews related work, Section III presents our architectural approach and implementation (both hardware and software) of this specific smart-lock application, Section IV discusses evaluation results, and Section V concludes with future directions.

2. Related work

Introduced in 2008 by Satoshi Nakamoto [10], Blockchains propose a radically new approach to computation. Whereas the original application was the exchange of digital currency (Bitcoin) through a purely decentralised architecture, it saw fast developments, such as with Ethereum in 2015 [1, 11], which proposed a distributed computing platform where a blockchain can execute arbitrary code ("smart contracts") and was no longer limited to maintaining a simple ledger. Since then, it has been applied or considered in virtually all domains, with varying degrees of success. Polygon, which we use in this project, is a platform compatible with Ethereum that guarantees fast transactions and reduces fees by up to 100 times through its scalable architecture [12].

CPS denotes the fusion of various engineering disciplines, such as mechanics, electronics and computer science [2-4], with the aim of reinventing the interaction between humans and computers by using sensors and actuators [13]. The positive synergies and potential of combining blockchains and CPS have been widely recognised [4-5].

In tangential areas, similar integrations have been reported, such as for Industrial IoT [5, 14], generic sensing architectures [15] (e.g., the Electricity grid [16]), and also includes local blockchains [17] (as opposed to public blockchains, that we use). Notably, our project was inspired by an existing open-source project. Specifically in the domain of smart locks, we note similar projects such as using One-Time Pads for increased secrecy [18], or a project similar to ours that used blockchains and reported comparable results [19].

3. Approach

Figure 1 depicts the hardware setup. A Raspberry Pi, is connected to the internet via Wi-Fi, which is electronically wired to control both an light-emitting diode (LED) (for notifications) and a relay connected to a lock. The project assumed two actors: the owner of the property (such as a "landlord") and a guest. The guest would receive the (digital) key from the landlord, who can revoke access at any moment.

Figure 2 shows a simplified architecture diagram where:

The blockchain provides the infrastructure, stores data and enforces rules over all the agents through a smart contract.

The frontend supplies a web interface, allowing users to request authorisation to unlock the door.

The door represents the physical component, receiving instructions from the blockchain on when to open the lock.

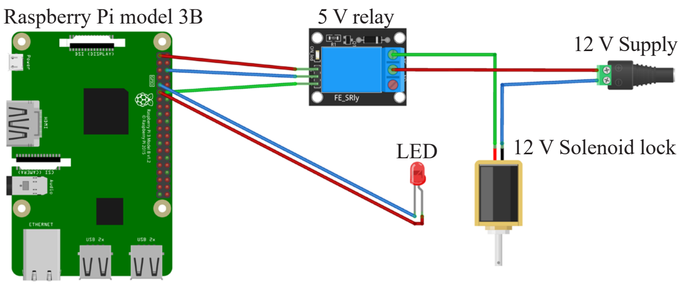


Figure 1. Hardware setup

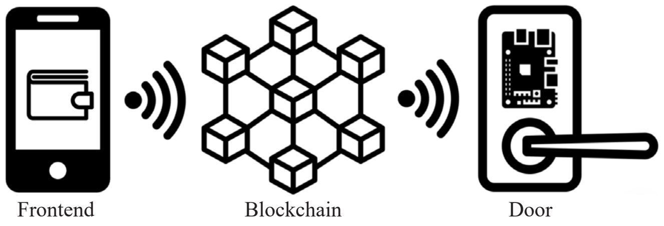


Figure 2. Simplified architecture diagram

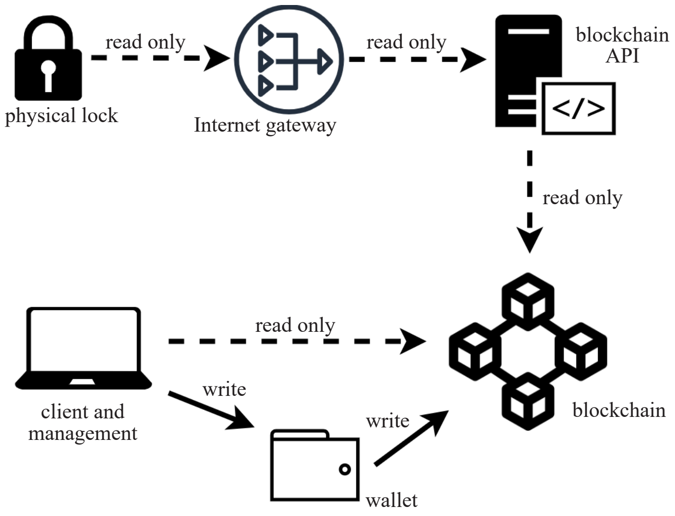


Figure 3. Architecture interfaces

Figure 3 depicts the interfaces. Rather than directly interfacing with the blockchain and considering a CPS scenario where devices are expected to be resource constrained (energy, storage, and connectivity), we used an external element providing APIs to the blockchain. Moreover, a cryptocurrency wallet is required for authentication and to facilitate operations on the blockchain and smart contracts.

3.1 *Architecture*

Our overall architecture uses specific modules: (1) authentication, (2) guest and (3) owner operations, and (4) CPS (door) interfaces.

Authentication-It begins by connecting to the blockchain wallet, which verifies account ownership (Figure 4). Once authenticated, the frontend retrieves the user's balance and role, which can either be an owner or a guest, and displays the corresponding interface.

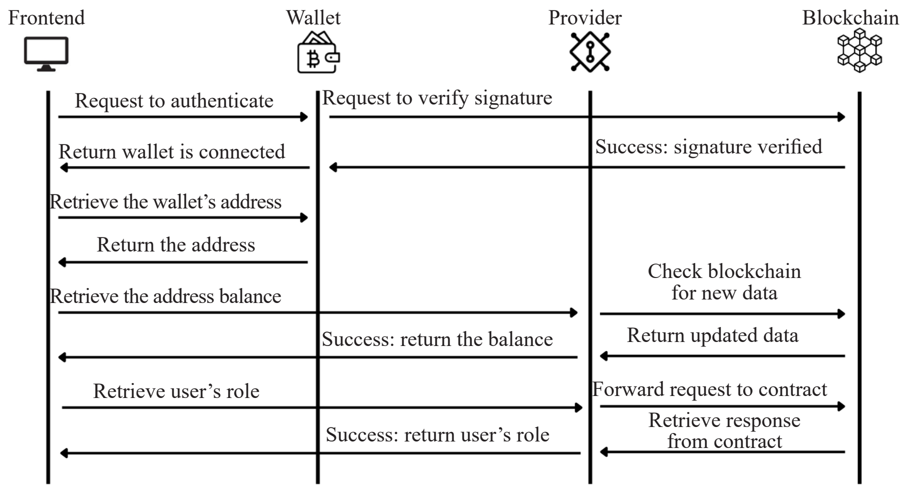

Frontend
Wallet
Provider
Blockchain
Request to authenticate
Request to verify signature
Return wallet is connected
Success: signature verified
Retrieve the wallet's address
Return the address
Check blockchain for new data
Retrieve the address balance
Success: return the balance
Return updated data
Retrieve user's role
Forward request to contract
Success: return user's role
Retrieve response from contract

Figure 4. Authentication

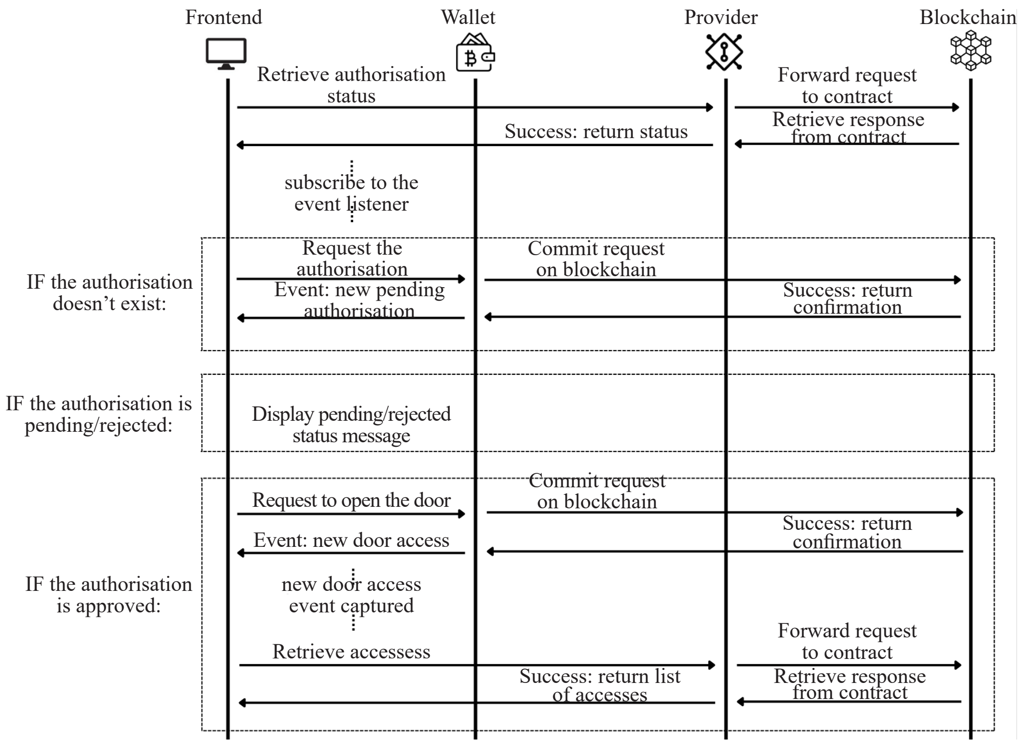

Frontend
Wallet
Provider
Blockchain
Retrieve authorisation status
Forward request to contract
Success: return status
Retrieve response from contract
subscribe to the event listener
IF the authorisation doesn't exist:
Request the authorisation
Commit request on blockchain
Event: new pending authorisation
Success: return confirmation
IF the authorisation is pending/rejected:
Display pending/rejected status message
IF the authorisation is approved:
Request to open the door
Commit request on blockchain
Event: new door access
Success: return confirmation
new door access event captured
Forward request to contract
Retrieve accessess
Success: return list of accesses
Retrieve response from contract

Figure 5. Guest operations

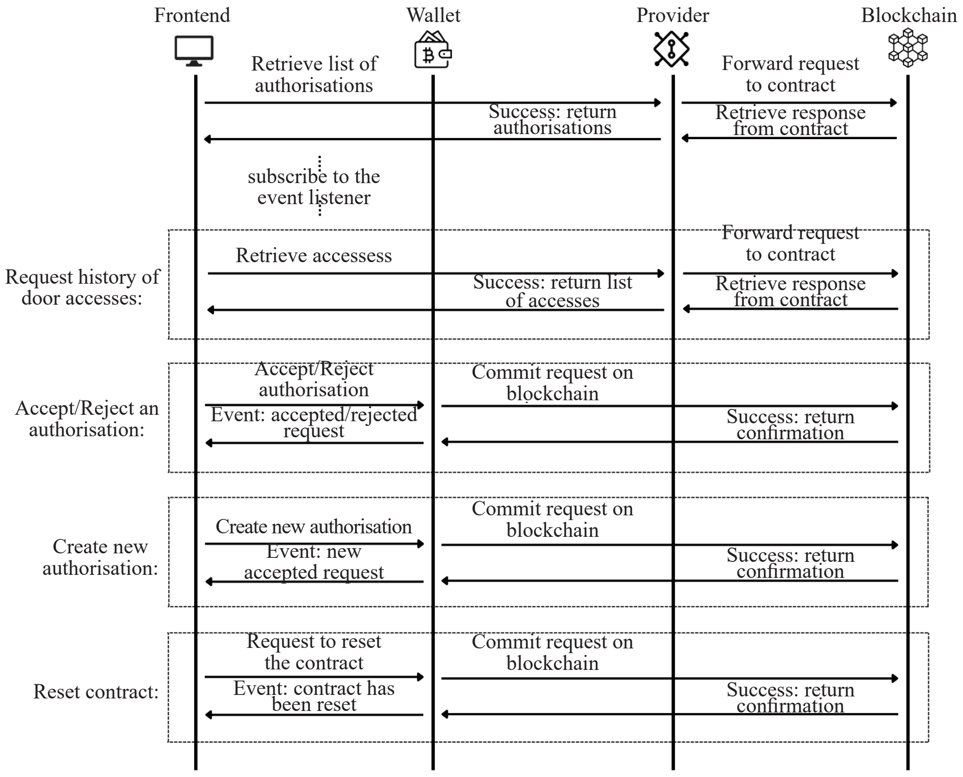

Frontend
Wallet
Provider
Blockchain
Retrieve list of authorisations
Forward request to contract
Success: return authorisations
Retrieve response from contract
subscribe to the event listener
Request history of door accesses:
Retrieve accessess
Forward request to contract
Success: return list of accesses
Retrieve response from contract
Accept/Reject an authorisation:
Accept/Reject authorisation
Commit request on blockchain
Event: accepted/rejected request
Success: return confirmation
Create new authorisation:
Create new authorisation
Commit request on blockchain
Event: new accepted request
Success: return confirmation
Reset contract:
Request to reset the contract
Commit request on blockchain
Event: contract has been reset
Success: return confirmation

Figure 6. Owner operations

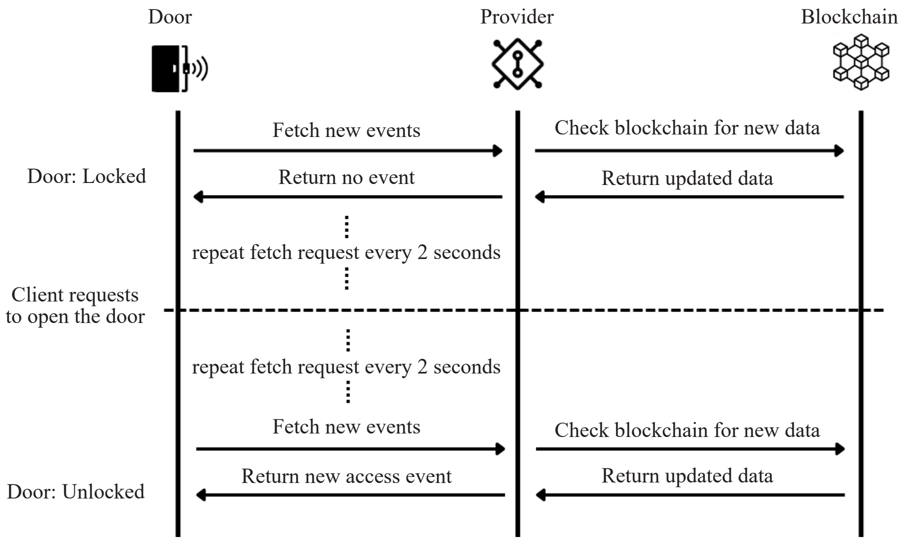

Door
Provider
Blockchain
Fetch new events
Check blockchain for new data
Door: Locked
Return no event
Return updated data
repeat fetch request every 2 seconds
Client requests to open the door
repeat fetch request every 2 seconds
Fetch new events
Check blockchain for new data
Door: Unlocked
Return new access event
Return updated data

Figure 7. Door operations

Guest-First, the frontend evaluates whether the guest has previously requested door access (Figure 5). If not, the guest can submit a request by paying a gas fee, which remains pending until the owner either approves or rejects it. Depending on the owner's choice, door access is granted or a rejection message is displayed. Once approved, the guest can open the door at their discretion.

Owner-The contract's owner has complete control over the data, with the ability to view authorisations and door access, approve, reject, or create new authorisations, and reset all recorded information (Figure 6). Any action taken by the owner that alters the smart contract's data requires the payment of a small gas fee.

Door-The door is interfaced with the smart contract through the blockchain provider, fetching incoming events (Figure 7). Once a new door access is detected, the lock is opened for a fixed amount of time.

3.2 *Implementation*

As mentioned, we used off-the-shelf components and services to implement our project.

3.2.1 *Off-the-shelf components*

We used Polygon as the blockchain platform due to its popularity, support, faster transactions, and low fees. Furthermore, the smart contract was developed using Solidity v0.8.x and deployed on the Polygon Amoy test network, both of which are among the most versatile options available in 2024. The frontend was developed using Next.js v14, a React-based framework combined with TypeScript, which provides type-checking capabilities to improve the robustness of the codebase. It ran locally on a Node.js server during the development phase. However, it can also be deployed remotely for global accessibility.

The lock consists of both hardware and software elements, powered by a Raspberry Pi 3B. The GPIO pins manage a 12 V solenoid lock, a 3.3 V LED, and a 5 V relay. The relay is required as standard electric locks require a 12-volt power supply, whereas the GPIO pins provide only up to 5 volts. Additionally, a Python 3.10.x script was developed to retrieve events from the smart contract and control when the LED and the relay must be activated, resulting in the lock being opened.

Furthermore, MetaMask was used as a browser-based wallet. It serves to store private keys, verify the user's account ownership, and manage their assets. Moreover, it played a crucial role in authentication and interaction with the smart contract. Finally, we used a third-party blockchain provider as a service via an Application Programming Interface (API), instead of a self-hosted blockchain node for scalability. Specifically, QuikNode was selected as a provider as it offers API endpoints to deploy smart contracts and fetch data via HTTP and WebSocket protocols, supporting synchronous and asynchronous communication.

3.2.2 *Smart contracts*

The initial focus was on identifying the data structures to store the users, authorisations and door access entries. Subsequently, several methods were defined and categorised into two groups: 'view' functions, which retrieve data, and 'state-changing' functions, which require a gas fee to alter or insert data, as shown in Table 1.

Table 1. Smart contract's methods

Method	Type	Role	Description
getRole	View	All	Retrieve user's role
getAuthorisation	View	Guest	Retrieve guest's authorisation
getAccessess	View	Guest	Retrieve door's accesses
getData	View	Owner	Retrieve the authorisations list
requestAuthorisation	State-changing	Guest	Request an authorisation
accessDoor	State-changing	Guest	Request to open the door
createAuthorisation	State-changing	Owner	Create a new authorisation
acceptAuthorisation	State-changing	Owner	Accept a guest's authorisation
rejectAuthorisation	State-changing	Owner	Reject a guest's authorisation
reset	State-changing	Owner	Reset the data of the contract

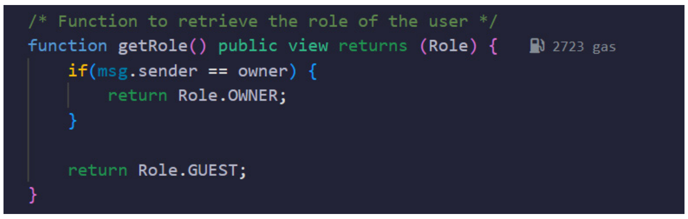

```
/* Function to retrieve the role of the user */
function getRole() public view returns (Role) {    2723 gas
    if(msg.sender == owner) {
        return Role.OWNER;
    }

    return Role.GUEST;
}
```

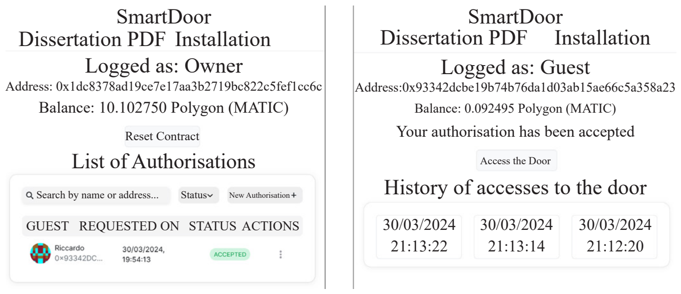
Figure 8. getRole definition (an example of "view" method)

As an example, Figure 8 illustrates the definition of a method in Solidity, getRole, a "view" method that retrieves the user's role. A method definition includes its name, the visibility, the returned data type and its instructions.

Asynchronous communication with the frontend and the door script is established through an event-based mechanism. Three event types were defined to notify when data is modified: one for new door access, one for the contract reset, and one prompting the guest and owner to fetch updated data.

3.2.3 *Frontend*

A web application was developed using Next.js, with separate interfaces: the owner can manage authorisations and reset the contract, while guests can decide when to access the door and view the history of previous accesses. Figure 9 illustrates the user interface.


SmartDoor
Dissertation PDF Installation
Logged as: Owner
Address: 0x1dc8378ad19ce7e17aa3b2719bc822c5fef1cc6c
Balance: 10.102750 Polygon (MATIC)
Reset Contract
List of Authorisations
Search by name or address... Status New Authorisation +

GUEST	REQUESTED ON	STATUS	ACTIONS
Riccardo 0×93342DC...	30/03/2024, 19:54:13	ACCEPTED	⋮

SmartDoor
Dissertation PDF Installation
Logged as: Guest
Address:0x93342dcbe19b74b76da1d03ab15ae66c5a358a23
Balance: 0.092495 Polygon (MATIC)
Your authorisation has been accepted
Access the Door
History of accesses to the door
30/03/2024 21:13:22 30/03/2024 21:13:14 30/03/2024 21:12:20

Figure 9. User interfaces for the owner and guests, respectively

Throughout the frontend development, the Web3.js library was used to interact with the smart contract. An event listener was configured to capture new events and update the interface to replicate the mechanism shown in Figure 10.

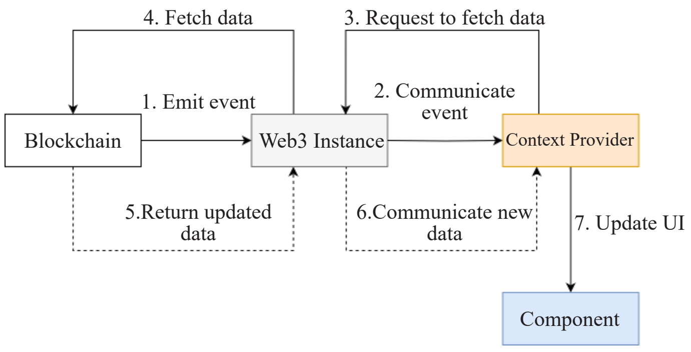
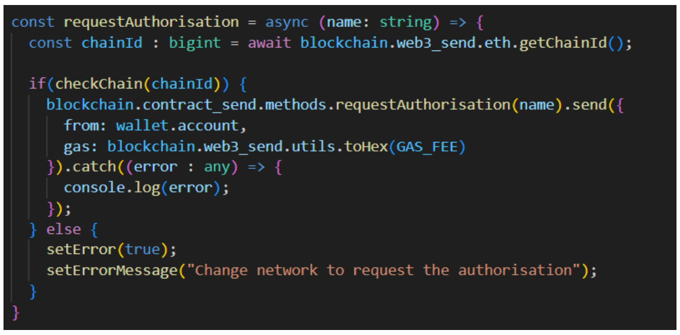


Figure 10. Capturing a new event



```
const requestAuthorisation = async (name: string) => {
  const chainId : bigint = await blockchain.web3_send.eth.getChainId();

  if(checkChain(chainId)) {
    blockchain.contract_send.methods.requestAuthorisation(name).send({
      from: wallet.account,
      gas: blockchain.web3_send.utils.toHex(GAS_FEE)
    }).catch((error : any) => {
      console.log(error);
    });
  } else {
    setError(true);
    setErrorMessage("Change network to request the authorisation");
  }
}
```

Figure 11. requestAuthorisation() contract's method call

Another interaction involves calling a contract's method by first verifying the wallet's network and then confirming the call by paying a specified gas fee (Figure 11).

3.2.4 *Door*

The door lock is controlled through a Python script that captures the event of a new door access, resulting in a signal being sent to certain GPIO pins to power an LED and open the lock for a fixed amount of time (Figure 12).

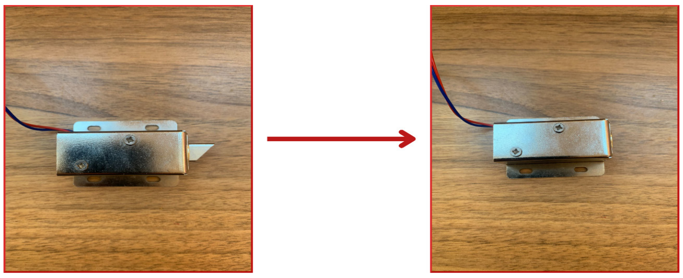
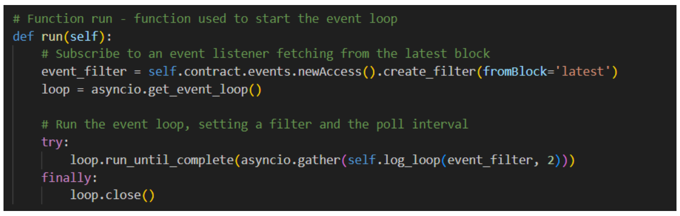


Figure 12. Solenoid lock transitioning from closed to open

A fundamental step in creating the script was configuring an event loop that polls for new events every two seconds and filters them to identify door accesses from the latest block (Figure 13).



```
# Function run - function used to start the event loop
def run(self):
    # Subscribe to an event listener fetching from the latest block
    event_filter = self.contract.events.newAccess().create_filter(fromBlock='latest')
    loop = asyncio.get_event_loop()

    # Run the event loop, setting a filter and the poll interval
    try:
        loop.run_until_complete(asyncio.gather(self.log_loop(event_filter, 2)))
    finally:
        loop.close()
```

Figure 13. Subscription to the event listener

4. Evaluation

Our approach was evaluated based on performance, the maintenance costs, and the security of the smart contracts.

4.1 *Performance*

The system's performance was measured by considering the interaction time with the smart contract's methods. To do so, our self-developed "evaluate_methods.js" script was used to retrieve the minimum, average, and maximum interaction times, based on 25 calls to each contract's method (Figure 14).

As a result, the average time for "state-changing" method calls, which alter data, is 4.047 seconds, whereas "view" method calls are much quicker, averaging 0.224 seconds. The accessDoor method, which commands the door to open, is the system's most time-constraining operation, with an average execution time of 3.501 seconds. It is worth mentioning that the experiment was conducted under good conditions, employing the Polygon Amoy test network with low congestion.

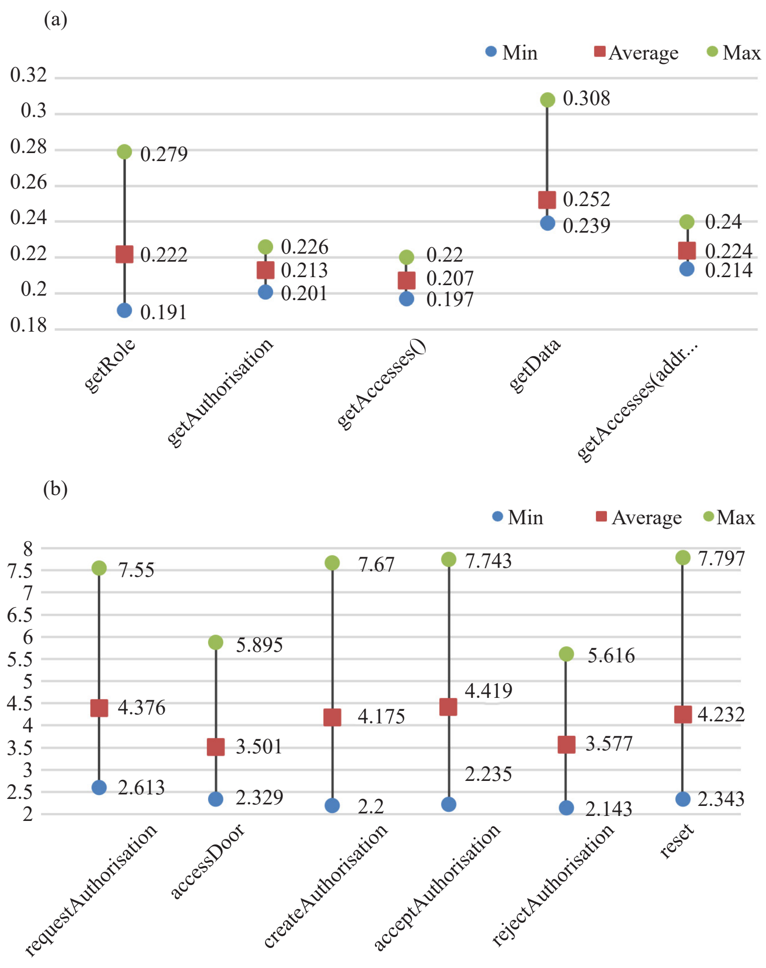


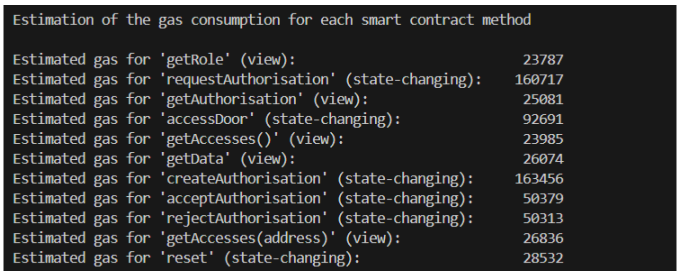
Figure 14. Evaluation: (a) view method times, (b) state-changing method times

We can also observe that neither the hardware (admittedly low end) nor the blockchain itself impacts scalability, at least at a local level. Whereas reconfiguring permissions and access is acceptable to be slow (yet, in the order of seconds) as it is an infrequent activity, gaining access can be a simple read process that takes no resources and can be done offline, if necessary, should the hardware and server have a local copy of the ledger. In any case, one notes that simply waiting ~ 5 seconds is well within expected usability parameters. On the other hand, a valid discussion point is whether a public, global blockchain can handle many such devices, a discussion connected to the underlying blockchain infrastructure and is out of scope in this paper yet will be explored in future work.

4.2 *Costs*

The cost of maintaining a system supported by a blockchain is defined by gas fees paid when interacting with state-changing methods. Moreover, view functions also consume gas, but this only measures the computational expenditure for the network, and users are not required to pay for it. The "estimate_gas.js" script (see Figure 15) was developed to retrieve the gas consumption for each smart contract method, expressed in GWEI (a cryptocurrency unit derived from "giga-wei", which is equivalent to one-billionth of an Ether).

Upon analysis, the lowest gas consumption is found in view methods, with values ranging from 23,787 to 26,836 GWEI. The key insight is that state-changing methods inserting new data incur higher fees than those updating existing data. For instance, the methods requestAuthorisation and createAuthorisation consume over 160,000 GWEI to insert a new authorisation into the list, in contrast to methods like acceptAuthorisation and rejectAuthorisation, which alter existing authorisations and incur lower fees of around 50,000 GWEI per call.



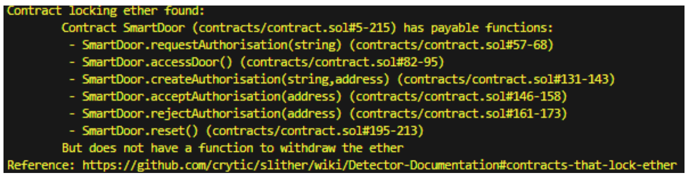
```
Estimation of the gas consumption for each smart contract method

Estimated gas for 'getRole' (view):                                23787
Estimated gas for 'requestAuthorisation' (state-changing):        160717
Estimated gas for 'getAuthorisation' (view):                       25081
Estimated gas for 'accessDoor' (state-changing):                   92691
Estimated gas for 'getAccesses()' (view):                          23985
Estimated gas for 'getData' (view):                                26074
Estimated gas for 'createAuthorisation' (state-changing):         163456
Estimated gas for 'acceptAuthorisation' (state-changing):          50379
Estimated gas for 'rejectAuthorisation' (state-changing):          50313
Estimated gas for 'getAccesses(address)' (view):                   26836
Estimated gas for 'reset' (state-changing):                        28532
```

Figure 15. Estimation of gas consumption for each method

This type of experiment is particularly useful for optimising costs and identifying bottlenecks in the smart contract design.

4.3 *Security*

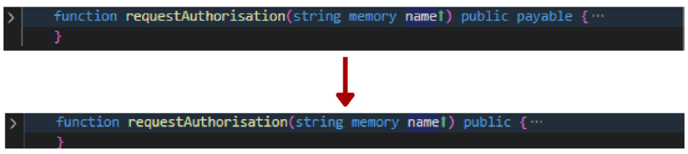
The security of the smart contracts was evaluated using Slither, which identified various threats across medium-risk, low-risk, and informational categories. Once identified, these threats were addressed by making changes to the contract's code. Medium-risk security threats underline bugs exploitable by malicious entities. One example, shown in Figure 16, relates to the locking of funds that occurs when a state-changing method is defined as "payable," allowing users to send funds but not implementing a mechanism to manage them. The fix involved removing the "payable" keyword from the impacted methods, as shown in Figure 17.



```
Contract locking ether found:
        Contract SmartDoor (contracts/contract.sol#5-215) has payable functions:
         - SmartDoor.requestAuthorisation(string) (contracts/contract.sol#57-68)
         - SmartDoor.accessDoor() (contracts/contract.sol#82-95)
         - SmartDoor.createAuthorisation(string,address) (contracts/contract.sol#131-143)
         - SmartDoor.acceptAuthorisation(address) (contracts/contract.sol#146-158)
         - SmartDoor.rejectAuthorisation(address) (contracts/contract.sol#161-173)
         - SmartDoor.reset() (contracts/contract.sol#195-213)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
```

Figure 16. Medium-risk threat: contract locking ether

Low-risk threats do not compromise the security of a smart contract; however, they serve as indications of best practices, and they might undermine maintainability. For example, the owner address must be defined as immutable, and before looping through an array, its length must be stored in a variable.



```
function requestAuthorisation(string memory name) public payable {···
}
```

```
function requestAuthorisation(string memory name) public {···
}
```

Figure 17. requestAuthorisation method redefinition

5. Conclusions and future work

The design for a Smart-Lock system supported by a Smart Contract is presented, with detailed design, code, and evaluation, allowing researchers and professionals to gain practical insights into the synergies of blockchains and CPS. The developed system presents numerous benefits. For instance, the system does not require a dedicated backend to store data and execute operations, as the blockchain assumes this role. This property allows the system to be accessible by everyone worldwide while offering good security, and virtualisation of functions such as auditing and access control. Moreover, stored data cannot be altered, resulting in the record of door accesses not being tampered with by malicious entities. Ultimately, another benefit relates to authentication, which presents a high-grade security standard, removing the need for a physical key to open the door.

To evaluate the system's effectiveness, a standardised process was created. As a result, it was observed that the performance, expressed as the time for the system to execute an operation, is highly variable, potentially influencing real-time interactions where fast response is important. Another factor analysed was maintenance costs, which are linked to gas fees. In this case, the most cost-expensive operations and the fact that network congestion impacts the costs at any given time were highlighted. Finally, the security of the smart contract was assessed, highlighting the ease of detecting and fixing vulnerabilities through Slither, a widely used security tool.

This project aims to advance future research on the integration of Cyber-Physical Systems (CPS) and blockchain technology. For instance, the proposed architecture provides a framework for evaluating the security standards of network protocols, not only by analysing threats at the application layer but through a comprehensive assessment of the entire blockchain infrastructure. Additionally, there is potential for scaling this architecture to support large-scale commercial systems, allowing the evaluation of key factors such as scalability, throughput, technological constraints, and environmental sustainability. Specifically, domains like rental property management and commercial building access control could greatly benefit from such a scaled-up solution. Ultimately, an emphasis on developing fault tolerance, health and safety protocols, and a more robust hardware framework would be beneficial to strengthen its resilience and practicality.

Conflict of interest

The authors declare that they have no conflict of interest.

References

[1] Buterin V. *A Next-Generation Smart Contract and Decentralized Application Platform*. Ethereum White Paper. 2014.

[2] Atlam HF, Alenezi A, Alassafi MO, Wills G. Blockchain with internet of things: Benefits, challenges, and future directions. *International Journal of Intelligent Systems and Applications*. 2018; 10(6): 40-48.

[3] Lee EA. The past, present and future of cyber-physical systems: A focus on models. *Sensors*. 2015; 15(3): 4837-4869.
[4] Lesch V, Züfle M, Bauer A, Iffländer L, Krupitzer C, Kounev S. A literature review of IoT and CPS-What they are, and what they are not. *Journal of Systems and Software*. 2023; 200: 111631.
[5] Zhao W, Jiang C, Gao H, Yang S, Luo X. Blockchain-enabled cyber-physical systems: A review. *IEEE Internet of Things Journal*. 2020; 8(6): 4023-4034.
[6] Fernández-Caramés TM, Fraga-Lamas P. A review on the use of blockchain for the internet of things. *IEEE Access*. 2018; 6: 32979-33001.
[7] Ali MS, Vecchio M, Pincheira M, Dolui K, Antonelli F, Rehmani MH. Applications of blockchains in the Internet of Things: A comprehensive survey. *IEEE Communications Surveys and Tutorials*. 2018; 21(2): 1676-1717.
[8] Dai HN, Zheng Z, Zhang Y. Blockchain for internet of things: A survey. *IEEE Internet of Things Journal*. 2019; 6(5): 8076-8094.
[9] Sengupta J, Ruj S, Bit SD. A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT. *Journal of Network and Computer Applications*. 2020; 149: 102481.
[10] Nakamoto S. *Bitcoin: A Peer-To-Peer Electronic Cash System*. Satoshi Nakamoto; 2008.
[11] Zheng G, Gao L, Huang L, Guan J. *Ethereum Smart Contract Development in Solidity*. Berlin/Heidelberg, Germany: Springer; 2021.
[12] Thibault LT, Sarry T, Hafid AS. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*. 2022; 10: 93039-93054.
[13] Baheti R, Gill H. Cyber-physical systems. *The Impact of Control Technology*. 2011; 12(1): 161-166.
[14] Xu H, Wu J, Pan Q, Guan X, Guizani M. A survey on digital twin for industrial internet of things: Applications, technologies and tools. *IEEE Communications Surveys and Tutorials*. 2023; 25(4): 2569-2598.
[15] Zhao W, Aldyaflah IM, Gangwani P, Joshi S, Upadhyay H, Lagos L. A blockchain-facilitated secure sensing data processing and logging system. *IEEE Access*. 2023; 11: 21712-21728.
[16] Musleh AS, Yao G, Muyeen SM. Blockchain applications in smart grid-review and frameworks. *IEEE Access*. 2019; 7: 86746-86757.
[17] Rawlins CC, Jagannathan S. An intelligent distributed ledger construction algorithm for IoT. *IEEE Access*. 2022; 10: 10838-10851.
[18] Srinivasan P, Sabeenian RS, Thiyaneswaran B, Swathi M, Dineshkumar G. OTP-based smart door opening system. In: *Intelligent Communication Technologies and Virtual Mobile Networks: Proceedings of ICICV 2022*. Singapore: Springer Nature Singapore; 2022. p.87-98.
[19] Han D, Kim H, Jang J. Blockchain based smart door lock system. In: *2017 International Conference on Information and Communication Technology Convergence (Ictc)*. IEEE; 2017. p.1165-1167.