EC544 Technical Report

Submitted to: Professor Babak Kia Montazam

by

Team 10

Team Members

Talha Kamran, tkamran@bu.edu
Juehao Lin, juehlin@bu.edu
Nuwapa Promchotichai (Prim), pnuwapa@bu.edu

Submitted: Wednesday May 4, 2022

I. Introduction

A smart lock is defined as an electromechanical lock which is designed to perform locking and unlocking operations on a door when it receives such instructions from an authorized device using a wireless protocol and a cryptographic key to execute the authorization process. Compared to conventional locks that required physical keys, it offered a more diverse functionality including remote access control, timed access control and prevention of physical key copy etc. However, these benefits come with new cybersecurity risks. A Centralized database for remote control or an insecure transmission protocol may lead to compromised access authorization.

In this project, we will focus on exploring and researching on the possibilities and feasibility of building a smart lock based on blockchain and smart contract. Such a system not only allows the owner of the lock to have full access control over the lock, it also provides a decentralized system that eliminates the central authority and possible related exploitations. Furthermore, our proof-of-concept project will also try to test our system via Ethereum wallet and web3 Application to simulate the ownership's management and the access of the smart lock.

II. Project Overview

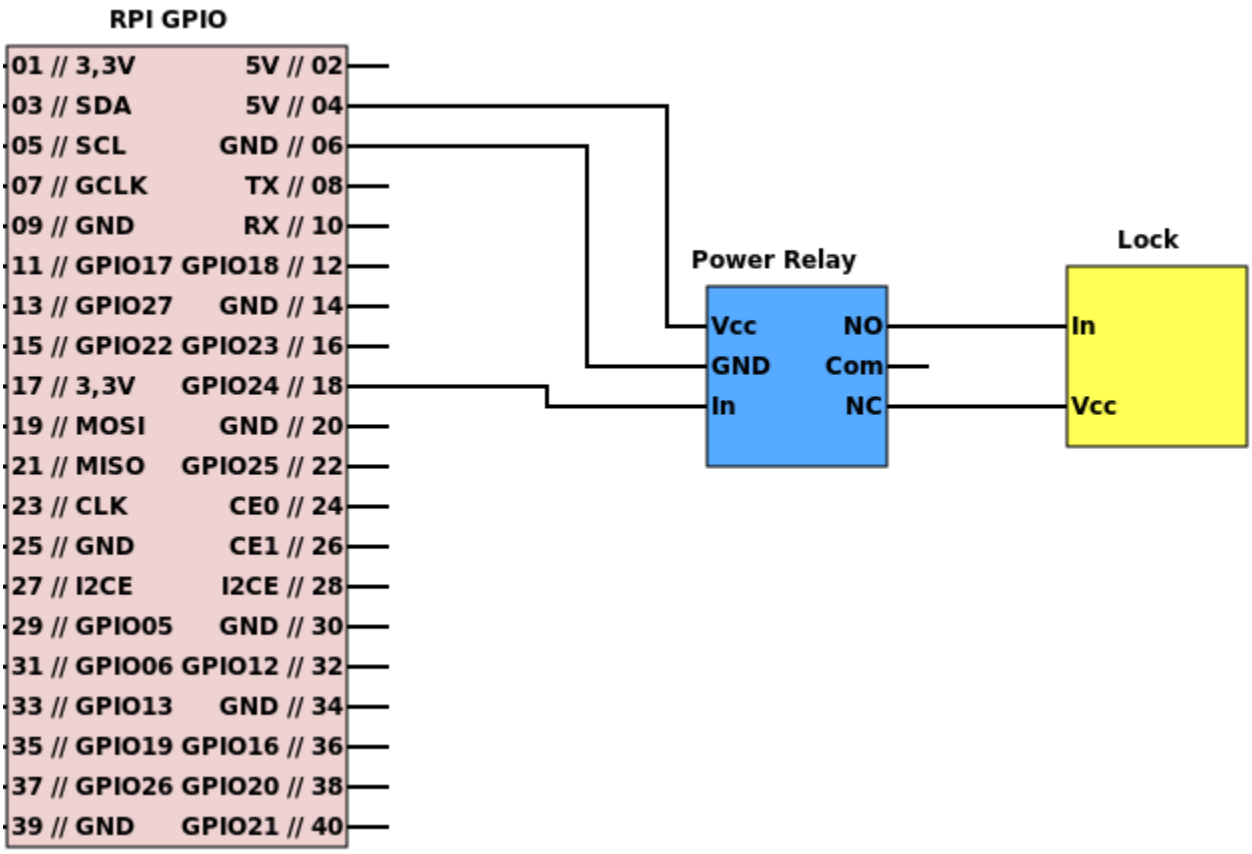
For this project, the system comprises of following components:

Hardware

- Raspberry Pi 4, MCU for the system
- Door locking/unlocking mechanism
    - 1, 5V Relay Module
    - 1, DC 12V Solenoid Electromagnetic Mini Door Lock
    - Female to female jumper wires
    - Male to female jumper wires
    - 1, 12V 2A Power Supply
    - 1, Female DC Power Jack Adapter

Refer to figure 1 for the hardware wiring diagram.

Figure 1: Hardware wiring diagram for door lock



Software

- Entry Portal: A web3 Dapp developed via Replit, a standard web development stack combining JavaScript, HTML, and CSS. It offers users the ability to connect to the user's ether wallet via ethers library.
- Polygon Side-Chain Network: Private test networks designed for emulating the Ethereum main network behavior. It enables private testing networks for Ethereum-based projects. Polygon supports smart contracts and decentralized applications and is a side-chain that runs parallel to Etheruem. In addition to that, it also provides a set of API that allows developers to integrate with web3 apps and Ethereum wallets. In basic terms Polygon is trying to be the AWS of the Web3 world.
- Metamask: Ethereum wallet that allows users to access the blockchain network and authorize.

III. System Architecture:

Hardware

12V Solenoid Electromagnetic Mini Door Lock was used to demonstrate the door lock mechanism. The MCU for this project is Raspberry Pi 4. We control a solenoid with a raspberry pi 4 using a 5V relay (refer to Figure 1 for wiring diagram). Since the goal of the project is to demonstrate the door lock mechanism based on a click of a button from web3, raspberry pi also needs to be able to read from the smart contract.

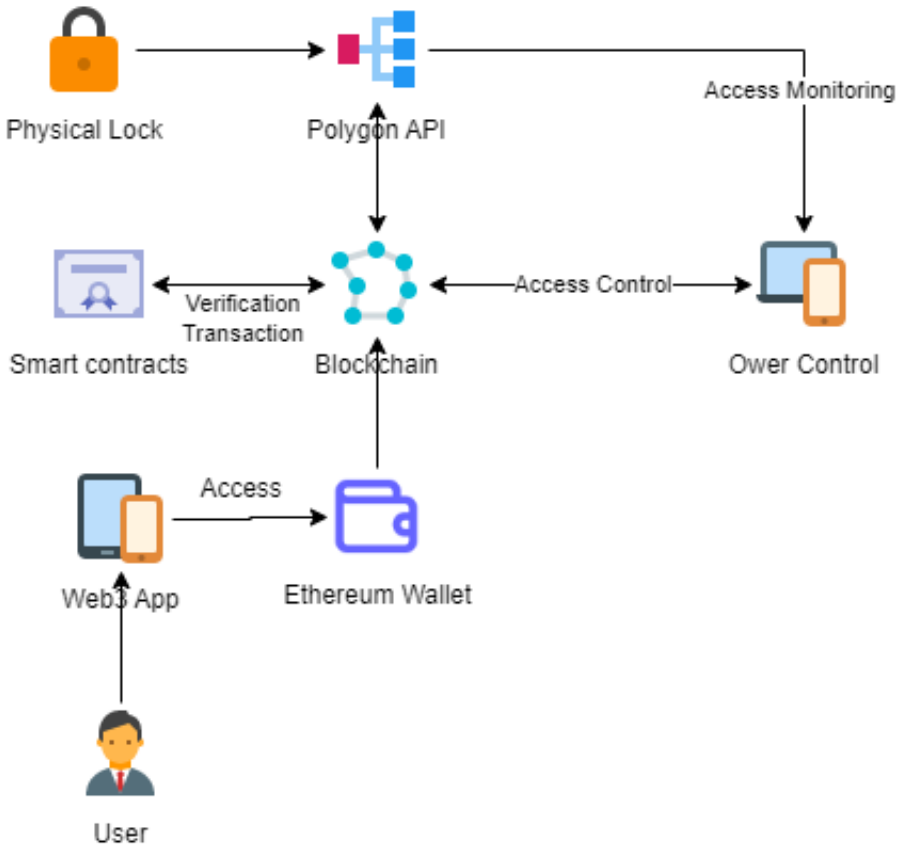
When a user clicks a button on the web3 to lock/unlock the door, it will trigger the toggle() function inside the contract. This toggle() function changes the state of the door lock inside the contract. For this project, state = true means the door is locked and state = false means the door is unlocked. By default, the state is set to true. The solenoid door lock will be triggered based on the state changes. In order for the raspberry pi to trigger solenoid based on the state changes, raspberry pi needs to call a function state() which is a getter function to return a state variable from the contract. We created an entryRead.js with web3.js library to connect to the contract via raspberry pi and get the state variable from the contract. Web3.js library has methods to connect to the deployed contract using the ABI, contractAddress, and RPC server of the contract. To make sure that raspberry pi constantly checks for state variables from the contract and triggers the lock accordingly, we created a bash script, checkState.sh. checkState.sh loops every 10 seconds to call node entryRead.js to get the state variable from the contract, and based on the state, triggers a GPIO pin to control the lock/unlock of a solenoid.

Software

To avoid security concerns based on the centralized access control method, we modified the structure of a conventional smart lock to deploy the authentication and verification process onto a decentralized system, which is the Polygon Network. The Polygon Network offers Ethereum Blockchain based architecture that allows us to develop a decentralized application to manage the door lock. The Polygon Network keeps track of the user's unique account address, and is also capable of transferring and managing ownership of the lock via ethereum transactions.

To access the Polygon Network, where all the decentralized information is held, we developed a web3 application that calls on Polygon API to verify the identity of the user via Ethereum wallet (Metamask) and access the Polygon Network. The smart contracts we deployed on the network will verify the identity and authenticity of the given account address given by the Ethereum wallet to check if the account and its owner have the authentication required to access the lock. If the answer is yes, it will unlock the lock by changing a public read-only variable on the network to unlock the lock, where the Raspberry Pi will read the variable using onboard .js program.

Figure 2: Software architecture diagram



IV. Technical Description

Front End

Lock hardware

The hardware components consist of two main functions:

1. Controlling the solenoid with raspberry pi and relay
    - We use raspberry pi 4 and 5V relay to supply and turn the 12V solenoid (refer to Figure 1. for wiring).
    - To trigger the lock/unlock of a solenoid, we run the bash script checkState.sh via raspberry pi. checkState.sh triggers pin GPIO 18 (solenoid) based on the state variable of true or false which we pull from the contract deployed.
2. Connecting to the smart contract and read state variable from it
    - entryRead.js uses web3.js library to connect to the deployed contract. Inside entryRead.js, we define methods to connect to the deployed contract and call function state() to read the state variable from the contract.
    - checkState.sh is a bash script which calls entryRead.js and assigns the result from entryRead.js (state variable) into a variable. We then trigger pin GPIO 18 based on the variable. If the variable is true, then we close the lock. Otherwise we open the lock.

Overall, to trigger the lock hardware, we created an entryRead.js file with web3.js library to connect to the deployed contract and read the state variable from it. Then we call the entryRead.js from checkState.sh script which loops every 10 seconds to pull the state variable (true/false) from the contract and trigger the solenoid lock accordingly.

V. Use cases

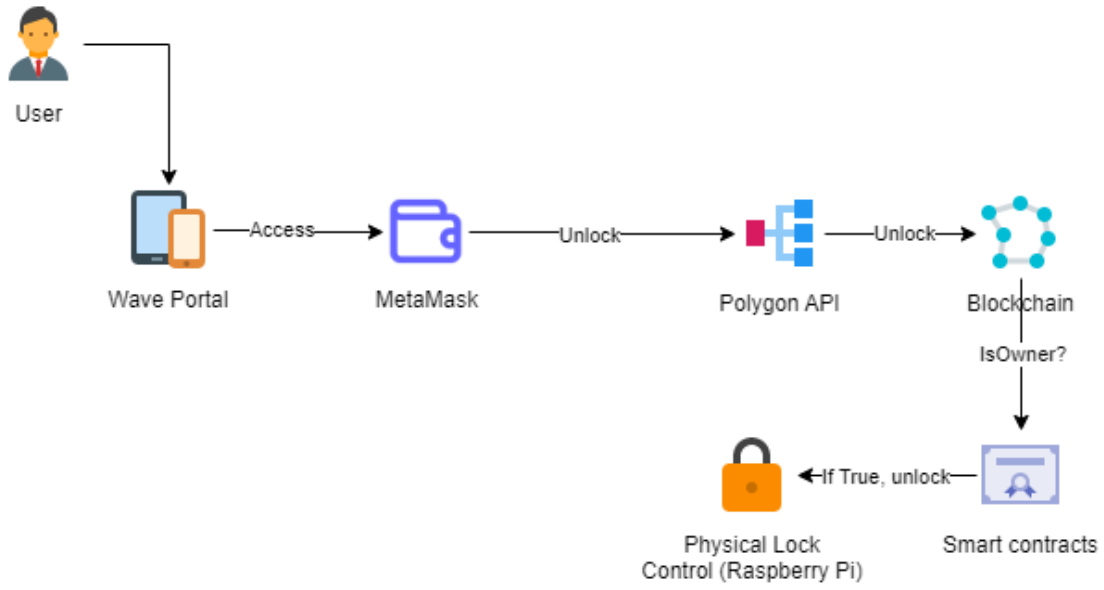
Figures and paragraph below show the process of two most important features on managing and accessing the Smart Lock.

Case 1: Unlock the lock

- User login via the Entry Portal
- Wave Portal invoke Metamask API to connect to User's Ether wallet
- After the verification can connecting process is completed, Wave Portal offer User option to operate the lock

- User send unlock request, Wave Portal invoke Polygon API to access the Polygon Test Net
- Polygon Test Net confirm account validity, call Smart Contract based on User request (Unlock)
- Smart Contract verifies request's authenticity, update lock state variable.
- MCU (Raspberry Pi) reads the change of state variable, unlock via GPIO command.

Figure 3: Process of accessing the lock

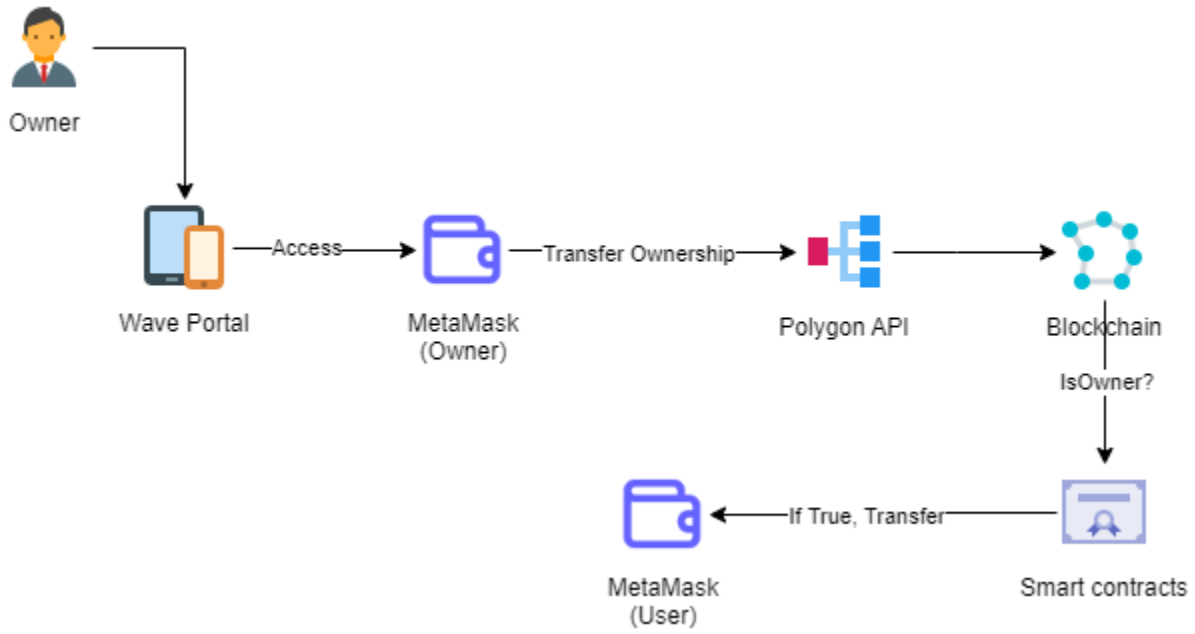


Case 2: Ownership transfer

- User login via the Wave Portal
- Wave Portal invoke Metamask API to connect to User's Ether wallet
- After the verification can connecting process is completed, Wave Portal offer User option to change lock ownership
- User send change lock ownership request, Wave Portal invoke Polygon API to access the Polygon Test Net
- Polygon Test Net confirm account validity and authentication, call Smart Contract to transfer ownership

- Smart Contract initialize transaction process, block begin verification process over all the nodes in the network (transfer)
- Verification process success, transaction and transfer of ownership complete.

Figure 4: Process of transferring ownership



VI. Discussion

A. Aspects of the project which were not implemented

1. NFC Module

One of main concerns when we begin the development of this project is identifying the users. Our original plan was to use a NFC module to identify the user's identity. However, from the information we learned later in the project research, we realized that the NFC method has several security flaws and the user's identity tag can be copied or stolen with ease[7]. Since The Radio Frequency Identification (RFID) technology uses the radio-frequency magnetic fields to identify and track user's identity, any device with capability to read RFID tags can replicate the user's identity token, which invalidates the whole point of developing a secure decentralized system. As a result, we abandoned the design of

the NFC module and decided to use Etherwallet and smart contract to verify the user's identity.

2. **Ethereum Fork**

   At the designing phase of the project, we need a proof of concept platform to deploy our blockchain. Our original idea was to fork the Ethereum mainnet for our testing purpose. But we soon realized that forking even part of the network will result in huge amounts of data which require weeks to complete the fork. Due to the immense high cost of testing on Ethereum, we later transfer our testing branch to Ganache, and later deploy to the Polygon Test Net.

3. **Amazon web services**

   While Amazon Web Services provide an excellent platform for a fully managed service that creates and manages scalable private networks using the popular open-source frameworks Hyperledger Fabric and Ethereum. However, the AWS platform lacks the support for web3 integration. On the other hand, the Polygon solution we choose not only has integrated support for web3 decentralized applications, but also integrated API for Metamask access and Polygon Ethereum blockchain network access, which is much more convenient and well integrated compared to the AWS platform.

4. **Ganache**

   Ganache itself is a useful tool for blockchain based project debugging and developing. However it only serves as a test branch and cannot simulate an actual network that's deployed on the cloud. Hence in the later stage of our project development, we transferred and deployed our Blockchain network on the Polygon Test Net.

5. **Infura**

   Infura provides the tools and infrastructure that allow developers to easily take their blockchain application from testing to scaled deployment. However it

B. Aspects which were implemented but not in original proposal

1. **Polygon Network**

Polygon Test Net is where we deployed our blockchain network for online testing with our web3 application. It enables developers to deploy personal Ethereum blockchain which we use to run tests, execute commands, and inspect state while controlling how the chain operates. It also offers a fictional currency token for debugging our architecture.

2. **Ganache**

   An extremely powerful local blockchain test environment. It offers blockchain for rapid Ethereum and Corda distributed application development. Furthermore, it enables us to develop, deploy, and test your dApps in a safe and deterministic environment.

3. **Remix**

   Remix IDE allows developing, deploying and administering smart contracts for Ethereum like blockchains. It is necessary to enable the deployment and debugging of our smart contracts on the blockchain network.

4. **Hard Hat**

   Hardhat is a development environment to compile, deploy, test, and debug Ethereum software. It helps developers manage and automate the recurring tasks that are inherent to the process of building smart contracts and dApps, as well as easily introducing more functionality around this workflow. This means compiling, running and testing smart contracts.

5. **Repl.it**

   Repl.it is an online frontend IDE, Compiler, Interpreter, and REPL that is easy to use and test our code instantly. It also is seamlessly integrated with GitHub and Visual Studio Code to offer us an integrated environment to develop our user-friendly interface to invoke the Metamask feature.

6. **Metamask**

   MetaMask is an extension for accessing Ethereum enabled distributed applications, essentially a key vault that works as the user’s ethereum wallet. It offers the blockchain an identifier to verify the user’s identity.

VII. Demonstration Issues

We have developed a more complex web3 Dapp via Replit with features to tell how many times a user has locked/unlocked the door as well as a timestamp when the user clicks unlock/lock. However, there is an issue with polygon network deployment to a public testnet and at the end we were unable to test the polygon network with Raspberry Pi to demonstrate the door lock mechanism. For demonstration purposes (video demonstration), we instead use Ganache blockchain and a simpler web3 Dapp developed with nextjs and ethers library to test connecting to the wallet and unlock/lock the door features. Ganache blockchain worked in this scenario because we were able to use an IP address as an RPC server. This essentially means that if raspberry pi is connected to the same wifi as Ganache, it can access and connect to the contract which is deployed on the same network. Please note that our system design is still the same as described throughout the report, except for demonstration purposes we used a different ethereum blockchain because we were unable to deploy a public testnet to test raspberry pi.

Another note for the demonstration is that it can take several seconds for the door to be unlocked because we need to wait until the transaction is mined. Future improvements will be discussed in the next section.

VIII. Conclusion And Future Work

In this project, we proposed and implemented a decentralized Smart Lock System based on Ethereum blockchain. Such systems allow users to have access control and enable users to manage it remotely. Furthermore, our system removed the necessity of a central authority and centralized database, which resulted in avoiding related cybersecurity risks.
During our development, we learned how to develop a user-friendly interface using Replit integrated development environment and distribute our application using a decentralized protocol known as the Web3 protocol, which data is stored in multiple copies of a P2P network. Furthermore, we learned how to use Solidity to write Smart Contracts which are programs that govern the behavior of accounts to implement our designed features into the blockchain architecture. It allows us to finish the authentication, authorization and verification process entirely in a decentralized manner.

Regarding the cost and performance of using the blockchain as our architecture, we tried to lower the cost of operating the lock by implementing public variables and read only functions

to avoid unnecessary transaction and call of function that require to be verified by all the nodes on the network (aka “mined”).

However, this project so far is still in the stage of prototyping. It has basic features like user verification and authentication, but lacks the capability to satisfy time sensitive user requirements. It can improve by implementing more real time verification algorithms such as the use of local buffers and a more optimized verification process. Furthermore, a multiple accounts system can also be implemented by modifying our ownership smart contract to achieve a multi-accounts feature. The hardware of the lock can be also improved by adding features like emergency backup power source and physical emergency uplock mechanism, to offer more options for the users of the smart lock.

In conclusion, this project demonstrated the possibility and feasibility of building a smart lock based on blockchain and smart contract.

References

[1] “Polygon Transaction Hash (Txhash) Details: PolygonScan.” *Polygon (MATIC) Blockchain Explorer*, https://mumbai.polygonscan.com/tx/0xb68e9f07529f660b1940753e05452cf520182c283fd1f91a18473e8adbc7bbf7 [Accessed May 04, 2022]

[2] “WEB3 Development Platform: Blockchain API and Node Service: Ethereum, Polygon, Arbitrum, Optimism, Flow and More Blockchains.” *Alchemy*, https://www.alchemy.com/ [Accessed May 04, 2022]

[3] Romano, Simone. “A Blockchain-Powered Smart-Lock.” Hackster.io, 11 Nov. 2020, https://www.hackster.io/simone-romano/a-blockchain-powered-smart-lock-21fad9 [Accessed May 04, 2022]

[4] “Polygon Faucet.” *Polygon Faucet*, https://faucet.polygon.technology/ . [Accessed May 04, 2022]

[5] OpenZeppelin. “OpenZeppelin/Openzeppelin-Contracts: Library for Secure Smart Contract Development.” *GitHub*, https://github.com/OpenZeppelin/openzeppelin-contracts/ . [Accessed May 04, 2022]

[6] Nuwapa, Lin, Kamran (2022) Project Proposal. https://learn.bu.edu/ultra/courses/_85260_1/cl/outline . [Accessed May 04, 2022]

[7] Chechi, Davinder & Kundu, Twinkle & Kaur, Preet. (2012). THE RFID TECHNOLOGY AND ITS APPLICATIONS: A REVIEW. International Journal of Electronics, Communication & Instrumentation Engineering Research and Development (IJECIERD). 2. 109-120.

[8] “Buildspace.” *Buildspace*, https://buildspace.so/ . [Accessed May 04, 2022]