6-10-2020

# Securely Updating IoT Using Blockchain

Rachael Brooks

# SANTA CLARA UNIVERSITY

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Date: June 10, 2020

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

**Rachael Brooks**

ENTITLED

## Securely Updating IoT Using Blockchain

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Yuhong Liu
Yuhong Liu (Jun 13, 2020 23:08 PDT)

Thesis Advisor

Nam Ling
Nam Ling (Jun 13, 2020 23:10 PDT)

Department Chair

# Securely Updating IoT Using Blockchain

by

Rachael Brooks

**SENIOR DESIGN PROJECT REPORT**

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 10, 2020

# Chapter 1

# Abstract

Because of the extensive growth of IoT devices in the last few years, researchers have been looking for ways to reduce the amount of power they use. One cause for concern is the energy used to communicate with manufacturers when an update has to occur. Our solution to this problem is to move those communications off the device and onto a blockchain, with the owner of the device as an intermediary. The update is handled through the blockchain and is mediated by a smart contract. Overall, the owner pays for the update, the manufacturer passes the encrypted keys for the update through the contract, and the owner downloads the update from a cloud service and installs the software on the device. So far, the data flow works, but the performance on the blockchain is subpar and needs to be adjusted. This is continuing research and will be improved over time.

# Table of Contents

# List of Figures

# Chapter 2

# Introduction

## 2.1 Motivation

IoT devices are becoming integrated into our society more and more every day. From smart home devices, to traffic sensors, to GPS trackers, the utility of IoT is adding functionality to our lives by allowing us to let our devices take responsibility for actions that are time consuming or menial. However, considering the data and functionality that these IoT devices have, it is vital for those devices to be protected and for manufacturers to take security into account. The importance of security was illustrated by the Mirai Botnet attack that happened in 2016, a DDoS attack that took place on IoT devices[1][2]. This attack utilized IoT devices to attack Microsoft game servers, but left much of the U.S. east coast periodically without internet for a full day. While security is clearly essential, IoT devices typically have low processing power and rely on battery power to run, and therefore must sacrifice either security or functionality in order to perform well.

Current solutions to this issue typically involve using cryptographic schemes that have the least amount of overhead while still being effective, such as Elliptic Curve based cryptography. However, as with most public key cryptography, this still requires a lot of computation.

## 2.2 Solution

Our solution to this issue is to move the majority of cryptographic operations off of the device and onto a blockchain. We considered moving operations simply to the owner of the device, but if an owner manages a large number of devices, such as for a university or research project, that might be unreasonable. Therefore, moving the majority of the cryptography to a blockchain provides the same level of security but puts less work on the IoT devices.

Before diving too far into our system, it is important to lay out some of the defining features of blockchain. First, it is decentralized. This means that there is no central body regulating it, which tends to increase trust in the records of the blockchain. There is a group of developers that contribute to the maintenance and upgrades of the blockchain, but those contributors are typically volunteers who do not receive payment for their work or have access to privileged

information. Decentralization also adds to reliability, as there is not a singular point of failure, as there is with a server owned by a company. Second, everything on the blockchain is visible. Every transaction placed on a blockchain is visible to everyone else and can be audited by everyone else. This is generally a good thing as it makes sure all transactions are legitimate and that past information in the blockchain cannot be changed. However, this does lead to concerns about security and malicious users having access to information they shouldn't. Third, smart contracts are described as literal contracts that are implemented through a blockchain for a variety of applications, from video games to verifying diamond supply chains. However, it may make more sense to think of them as programs that are built to work with the blockchain and its transactional system.

Introducing blockchain adds benefits to our system, notability: decentralization, security, and auditability. The benefits of decentralization are discussed in the previous paragraph, but security is a focal point of blockchain that is bolstered by its decentralized property. For our project, by using the security of the blockchain, we reduce the strain on IoT devices while maintaining a secure transfer of information. Auditability has become one of our major benefits. Blockchains have a ledger of all past transactions and events. We use this ledger so that manufacturers, owners, and regulators can verify that an update occurred.

It is important to mention that the assumption made in our original design is that the update is sent to the device over a secure network. Future implementations of this project with the purpose of making it viable for production will need to address this, as it is a dangerous assumption outside of research. As the benefits of our project extend outside of security, and still add an extra layer, we do not see this future addition as a potential devaluation of our project.

This system works by:

1. Having a manufacturer notify an owner of an available update.
2. Once the update is accepted and encrypted, it will be uploaded to a cloud server by the manufacturer where the owner can eventually access it.
3. The owner then goes to the blockchain where the manufacturer and owner will share encrypted keys and verify the update through a transaction specified by a smart contract. In this process, the owner does pay a fee to the manufacturer in exchange for the keys.
4. The owner will then download and decrypt the update and send it to the device.
5. The device will install the update and send a confirmation to the owner, which is then posted on the blockchain.

This system's implementation is discussed in "Blockchain based Owner-Controlled Secure Software Updates for Resource-Constrained IoT" by Gabriel Solomon, Peng Zhang, Yuhong Liu, and Rachael Brooks[3]. We agree that a successful implementation of the project is largely reliant on a successful smart contract executed by a functional blockchain that results in an IoT device receiving a file.

# Chapter 3

# Requirements

Our requirements are split into three sections: Functional, Non-Functional, and Design Constraints.

## 3.1 Functional

”The system will have...”

**Critical Requirements**

- A smart contract that accounts for an encrypted symmetric key, the hash of the update, the link to the update, and a fee for the miners.
- A functional interface for both manufacturers and owners.
- Ability to process an update from request to download to Owner.
- Ability to work on a private blockchain.
- Single-party payment transaction capability

**Recommended**

- Auditing capabilities for both owner and manufacturer.
- Ability to process an update from request to download to IoT Device.
- Authentication for manufacturers.
- Secure message transmission from owner to IoT device.
- Encrypted elements in the website, but not on the blockchain.

**Suggested**

- Capability to work on a public blockchain, specifically Ethereum.
- A decentralized file system (IPFS).

## 3.2 Non-Functional

"The system will be..."

**Critical**

- Secure.
- Of reasonable size for both IoT and blockchain technologies.

**Recommended**

- Well-performing, considering blockchain technologies.
- Able to be uploaded to a public blockchain.
- Simple to use.

**Suggested**

- Well designed, regarding non-functional elements, such as coloration.

## 3.3 Design Constraints

- Must follow original research paper and any updates to that research.
- Must work with Raspberry Pi for testing.

# Chapter 4

# Use Cases

This system has a few basic use cases.

The preconditions are:

- The devices are registered by both the manufacturer and the owner.
- The smart contract is posted to the blockchain
- The manufacturer has been verified on the blockchain.
- The owner and manufacturer have accounts with the system.

The postconditions are:

- The IoT device has been updated, if all conditions are met.
- The manufacturer and owner know what the outcome was, so it can be redone if needed.
- All lists of devices reflect the change

Initially in Figure 4.1, both actors, the manufacturer and the owner, would have accounts with the system. The manufacturer should register built devices with the system that are to be sold, and the owner will have registered their purchased devices as well. This use case diagram does not account for errors that may occur in the system, or the case where the manufacturer or owner needs to restart a transaction (for example if the smart contract isn't fulfilled in a 3 week period), which is set by the Ethereum blockchain. The end goal is to have the devices successfully and securely updated with the software provided by the manufacturer.

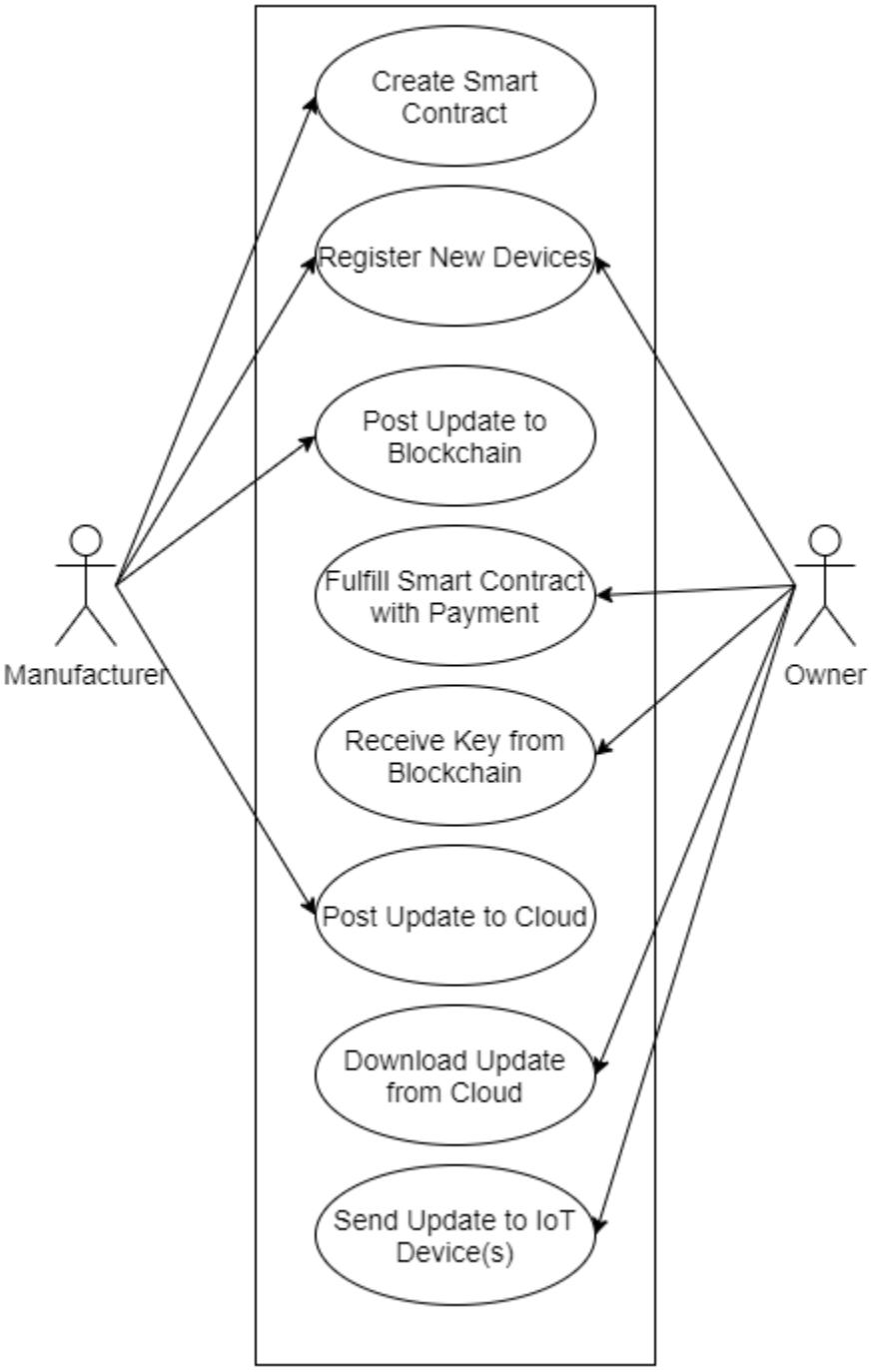


Figure 4.1: Use Case Diagram

# Chapter 5

# Activity Diagrams

The two diagrams in this section represent the activities manufacturers and owners can take in our system. Importantly, there is choice represented in this diagram as owners have the opportunity to deny updates. However, our initial implementation did not fully incorporate this because we were primarily focused on the process being fulfilled completely when accepted.

Figure 5.1 is of the Manufacturer Activity Diagram. In a finalized system, the manufacturer would not remake the devices on the blockchain every time the system was accessed, but in our scaled down version, on a private blockchain, it does need to happen. Then, the manufacturer can notify the owner of a new update. Should the owner accept, the manufacturer prepares the updates and keys and waits for the owner to fulfill the contract. If the owner does, then the keys are sent and the owner can access the updates.

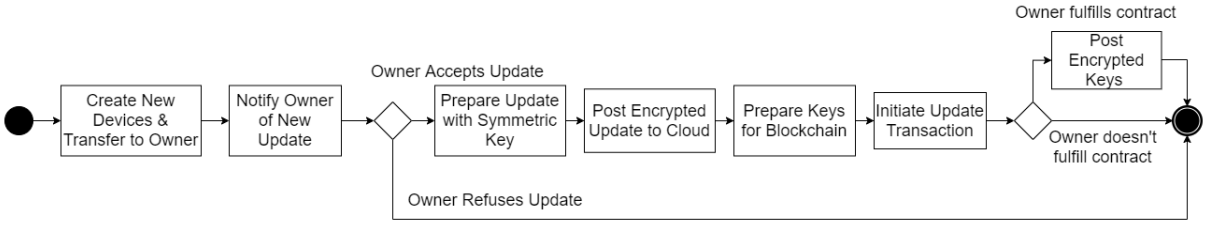


Figure 5.1: Manufacturer Activity Diagram

Figure 5.2 mirrors Figure 5.1 from the owner's point of view, taking much the same steps in accepting and fulfilling updates, but ending with sending the updates to the devices and performing a final confirmation with the blockchain.

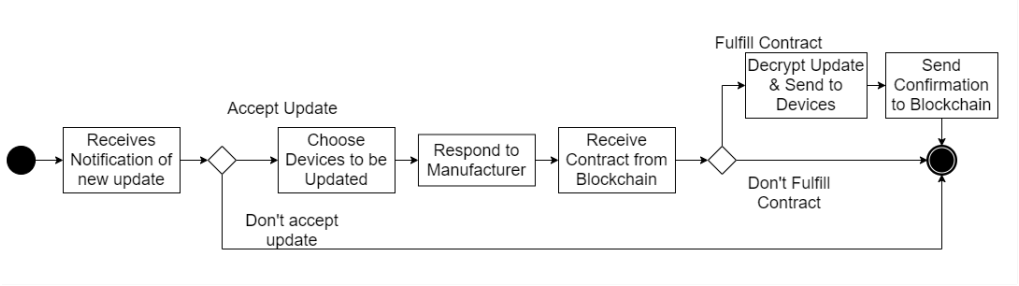


Figure 5.2: Owner Activity Diagram

# Chapter 6

# Architectural Diagram

The architecture for our project diagrammed in Figure 6.1 is a Service Oriented Architecture (SoA) with three services, a system, and a connection to the cloud. As far as my research team has seen, there are not standard architectural diagrams designed for blockchain technologies. Blockchain adds complexity because of its decentralization. Several architectures that seems to have potential for our project, such as a cli ent-server model, were rejected as the decentralized nature of blockchain kept them from being good fits. Eventually, we decided upon SoA because of its ability to have all the elements in ”black boxes”. SoA worked with our requirements of decentralization and keeping the method simple. However, I would happily reconsider the classification of our system with the introduction of architectures that include blockchain technology, as the parts still have to communicate more than is probably reflected by this model. For now, when a transaction takes place, the system provides different services in sending the final update to devices, auditing for owners and manufacturers, posting and downloading updates from the cloud, and communicating with the blockchain.

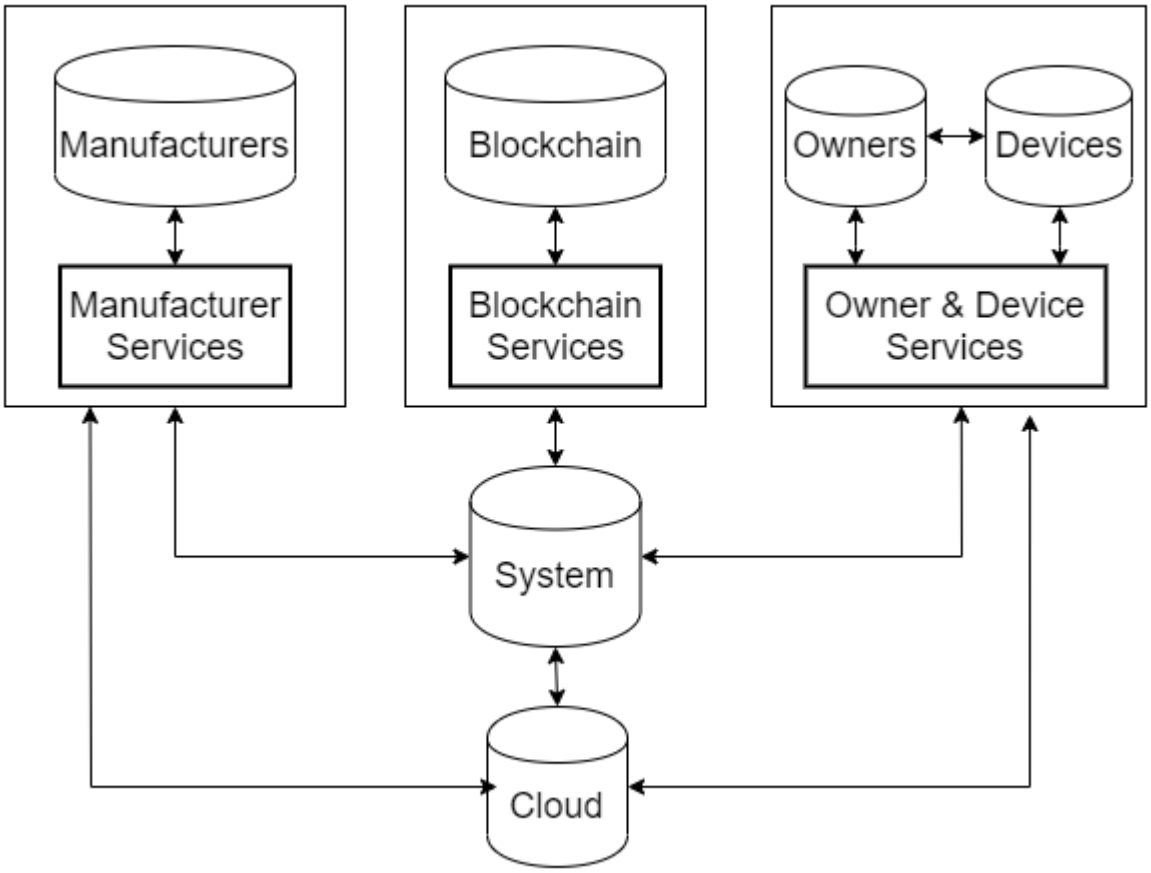


Figure 6.1: Architectural Diagram

# Chapter 7

# Technologies Used

The technologies we used are:

- Truffle Suite
- Ganache
- Metamask
- IPFS
- Languages: Solidity, JavaScript, Python
- Ethereum and Ropsten Testnet
- Raspberry Pi

# Chapter 8

# Design Rationale

The following is a list of the technologies used and why they are used.

- Truffle Suite: This is an open source software that provides a basic setup for smart contracts to be uploaded to Ethereum. The software also includes tutorials on how to build and upload a smart contract and information on how to develop it. We chose to use this software because we are relatively inexperienced with developing smart contracts, and this has a reputation of being a good place to start.
- Ganache: Ganache is a part of Truffle Suite, but it simulates a blockchain by building a virtual one on your computer. This is not ideal for testing errors that may come from what individual miners might do, but it does allow a platform to test our smart contract and prove that it works before uploading it to Ethereum or Ropsten. It could also be used as a presentation feature for a live demo at the Senior Design Conference.
- Metamask: This is a cryptocurrency wallet that is recommended to use by Truffle and can be configured to be used with a private blockchain. Ideally it would also be used for testing on the public blockchain as well.
- IPFS: This is a decentralized file storage system that we used for cloud storage. We were not able to fully implement it because of cost and limitations, but this is not an issue as we were able to use it for testing and decentralized storage was not the focal point of this project.
- Lanugages: Solidity is a newer language based on C++ and JavaScript that is used to build smart contracts. JavaScript is used to build the interface for the smart contract and private blockchain. Python and Arduino are used for programming the devices used for testing.
- Ethereum and Ropsten Testnet: Ethereum is one of the major blockchains that is known for being able to add smart contracts and therefore have customized transactions. Ropsten Testnet is a testing blockchain that is designed so developers can test their smart contracts with real miners and customers before deploying to Ethereum. This is critical because items that are posted on Ethereum are immutable, so most errors cannot be fixed later.

- Raspberry Pi and Arduino: The goal with using one or both of these devices is for testing and to download an update onto the device at the completion of this project.

# Chapter 9

# System Description

The system we implemented required few changes from our initial plans because of errors discovered in the original paper and constraints on time and cost that prevented us from having as robust of a system. Though there are some aspects that still need to be incorporated, the ones that were critical for my Senior Design Project and my team's research were successfully done.

## 9.1 Smart Contracts

The system is built around three smart contracts: DeviceCreate, IoTUpdate, and DeviceOwner. These contracts do utilize some open source contracts, namely Ownable, which allows certain functions in the contracts to be performed by only the owner. Each contract has a different main function, but they all serve to exchange keys on the blockchain. Though we hope that this system could be implemented on different blockchains in the future, Ethereum is the optimal blockchain to use for because its smart contract system is the most advanced, and it is where we did most of our work.

First, DeviceCreate registers devices with the contracts and assigns them an index position in the device's array. Each device is stored as a structure, with unique identifying information like a serial number or device type. Storing all of the devices on the blockchain does cause concern for storage issues, but all devices are assigned an index at time of creation that should be accessible in O(1) time with indexing. Each device is also mapped to an owner, which is represented by their address on the blockchain, through a hash map. Apart from creating the device, this contract largely has support functions, such as getting the owner of a device, getting the number of devices an owner has, and getting the information stored in the device structure.

Second, DeviceOwner manages transferring the ownership of the device between owners. This is largely based on the contract ERC721.sol, which is also open source. While these transfers of ownership do not currently require manufacturer approval, the contract could be adjusted so the manufacturer approves any transfers of items.

Third and most critical, IoTUpdate manages the exchange of keys between a manufacturer and an owner. Currently, the terms of the update are assumed to be negotiated off the chain. When both parties agree, the manufacturer initiates

an update for a device using a structure. Information like the current version is put into the structure with no other information. The owner then pays for the update, and in return they receive encrypted and signed information to access the update through a link, and the proper keys to decrypt it. At this point, the owner reports to the blockchain that they have received the update and sends the update to the device. Upon completion of the update transactions, the contract releases an event on the blockchain, which can be searched for and found at a later time, with the encrypted information about the update so that manufacturers and owners could audit the information later. For each update that occurs, the information from the previous update is written over, but because events have been released on the blockchain with the necessary information, no data should be lost.

The largest security concerns come from anyone being able to call functions on any device, the owner being able to gather information before they should, and the visibility of a version that a particular device may have. The issue of function calls is taken care of with modifiers that require a callee to be the manufacturer or own the device being updated and verify the update is at the correct state in the transaction. These requirements prevent spoofing, as no one can pretend to own a device they do not or be a manufacturer, which could cause malicious users to send malware or prevent users from getting their updates, and backdoor attacks, because even if the owner is legitimate, it is important that they cannot access the keys to decrypt the software update until they have paid for it.

## 9.2 IPFS

Because storage on the blockchain can be costly, we decided to keep the encrypted update on a cloud service to save on costs. Initially, we were planning to use a simpler storage service, like Google Drive, to run our system. However, we were told about IPFS during the Peer Review process and decided to look into and, eventually, implement it. The primary benefit of using IPFS with our system is that it is a decentralized cloud storage service. So by using IPFS, we were able to keep the decentralized property of our project. This is not necessary for our project to work, but decentralization is one of the major appeals of blockchain, so keeping it is beneficial to our research and the future of the project.

## 9.3 Interface

My partner Gabriel Solomon created the Interface for the website using Web3js and React. Web3js is a JavaScript derivative that is specifically designed to work with blockchains and smart contracts. The interface follows the framework laid out in our original paper [3], along with some adjustments made to the order of actions made for security concerns, and has allowed for testing and debugging. There are three tabs on the interface: Manufacturer, Owner, and Administrator. The Manufacturer tab allows you to choose an ”update” from a list and begin a transaction with the Owner by hitting the upload button. On the Owner tab, once a transaction is completed through the smart contracts

and the cryptocurrency wallet, MetaMask, the owner can download it to the device. The Admin tab is designed for debugging and adjusting settings for demos and testing. Clearly, this is not a finalized version, as all three tabs are available to any user that runs it. However, it works for our primary purpose of implementing the smart contracts to know they work and how they could be improved from our original research.

# Chapter 10

# Testing

Our testing process has, for the most part, consisted of unit and integration testing. However, testing for known security issues and the cost of operations have been happening as well.

Unit and integration tests have been happening to make sure everything is functional. While we are happy with how everything has turned out, there certainly needs to be more tests to find bugs that may not have been realized, and add in more fail-safes for if those bugs come to pass.

We have also been able to test our code through a service called *Securify*. This service analyzes code in smart contracts and delivers a report if any known issues or potential attacks is found. We have used the information from this report to fix a few issues that have come up and will likely use this service again, as the project continues to develop in the coming months.

Finally, we tested the performance of our smart contracts. While all of them have a typical time complexity of O(1), the cost of storage means that the gas cost for an update is around 280,000, or roughly $1.12 per update. Considering the large number of IoT devices that a company may own, this is much too expensive. One update for 10,000 devices would cost a $11,200 between the owners and manufacturers. Consider further how often updates occur on devices and how many devices an owner may have, and the unreasonable side of this becomes apparent. Future work for our project does involve implementing optimizations and continuing testing new optimizations to keep costs reasonable.

# Chapter 11

# Difficulties Encountered

While we were able to get to a comfortable point in our project, there were several difficulties that we encountered.

Obviously, the shelter-in-place orders for COVID-19 disrupted our flow. Fortunately, any in-person aspects of our project had been completed prior to the Shelter in Place order, but there was an adjustment period of a few weeks as we transitioned to remote work.

We also encountered difficulties with a mismatch between our parts. After our paper had been published and we started implementation, one of us discovered a bug in our original paper that would have allowed the owners access to the encrypted keys before making a payment. A miscommunication occurred when discussing the bug, so the data flow of the interface did not match the data flow of the smart contracts. This led to integration taking much longer than it should have.

Another contributor to the previous obstacle was the approach we took to the project, where one party largely focused on the smart contracts, while the other focused on the wider data flow and interface. Because we worked separately and brought our pieces together, rather than more diligently integrating our pieces during development, that also contributed to the extended integration period.

Finally, I would say a large difficulty for me, as the undergraduate student on this project, was learning to balance my responsibilities and prioritize this project outside of a regular class or work setting.

While these difficulties certainly impacted our project, good planning and consistent communication meant that we were able to finish the core aspects of our project in time.

# Chapter 12

# Suggested Changes

This system continues to be a work in progress. First, we would recommend making a change to the notification system. We would have used a different method if it was critical for our research, but as it was not, we just used smart contracts. This is incredibly expensive and inefficient, but it works as a temporary fix.

Second, we would adjust the optimizations, especially for scalability. While our code is functional, it is not efficient. We are currently spending far too much on storage costs, as there is a price to store and edit data on the blockchain. On a similar note, we are looking at ways to increase scalability by reducing the number of transactions that need to occur. For example, in a university with thousands of key card readers, we would want to be able to update those readers with a minimum number of transactions (ideally, one transaction), while still providing the user a choice of which devices to update and when. Current proposals for this involve using another section of the update or the encrypted message to indicate which devices should be updated.

Third, a few critical components for security still need to be implemented and reviewed. Primarily, we are in the process of adding encryption to our code so that we can have the necessary security. Other security issues being addressed are ensuring the final owner confirmation proves that the update was sent to the devices, adjusting the update flow so that transactions can be aborted or disputed in multiple places, and making sure that any device transfers between owners are desired.

# Chapter 13

# Lessons Learned

There were three main takeaways from this project that will inform how I approach future endeavors.

## 13.1 Communicate Early and Often

Though COVID-19 was a chaotic event, fortunately, my partner and I started working on this project in October after months of design work. As a result, we had made significant progress and had acquired all of our materials before quarantine took place. We had also planned the majority of the steps early on. Some parts of the project certainly took longer than they should have, but knowing what needed to happen early meant that we were more prepared for the obstacles that met us in March.

## 13.2 Prioritize Working With Partners

Although my partner and I kept up regular communication and held weekly meetings, miscommunications still occurred and we failed to inquire about important work the other person was doing, so our integration and implementation time took longer than it should. Specifically, communicating more clearly about a security issue and how the interface worked in January likely would have meant that the integration would have taken less time. For the future, I need to allocate more time to understanding the work any team members are doing and strive to explain the details of my own work better.

## 13.3 Document Everything

I was not particularly good at this, but while completing this write up, and rediscovering old bugs in the system, the importance of documentation has been enforced. My partner was fantastic at this, and the documentation he did has been useful many times. This is a skill I still need to develop, but I'm glad I picked up tips from my partner on documentation and version control, which I had not extensively used prior to this project.

# Chapter 14

# Societal Issues

## 14.1 Ethical

The primary ethical question our project raised related to paying for updates. The concern lies in the cost, specifically for security updates: would people refuse updates if they require payment? How damaging could this be? Is it reasonable to have a method of updating that can only be done through requiring payments? These are the primary questions that I have grappled with, and frankly, I am unsure what the answer is. There is potential that, if it is severe enough, a company may send the cost of the update to the owner in a transaction, so the owner would not have to pay. But, would a company take that on? Part of the reason we decided to target larger organizations was that they should have the funds to pay for necessary updates, and be less deterred by costs if the update was critical enough. I still don't have the answer for a lot of these questions, but I think posing our solution, so that it can be improved upon, is a step in the right direction to explore answers.

## 14.2 Social

While there is some potential for social consequences because we charge for updates through our system, and this might be an unprecedented cost, our target consumers were not intended to be individual people and homes, but larger institutions, such as universities and office buildings. These consumers would have many devices that require updates (key card readers, smart lightbulbs, etc.), and have more need for an auditable updates. As such, the cost of an update would be worth it, and if scalable, the cost could decrease.

## 14.3 Political

This project is not inherently political, but there has been interest from a regulatory standpoint, as this project allows for the auditing of updates. Specifically, we have received interest in this project from employees at car companies who need to prove that updates for vehicles have occurred, which can be proven through the blockchain ledger.

## 14.4 Economic

The economic implications of our project was not taken lightly. We have been discussing the ethics and implications of requiring a charge for updates, as the cost of gas on an update is still a concern, even if the manufacturer charges nothing. As such, we are working on how to optimize our code while still remaining secure, so that we can keep costs down and provide guidelines to maintain those optimizations.

## 14.5 Health and Safety

While I doubt this project impacts health, I would argue that it could have an impact on safety, specifically security of information. I do not believe our current model is the end result of our research, nor is its current conception as secure as it should be, should it be commercialized, as the assumption of a secure home network is a glaring risk. However, as we continue to do research and consider methods to improve and advance our current design, I believe this research could be a positive benefit to computer security by providing and inspiring new methods to address issues of power and security because using blockchain, a notoriously resource intensive service, is a counter-intuitive idea for addressing resource constraints for IoT devices.

## 14.6 Manufacturability

As the primary goal of this project was not manufacturability and it should not be manufactured because of the security risks in our assumption, the lack of notification service, and the necessity for more thorough testing, that does not mean that it could never happen or that it is not viable. It would not be unreasonable for our software to be manufactured, though important changes to security and the contracts that have been designed need to reach a deeper level of security and resistance to bugs and threats. Furthermore, I am sure that there are aspects of the design that could be improved for the purpose of manufacturability, specifically improvements to scalability, that would make the product more appealing.

## 14.7 Sustainability

Once this project is fully completed, I believe that it will be sustainable, to an extent. We are attempting optimizations to limit the blockchain resources used by our contracts, but as the world learns more about blockchain technology, the technology changes, not only in how the blockchain operates (proof of work to proof of stake[4], adding storage rent[5]), but also in the security risks that arise. If we can make our contracts as secure as possible, then they will last as long as their security does. However, there may be a need to republish updated contracts with changes to the blockchain, as contracts are immutable. We have taken precautions with this and allowed for some ability to change, but it is still a concern for our contracts, and all contracts, on blockchains.

## 14.8 Environmental Impact

Blockchain, especially in the Proof of Work model, is not good for the environment. It is estimated that cryptocurrency mining consumes more energy than Switzerland [6]. While Ethereum has been working to move to a Proof of Stake model, which is expected to reduce the energy usage by a significant amount, it has not occurred yet. Furthermore, our project is fairly resource intensive, requiring a decent amount of storage and rewrites to that storage during updates. We are working on optimizations to reduce this storage usage in the context of Ethereum, and hopefully keep our environmental impact minimal. However, if another blockchain arrives and offers a system for smart contracts as robust as Ethereum's with less energy usage, we may consider switching to that option for the environmental benefit.

## 14.9 Usability

For the most part, I would say our system is fairly usable. I would not recommend it so much for an average user in its current state, but an IT team could set it up and average users should be able to interact with it. However, if a company would like certain customizations to their contracts or methods, it should be done by someone with experience in these areas, as security and bugs are a major concern.

## 14.10 Lifelong Learning

Because this project has been based on a research paper I co-authored, it has been a central part of learning about the academic and research processes, essentially a definition of lifelong learning. But apart from the research, I have learned a lot about learning on my own. I spent many, many hours researching specific details about blockchain and smart contract implementation to figure out how to write, test, debug, and implement my code.

## 14.11 Compassion

It is hard to think of this project in terms of compassion. Many of the projects I saw my peers work on attempted to solve problems for communities in need, or create something to help a specific person or group. However, my project's compassion is several steps removed from the core of helping to relieve others' suffering. We are certainly working to address a problem in the world, the power consumption of IoT devices in communications and updates, but it is hard to pinpoint how it addresses human suffering.

# Chapter 15

# Conclusion

My major contribution to this research was to develop the smart contracts and integrate them into the interface. As a result of this primary contribution, I also assisted with catching errors in our original design, researching blockchain to support everyone in the group, and assisting with developing papers for research and patents. My partner build the interface and ensured that the data flow surrounding the smart contracts was set up. This involved building the website, setting up IPFS, and setting up the devices for our demo. Combined, we built a system that can transfer files from one entity to another in a decentralized, secure, and auditable way.

From this project, we have learned that our method of sending updates using blockchain technology needs work. Although our method is mostly secure, there are changes that need to be made to our system in order for it to be commercialized or expanded. This is especially important for the optimizations of data storage, as our performance is too large to consider being widely usable. However, while our project still needs a lot of work, I am hopeful that it can be developed to where it can be reasonably used and implemented outside of research. In general, however, I can say that I learned a lot about the details of smart contract implementation and some blockchain properties. I still have plenty to learn, but that knowledge has improved our project and prevented risks that we had not noticed until we explored the parameters and properties of blockchain more deeply.

The advantages of our work is that it will be decentralized, secure, and auditable. These are properties that can certainly be appealing to companies and groups. However, our project does require a payment for an update to occur. This clearly has negatives in cases of urgent security updates, and workarounds, such as increasing the number of device updates associated with a transaction to reduce overall cost, need to be developed for these cases.

For the future, we need to ensure that data passing between devices and owners is encrypted at some level. A less intensive symmetric key encryption and decryption process may take care of this, but we need to research this field and find one that can be applied. Other aspects of our full implementation need to take place as well, such as allowing owners to refuse updates and adding optimizations for data storage on the blockchain. As a distant goal, we should also prepare to reevaluate our system if storage rent on blockchains becomes more widespread. While a lot of

work still needs to happen on this system, the progress that has been made is promising, and we hope that our further research into this project allows us to build up our system and we can continue to improve upon it.

# Bibliography


[1] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[2] Newman. What we know about friday's massive east coast internet outage, October 2016. URL https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/.

[3] Liu Y. Solomon G.J, Zhang P. and Brooks R. Blockchain based owner-controlled secure software updates for resource-constrained iot. *NSS 2019*, 11928(9):371–386, December 2019. ISSN 0001-0782. doi: 10.1007/978-3-030-36938-5_22. URL https://doi.org/10.1007/978-3-030-36938-5_22.

[4] Jeffery Hancock. What happens after ethereum goes switches on the proof-of-stake algorithm?, February 2020. URL https://medium.com/ethexbet/what-happens-after-ethereum-goes-switches-on-the-proof-of-stake-algorithm-4b1ac9eb512e.

[5] Ivan Kamakin. Ethereum storage rent: What should we expect?, March 2019. URL https://medium.com/going-byzantine/ethereum-storage-rent-a2b89e96bd35.

[6] Chris Baraniuk. Bitcoin's energy consumption 'equals that of switzerland', July 2019. URL https://www.bbc.com/news/technology-48853230#:~:text=Bitcoin%20uses%20as%20much%20energy,the%20University%20of%20Cambridge%20shows.&text=Currently%2C%20the%20tool%20estimates%20that,0.21%25%20of%20the%20world's%20supply.

# Appendix A

# Installation Guide

1. Install Truffle and Ganache. Instructions can be found on the Truffle Website: https://www.trufflesuite.com/. You will also need to install Metamask, though instructions for set up are on Truffle's website at: https://www.trufflesuite.com/docs/truffle/getting-started/truffle-with-metamask. If you do not have npm already installed, do so before proceeding.

2. There are other libraries and dependencies to install before usage. They are:

```
npm install react-scripts@latest
npm install web3-js
npm install typeface-roboto --save
npm install @material-ui/core
npm install @material-ui/icons
npm install ipfs
npm install js-ipfs
pip install Pillow
run create\_uploads.py python script
```

3. Download the project. There are several files that require paths to the location of the project, all of which should be in the folder "main"

   - In folder "server": manufacturer_file_io.js, manufacturer_ipfs.js owner_raspberry_pi.js
   - In folder "create_uploads": create_uploads.py
   - If you have set up Raspberry Pis and Twitter Bots, in "raspberrypi_client twitterbot" in twitterbot.py, change the host-name to accommodate your computer and path. However, even without Raspberry Pis set up as Twitter bots, you should be able to upload a file and download it to a different folder.

4. Once downloaded and all paths are corrected, Go to main → client → src → App.js. This file contains several configurations for running the code. Primarily, useTestSmartContracts, useTestWithLessManufacturer, useTest-

WithLessDevice, useCrypto, which are consecutive starting on line 100. If you would like to test the code using the full smart contracts. For now it is recommended to have these settings to: false, true, true, false, respectively.

At this point, you should be able to run the project.

# Appendix B

# User Manual

This section holds the user manual, Installation goes through the Installation Guide.

At this point, you should be able to run the project. Start by compiling the contracts. Go to Terminal (we were on Windows and used PowerShell) and enter the ”main” directory in the project. Run:

```
truffle compile
```

There should not be any errors. At this point, possibly while the contracts are compiling, open Ganache and, for now, hit ”Quickstart”. Then, go to ”Settings”, which is a Gear button. Hit the button that says ”ADD PROJECT”, and then click truffle-config.js in the main directory. Hit Restart. This will allow you to view the transactions made with specific contracts.

Once Ganache has been set up and the smart contracts have compiled successfully, run:

```
truffle migrate
```

This will move the contracts onto the private blockchain that you started running with Ganache. It will take the ether from the account 0 to do this.

Once migrations have finished successfully, you can set up your Metamask wallet. We used a Google Chrome extension and will be offering instructions based on that. You will need to create an account and password, however, you will have a seed phase available on Ganache that Metamask can connect to. It is under Mnemonic and may be different on different computers. Set up your account on Metamask and set it up to work on localhost network: ”HTTP://127.0.0.1:7545”.

IMPORTANT: This account and seed phrase WILL NOT work on the actual Ethereum blockchain, so take care when conducting transactions on Ethereum and on the private blockchain.

Now that all has been set up, you should be able to start running the program. In terminal, run:

```
cd client
run dapp-start.ps1
```

At this point, the program should load and the first contracts to create devices should show up. If you would like to upload a file, hit the upload button on the manufacturer tab, and to download hit the download button on the desired

version on the owner tab. You should see a downloads folder in main that contains any ”updates” you have uploaded and subsequently downloaded.