Vietnam National University Ho Chi Minh City
Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

_____ * _____

GRADUATION THESIS

Implementation of Publish-Subscribe Protocol for IoT Data Sharing Management using Re-Encryption and Blockchain

Major: Computer Engineering

Committee : **Computer Engineering**
Supervisor : **Pham Hoang Anh, Ph.D**
Reviewer : **Pham Quoc Cuong, Ph.D**

Student 1 : **Duong Phi Long - 1811039**
Student 2 : **Nguyen Minh Phuc - 1852666**

Ho Chi Minh City, May 2022

ĐẠI HỌC QUỐC GIA TP.HCM

TRƯỜNG ĐẠI HỌC BÁCH KHOA

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

KHOA:KH & KT Máy tính____
BỘ MÔN: KT MÁY TÍNH _____

NHIỆM VỤ LUẬN ÁN TỐT NGHIỆP

Chú ý: Sinh viên phải dán tờ này vào trang nhất của bản thuyết trình

HỌ VÀ TÊN: DƯƠNG PHI LONG _____ MSSV: 1811039 _____
HỌ VÀ TÊN: NGUYỄN MINH PHÚC_____ MSSV: 1852666 _____
HỌ VÀ TÊN: _____ MSSV: _____
NGÀNH: KỸ THUẬT MÁY TÍNH _____ LỚP:_____

1. Đầu đề luận án:

Implemention of Publish-Subscribe Protocol for IoT Data Sharing Management using Re-Encryption and Blockchain

2. Nhiệm vụ (yêu cầu về nội dung và số liệu ban đầu):

- Investigate the existing solutions for IoT access control and data sharing models
- Study the related works such as Blockchain technology, re-encryption technique, pub/sub protocol
- Design and implement the proposed solutions
- Conduct experiments for performance evaluation

3. Ngày giao nhiệm vụ luận án: 10/01/2022

4. Ngày hoàn thành nhiệm vụ: 03/06/2022

5. Họ tên giảng viên hướng dẫn: **Phần hướng dẫn:**

1) Phạm Hoàng Anh 100 %

2)

Nội dung và yêu cầu LVTN đã được thông qua Bộ môn.

Ngày ..... tháng ..... năm .....

CHỦ NHIỆM BỘ MÔN **GIẢNG VIÊN HƯỚNG DẪN CHÍNH**
(Ký và ghi rõ họ tên) *(Ký và ghi rõ họ tên)*

Phạm Quốc Cường **Phạm Hoàng Anh**

PHẦN DÀNH CHO KHOA, BỘ MÔN:
Người duyệt (chấm sơ bộ):_____
Đơn vị: _____
Ngày bảo vệ: _____
Điểm tổng kết: _____
Nơi lưu trữ luận án: _____

TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KH & KT MÁY TÍNH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Ngày 25 tháng 05 năm 2022

PHIẾU CHẤM BẢO VỆ LVTN

(Dành cho người hướng dẫn)

1 Họ và tên SV: **Dương Phi Long, Nguyễn Minh Phúc**
MSSV: **1811039, 1852666** Ngành (chuyên ngành): Computer Engineering

2. Đề tài: **Implemention of Publish-Subscribe Protocol for IoT Data Sharing Management using Re-Encryption and Blockchain**

3. Họ tên người hướng dẫn: **Phạm Hoàng Anh**

4. Tổng quát về bản thuyết minh:
Số trang: Số chương:
Số bảng số liệu Số hình vẽ:
Số tài liệu tham khảo: Phần mềm tính toán:
Hiện vật (sản phẩm)

5. Tổng quát về các bản vẽ:
- Số bản vẽ: Bản A1: Bản A2: Khổ khác:
- Số bản vẽ vẽ tay Số bản vẽ trên máy tính:

6. Những ưu điểm chính của LVTN:
- Students have completed most of the requirements of a capstone project, including:
  - Investigate the related works to find the things for improvement
  - Propose a secure publish-subscribe protocol for data sharing management using Blockchain and reencryption.
  - Implement a prototype to demonstrate the proposed approach in IoT context.
  - Conduct experiments to evaluate the implementation
- The report is well written but students should revise the typos and correct the citation.

7. Những thiếu sót chính của LVTN:
- Students should add more discussion why you choose Ethereum Blockchain for implementation.

8. Đề nghị: Được bảo vệ ☐ Bổ sung thêm để bảo vệ ☐ Không được bảo vệ ☐
9. 3 câu hỏi SV phải trả lời trước Hội đồng:
a.
b.
c.

10. Đánh giá chung (bằng chữ: giỏi, khá, TB): **Điểm: 9.2/10**

Ký tên (ghi rõ họ tên)

Phạm Hoàng Anh

TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KH & KT MÁY TÍNH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

Ngày 28 tháng 5 năm 2022

PHIẾU CHẤM BẢO VỆ LVTN

(Dành cho người ~~hướng dẫn~~/phản biện)

1. Họ và tên SV:

Dương Phi Long	MSSV: 1811039	Ngành: Kỹ thuật Máy tính
Nguyễn Minh Phúc	MSSV: 1852666	Ngành: Kỹ thuật Máy tính

2. Đề tài: Implementation of Publish-Subscribe Protocol for IoT Data Sharing Management using Re-Encryption and Blockchain

3. Họ tên người ~~hướng dẫn~~/phản biện: Phạm Quốc Cường

4. Tổng quát về bản thuyết minh:

Số trang:	Số chương:
Số bảng số liệu	Số hình vẽ:
Số tài liệu tham khảo:	Phần mềm tính toán:
Hiện vật (sản phẩm)	

5. Tổng quát về các bản vẽ:

- Số bản vẽ:	Bản A1:	Bản A2:	Khổ khác:
- Số bản vẽ vẽ tay		Số bản vẽ trên máy tính:	

6. Những ưu điểm chính của LVTN:

- Students fulfill most requirements of the capstone project;
- Students proposed a suitable model to share IoT data using the publish-subscribe protocol with an asymmetric encryption method
- The thesis is well written and organized although there exist some typos and grammatical errors.

7. Những thiếu sót chính của LVTN:

- The proposed and developed system is not feasible for most users especially government-related organizations due to illegalized cryptography currency.
- Students have not analyzed well the characteristics of IoT data sharing when designing and implementing the system.

8. Đề nghị: Được bảo vệ □ Bổ sung thêm để bảo vệ O Không được bảo vệ O

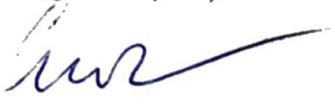
9. 3 câu hỏi SV phải trả lời trước Hội đồng:

- It seems that the proposed system can be used for any data sharing applications, what is the key point for IoT data management in the system?

- Many data sharing and management applications like Netflix, and FPTPlay... have been deployed and operated properly. What are the advantages and disadvantages of your proposed system compared to theirs?

10. Đánh giá chung (bằng chữ: giỏi, khá, TB): Good Điểm: 9/10

Ký tên (ghi rõ họ tên)



Phạm Quốc Cường

Commitment

The team commits that everything presented in the report and the product are the work of the team - except for cited references as well as supporting libraries by the manufacturer itself, the open-source community provided. All the content is not copied from any other copyrighted source. If there exist any evidences that go against our group statements, the team will take all responsibility before the Dean of the Faculty and the School Board of Directors.

Acknowledgement

First and foremost, we would like to express heartfelt gratitude to everyone who has always supported and assisted the group during the final period of thesis work. provided invaluable feedback on my analysis and framing, at times responding to emails late at night and early in the morning.

Especially, we extremely thankful to our supervisors Ph.D Pham Hoang Anh, who revise our thesis numerous time, provided invaluable feedback on our design and thesis, at times responding to emails late at night and early in the morning. He also set up all of the necessary conditions for the group to realize their ideas, be creative, and reach their full potential without fear of impediments.

The group would also like to express its gratitude to all of the faculty members of the Faculty of Computer Science and Engineering, as well as all of the faculty members of Ho Chi Minh City University of Technology in general, for their dedication to teaching and assisting the group in acquiring the knowledge and skills required. Show the group how much fun the lessons are so they may keep pursuing their passions and achieving the group's goals.

The group would also like to thank all the teachers in the Faculty of Computer Science and Engineering in particular and the teachers of Ho Chi Minh City University of Technology in general, for their dedicated teaching and helping the group to grasp the knowledge and skills needed by the group. Show the group the excitement in the lessons so that they can continue to pursue their passions and conquer the group's ambitions.

Abstract

The industrial revolution 4.0 is bringing amazing changes to the globe, particularly in areas like the internet of things, machine learning, and real-time data. Over the last two decades, the Internet of Things has grown at an incredible rate. The amount of data created by sensors is exceedingly enormous and sensitive, hence, posing challenges in terms of data storage, management, and sharing. Because most IoT devices have weak processing capabilities and are prone to assaults owing to their tiny size, and are designed to focus on a certain task. Solving the aforementioned problem using standard storage, administration, and sharing techniques by a centralized approach is problematic. Blockchain, another achievement of the industrial revolution 4.0 in terms of security, traceability, immutability, and decentralized model emerged as a solution to the aforementioned dilemma.

In this study, our research team aims to demonstrate the feasibility of developing an IoT data sharing management system utilizing the publish-subscribe protocol to combine IoT data with blockchain. In the process of data exchange, the system has maintained integrity, security, and fairness. The system also offers a web app with a simple, attractive UI. When transferring data between the aggregator and the data consumer, we additionally deploy Proxy Re-encryption technology as a security layer to improve data security and integrity. Finally, the system will be assessed to see if it is feasible in practice.

Contents

Commitment iv

Acknowledgement v

Abstract vi

List of Figures xi

List of Table xii

Terms xiii

1 Introduction 1

1.1 Motivation and Purpose 1

1.2 Scope and Objectives 2

1.3 Thesis report structure 2

2 Preliminary Studies 3

2.1 Blockchain 3

2.1.1 Definition 3

2.1.2 Component 4

2.1.3 Properties 5

2.1.4 Virtual machine 5

2.1.5 Smart contract 5

2.2 Internet of thing 7

2.2.1 Concept of IoT 7
2.2.2 Basic component of IoT 7
2.2.3 Application 8
2.2.4 Publish/Subscribe protocol 10
2.3 Cryptography 10
2.3.1 Symmetric key encryption 10
2.3.2 Asymmetric key encryption 10
2.3.3 Proxy Re-encryption 10
2.4 Related Works 11

3 The Proposed Approach **15**
3.1 Scenario and solution description 15
3.2 System design 16
3.3 Requirement analysis 19
3.3.1 Functional requirements 19
3.3.2 Non-functional requirements 19
3.4 Implementation 19
3.4.1 General architecture 19
3.4.2 General sequence diagram 26
3.4.3 Functional implementation 30
3.5 Implementation results 39
3.5.1 Register and Login 39
3.5.2 Web application for data user 40
3.5.3 Web application for data user 43

4 Experiment and Evaluation **48**
4.1 Risk mitigation in Smart Contract 48
4.1.1 Re-reentrancy attack 48
4.1.2 Integer overflow and underflow 49
4.2 Data security in proposed model 49

4.3 Testing Smart Contract 50
4.4 Gas fee of transaction on Blockchain 54

5 Conclusion **58**

5.1 Main contribution 58
5.2 Achievement 58
5.3 Limitation 59
5.4 Future Improvement 59

Appendix **62**

A Prerequisite software 62
B Build 63
B.1 Pull repository 63
B.2 Install nodejs packages 63
B.3 Initailize database 63
B.4 Deploy smart contract 63
B.5 Start IPFS 64
B.6 Run the application 64
B.7 Test the smart contract 64
C Manual 65

List of Figures

2.1 System model of access control framework [1] 11
2.2 Information structure [1] 12
2.3 System structure [2] 13
2.4 Blockchain IoT Data Sharing Model [3] 14

3.1 The architecture of the proposed model 17
3.2 Implementation of the proposed model 20
3.3 Proxy Re-encryption Model [4] 24
3.4 Sequence diagram of subscribe data between data owner and data User 27
3.5 Sequence diagram of subscribe data between data owner and data User 29
3.6 Sequence diagram of DO register device 30
3.7 Sequence diagram of DO public data 31
3.8 Sequence diagram of DU subscribe future data 33
3.9 Sequence diagram of DU subscribe published data 34
3.10 Constant in Smart Contract 35
3.11 Struct in Smart Contract 36
3.12 Mapping variable in Smart Contract 36
3.13 Sign up (Webapp) 39
3.14 Sign in (Webapp) 40
3.15 DO register device (Webapp) 40
3.16 DO's devices (Webapp) 41

3.17 Publish data (Webapp) 41
3.18 Transaction need to update 42
3.19 Update re-key 43
3.20 Available device (Webapp) 43
3.21 Subscribe device (Webapp) 44
3.22 Data summary card (Webapp) 45
3.23 Received Devices Data (Webapp) 45
3.24 Received Data (Webapp) 46
3.25 Decrypt Data (Webapp) 46
3.26 All Subscription (Webapp) 47
3.27 Confirm Data (Webapp) 47

4.1 Re-entrancy attack example 49
4.2 Test smart contract result 54
4.3 Gas price 56

1 Blockchain network configuration 63
2 Smart contract address 64

List of Tables

2.1 Component of IoT 8
2.2 Field application of IoT 9

4.1 Gas fee table 55
4.2 Cost in Ethereum 56
4.3 Cost in BSC 56

Terms

ABI ..... Contract Application Binary Interface: is a standard set by Smart contract provides for applications to communicate with it.

EOA ..... Externally Owned Account: ethereum account of user

API ..... Application Programming Interface

ABI ..... Application Binary Interface

D-app ..... Decentralized application

IoT ..... Internet of Thing

P2P ..... Peer to peer

PoW ..... Proof of Work

PoS ..... Proof of Stake

PoA ..... Proof of Authority

PoSA ..... Proof of Staked Authority

DO ..... Data owner

DU ..... Data user

pk ..... Public key in asymmetric key pair

sk ..... Secret key (private key) in asymmetric key pair

BC ..... Block-chain

SC ..... Smart contract

rk ..... Random key (a symmetric key)

CID ..... A content identifier, is a label used to point to materials in IPFS.

1 Introduction

1.1 Motivation and Purpose

Nowadays, IoT plays an important role in our life from inside to outside of society. IoT is everywhere, even though we don't always see it or know that a device is part of the IoT. The number of IoT devices is predicted by Statista to exceed 25 billion by 2030 [5]. The IoT is turning physical objects into an ecosystem of information shared between devices that are wearable, portable, even implantable, making our lives technology and data-rich. However, in order to build a sufficient database system to manage and storage those IoT devices data spend a lot of time, money and is inefficient. Poor management might result in the loss of sensitive data such as health status, lifestyle patterns, and device control. For example, you may lose control of your car while sitting in it, or you may be unable to enter your home due to a malfunctioning smart door system.

Traditional IoT access management programs are mainly built on popular access management models such as RBAC (Role-based access control model), ABAC (the attribute-based access control model), CapBAC (the capability-based access control model). It's worth mentioning that in the aforementioned management methods, a centralized entity confirms object access permissions. As a result, it can lead to a single point of failure. To address this issue, distributed CapBAC models have been developed, in which IoT devices themselves, rather than a centralized entity, authenticate access. However, IoT devices are small, low-power, and have limited computing capacity, thus they can't serve as an access authorization entity.

Meanwhile, following its success in the financial sector since 2008, Blockchain has demonstrated its benefits, the most important of which is the decentralized model. Therefore, Blockchain is a good candidate for IoT data management.

Blockchain combined with IoT will be a good solution to solve the problems encountered in centralized architecture when managing IoT data.

In this project we aim to implement publish/subscribe protocol based on Block-chain technology in order to solve the sharing IoT data problem. We also apply proxy re-encryption to enhance the security and privacy of shared data.

1.2 Scope and Objectives

In the Internet of Things, access management can take several forms. Device ownership management, data disclosure management, data sharing management, and so on are examples. Within the scope of this paper, however, our research team suggests a data management system that can manage and distribute data between people who own the equipment that generates the data and those who need to access and utilize it.

The project's objective is to focus on building a system utilizing block-chain to implement a publish-subscribe protocol in an IoT access management solution, as well as re-encryption to enhance the integrity and security of IoT data sharing management. All data sharing transactions between participants of the system are kept in a visible, public, and unchangeable manner. From there, tracking transactions and settling inter-party disputes are simple and non-repudiation is assured. Some factors like smart contract security and transaction costs are taken into account by the study team while evaluating the system.

1.3 Thesis report structure

The remainder of this thesis is organized in the following manners. Chapter 2 explains some terms and the background knowledge of blockchain, IoT, cryptography and summarize some related works. Chapter 3 presents the architecture of our proposed model with explanation in details as well as how we implementing our prototype and our result. The research discuss about some security hole in smart contract and evaluate our implemented system in Chapter 4. In the final chapter chapter 5, the research team summarizes the main contributions of the thesis, the results achieved, and some limitations as well as the future development of the topic.

2 Preliminary Studies

2.1 Blockchain

2.1.1 Definition

Blockchain is a distributed database shared by nodes in a computer network [6]. As a database, blockchains store information electronically in digital form. Blockchain is best known for its important role in providing a secure and decentralized record of transactions in cryptocurrency systems such as Bitcoin. The innovation of blockchain is that it ensures the accuracy and security of data records and builds trust without the need for trusted third parties.

One of the key differences between a typical database and blockchain is the way data is structured. Blockchain collects information into groups known as "blocks" that contain sets of information. Blocks have a specific storage capacity, are closed when filled, and linked with previously filled blocks to form a data chain known as a "blockchain". Any new information following a newly added block is compiled into a newly formed block, which is also added to the chain after being filled.

Databases typically structure data into tables, whereas blockchains, as the name suggests, structure data into chunks (blocks) that are linked together. This data structure, when implemented in a distributed fashion, creates an inherently irreversible data timeline. When the block is full, it is pinned to the stone and becomes part of this timeline. Each block in the chain is given an accurate timestamp when added to the chain.

2.1.2 Component

Blockchain has 3 main components:

- **Node application**: Each computer need to connect to the internet, install and run a specific application computer to the blockchain ecosystem. Using the case of ethereum as an example ecosystem, each computer must be running the ethereum wallet application.
- **Shared Ledger**: This is a logical component. A shared ledger is a data structure managed within a node application. Data of a shared ledger will be distributed to every node application in that blockchain network and this data can only be read or written, can not be edited or deleted. The structure of shared ledger is a backlinks list and when a new transaction is created, this transaction will be contained in a new block. This new block will link to the previous block and shared ledger will extend overtime.
- **Consensus mechanism**: The consensus mechanism is implemented as part of the node application, providing the rules for how the ecosystem will arrive at a single view of the ledger. Presently, Blockchain has 4 most popular consensus mechanisms:
    1. **Proof of Work - POW**: Proof of Work is the original consensus algorithm for blockchain networks. The algorithm is used to confirm the transaction and creates a new block in the chain. This algorithm involves minors (groups of people) who compete with each other to complete a transaction on the network. The process of competing with each other is known as mining.As soon as the miner successfully created a valid block with a minimum time, that miner gets rewarded. A famous blockchain network using this mechanism is Bitcoin blockchain
    2. **Proof of Stake - POS**: POW makes waste in solving resources in that blockchain.Therefore, the published of POS is to fix this problem.The concept of POS is that block transactions can be mined or validated depending on the number of coins the miner owns. This means that the more coins a miner has, the more mining power they have. This mechanism is used Ethereum
    3. **Proof of Authority - POA**: POA consensus mechanism leverages value of identities, which means that block validators are not staking coins but their own reputation instead. With POA, the right to create a new block is assigned to a node that has proven to do so. To get

this right and the right to generate a new block, the node must pass a preliminary authentication. This method is used in Eurus

4. **Proof of Staked Authority - PoSA**: This mechanism is a hybrid of PoS and PoA. Only authorized identity can be come a validator, and these person will be chosen by the staking mechanism. In Binance smart chain (BSC) there are only 21 validator which will take turn verify blocks. This method will shorten validating time and reduce cost.

2.1.3 Properties

Blockchain have 3 unique properties comapare to other technology is:

1. **Decentralize**: Blockchain do not have a single point of failure due to the mechanism of P2P network. All of the data is distributed, therefore make the data almost always available.
2. **Immutable**: Since all the data in a block is hashed and all the block is linked together therefore if data changed all the following block would be changed.
3. **Transparency**: Since every one can access to public blockchain and check all the transaction from the beginning.

2.1.4 Virtual machine

The virtual machine is the final logical component implemented as part of the node application that every participant in the ecosystem runs.A virtual machine is a representation of a machine created by a computer program and manipulated by language instructions. This is an abstraction of the machines contained in the machine. The virtual machine itself exists solely for the purpose of keeping the continuous, uninterrupted, and immutable operation of this special state machine. For instance in ethereum, ethereum virtual machine is the environment in which all ethereum accounts and smart contracts live.

2.1.5 Smart contract

Smart contract can be understand as a program that can run on Virtual Machine in a blockchain network. Smart contract can be written in high programming language such as **Solidity, Rust, Vyper, ...**, then this code will be compiled

into bytes code and stored in blockchain. Once smart contract is deployed, it is immutable since it is stored in blockchain.

2.1.5.1 Protocol

Since smart contract is quite new. It became popular and widely known in 2015 by its application run EVM (Etherrum Virtual Machine) on Ethereum blockchain network. Therefore, protocol have been proposed by developer to make coding and communicating between smart contract easier. These proposal is name as Ethereum Improvement Proposals (EIPs) [7]. These proposal including only interfaces that tell which and how function should be implement. In this thesis proposal, we have studied and spotted out these relevant EIP:

- **EIP-721 (Non-fungible token):** This is an interface for non-fungible token (NFTs) to help to creating and control ownership of data. Each token in this interface contain a unique data, therefore can help us in identify ownership of corresponding data.
- **EIP-1155 (Multi Token Standard):** A standard interface for contracts that manage multiple token types. A single deployed contract may include any combination of fungible tokens, non-fungible tokens or other configurations.
- **EIP-1967 (Standard Proxy Storage Slots):** Due to the immutable property of blockchain, smart contract once deployed its logic can not be modified. However, programming always evolve new optimization can be made and new security risk can be found. Therefore, EIP-1967 proposed a new way to upgrade deployed contract without changing the old address using proxy upgrade. User will interact with this proxy contract and in this contract, it will store the logical contract. When ever we want to change the logic we can update the logic contract and the user can still interact normally with the same proxy contract.

2.1.5.2 Smart contract security

Just like any other program smart contract also have vulnerability that malicious party could exploit. Depend on difference programming language will have their weakness. In this part, we will focus on vulnerability of **Solidity**.

There are many security issued happen in solidity such as integer overflow/underflow, memory overflow. But these error had been fix from Solidity

version 0.8.0 and above [8]. How ever many other problems still there in Solidity such as race condition, transaction order dependency (TOD), re-entrancy,... [9].

- **Transaction order dependency (TOD):** This issue is mentioned in SWC-114 (Smart Contract Weakness Classification) [10]. Miner can looked the transaction and choose which one to include to a block base on it's gas price. If a malicious party running the blockchain node can tell which transaction going to be occur before the block finalized. Then malicious party could make an other transaction with high gas price so that their transaction can be processed first. If a contract having TOD malicious party can exploit this.
- **Re-entrancy:** This issue is mentioned in SWC-107 (Smart Contract Weakness Classification)[11]. This can happen when smart contract don't change the state variable before making call to external smart contract and the external contract call the original contract again. When this happen the, it will alter the control flow of the function and making in-predicted interaction.

  This issue will be discuss further in section 4.1.1

2.2 Internet of thing

2.2.1 Concept of IoT

The term Internet of Things generally refers to situations where networking and computing capabilities extend to everyday objects, sensors, and objects that would not normally be considered computers, allowing devices to creates, exchanges, and consumes data with minimal human intervention.[12]

2.2.2 Basic component of IoT

IoT is not a single technology, but it is the combination of technologies. According to [13] A complete ecosystem of IoT including 4 building blocks: Things, Gateways, Network Infrastructure (NI), and Cloud Infrastructure (CI).

- Things are centralized areas where information is obtained by sensor elements or actuators.
- Gateway blocks are used for connectivity purposes and are intermediate blocks between things and your network or cloud infrastructure (CI).

- The network infrastructure (NI) block assists in order to control the information provided and ensure that the information flows safely and smoothly.
- Cloud infrastructure (CI) with information storage and data processing capabilities.

The detail of each component are shown in the table below:

IoT building blocks	IoT devices	Features
Things	Sensors, Actuators	Allows to communicate and collect the information from the objects of focused areas without any human interaction.
Gateways		Acting as an intermediate block and enables the strong connectivity between the things and cloud infrastructure. It also provides the security and manageability abilities during the data flow.
Network infrastructure (NI)	Routers, Aggregators, Gateways, Repeaters	It allows the control over the data flow from things to the cloud infrastructure. It also enables the security during the information flow.
Cloud infrastructure (CI)	Virtualized Servers (VS), Data Storage Units (DSU)	It allows the analytical, logical, and advanced computing abilities.

Table 2.1: *Component of IoT*

2.2.3 Application

IoT technologies have a wide range of application because it is adjustable to almost any technology that is capable of providing relevant information about its own operation, about the performance of an activity, and even about the environmental conditions that we need to monitor and control at a distance. Thanks to IoT's application, concepts like smart home, smart office, farm, factory, or even smart city are no longer strange to many people. According to [14] IoT's application presents in 9 fields in the table below:

Index	Area of usage	Description	Examples
1	Human	Devices attached or inside the human body	Devices (wearables and ingestible) to monitor and maintain human health and wellness; disease management increased fitness and higher productivity.
2	Home	Buildings where people live	Home controllers and security systems.
3	Retail Environments	Spaces where consumers engage in e-commerce	Store s, banks, restaurants, arenas anywhere consumers consider and buy; self-checkout, in-store, inventory optimization .
4	Offices	Spaces where knowledge workers work	Energy management and security in office buildings;improved productivity, including for mobile employees .
5	Factories	Standardized production environments	Places with repetitive work routines, including hospitals and farms; operating efficiencies, optimizing equipment use and inventory .
6	Worksites	Custom production environments	Mining, oil and gas, construction; operating efficiencies, predictive maintenance, health and safety .
7	Vehicles	Systems inside moving vehicles	Vehicles including cars, trucks, ships, aircraft, and trains; condition-based maintenance, usage-based design, presales analytics.
8	Cities	Urban environment	Public spaces and infrastructure in urban settings; adaptive traffic control, smart meters, environmental monitoring, resource management .
9	Outside	Between urban environments (and outside other settings)	Outside uses include railroad tracks, autonomous vehicles (outside urban locations), and flight navigation; real-time routing, connected navigation, shipment tracking.

Table 2.2: *Field application of IoT*

2.2.4 Publish/Subscribe protocol

According to [15] Publish-subscribe is a common message pattern in which a message is not programmed for a particular recipient. Instead, the user creates topics and publish a message to those topics. A group of servers called message brokers store messages and forward them to topic subscribers. The Publish-subscribe system is typically not used directly by end users. Developers use Publish-subscribe to share data with their applications.

2.3 Cryptography

2.3.1 Symmetric key encryption

Symmetric encryption is a type of encryption that uses only one key (private key) for both encryption and decryption of electronic information. Entities that communicate over symmetric encryption need to exchange keys for use in the decryption process. This encryption method differs from asymmetric encryption, which uses a public / private key pair to encrypt and decrypt messages.

2.3.2 Asymmetric key encryption

Asymmetric key encryption, also known as public key cryptography, is a method of encrypting data using two different keys and providing one of the public keys that anyone can use. Another key is known as the private key. Data encrypted with the public key can only be decrypted with the private key to protect the data packets from unauthorized access or use.

2.3.3 Proxy Re-encryption

Proxy re-encryption allows a proxy server to convert a ciphertext encrypted with one key to encrypt the same message with a different key. The basic idea is to expose as little information as possible to the proxy server so that the translation can be done with as little trust as possible. At a minimum, the proxy should not know the participant's key or the content of the message it re-encrypts.

2.4 Related Works

The authors [1] proposed a Blockchain-based access control framework with privacy protection in cloud, named AuthPrivacyChain. As described in Figure 2.1, Data Owner (DO) will upload data resource to a centralized cloud. Then whenever Data User (DU) want to access the data, they must request to the cloud and the cloud would find permission from blockchain then verify the access. If it is valid the centralized cloud would give back the data to DU.

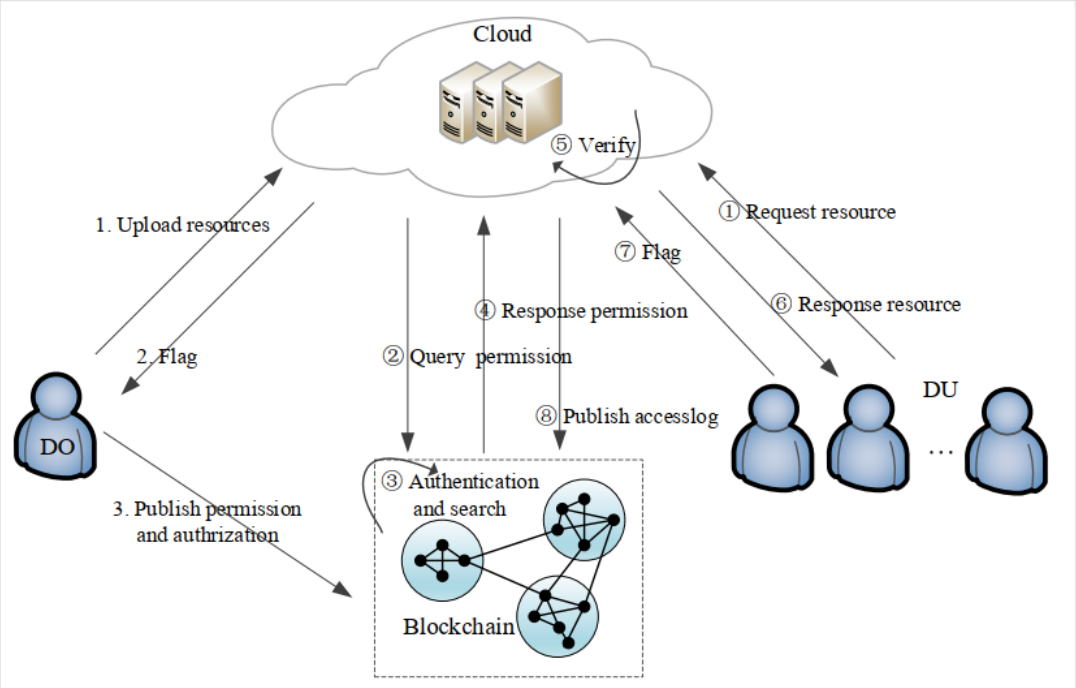


Figure 2.1: *System model of access control framework [1]*

We can see that all the interaction between and Data User (DU) must go through to a centralized cloud. Cloud played as both as database and access control unit taking access control data from blockchain. This system which is most likely a centralized system with a single point of failure is the cloud system. Beside from its inappropriate architect. We had learnt what information of a resource that we should stored in blockchain as shown in Figure 2.2.

Set/Field	Description
resInfo	It represents a set of resource information. **resInfo** ={*resOwnerID, resID, resName, reshash, resCAP, resURL*}. where *resOwnerID*: host of the resource. *resID*: the unique identifier of the resource. *resName*: resource name. *reshash*: resource hash. *resCAP*: resource access permission. *resURL*: resource access address.
resCAP	It represents access permission of resource. ***resCAP*** = {*accessPolicy*, *acessConstCond*, *accessMode*, *timespan*, *resCAPhash*}. where *accessPolicy*: access control policies, such as DAC, BLP, RBAC, etc. *acessConstCond*: access constraints and conditions. *accessMode*: resource access mode, *accessMode*= {up, down, migrate,delete}. *timespan*: access time interval, including start and end. *resCAPhash*: *resCAP* hash.
resCAP_S	It represents encrypted *resCAP*.
bresInfo	It represents resource information in blockchain. ***bresInfo***={ *actionInfo*, *resInfo* }, where *actionInfo*: the added information of the resource.
cloudInfo	It represents *Cloud* basic information set, ***cloudInfo*** ={ *CloudID*, *CloudName*, *CloudUrl*}, where *CloudID*: cloud record number). *CloudName*: cloud name. *CloudUrl*: cloud access address.
tran	It represents a transation of blockchain set, ***tran***={*tranID*, $Addr_{from}$, $Addr_{to}$, *value*, *bresInfo*}, where *tranID*: transaction number. $Addr_{from}$: transaction originator. $Addr_{to}$: transaction receiver. value: transaction value. *resInfo*: resource information.

Figure 2.2: *Information structure [1]*

The author [2] proposed a model leveraging Blockchain to enhance data privacy in IoT-based applications as depicted in Figure 2.3. Our method is mainly developed based on this work that solves the problem of sharing data that the aggregator want to share theirs collected data and does not have to reveal the sk subcribers to decrypt the data packet.

Each data slot along with each subcriber will have a distinguish re-encryption key, this will avoid subcribers read all the data in the system. Moreover, with this system both aggregator and publisher all interact with blockchain therefore could avoid a single point of failure and all the action is taken place record in blockchain.

Using blockchain in the system also makes the transactions verified and trustable. When Aggregator publishs data , they have to using there private key to sign. This helps in proving the owner of that data packet. Using distributed hashtable, data are separated into chunks and are stored across different storage nodes, making someone hard to assemble data.

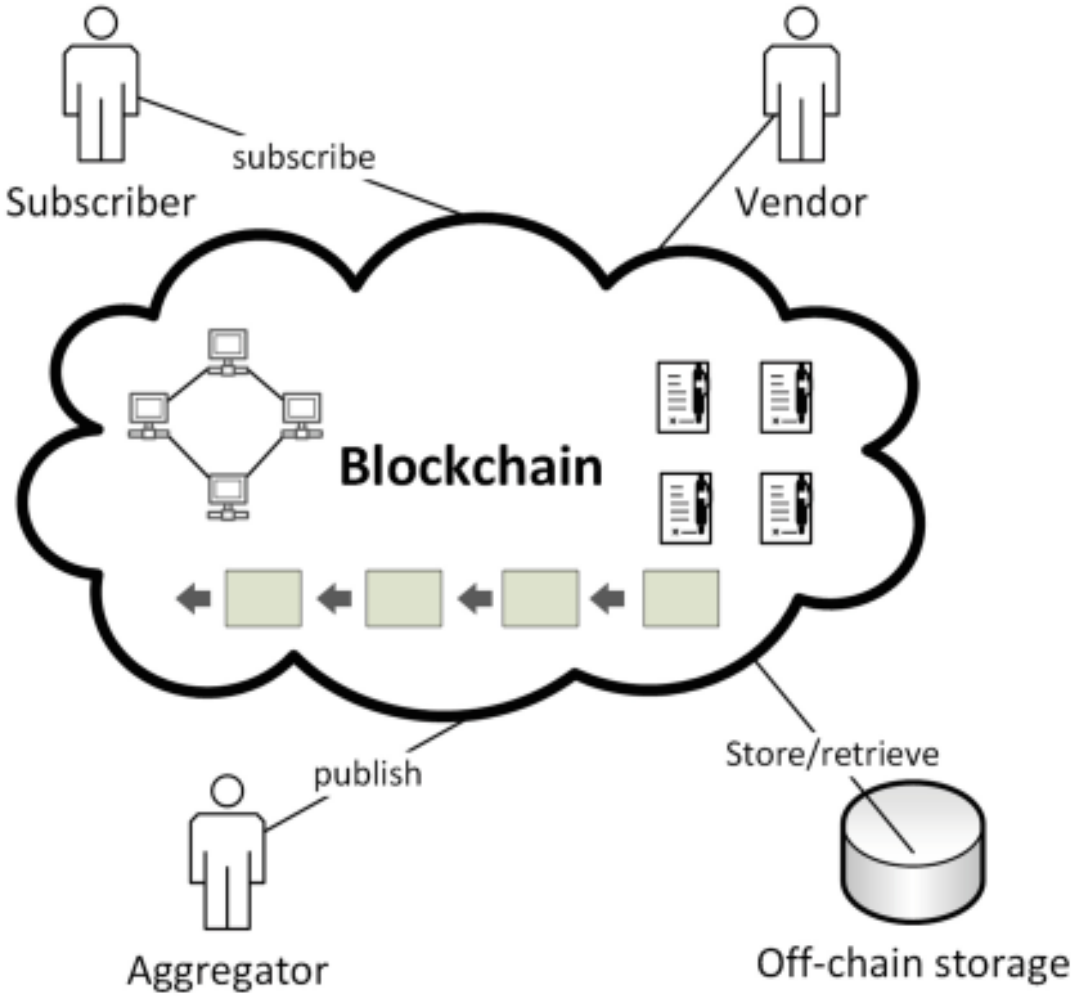


Figure 2.3: *System structure [2]*

A similar work has been proposed by authors in [3]. Our proposed approach will try to improve this work. As shown in Figure 2.4, the idea of this report is DU can buy data packages from DO through smart contract. After the transaction complete the gateway of DO will decrypt the data information and send to DU. The system is primarily comprised of a gateway that receives and processes data

from IoT devices before re-encrypting and sending it to DU. However, with the current model, there are still some security and performance issues.

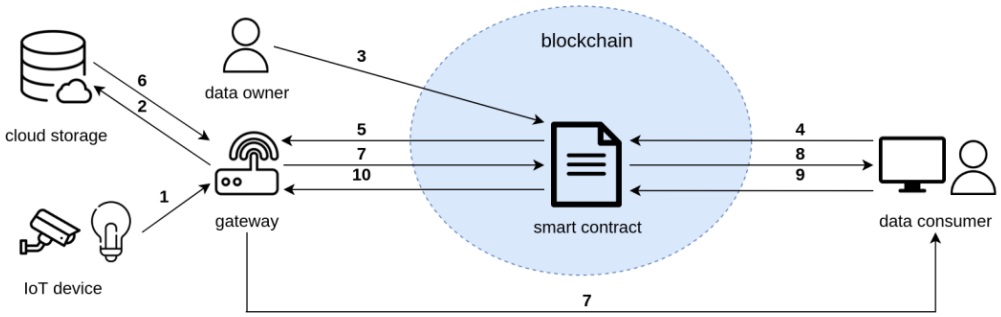


Figure 2.4: *Blockchain IoT Data Sharing Model [3]*

Firstly, the Gateway will listen to all events from SC; but, if the number of queries is significant enough, bottleneck concerns may arise. For example, if all requesters try to request data at the same time, the gateway will listen to all requests from SC, causing the gateway to become overloaded, resulting in a single-point failure. The author also attempted to tackle the problem by limiting access to certain times. This technique, however, is not practicable in practice, and it creates an overhead for managing access in the smart contract.

Second, after listening to the events, the gateway will retrieve data from cloud storage, encrypt it with DO's private key and DU's public key, and transfer the data packets to DU. To do the re-encryption, the gateway will need to store DO's private key. Furthermore, the gateway can be easily harmed. As a result, if hostile parties attack the gateway, they can obtain the DO's private key as well as data packets, allowing information to leak.

3 The Proposed Approach

3.1 Scenario and solution description

Access management in the IoT may be handled in a variety of methods, as discussed in the preceding sections. However, in this research, we focused on the issue of data storage and sharing management in the Internet of Things.

Different from the traditional model, the proposed system eliminates the role of the third-party service during data exchange between *Data Owner (DO)* who has data to share or sell, and *Data User (DU)* who wants to buy the data. Our proposed model employs a decentralized architecture by using Blockchain to enhance the security of the data exchange process, in which all operations are executed via smart contracts.

For example, Data Owner is the owner of a smart hatchery consisting of hundreds of IoT devices that collect biological indicators of livestock on his farm. Collected data as well as biological index data of the breeding stock are stored by the owner for monitoring as well as sharing for organizations and individuals that need to use this data. The Data Owner will link IoT devices to the Gateway as the data collecting and communication gateway with the Internet environment since IoT devices have minimal processing capability, limited power, and employ a variety of data transmission protocols. The Data Owner stores their data in Off-chain storage (data that is not stored on the Block-chain) and publish it to blockchain. Data Requester wishes to access and use the Data Owner's data. Data Requester, for example, is a company that studies the growth of breeding stock and needs to analyze data on biological indicators from a variety of breeds at various ages. Because the Requester needs the data received from the sensors on the animal in order to measure it, they must subscribe to the farm's data. As a

result, the Requester can only access and use this data source whenever the Data Owner publish data.

The identification of the data owner or data requester is done by using the Block-chain's key system. A key pair will be owned by everyone on the block-chain network (including a public key and a private key, the public key is generated from the private key). The address produced from the public key will be used to identify each user in the system. Key pairs are used for encryption during data storage and exchange in addition to creating IDs for users on the network. The system will use the Block-chain network environment (ex: Ethereum's network) to allow the Data Owner and Data Requester to exchange data. Instead of relying on a third-party intermediary for data sharing, the system performs data exchange using sets of rules and management rules built on Smart contracts.

In contrast to the old manner, the system's two above objects will share data via Smart Contract. As a result, data sharing is transparent and public. Every share transaction is recorded on the decentralized network, making changing the transaction history extremely impossible. At the same time, utilizing the blockchain network platform's public key - private key combination helps to secure data integrity and security, avoiding the possibility of intermediaries stealing, decrypting, and exploiting data (a man-in-the-middle attack).

Overall, the proposed approach has the following benefits in compare to the traditional approach:

- The ability to retrieve transaction history and ensure that the transaction's content is correct.
- Remove the involvement of a third party (centralized entity) in data access verification.
- Enhance data transparency, integrity, and security, as well as the privacy of users' data.

3.2 System design

Figure 3.1 depicts the architecture of our proposed model that consists of two processes, including data collection and data exchange. Regarding data collection process, each data owner can have multiple *IoT end-devices* to collect IoT data that will be then transferred to the *Off-chain Storage* through the *Data*

management system.

In this model, we will use IPFS as our *Off-chain Storage* since it is a distributed system for storing and accessing files and everyone can access the file through content identifier (CID). The detail of the data collection process can be summarized as follows:

- Data owner aggregate data from IoT end-devices. The data includes sensing data and corresponding timestamps. Additionally, each IoT end-device will be assigned a unique number for device identity management.
- Once DO upload the data from IoT end-devices, the *Data management system* (DMS) will create asymmetric key pair and encrypt these data. Then push them to the off-chain storage (IPFS) by pre-established protocols. DO will store CID of data along with deviceID, startDay, endDay of data for classification.

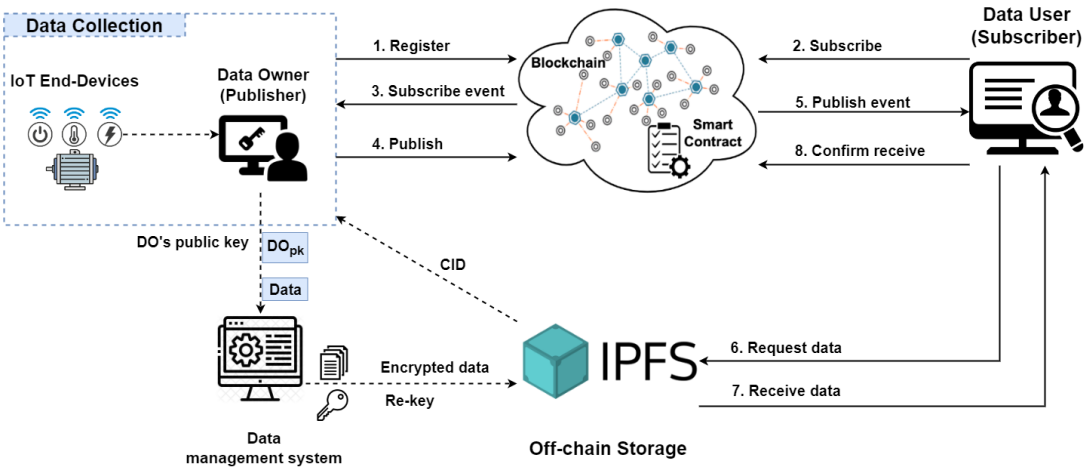


Figure 3.1: *The architecture of the proposed model*

Regarding the data exchange process, the owner will register and publish the data they wanted to share on Blockchain via smart contracts. DU must send the corresponding amount of money with the transaction to the smart contract to request for data from the owner and has 2 options in order to achieve the data:

- DU subscribe for the upcoming data in the future.
- DU subscribe for the data that have been published in the past.

An overall information trade process for subscribe data is acted in eight stages as numbered in Figure 3.1 and depicted as follows.

1. The DO register their IoT devices information to smart contract. Information including: name of device, description, price per day. The price of the IoT device will be used as a parameter to calculate the amount of money the subscriber has to pay.

2. When DU subscribe to a device, they have to send the subscription fee to the smart contract corresponding to the device and duration that they want to retrieve the shared data. The fee will be stored in a smart contract until the DU confirms it, after which it will be transferred to DO.

3. DO will receive an event whether DU subscribes to future data or subscribe to past data from the smart contract

4. In case DU subscribes to future data. Whenever DO want to publish the data, they will collect the data from IoT devices and send these data to the DMS. DMS will generate a random asymmetric key pair, then use the pk (from the newly generated key pair) to encrypt the aggregated data. Next, DMS will create re-encryption key for corresponding subscribers from the sk of the data and subscriber's public key. DMS then upload both encrypted data and list of re-encryption key to IPFS. Finally, DO publish CID of both encrypted data and list of re-encryption key received from IPFS to smart contract; In case DU subscribes to published data. DMS will create new re-encryption key for new DU, then upload new list of re-encryption keys to IPFS. Finally, DO update CID of new list of re-encryption key to SC.

5. DU listen and receive the publish data event to get the CID of encrypted data and list of re-encryption key or receive update key event in order to get new list of re-encryption key from Smart contract.

6. DU go to the IPFS to get both the encrypted-data and the list of re-encryption keys.

7. DU extract their re-encryption key from the keys list. In order to get the raw data, Du will perform 2 steps of decryption sequentially. First DU will re-encrypt the encrypted-data with re-encryption key, then continue to decrypt re-encrypted data with private key of DU.

8. If the data is valid, the consumer will confirm on the smart contract that the data was precisely received. The smart contract will transfer subscription fee

paid by the DU to the DO once the DU sends the confirmation of valid data reception.

3.3 Requirement analysis

3.3.1 Functional requirements

- DO is able to registers IoT device to Smart contract.
- DO can publishes data to SC.
- DU can subscribe to registered device.
- DU can access subscribed data.
- Making payment for data between DO and DU
- Applying Proxy Re-encryption
- Encrypt and store data of DU in off-chain storage
- All sensitive data is encrypted
- Web-app is able to interact and communicate with Smart Contract.

3.3.2 Non-functional requirements

- The user interface is build on web-base.
- Simple, elegant user interface, easy to use.
- The application can operate 24/7.

3.4 Implementation

3.4.1 General architecture

In order to evaluate the feasibility of our proposed model, we have developed a prototype of an application for IoT data sharing management on Blockchain Network that is described in Figure 3.2. Basically, there are four major objects in our proposed model, including *DO*, *DU*, *Data management system*, and *Smart*

Contract. Each DO or DU is an externally owned account (EOA), which is controlled by a public/private key pair, in the Ethereum Blockchain system. The purpose of DMS is to create, store and manage list of key (re-key, pk, sk). DO use DMS to manage public key and private key of each data package. DU also use DMS to manage their public key, private key and package re-key. All private key stored in DMS is encrypted with a symmetric key phrase. The smart contract is used to handle the data exchange process between the DO and the DU.

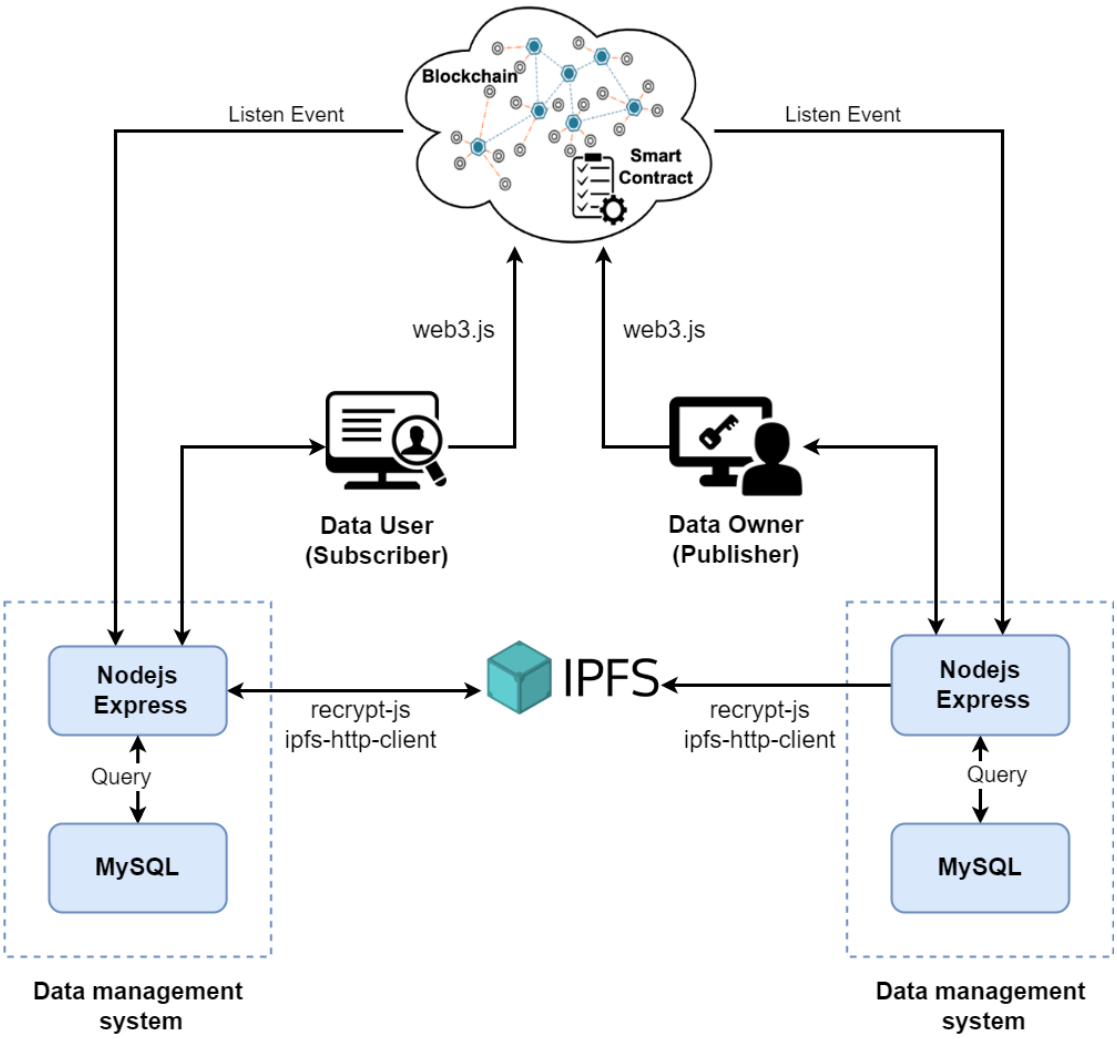


Figure 3.2: *Implementation of the proposed model*

The Figure 3.2 contain many technologies that we choose, and utilize in order to build our system. A brief introduction as well as the reason that we choose these technologies is discussed in the following subsection below.

3.4.1.1 Solidity

To design smart contracts, Ethereum supports a number of high-level programming languages, including:

- Solidity: Object-oriented, high-level language, statically typed, influenced by C++.
- Vyper: a programming language with a heavy influence of Python, procedural programming, simplicity, code is human-readable.
- Yul: intermediate language, able to compile to bytecode.
- Serpent: procedural programming, syntax similar to python.
- FE: Statically typed language for the Ethereum Virtual Machine, inspired by Python and Rust, development is in early stages.

We choose Solidity as a language to develop our smart contract because it being the first smart contract programming language, has large community to support and library development and tools. Solidity supports many data types (uint, uint8, uint16, bytes, ...), various control structures (if else, for, while) inspired from C++, JavaScript. Solidity also has wide market adoption and is being used to build many decentralized applications.

3.4.1.2 Truffle, Ganache

Truffle is the development environment, asset pipeline, and testing framework of the Truffle Suite ecosystem. Truffle is a hugely popular development framework for Ethereum dApp development, and there is a large community behind the tool. Furthermore, Truffle is using the EVM as a basis, and one of its purposes is to make the development of smart contracts more straightforward and more accessible.

Ganache is a tool that allows us to spin up our own local Ethereum blockchain. The blockchain can be utilized throughout all parts of the development process, making this tool highly useful. As we set up our local blockchain, Ganache allows us to deploy, develop and test all our dApps in a safe and deterministic environment.

We use Truffle and Ganache as a development, testing environment to develop our application.

3.4.1.3 HTML, Bootstrap 5, and JavaScript

HTML is an abbreviation for "hypertext markup language." It is a programming language used to construct web pages that can be seen by a web browser. The majority of web pages on the internet and online applications are saved as HTML files. A hypertext is a text that is used to reference other pieces of text, while a markup language is a series of markings that tells web servers the style and structure of a document. Websites are collections of connected HTML pages that are saved on a server somewhere.

Bootstrap, the most popular front-end framework built to design modern, responsive, and dynamic interfaces for professional design web pages. The newest version of Bootstrap is Bootstrap 5 which initial release on June 16, 2020. Bootstrap 5 benefits us the following to design and develop the Webapp front end.

- Easy to use: Anybody with just basic knowledge of HTML and CSS can start using Bootstrap.
- Responsive: Bootstrap's responsive CSS adjusts to phones, tablets, and desktops.
- Compatibility: Bootstrap 5 is compatible with all modern browsers (Chrome, Brave, Edge, Safari, and Opera).

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc.

We choose html, bootstrap and javascript to build front-end because it is simple to create and manipulate document on the website. It also provide pre-define css class help us with make up our website.

3.4.1.4 Nodejs and Express

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest stable version is v16.15.0. The definition of Node.js as supplied by its official documentation is NodeJs as follows:

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Express.js is a Node.js web application framework that is free and open source. It is used to quickly and easily design and create web apps. Web apps are web applications that may be accessed through a web browser.

We utilizing Express to building the DMS because:

- Express.js simply requires javascript, programmers and developers may quickly create web applications and APIs.
- Express.js is a Node.js framework, which implies that most of the code is already created and ready for programmers to use.
- Express.js can be used to create single-page, multi-page, or hybrid web applications.
- Express.js is a lightweight framework that aids in the server-side organization of web applications into a more ordered MVC architecture.

3.4.1.5 Proxy Re-encryption

- Problem Statement:

  Assume that you have 10MB of data and you want to securely save on Online Storage (Google Firebase, IPFS, ...). To ensure that no one can access the contents of your data, encrypt it using your public key and then save the encrypted material on Online Storage. To view the data's contents, you must first download the encrypted material from Online Storage and decrypt it using your private key.

  However you want to share the contents in the file with your friends? In such scenario, you must download the file, decrypt it using your Private Key, encrypt it again with your friend's Public Key, and upload it back to the cloud. Your friend may now download and decode it using his private key. Decrypting and encrypting data of modest size (for example, 10MB) using an asymmetric key will be rapid, and saving on online storage will not take up much space.

The aforementioned scenario appears to be without issue. But what if your data is larger than 5GB? You must repeat the preceding steps for each friend with whom you wish to share the content. It indicates that if you wish to share with 10 people, you must do it 10 times. However, decrypting and encrypting large amounts of data with an asymmetric key takes a long time, and each time you must also store new encrypted data, which consumes 5GB more of online storage.

- Solution:

  Proxy Re-encryption is solution for this situation. Proxy re-encryption (PRE) schemes are cryptosystems which allow third parties (proxies) to alter a ciphertext which has been encrypted for one party, so that it may be decrypted by another.

- Operation method:

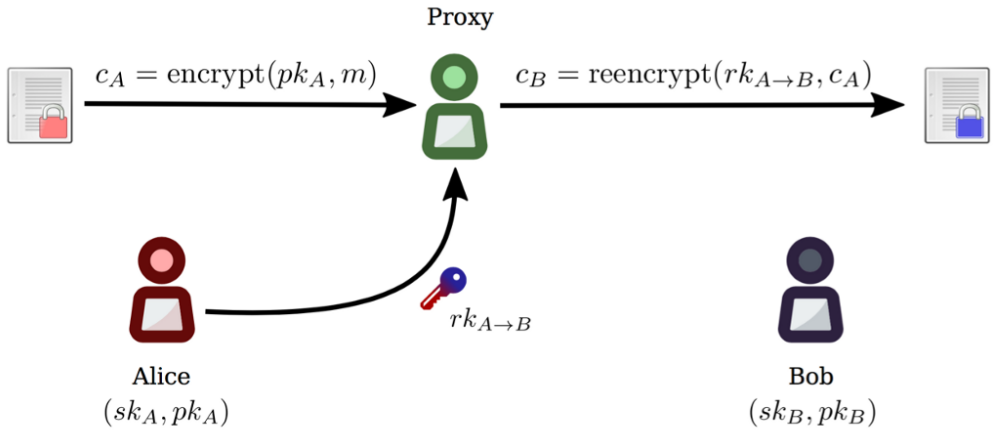


Figure 3.3: *Proxy Re-encryption Model [4]*

  - Assuming that Alice encrypts a message m, with Alice's public key pk_A, the result is the ciphertext c_A.
  - Alice wants to share the content of message m with Bob, who has a key pair (pk_B,sk_B).
  - Alice generates the re-encryption key with her private key and Bob's public key $rk_{\text{A->B}} = rekey(sk_A, pk_B)$
  - Bob will re-encrypt c_A and will receive c_B after encrypting.
  - Bob can decrypt data c_B by using his private key sk_B

To implement the Proxy Re-encryption technology, we utilize this repository Recrypt-js from GitHub. Below is a brief pseudo code that shows how key pair and re-key are generated as well as how to encrypt, re-encrypt, and decrypt data:

```
var kp_A = Proxy.generate_key_pair();
var sk_A = Proxy.to_hex(kp_A.get_private_key().to_bytes());
var pk_A = Proxy.to_hex(kp_A.get_public_key().to_bytes());

var kp_B = Proxy.generate_key_pair();
var sk_B = Proxy.to_hex(kp_B.get_private_key().to_bytes());
var pk_B = Proxy.to_hex(kp_B.get_public_key().to_bytes());

let obj = PRE.encryptData(pk_A, "test data")
console.log(obj)
let rk = PRE.generateReEncrytionKey(sk_A, pk_B);
PRE.reEncryption(rk, obj)

let decryptData = PRE.decryptData(sk_B, obj)
console.log(decryptData)
```

3.4.1.6 Hashing and Symmetric Encrypt/Decrypt

In order to implement the hashing process in our project, we utilize this repository Bcrypt from GitHub.

To encrypt and decrypt using symmetric key in our system, we utilize this repository Crypto-js from GitHub.

3.4.1.7 IPFS

IPFS is a new method to store data, and was initial release in February 2015. The term IPFS stands for InterPlanetary File System, which is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

According to [16], IPFS has 3 main component:

- **Distributed Hash Table(DHT)**: It is used to store and retrieve data between network nodes. It's a class that looks like hash tables. Any node on the network can use a DHT to get the value associated with a hash key.
- **Block Exchange:** It is used to swap data between nodes in the BitTorrent

Protocol (also known as BitSwap). It's a peer-to-peer file-sharing protocol that helps untrustworthy swarms organize data exchange. It employs a tit for tat technique, in which nodes who contribute to each other are rewarded, while nodes that merely seek resources are punished. This allows an IPFS node to retrieve several chunks of data at the same time.

- **Merkle DAG**: It makes use of a Merkle Tree or Merkle DAG, which is comparable to the Git Version Control system. It's used to track changes to network files in a distributed-friendly manner. The cryptographic hash of the content is used to address data.

We choose IPFS as our main storage for data and re-key because it has some benefits:

- **Availability** Users in a local network can communicate with each other, even if the Wide Area network is blocked for some reason.
- **Integrity :** Since it is a content-addressing every file will have a unique CID base on the content. Therefore, the integrity of the data inside is preserve.
- **Immutability:** You can not modify the content inside once uploaded to IPFS. Since even with a little change to the content, it will resolve to a complete difference CID.
- **Cost efficiency:** By eliminating the need for a series of docking stations and Internet servers, the overall cost is markedly reduced.

3.4.2 General sequence diagram

Each participant (e.g., data owner and data user) in the system will be identified by a unique number string that is also the address in Ethereum Blockchain. Smart contract stores a list of registered device and their owner, which will be provided to DU when they do searching. The DO also has to provide information about the device that they want to share, such as description and subscription fee.

Since smart contracts are executed in the Ethereum virtual machine (EVM) environment, they can not inform the outside what is happening inside EVM. However, smart contracts support creating events that can attach certain information for whom outside of Blockchain network.

As mentioned in Section 3.2, our prototype has 2 main sequences to handle subscription data in the future or in the past. These sequences are depicted in order below:

Figure 3.4 depicts the whole sequence diagram of the data exchange process between the DO and the DU based on events and execution of the corresponding smart contracts. This sequence diagram executes correctly to eight steps described in the proposed model of subscribing device in the future.

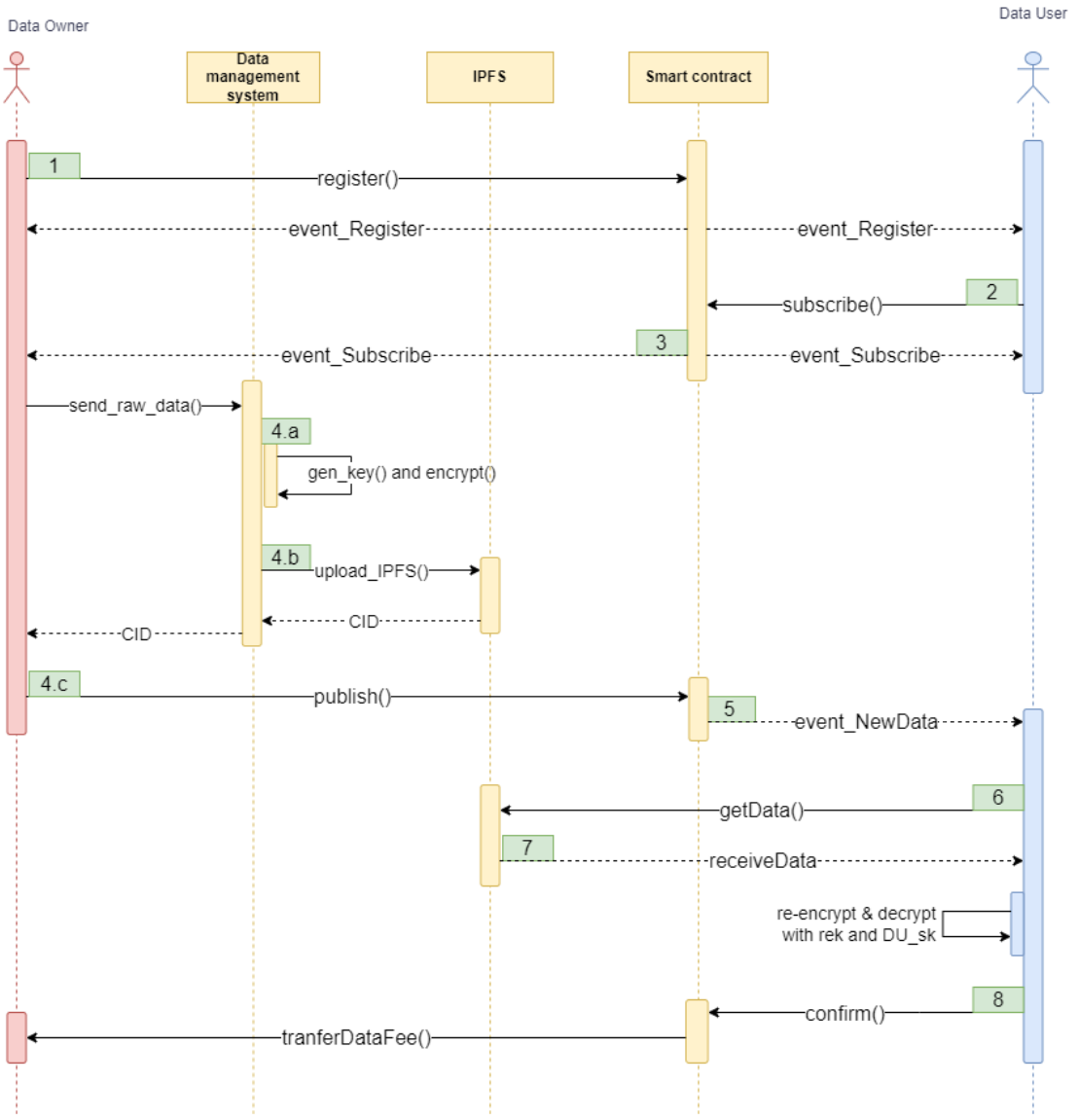


Figure 3.4: *Sequence diagram of subscribe data between data owner and data User*

1. The DO calls *register* to register a new device to the system. If the registration is successful, the system will emit an event *Register* to all participants

for updating information.

2. The DU call *subscribe* along with their public key to ask for sharing data from the selected device. The DU, also, has to transfer data fee to the smart contract.

3. Smart contract validates the data request from the DU. If the request is valid, the system will emit an event *Subscribe* to all participants.

4. DO gather and send data from IoT end-devices to DMS once they are available. DMS will create a asymmetric key pair and encrypt data using the newly created public key. Then generate re-key for DU .The re-key and encrypted data are then uploaded to IPFS via DMS. DO calls *publish* to send the newly data CID and re-key CID to SC.

5. The SC sends an event to DU indicating that the new data has been published.

6. DU will use the received CID from *NewData* event to get the encrypted data and list of re-key.

7. Once encrypted data and list of re-key is received. DU will decrypt the DU's sk with secret key phrase. Then re-encrypt the encrypted-data with re-encryption key. Then continue to decrypt it with pk of DU.

8. The DU validates the integrity of the received data. If the data is valid, the consumer calls $confirm$ to confirm that the data was precisely received. The smart contract transfers the data fee to the DO once the DU confirms the validity of data reception

Figure 3.5 depicts the whole sequence diagram of the data exchange process between the DO and the DU based on subscribe past data events and execution of the corresponding smart contracts.

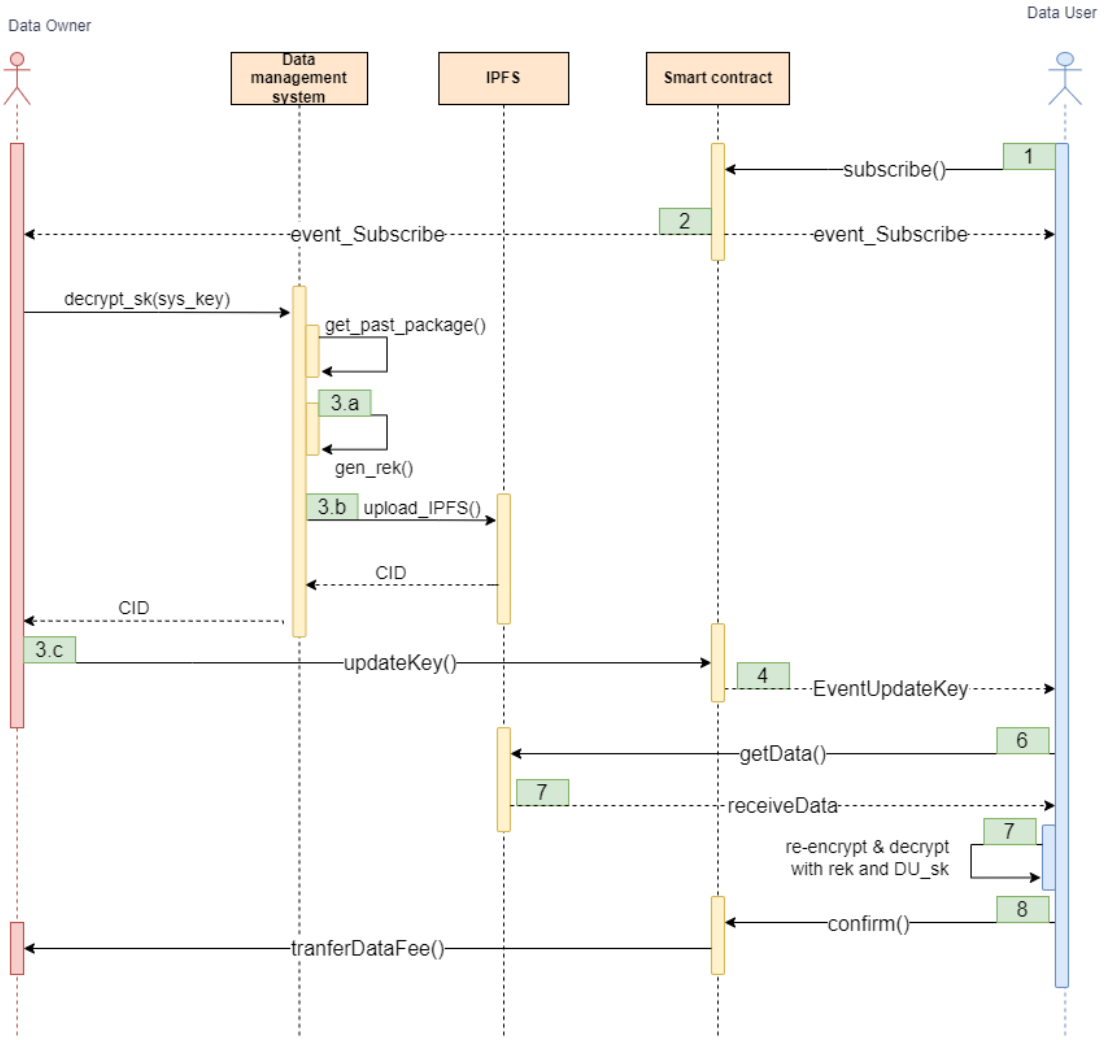


Figure 3.5: *Sequence diagram of subscribe data between data owner and data User*

1. The DO calls *subscribe* to buy published data and transfer fee to smart contract.

2. Then system will emit an event *Subscribe* to all participants for updating information.

3. When DO receive *Subscribe* event with start time less then the latest package. DO will find all the package with-in the time duration, then DO will decrypt packages's sk with DU's secret key phrase. Then create list of re-key for subscribers.

4. Then upload list of re-key to IPFS.

5. DU then calls *updateKey* in SC with new CID to update the CID to re-key.

6. The SC sends an event to DU indicating that the new key has been added.
7. DU will use the received CID from $UpdateKey$ event to get the encrypted data and list of re-key.
8. Once encrypted data and list of re-key is received. DU will decrypt the DU's sk with secret key phrase. Then re-encrypt the encrypted-data with re-encryption key. Then continue to decrypt it with sk of DU.
9. The DU validates the integrity of the received data. If the data is valid, the consumer calls $confirm$ to confirm that the data was precisely received. The smart contract transfers the data fee to the DO once the DU confirms the validity of data reception

3.4.3 Functional implementation

Each Data Owner and Data Requester is each EOA account in the Ethereum Blockchain or Testchain (local blockchain) system. To perform transactions through Smart Contracts, Data User and Data Owner are required to have an Ethereum addres and linking the wallet to the system through the Metamask wallet application. Users interact with the system through a web interface developed in Html applying Bootstrap 5 framework along with the Backend and Local Database are implemented in Nodejs, Express, and MySQL. The data is stored in distributed database IPFS.

3.4.3.1 Data owner

The system has 2 main process when the Data Owner interact with it, these function are described in detail using below sequence diagrams.

- Process of register device id is depicted figure 3.6:

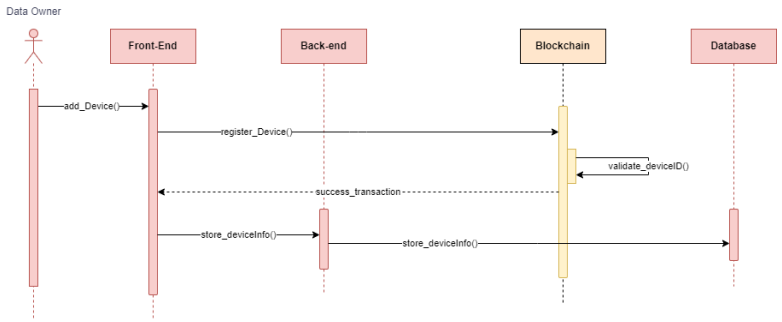


Figure 3.6: *Sequence diagram of DO register device*

In this process, when the Data Owner wants to register a new device, they fill in the detail of device and call *add_Device*() in the front-end of Webapp. The front-end then calls *register_device*() through Web3 to interact with Smart Contract (Blockchain). Then the Smart Contract will validate whether the device id exists or not. If not, SC store the new device and emit an event Register to the front-end of Webapp. Finally, the front-end calls back-end of Webapp to store the device info into the local database(MySQL).

- Process of publish data is depicted in figure 3.7:

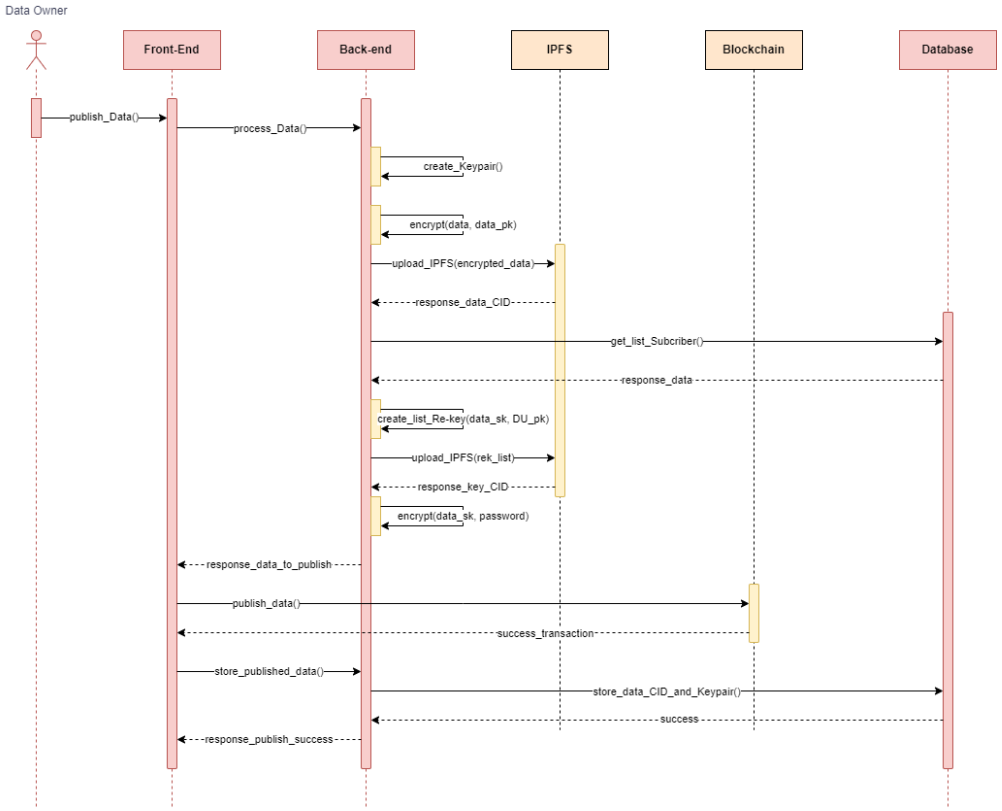


Figure 3.7: *Sequence diagram of DO public data*

In this process, Data Owner call *publish_Data*() after uploading the data file and filling in the data start-day, end-day. The front-end then transfers these data to the back-end of the Webapp (Data management system). After receive the *process_data*() request, first, the DMS call *create_keypair*() to create an asymmetric key pair (public key, secret key) for the data. Then, DMS uses *encrypt_Data*() with the newly created public key to encrypt the data. After that, the DMS call *upload_IPFS*() to upload the encrypted data to the distributed database IPFS. After the upload is completed, the DMS receives the data CID from the IPFS. Then, the DMS calls *get_list_Subscriber*()

to the local database (MySQL) to get the valid subscribers' data (DU's pk, DU's address) from start-day to end-day period.

To perform the *create_list_Rekey()*, the DMS execute the following step. First, the Data Owner inserts the account password to the front-end of Webapp. After that, for each subscriber, the DMS creates the re-key from the secret key of the data packet and subscriber's public key and appends these re-keys corresponding with subscriber address to a .json file. This file, then, is uploaded to the IPFS and response a re-keys CID to DMS. Finally, the DMS executes *encrypt(data_sk, password)* to encrypt the data's sk (from asymmetric key pair of the data packet) with the password of DO and store encrypted-sk to session.

After receive the processed data from DMS, the front-end call *publish_data()* to the Smart Contract. Once, the publish is successful, the Smart contract emits an event *Newdata* to both Data Owner and subscriber(Data User). Then, the DMS queries local database (MySQL) to store start-day, end-day, data id, device id, data CID, re-key CID, and encrypted data's sk from the session. Finally, the DMS response publishes success to the front-end and refresh the page.

3.4.3.2 Data user

When handling request from DU, the system has to execute 1 of 2 main scenario. Both scenario are described in detail using below sequence diagrams.

- Process of subscribe future data is depicted in figure 3.8:

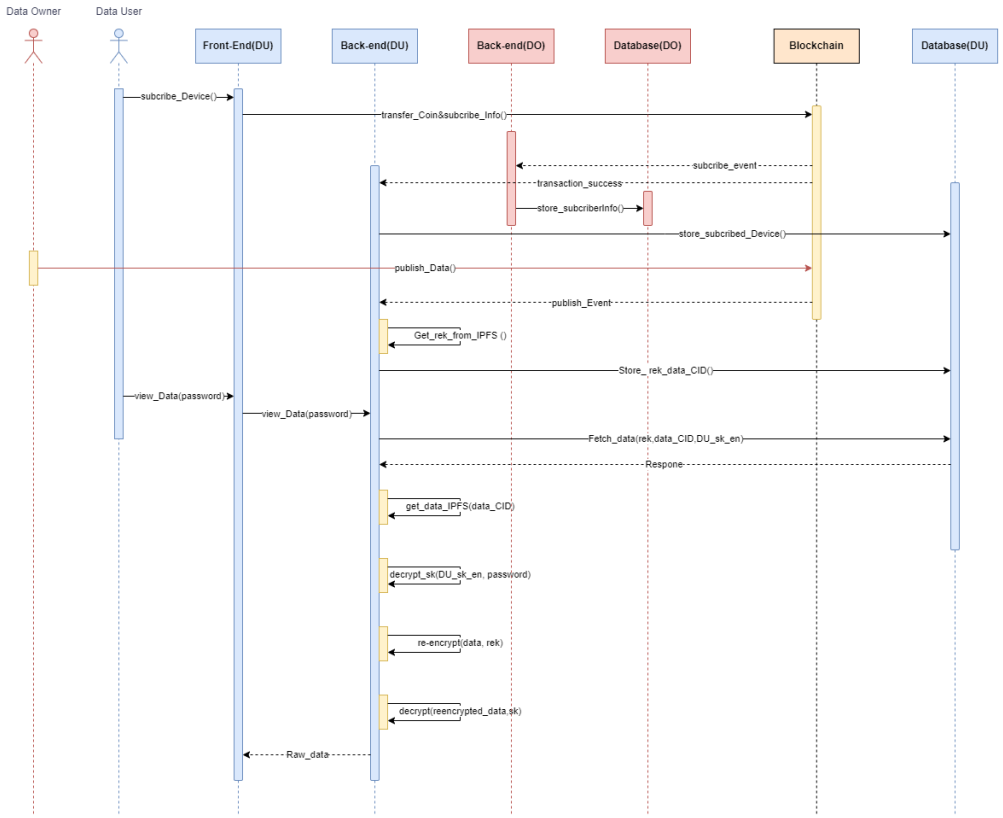


Figure 3.8: *Sequence diagram of DU subscribe future data*

In this process, Data User choose device and fills in the subscription time in the front-end of Webapp. When Data User presses the confirm button, the front-end executes *transfer_CoinaAndpurchased_Info()* through Metamask wallet to transfer the right amount of cost and subscription info to Smart Contract. Then, DU will receive success transaction from blockchain and the Smart Contract will emit an event *Subscribe* to Data Owner. The DMS (back-end of DO) receive the *Subscribe* event from the Smart Contract and query *store_subscriberInfo()* to local database to stores device ID, subscriber ID, subscription start-day, end-day.

Whenever Data Owner perform *publish_Data()* successfully, Smart Contract will emits an event *Newdata* to both Data User and Data Owner. The back-end of DU listen to the event and receive data ID, device ID, from, to, data CID, and re-key CID. DU's back-end execute *get_rek_from_IPFS()* to get list of re-key from IPFS. Then, DU's back-end extracts the re-key corresponding to DU's address from list of re-key. DU's back-end queries *store_rek_CID()* to local database(My SQL) to stores Data User ID, data ID, re-key.

Data User insert password to the front-end of DU, then, front-end post *view_Data(password)* to the back-end of DU. DU's back-end execute *fetch_data(rek,data_CID,DU_sk_en)* to fetch these data from local database (My SQL). After that, DU's back-end perform *get_data_IPFS(data_CID)* to get the encrypted data from IPFS.

After achieving all needed data, DU's back-end executes the following step to get the raw data. First, DU's back-end executes *decrypt_sk(DU_sk_en, password)* to decrypt encrypted DU's sk with password received from Data User. Next, DU's back-end perform *re-encrypt(data, rek)* to re-encrypt the encrypted data receive from IPFS with re-key. Finally, DU's back-end uses *decrypt(re-encrypted_data,sk)* to decrypt re-encrypted data with Data User's secret key and return the raw data to the front-end and show it to Data User.

- Process of subscribe published data is depicted in figure 3.9:

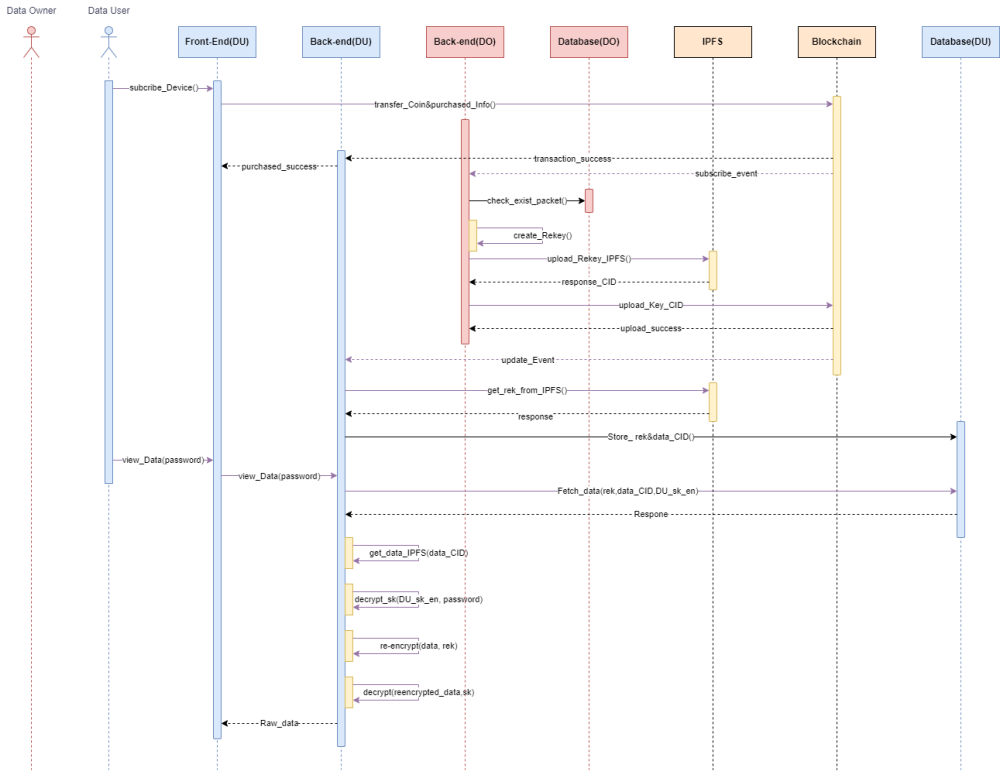


Figure 3.9: *Sequence diagram of DU subscribe published data*

In this process, the published subscribed data stream is similar to the future subscribed data stream. The execution flow differs only in that the DMS (DO's back-end) updates the key and uploads it to IPFS and publishes to Smart Contract.

After the DMS (back-end of DO) receive the *Subscribe* event from the Smart Contract, it execute *check_exist_packet()* to query all data packet info(start-day, end-day, data's pk, data's sk, data CID) that has been published in the subscription period if exist the DO will continue the below step.

To perform the *create_Rekey()*, the DMS execute the following step. First, the Data Owner insert the account password to the front-end of Webapp to decrypt the encrypted Data's sk. After that, for each subscriber, the DMS will create the re-key from the secret key of the data packet and subscriber's public key and append these re-keys corresponding with subscriber address to a .json file. This file, then, is uploaded to the IPFS and response a re-keys CID to DMS and issue *update_Key_CID* to SC.

The remain flow is the same with subscribe future data.

3.4.3.3 Smart contract

Our SC is written in Solidity and it contain the follow properties.

- Constant :

```
uint256 private constant MISECDAY = 86400000;
```

Figure 3.10: *Constant in Smart Contract*

  - MISECDAY is used to store number of millisecond a day.

- Struct:

```
struct Devices {
    address owner;
    string name;
    string describe;
    uint256 pricePerDay;
}

struct SensorData {
    bool isUsed;
    string uri;
    uint256 from;
    uint256 to;
}
```

Figure 3.11: *Struct in Smart Contract*

- Devices: is used to information about the device including DO address, name of the device, device's description and data subscription price per day in Wei (1 ETH = 10^{18} Wei).
- SensorData: will contain meta data about the data packet including a Boolean to indicate this device is used or not, data uri, the start day and end day of this packet.
-

- Mapping:

```
// deviceID to device
mapping(bytes32 => Devices) userDevices;

// deviceID to bool
mapping(bytes32 => bool) public usedId;

// from deviceID to dataID
mapping(bytes32 => mapping(bytes32 => SensorData)) devicesData;

// from deviceID to dataID to  rek
mapping(bytes32 => mapping(bytes32 => string)) rek;

// from DO to DU to transactionID to amount
mapping(address => mapping(address => mapping(bytes32 => uint256))) transactionList;
```

Figure 3.12: *Mapping variable in Smart Contract*

In this mapping, array is not used since it is a complex data type mentioned

in solidity documentation and will consume more gas price when deal with. Our mapping will contain:

  - userDevices: will resolve a Device struct with an device ID .
  - usedId: is used to check if the device ID is used or not.
  - devicesData: will mapping from deviceID to dataID and will resolve a SensorData struct.
  - rek: will return the address of the list of re-encryption key in IPFS with the according deviceID and dataID.
  - transationList: store the amount of Wei that DU have transfered to the SC, and will be deduce each time DU confirm a packet to transfer ETH to DO.

- Event:

```
event Register(
    address indexed owner,
    bytes32 indexed id,
    string deviceName,
    string _decribe,
    uint256 price
);
event Subscribe(
    address indexed from,
    address indexed to,
    bytes32 indexed deviceID,
    bytes32 txID,
    uint256 start,
    uint256 end,
    uint256 price,
    string pk
);

event NewData(
    bytes32 indexed deviceID,
    bytes32 indexed dataID,
    uint256 _from,
    uint256 _to,
    string _dataUri,
    string _keyUri
);

event NewKey(
    address to,
    bytes32 txID,
```

```
31      string keyUri
32  );
33
34  event Confirm(
35      address indexed from,
36      address indexed to,
37      bytes32[] dataID,
38      uint256 amount,
39      bytes32 indexed txId
40  );
```

Listing 3.1: Smart Contract event

Event is used as a way for SC to communicate with the outside programs. When an event emitted the data in the event will be stored inside the transaction log. Therefore, it will remain with the existence of the blockchain and we always can get back emitted event from a specific SC. With indexed field in the event specify that this field can be used as parameter for query the event.

- Register: This event will be emitted when DO register a new device to the system. This event will include information about the owner, id of the device, device name, short describe about the device and finally the price in Wei which which be used to calculate subscription price.
- Subscribe: When ever a DU call subscribe and transfer enough ETH to the SC an *Subscribe* event will emit. This event will *include from* (address of DU), *to* (address of DO), device ID, transaction ID which will be used in confirm data packets, start and end day of the subscription, amount DU have transfer and pk for DO can create re-key.
- NewData: When a data publish by DO this event will be triggered to inform DU about the new data that have been added. The information including id of the data and the deivce, duration of the device and the Uri of the data and the re-key in IPFS.
- NewKey: Emit by DO when DU subscribe duration contain published data packet. This will inform DU subscription's transaction ID and the corresponding re-key.
- Confirm: When DU have validate the data packet. They will confirm packets that they have receive and emit this event to inform packets that they have confirm and the Wei that they have transfer.

3.5 Implementation results

We have implemented a website for DU and DO to interact with the SC on blockchain network. The website will consist 2 different route for DU and DO.

3.5.1 Register and Login

In order to interact with the system, first, both DO and DU have to register an account on the system and link it with an account on the Metamask wallet. The system do not allow to register account with the same Metamask address or username. Users have to fill in username, password, link with a valid Metamask account address and choose their role (Data User or Aggregator) to register with the system (figure 3.13).

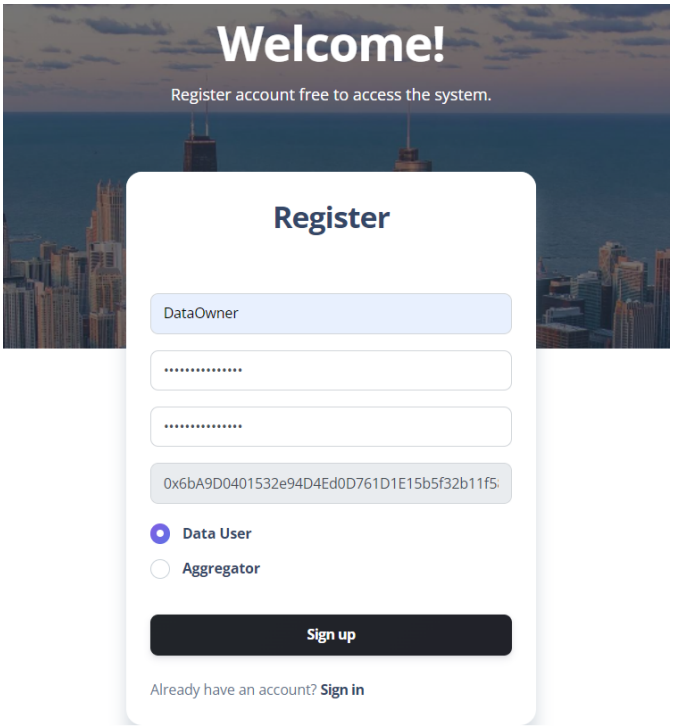


Figure 3.13: *Sign up (Webapp)*

User use the previous register data to login into the system (figure 3.14).

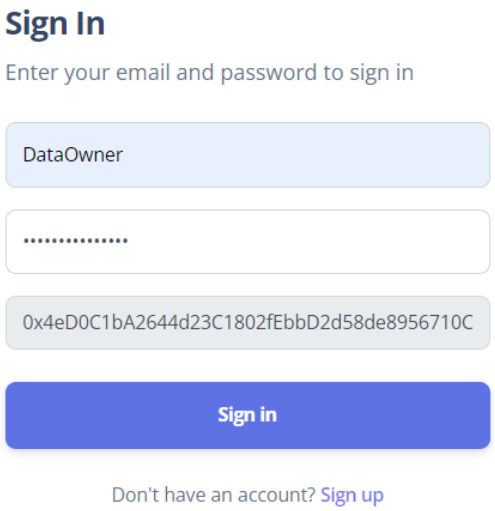


Figure 3.14: *Sign in (Webapp)*

3.5.2 Web application for data user

First, Owner can register their device. The device ID is generated randomly when DO press 'Generate ID'. The device ID are unique for each device (figure 3.15).

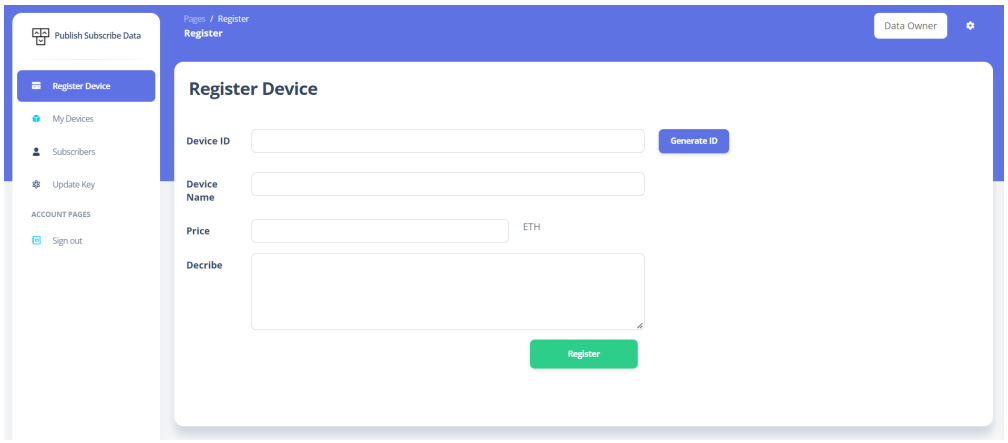


Figure 3.15: *DO register device (Webapp)*

After registration DO can view their device and choose the device that they want to publish the data. The device ID is convert from Unicode to hex data type (figure 3.16).

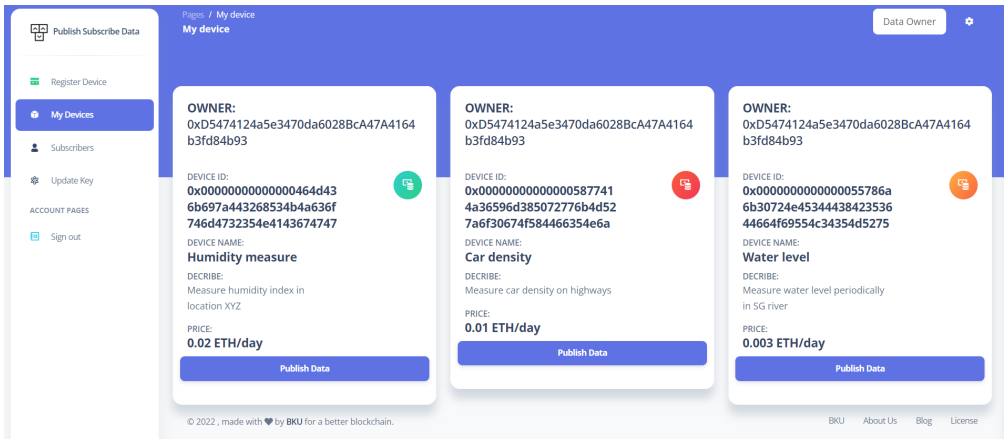


Figure 3.16: *DO's devices (Webapp)*

At publish page DO will choose the duration of the file and upload the file and fill in the login password to decrypt DO's sk and encrypt data sk. The duration of new data must not coincide with the duration of the data that was previously published. They can see the estimate price of the data and number of subscriber to this device in specified time (figure 3.17).

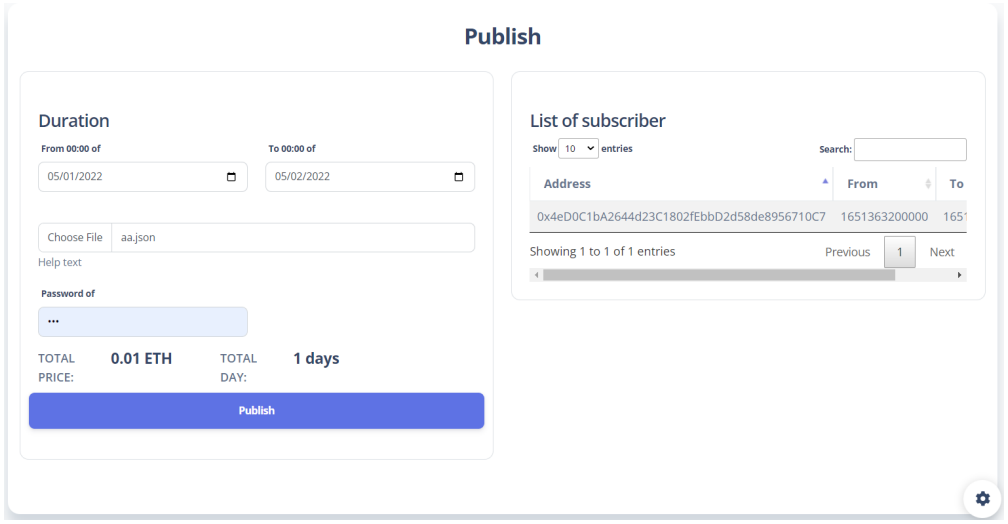


Figure 3.17: *Publish data (Webapp)*

When the period of a DU subscription includes some data that has been released, DO can view all transactions that need to update re-key by choose "Update Key". DO has access to detailed information such as Transaction ID, Subscriber ID, Device ID, subscription period, and pricing.

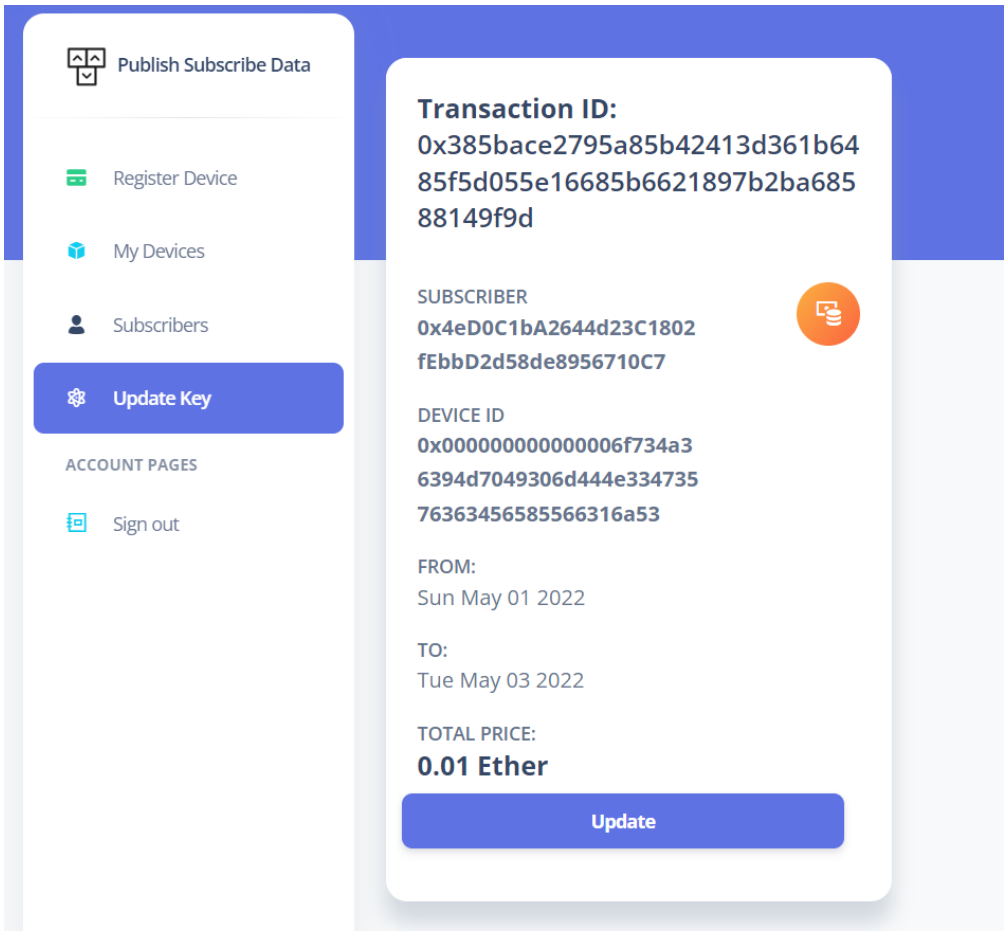


Figure 3.18: *Transaction need to update*

DO can choose a transaction that they want to update re-key. After press update, they can see list of data packet belong to this transaction that need to update the key. DO perform update key by press update.

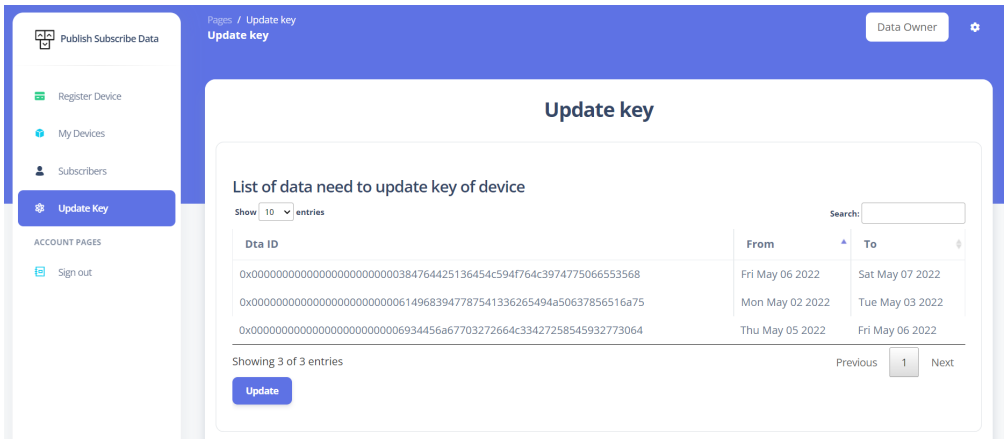


Figure 3.19: *Update re-key*

3.5.3 Web application for data user

For DU they can see all available device by choosing "View Devices". They can see the price/day and device detail such as: owner of device, device id, device name and description (figure 3.20).

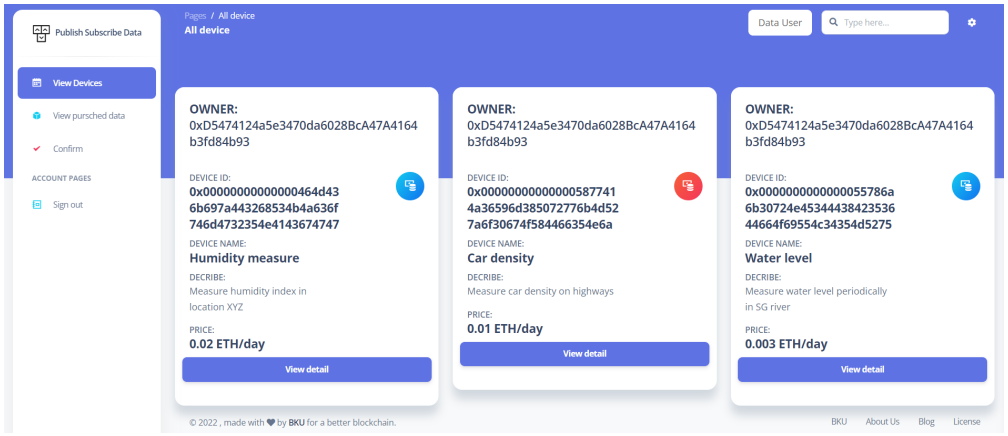


Figure 3.20: *Available device (Webapp)*

DU can choose a device that they want to subscribe. After press view detail, they can see form for subscription. This form will contain the device detail and duration for DU to choose for subscription. When DU press subscribe, they have to transfer the according ETH to the Smart Contract. The duration of new subscription must not coincide with the duration of the previous subscription (figure 3.21).

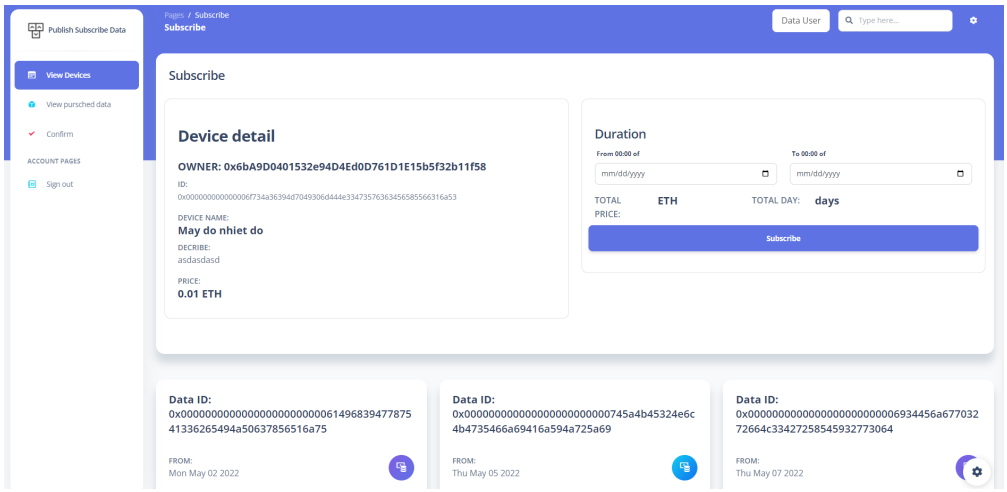


Figure 3.21: *Subscribe device (Webapp)*

Below the subscription form, DU can view brief summaries of previously published data include: Data ID, from, to, data uri, re-key uri (figure 3.22).

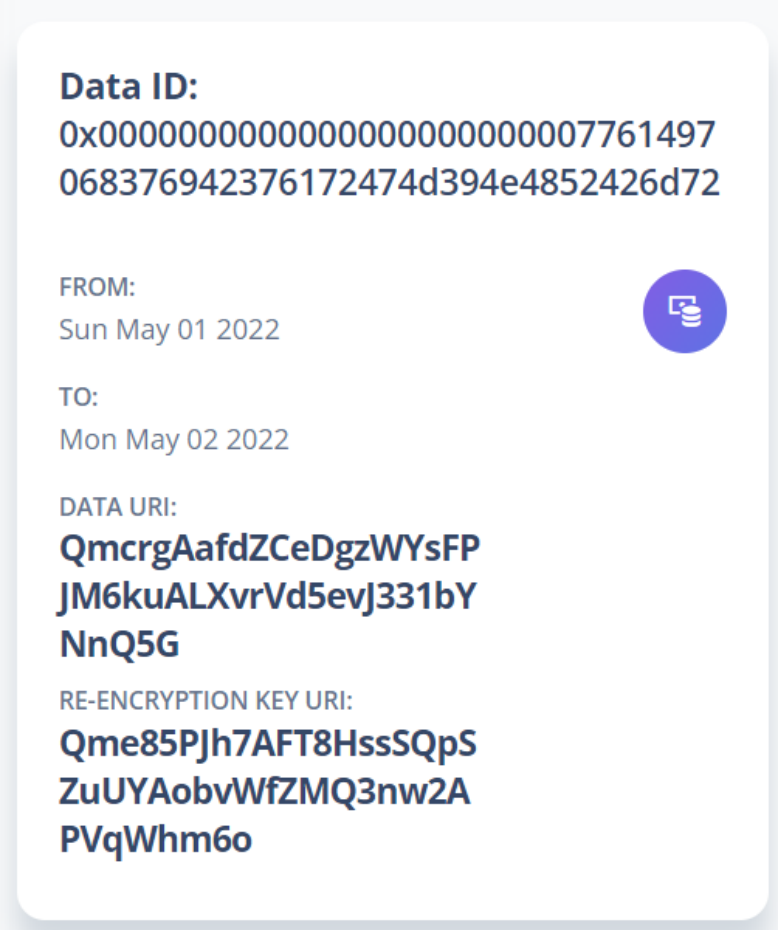


Figure 3.22: *Data summary card (Webapp)*

Data User can choose "View purchased data" to view all device that they have received data (figure 3.23).

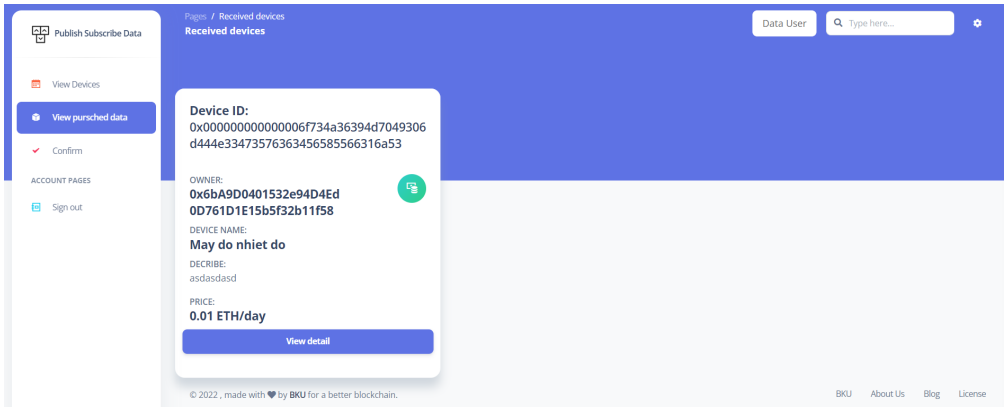


Figure 3.23: *Received Devices Data (Webapp)*

When Data User press on a device, they can see all the data that has been published by that device in their subscription period (figure 3.24).

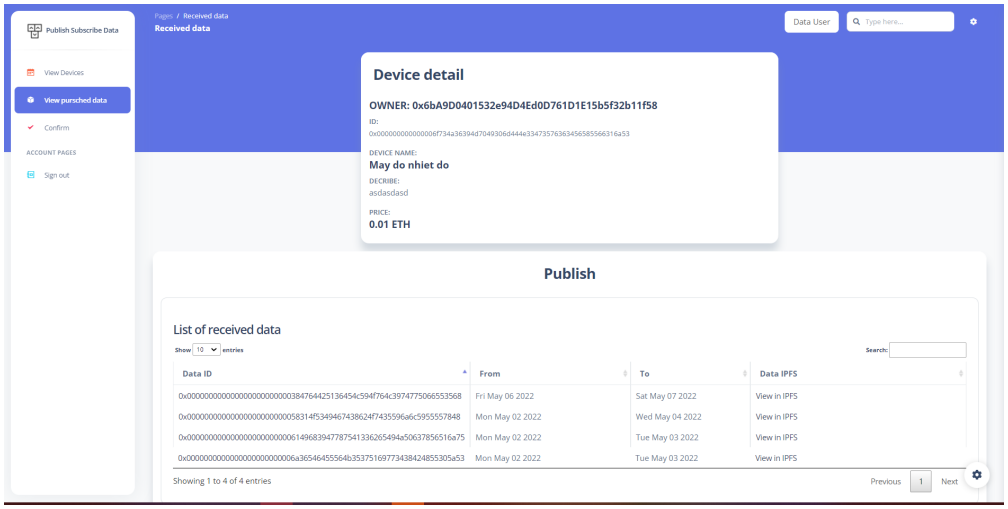


Figure 3.24: *Received Data (Webapp)*

Data User can view their data detail by click on the data on the list of received data. To view the plain text of data, Data User have to fill in their account password and press "Decrypt Data" to decrypt and view raw data (figure 3.25).

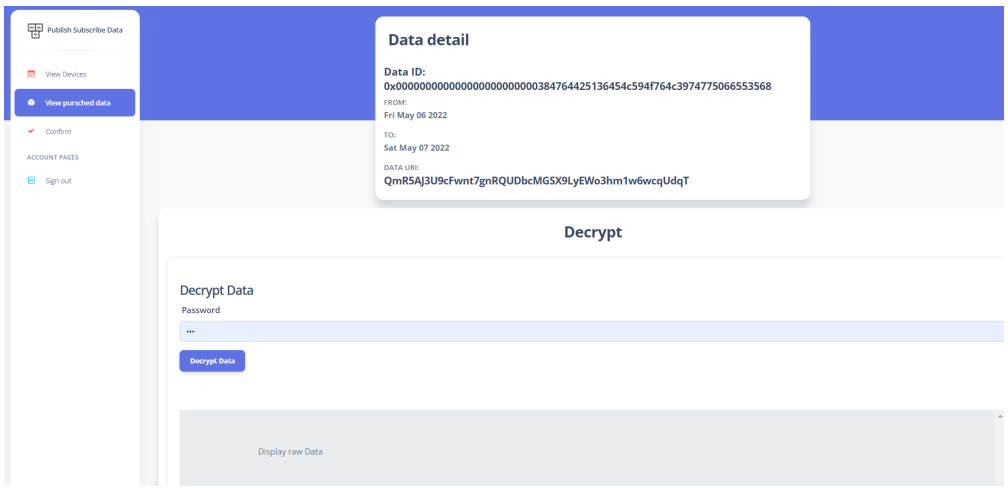


Figure 3.25: *Decrypt Data (Webapp)*

We also create a page for Data User to view and check whether they confirm receive a data or not. Data User can choose "Confirm" to view all subscription (figure 3.26).

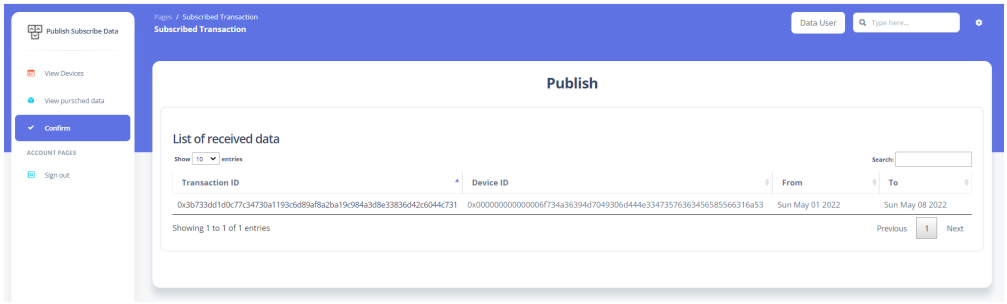


Figure 3.26: *All Subscription (Webapp)*

After choose a device , Data Owner will see all data published by this device and their confirm state. Data Owner choose data and click "Confirm Packet" to notify to Smart Contract that the data they receive is valid. The Smart Contract will transfer the corresponding ETH (received previously from DU) to Data Owner (figure 3.27).

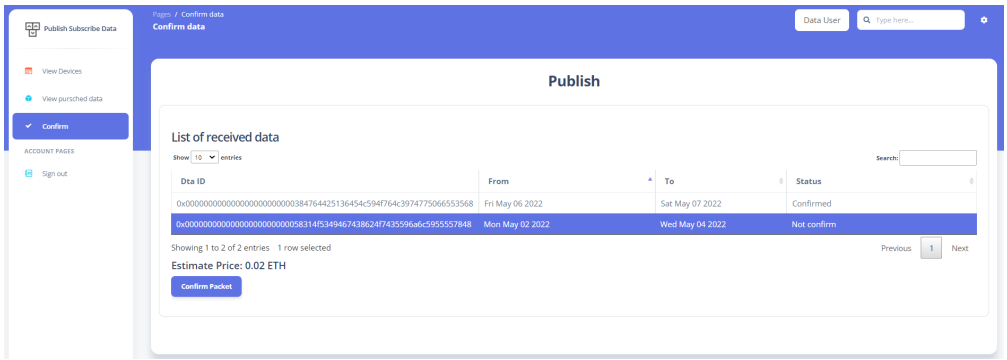


Figure 3.27: *Confirm Data (Webapp)*

4 Experiment and Evaluation

4.1 Risk mitigation in Smart Contract

By leveraging smart contracts and Blockchain, the proposed model eliminates the third-party in the traditional model. All operations in the data exchange process will be executed via smart contracts. Therefore, the implementation of smart contracts is a critical requirement.

However, the programming language for SC such as *Solidity* is still developing. Therefore, there are lot of security risk lie in the programming language itself. Since smart contract can hold a lot of value in itself, smart contract can become the subject for hacker. Our team has identified several vulnerability that can lead to losses in SC and apply prevention in our implementation to avoid them.

4.1.1 Re-reentrancy attack

This attack happen when a function in SC calling to a function in an external malicious SC. Then the malicious SC now then can call back to the original one. With this flow the attacker can drain the fund of the smart contract. We can see the example in Figure 4.1

In Figure 4.1 When we call transfer function. It will check balance of the caller. If the condition is satisfied, it begins to transfer the cryptocurrency to receiver without updating the balance. The victim SC will call the malicious SC in order to transfer cryptocurrency. Now the malicious SC call transfer again since the balance is not updated the attacker still can transfer normally. Therefore an attacker can drain all the money inside the victim's contract.

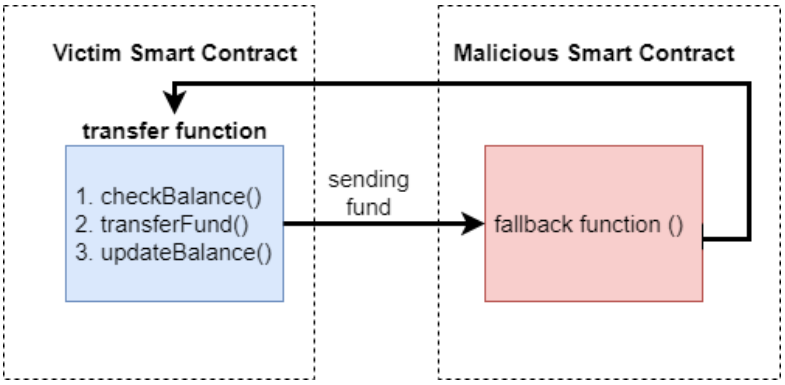


Figure 4.1: *Re-entrancy attack example*

This attack can be avoided by using Checks-effects-interactions pattern in our SC. This pattern indicates that we have to check the condition first then resolve all the effects, and finally make interaction to external contract. In our SC implementation, we have to transfer cryptocurrency to DO and DU on data confirmation. Therefore we have applied this pattern to prevent Re-entrancy attacks.

4.1.2 Integer overflow and underflow

Just like in some low-level programming languages such as: C, C++, C# and Java. This happen when unsigned integer reach its bytes size. For instant, a uint8 (contain 8 bit) variable can take a maximum value is $11111111_2 = 255$ when we add 1 unit to this number it will become $00000000_2 = 0$ this incident is called as overflow. Underflow will happen in same way when we try to subtract 1 unit from 000000000_2 it will become $11111111_2 = 255$. These faults are a major vulnerability for hackers to perform inappropriate actions on SC.

Previously, before Solidity version 0.8.0, to deal with this problem we can use a library called *SafeMath* to prevent underflow and overflow. However, from Solidity version 0.8.0 and above this problem is taken care of by the programming language itself. So our implementation had applied Solidity version 8.0 to avoid this security risk.

4.2 Data security in proposed model

Data security of the proposed model is constructed according to four main criteria as follows:

- **Confidentiality**: The asymmetric encryption based on public and private key pairs combine with re-encryption will enables only the DO and authorized DU to access and read the shared data. All the stored SK in the database is encrypted with a secrete key phrase, therefore all the key is confidential.
- **Integrity**: Since IPFS is a distributed content addressing database, Which mean that for the CID of a file is like the hash of a file. Each content will generate a unique CID, therefore with a given CID we don't need to worry about the integrity of the data . Additionally, Blockchain ensures that transactions are not manipulated and the entire transaction process is transparent and traceable.
- **Availability**: The decentralized architecture of Blockchain networks and files storage to avoids single-point of failure and mitigates DoS attacks to the services provided by smart contracts and thereby increases the service availability.
- **Immutability:** DU can trust that once DU received the data and re-key CID. No one can revoke the that privilege including DO thanks to the combination of SC on blockchain and decentralized database (IPFS). Data once publish will be record permanently and DU always can retrieve the data.

4.3 Testing Smart Contract

In order to test the smart contract, our team uses truffle. Truffle is a framework for constructing decentralized apps on Ethereum, with services such as compile and build, testing, deployment, and integration. Truffle allows users to do unit tests on smart contracts written in Javascript. Furthermore, Ganache is used by the team to construct the Ethereum environment (also a tool of the Truffle framework). When performed, Ganache builds a virtual Ethereum network on a personal workstation, pre-initializes 10 accounts, and each account holds 100 ether, allowing users to utilize in the building of DApps.

We have constructed a script written in JavaScript to test the ability of the Smart Contract. The script covers nearly all test situations that might occur in practice. Below is the description of the script file to test the smart contract

First, we create an exception.js file to construct some utility functions.

```
1 const PREFIX = "Returned error: VM Exception while processing
   transaction: ";
2
```

```
3  async function tryCatch(promise, message) {
4    try {
5      await promise;
6      throw null;
7    } catch (error) {
8      assert.isNotNull(error, "Expected an error but did not get one
        ");
9      assert.equal(
10       error.message.startsWith(PREFIX + message),
11       1,
12       "Expected an error starting with '" + PREFIX + message + "'
          but got '" + error.message + "' instead"
13     );
14   }
15 }
16
17 async function tryCatchNoReason(promise) {
18   try {
19     await promise;
20     throw null;
21   } catch (error) {
22     assert.isNotNull(error, "Expected an error but did not get one
        ");
23   }
24 }
```

Next, we begin with require some needed modules in test file. As can be seen, line 1 using to load the smart contract itself to the test file. Line 2, 3 load the catch error function from exception.js

```
1 const PubSub = artifacts.require("PubSub");
2 const catchRevert = require("./exceptions.js").catchRevert;
3 const catchRevertReason = require("./exceptions.js").
   catchRevertReason;
```

After prepare the environment to test the smart contract, we start to coding the scenario for each case. Below is some brief code show how we implement the test file.

```
1   it("Should deploy", async () => {
2     const pubsub = await PubSub.new();
3     assert.isNotNull(pubsub, "Can't deploy contract");
4   });
5
6   it("Should register", async () => {
7     const pubsub = await PubSub.deployed();
8     deviceID = stringToBytes32("Test 1");
```

```
9     await pubsub.register(deviceID, "Heat Sensor", "Measure
       temperature in Bien Hoa city 100 time a day", milliEtherToWei
       (1));
10    let device = await pubsub.getDevice(deviceID);
11    assert.isNotNull(device);
12  });
```

Each test case script is contained in *it()* function. In the first time, we have to deploy the smart contract in the testing environment by using *PubSub.new()* in line 2. For the continue test, we reuse the deployed smart contract by using *PubSub.deployed()*.

The test file contains a total of 10 scenarios to test the smart contract from the basic one to the most complex case. The detail of each testcase is described as below.

1. **Testcase name**: Should deploy.

   Description: Deploy Smart Contract in Truffle environment.

   Expected result: Deploy Smart Contract success.

2. **Testcase name**: Should register.

   Description: Data Owner register a new device with non exist device ID.

   Expected result: We can retrieve the device information from the deviceID.

3. **Testcase name**: Should not register - Duplicate device ID.

   Description: Data Owner register a device with exist device ID in Smart Contract.

   Expected result: Can not register. Raise an error message with duplicate device ID when register.

4. **Testcase name**: Should subscribe.

   Description: Data User should be able to subscribe to a device that has registered in Smart Contract.

   Expected result: Subscribe device success.

5. **Testcase name**: Should not subscribe - Lacking 1 wei.

   Description: Data User subscribe to a device that has registered in Smart Contract. However, the amount of fee Data User tranfer to SC is lacked 1 wei to perform the payment.

Expected result: Can not subscribe device. Raise a "not enough money to subscribe" error.

6. **Testcase name**: Should not subscribe - Not exist device.

   Description: Data User subscribes to a device that has not registered in Smart Contract.

   Expected result: Can not subscribe device. Raise a "non exist device" error.

7. **Testcase name**: Should publish.

   Description:Data Owner publish a data with a nonexistent data ID belonging to their device registered on SC.

   Expected result: Data Owner able to publish.

8. **Testcase name**: Should not publish - duplicate ID.

   Description: Data Owner publish a data with duplicate data ID with another packet from their device registered on SC.

   Expected result: Can not publish data. Raise a duplicate data ID error.

9. **Testcase name**: Should not publish - Not owner of a device.

   Description: Data Owner publish a data of a device which do not belong to them.

   Expected result: Can not publish data. Raise a non-self-owned devices error.

10. **Testcase name**: Confirm multiple packet and check balance.

    Description: Perform the test from the beginning. Data Owner register a new device. Data User subscribe to that device in 2 days period. Data User transfer enough money to Smart Contract. Data Owner publish 2 packet data to Smart Contract corresponding to 2 days. Data User confirm receive 2 data packet. Smart Contract transfer the money to Data Owner.

    Expected result: When Data User confirm received data packets successfully. The balance of Data Owner is expected to increase by the amount of 2 days worth of fee.

After execute the test file, the testing result is summarized in Figure 4.2, which shows that all 10 scenarios are successfully passed.

Figure 4.2: *Test smart contract result*

4.4 Gas fee of transaction on Blockchain

Gas fees are payments made by users to compensate for the computing energy required to process and validate transactions on the Ethereum blockchain. "Gas limit" refers to the maximum amount of gas (or energy) that you're willing to spend on a particular transaction.

Besides security threats, gas fee optimization is another parameter that should be considered. The gas fee will affect a lot on the topic's practicality. One of the ways that we can reduce the cost fee on each function call without changing the structure of the code is to put the appropriate data location of the variable and function argument. There is 3 location we can store our variable $storage, memory, calldata$. The fee of those 3 locations is different and we can decide where to store depending on our purpose.

- *Storage* is the most expensive one since the lifetime of the variable is limited to the lifetime of a contract.
- The less expensive one is *memory*, which is mostly used for function arguments since it only lasts in the function call, you can read and modify the variable in memory.
- The least one is *calldata*, which behaves mostly like memory but is not able

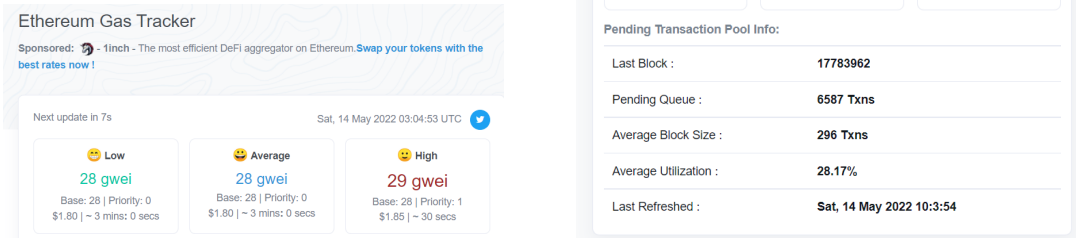
to modify the variable. In our SC we mostly chose *calldata* for our complex variable location which will reduce the gas fee for each function call.

Regardless of the content, the length of each data CID is fixed at 46 characters (base58), so the CID size is the same for all data. The gas price of our SC will not depend on the length of the data packet because we only store the link of data (CID of IPFS) on the blockchain. After perform multiple test on our system, we conduct a table of approximate gas fees for each interaction with the smart contract as below:

Action	**Gas**
Deploy Smart Contract	2606585
Register Device	174668
Publish Data	216969
Subscribe Device	70763
Confirm (2 data packet)	58359

Table 4.1: *Gas fee table*

From gas fee used we can compute the cost for each transaction to those action by using this formula: $fee = gasFee * gasPrice$ while gas fee will stay the same for the same workload, but the gas price will vary depend on the blockchain network. Each user can offer their own gas price, the higher the gas price the more likely their transaction will get picked. If the gas price is too low the transaction will be never picked. We also can get the suggested gas price for different blockchain network. Figure 4.3 show the gas price for 2 popular blockchain network Ethereum and Binance Smart Chain (BSC) network. The price is not always the same and will change from time to time.



(a) *Ethereum* **(b)** *Binance Smart Chain*

Figure 4.3: *Gas price*

The gas price and crypto price was gathered from 2 famous block exploring website Etherscan [17] and Bscscan[18] at 12:08 PM on 5/14/2022. In this calculation, we will use the average gas price: 28 Gwei for Ethereum and 6 Gwei for BSC and use the price of ETH and BSC is approximately 2,015$ and 293.53$ sequentially.

Action	**Gas**	**Cost(ETH)**	**Cost(USD)**
Deploy Smart Contract	2606585	0.073	147
Register Device	174668	0.0049	10
Publish Data	216969	0.0060	12
Subscribe Device	70763	0.0020	4
Confirm (2 data packet)	58359	0.0016	3

Table 4.2: *Cost in Ethereum*

Action	**Gas**	**Cost(BSC)**	**Cost(USD)**
Deploy Smart Contract	2606585	0.0156	4.6
Register Device	174668	0.00105	0.3
Publish Data	216969	0.00130	0.4
Subscribe Device	70763	0.00042	0.1
Confirm (2 data packet)	58359	0.00035	0.1

Table 4.3: *Cost in BSC*

From Table 4.2 and Table 4.3 we can see that all transaction cost in ethereum

is much higher than the cost in BSC ($\approx$ 32 time). With lower cost, DU and DO can interact with SC easier and gain more benefit and don't have to worry so much about the transaction fee. So it is more reasonable to deploy and interact with the SC in BSC to lower expenses and increase the feasibility of the system.

5 Conclusion

5.1 Main contribution

The research team's theme focus on resolving a critical and pressing issue in today's IoT: data exchange. As a consequence, the authors address the issue of lack of transparency, integrity, and data security in the data-sharing process. In this thesis, we have designed an architect that allows DO to share the data of their IoT devices to DU in Publish-Subscribe manner through Blockchain. We also use re-encryption to utilize the storage as well as enhance the security for the data in a transparent environment like blockchain. A DApp is created by our team to prove the feasibility of our design that allow DO and DU to interact with the blockchain.

5.2 Achievement

Following are some of the accomplishments that our system achieved as a result of the testing and assessment procedure.

- Convenient and easy-to-use web-based application for both DO and DU
- Allow user to communicate through blockchain.
- Using smart contracts eliminate the need for third-party middlemen.
- Prevent re-entrency attack.
- Using re-encryption to enhance the confidential of the data but still minimize storage.
- All important information is store in cipher text.

- Each data is encrypted by a unique key. As a result, even if one key is compromised, all other data remains secure.
- Prevent exposure of private information of DO, DU and data packet.
- The system guaranty three security requirements: Confidentiality, Integrity, Availability

5.3 Limitation

Our team, however, discovered certain limitations in the current model throughout the systems development and evaluation.

- Each transaction will cost fee, the gas fee will remain the same but gas price will vary depend on differences blockchain network.
- Each transaction will have a small delay depend on the network.
- DO still need a management system to manage all the key pair of the data packet.
- DU still need to manually confirm the data receive from DO.

5.4 Future Improvement

Within the scope of this report, the goal of the research team is to propose and prove the feasibility of the topic, not yet focusing on developing it into a practical application. However, in the future, the research team proposes a number of improvements to make the test application more complete and can become an application in practice:

- Apply the system to real-world IoT applications such as smart cities, smart healthcare devices, and smart agriculture, where device owners can collect data and only share it with those who need it.
- Add rating and commenting function for a more objective and reliable data exchange environment
- Improve the user interface, arrange the components in a more intuitive way
- Reduce more gas fee for all action. Making deployment and interaction with SC cheaper.

Bibliography

[1] Yang, Caixia and Tan, Liang and Shi, Na and Xu, Bolei and Cao, Yang and Yu, Keping, "Authprivacychain: A blockchain-based access control framework with privacy protection in cloud," *IEEE Access*, vol. 8, pp. 70 604–70 615, 2020.

[2] Nguyen, Truc and Pham, Hoang-Anh and M, Thái, *Leveraging Blockchain to Enhance Data Privacy in IoT-Based Applications*, 12 2018, pp. 211–221.

[3] Le Trung Kien and Pham Thi Ngoc My and Nguyen Hoai Quoc Trung, *Hien thuc giai phap quan ly quyen truy cap trong cac ung dung IOT su dung cong nghe Blockchain*.

[4] "Re-encryption figure," accessed: 2022-02-14. [Online]. Available: https://www.nucypher.com/proxy-re-encryption

[5] "Number of internet of things (iot) connected devices worldwide from 2019 to 2030." [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[6] Luke Conway, "Blockchain explained, 2021." [Online]. Available: https://www.investopedia.com/terms/b/blockchain.asp

[7] "Ethereum improvement proposals." [Online]. Available: https://eips.ethereum.org/

[8] "Solidity v0.8.0 breaking changes." [Online]. Available: https://docs.soliditylang.org/en/v0.8.10/080-breaking-changes.html

[9] Petar Tsankov and Petar Tsankov and Dana Drachsler-Cohen and Arthur Gervais, and Florian Bünzli and Martin Vechev, "Securify: Practical security analysis of smart contracts."

[10] "Smart contract weakness classification swc-114 transaction order dependence." [Online]. Available: https://swcregistry.io/docs/SWC-114

[11] "Smart contract weakness classification swc-107 reentrancy." [Online]. Available: https://swcregistry.io/docs/SWC-107

[12] Karen Rose and Scott D. Eldridge and Lyman Chapin, "The internet of things : An overview understanding the issues and challenges of a more connected world," 2015.

[13] Nallapaneni Manoj Kumar and Pradeep Kumar Mallick, "The internet of things: Insights into the building blocks, component interactions, and architecture layers," *Procedia Computer Science*, vol. 132, pp. 109–117, 2018, international Conference on Computational Intelligence and Data Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918309049

[14] Karen Rose and Scott D. Eldridge and Lyman Chapin, "The internet of things : An overview understanding the issues and challenges of a more connected world," 2015.

[15] Raymond Chengy and William Scotty and Bryan Parnoz and Irene Zhangy Arvind and Krishnamurthy and Thomas Andersony, *Talek:a Private Publish-Subscribe Protocol*.

[16] "Inter planetary file system." [Online]. Available: https://www.geeksforgeeks.org/interplanetary-file-system

[17] "Ethereum gas explorer," accessed: 2022-05-14. [Online]. Available: https://etherscan.io/gastracker

[18] "Binance gas explorer," accessed: 2022-05-14. [Online]. Available: https://bscscan.com/gastracker

Appendix

Link to source code of Implement pub sub in blockchain project from GitHub: https://github.com/gacontrolai/Implement-pub-sub-in-blockchain

A Prerequisite software

The following software must be installed first to be able to launch the project. The versions of these software should be the same as the version when we launch the project to avoid software incompatibility errors.

Link to download these software are below:

- **MySql** (v8.0.28). Link to download: https://dev.mysql.com/downloads/installer/
- **Nodejs** (v14.17.4) and **npm** (v6.14.14). Link to download: https://nodejs.org/en/download/
- **Ipfs Desktop** (go ipfs v0.12.2). Link to download: https://ipfs.io/#install
- **Truffle** (v5.5.10). Link to download: https://www.npmjs.com/package/truffle
- **Meta mask** Browser extension.
- **Ganache**(optional) (v7.0.3). Link to download: https://trufflesuite.com/ganache/

B Build

B.1 Pull repository

First, pull the following repository Pub/Sub in Blockchain from GitHub.

B.2 Install nodejs packages

Run command `npm install` to install all the relevant packages.

B.3 Initailize database

Open file **.env** in the main repository and change variable **DATABASE-NAME** to **lvtn**, **SALTROUND** to 10 and change **DATABASEPASS** and **DATABASEUSER** according to your database password. Then run command `node init_database` to create a database.

B.4 Deploy smart contract

The source code of the smart contract located in **contracts/PubSub.sol**

The configuration of the SC deployment can be found in **truffle-config.js** file. The default blockchain network that we use is Ganache blockchain network. If you want to change the network, then change 3 variable **host**, **port**, **network_id** in figure 1 to your desire network. After that, Smart contract can be deployed by calling `truffle deploy`

```
networks: {
  // Useful for testing. The `development` name is special - truffle uses it by default
  // if it's defined here and no other network is specified at the command line.
  // You should run a client (like ganache-cli, geth or parity) in a separate terminal
  // tab if you use this network and you must also set the `host`, `port` and `network_id`
  // options below to some value.
  //
  development: {
    host: "127.0.0.1", // Localhost (default: none)
    port: 7545, // Standard Ethereum port (default: none)
    network_id: "*", // Any network (default: none)
  },
```

Figure 1: *Blockchain network configuration*

After successfully deploy the smart contract, open file **buid/contract-**

s/PubSub.json copy the address of the smart contract belong to the destination blockchain network ID figure 2 and the **abi** (if SC is modified). Then open file **public/javascripts/metadata.js** paste the value to **addressOfContract** and **abi** variable.

Figure 2: *Smart contract address*

Change **networkID** variable in **subscribesEventSC.js** file to match the destination network.

Note: Since one database is designed and build for only 1 smart contract. If the smart contract is deployed again, ***the database must be initialized*** *for data consistency.*

B.5 Start IPFS

IPFS has to be started before running the application. IPFS can be started by running the ipfs desktop version or by the command `ipfs daemon`

B.6 Run the application

The application can be run with `npm run start` (for normal deployment) or `npm run dev` (for development). If the application is successfully deployed the terminal will show the massage inform the port that the application is running on.

B.7 Test the smart contract

The test file for the smart contract in located in **test/test_pubsub.js**. To run the test you can run the command `truffle test`.

Note:This command will deploy a new test smart contract and run all the test scenario every time it runs. Therefore, it will cost fee every time it run so it's advisable to check your balance before running this and run this on the local blockchain network or testnet only.

C Manual

We have listed all the interaction and functionality as well as how to use the website in Section 3.5.