University of Pittsburgh

School of Computing and Information
Report for Independent Study

Report for timed-lock message release deployment based on blockchain

Student Name : Haocheng Wang
Student Email. : HAW188@pitt.edu

April 2023

Abstract

Most information publishing platforms have a feature that allows users to schedule messages for publication at a specific time. Typically, these messages are stored on a company's centralized server and released at the designated time. However, this method is not entirely secure. If someone were to hack the company server or engage in unauthorized operations, the messages could be leaked prematurely.

In this blockchain-based time-lock release system, users no longer have to worry about such leaks. Blockchain technology offers tamper-evident and decentralized features, and this system utilizes secret sharing, multiple encryption, and authentication mechanisms to prevent message leaks when using delayed publishing. This independent study project aims to develop a time-lock messaging system based on blockchain technology, featuring a user interface similar to Twitter. Users can schedule messages for publication at specific times by focusing on the `message content`, `delay duration`, `secret-sharing threshold`, and `secret-sharing number`. A designated number of users registered on the blockchain network will be a role of transporting and reconstructing the message at the scheduled time.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to Blockchain

Blockchain technology, proposed in 2008, has been employed in various fields since its inception. The first known application of blockchain likely stems from Bitcoin, which effectively leveraged blockchain's features, elevating the use of cryptocurrency to new heights. Additionally, numerous applications are being developed or will be developed based on blockchain's unique features, such as Ethereum.

Smart contracts, a popular software development on the Ethereum platform, are based on blockchain technology. Due to their decentralized, durable, and auditable nature, smart contracts offer numerous advantages over traditional software. For instance, conventional transaction systems are centralized, with each transaction requiring verification through a central trust (e.g., a central bank), inevitably leading to cost and performance bottlenecks on central servers. In contrast, blockchain eliminates the need for a third-party centralized system. Consensus algorithms within blockchain maintain data consistency across a distributed network. Owing to its anonymity, users can interact with the blockchain using a generated address that does not reveal their true identity.

Blockchain technology allows data that record transactions to be accessed by anyone. However, thanks to its ingenious design, combined with cryptography and consensus mechanisms, the way data is recorded in the blockchain makes it extremely difficult to modify a single piece of data without altering all subsequent data records. Below is an image illustrating the basic structure of blockchain: 1.1

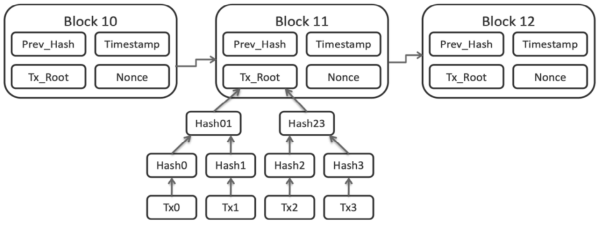

FIGURE 1.1: Blockchain structure

According to these advantages of blockchain, blockchain are widely used in a large number of fields. In this article, I deploy a time-lock data release based on Ethereum and I will introduce it in the next part.

1.2 Time-lock data release system

We can notice that `Twitter` has a function of schedule publish. The system extension twitter's interface design. Specifically, it is not to make too many changes in terms of user experience. Comparing to original Twitter UI, user only need to add two more parameter which is secret sharing threshold and number and all of the evaluation will be not noticeable to the user. When time is up, everyone can get the message. Message revealer can also verify the publish message is same to publish or not.

1.3 Outline

1. Chapter 2 talks about the basic component of dapp.
2. Chapter 3 give more details about each components including how to design and realize function.
3. Chapter 4 introduces multi-layer encryption and Verification mechanism.
4. Chapter 5 give the whole process of how to operate this system.
5. Chapter 6 summarize the shortcomings of the system.

Chapter 2

Basic process and component of dapp

Nowadays, there are many coding language to develop dapp. In this project I use 'Solidity' because it is the most popular language nowadays. In order to realize the interaction between Ethereum contract and user node, I use `Web3.js` based on `Javascript`. `Web3.js` is a general library that allow developers to interact with a remote or local Ethereum node using HTTP, IPC, or WebSocket. Using this library, it can develop websites or clients that interact with the blockchain. After that, I use `React.js` and `Next.js` to build a web interface to present my service. The web interface referenced from Twitter UI. It is convenient for user under understand how to use it because of large user base.

In this deployment part, I choose `Truffle` as a framework. `Truffle` brings together a variety of functions. It provides great convenience to setup the basic component just like Ethereum compile, deploy and front-end construction. In Truffle framework it is easy for `web3.js` to invoke `.json` file which the output file of Solidity compile, to realize interaction between Ethereum contract and Front-end. Then I use Ganache to be the blockchain network. Ganache is a good test blockchain network deployed on local environment. It not need to acquire ETH. In the Gananche test network, user have 10 EOA and adequate curruency.

In the next part, I will give more details about each component.

Chapter 3

System design

3.1 Components of system

The system composed by 1.Shamir's secret sharing, 2.Ethereum function, 3.Truffle deployment and 4.Front-end web. At first, I want to introduce the core encrypt algorithm in the system: Shamir's secret sharing.

3.1.1 Shamir's secret sharing

This system we want to using distributed feature to realize scheduled release function. In normal centralized system, there is the risk of centralized system data stolen. One of the solution is blockchain because it is distributed. In this system, we split and encrypt user's plaintext into many sharing and send to Holder. Holder will use a time-lock function to hold and release sharing when the time up. Then everyone can get sharing and assemble to plaintext. Normally, if user want combine sharing and get plaintext, they need collect all fragments. However, what if holder do not release sharing in time leads to others can not get all fragment on time? It will lead plaintext can not be shown on time. Or receiver decrypt ciphertext in advance? It will lead to early leakage of information. This is reason why we use `Shamir's secret sharing`.

In secret sharing, sender split plaintext into N fragment and set number T to be a threshold. When assembler get at least T number fragments, assembler can restore completed plaintext. We use this method in our system. Under normal circumstances,

each receiver can not decrypt in advance by their own fragment and prevent some receiver not publish fragment on time because satisfy the threshold number fragment to restore plaintext. It has fault tolerance.

In this system, `Shamir's secret sharing` to be a script in sender and assembler node, interacting with smart contract. The script provide some function API: plaintext setting, split plaintext in to ciphertext fragments and recombine ciphertext into plaintext.

In the next part, I will introduce the Ethereum function.

3.1.2 Ethereum function

In this part, I design 3 kinds of contract. They are `Sender`, `Holder` and `Assembler` and all of them correspond to specific EOA. The following part only the basic process of `Secret Sharing`. In the next section, I will give more evaluation of the system.

In `Sender` part, the sender node which is off-chain get a plaintext and split it to many `sharing`. Then the Sender node randomly select `sharing` and `Holder`, sending `sharing` to Holder through on-chain contract with user-specified schedule release time. Sender will also save a receipt to record which sharing gives to which B.

In the `Holder` part, when Holder received a sharing, it will generate a 'Old(Time stamp)'. The Old(Time stamp) means the time when holder receive the sharing. When holder want to send sharing to Assembler, it will determin the 'Now(Time stamp)' is bigger than 'Old(Time stamp)+delay time'. If the condition satisfy, it will send, else not.

After `Assembler` want to reveal plaintext, it will try to assemble all of the sharing he have. If the sharing quantity is sufficient, the assemble node will assemble the plaintext and published else not. Finally, the assemble will check he receive all of the sharing or not. If not, he will report to Sender. According to Sender receipt, he will know which holder violate and give punishment.

After we have the completed function, we need to compile and deploy Ethereum.

3.1.3 Truffle deployment

The most popular solution of integrate all dapp components is truffle. In truffle part, user use truffle command to realize interaction among 'front-end', JS and Solidity. In truffle, you can find many box to quickly build the framework. Here I choose the `React.js` to be the front-end part. It is easy to establish it with truffle.

After deploy all contract in block chain system with truffle, I would like to show the web interface.

3.1.4 Front-end web

This project use `React.js` and `Next.js` to be the Front-end frame. Here is the main UI:

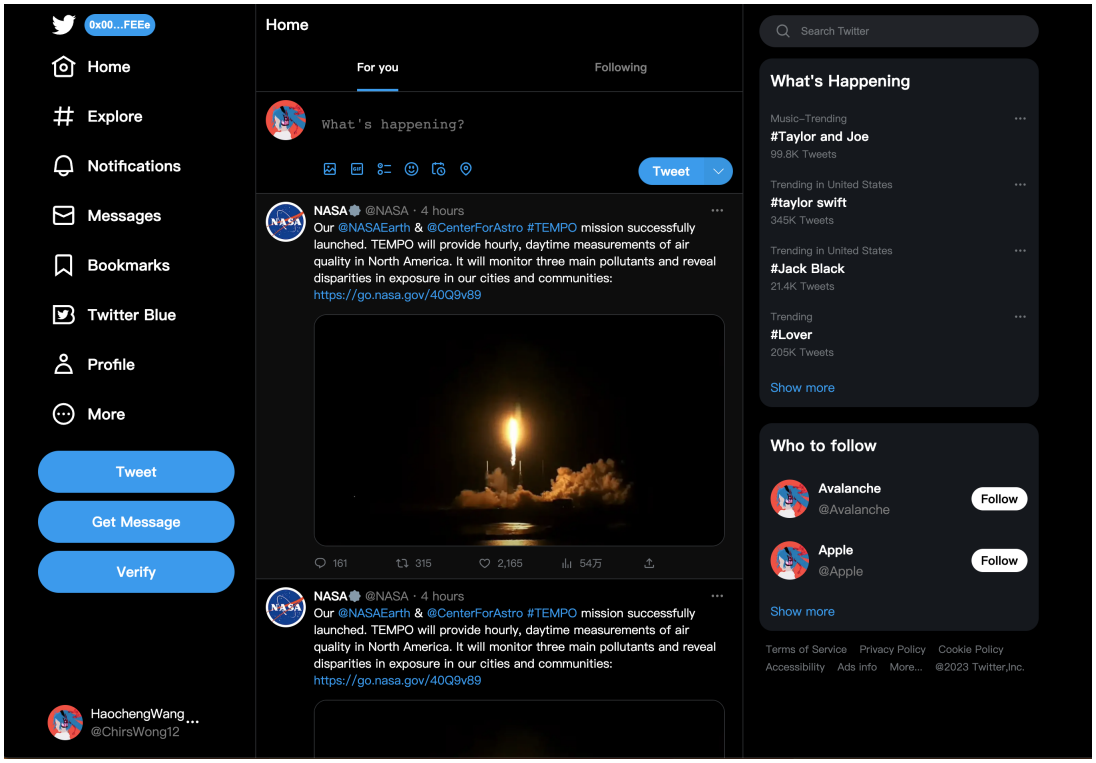

FIGURE 3.1: Main UI

On the top left corner user need to log in its account. The middle and right is same to twitter UI. In the left side, sender can tweet message same as Twitter:

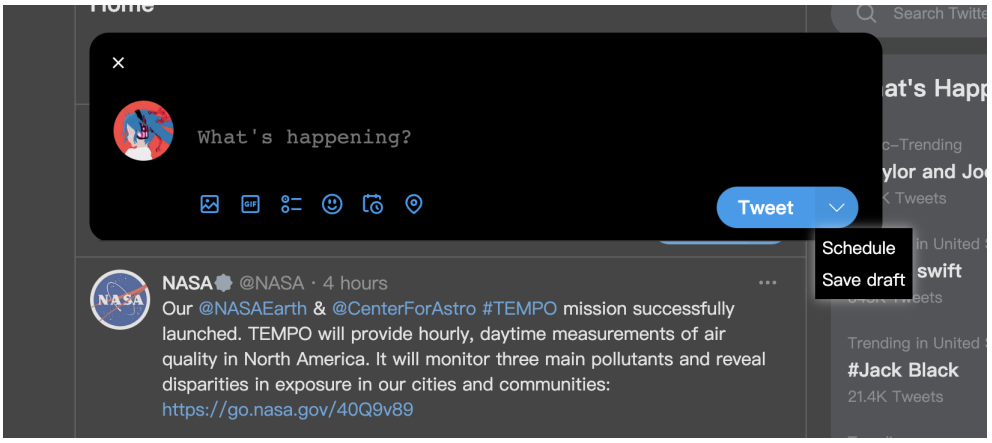

FIGURE 3.2: Publish UI

Here sender can click the `Schedule` to execution publish schedule:

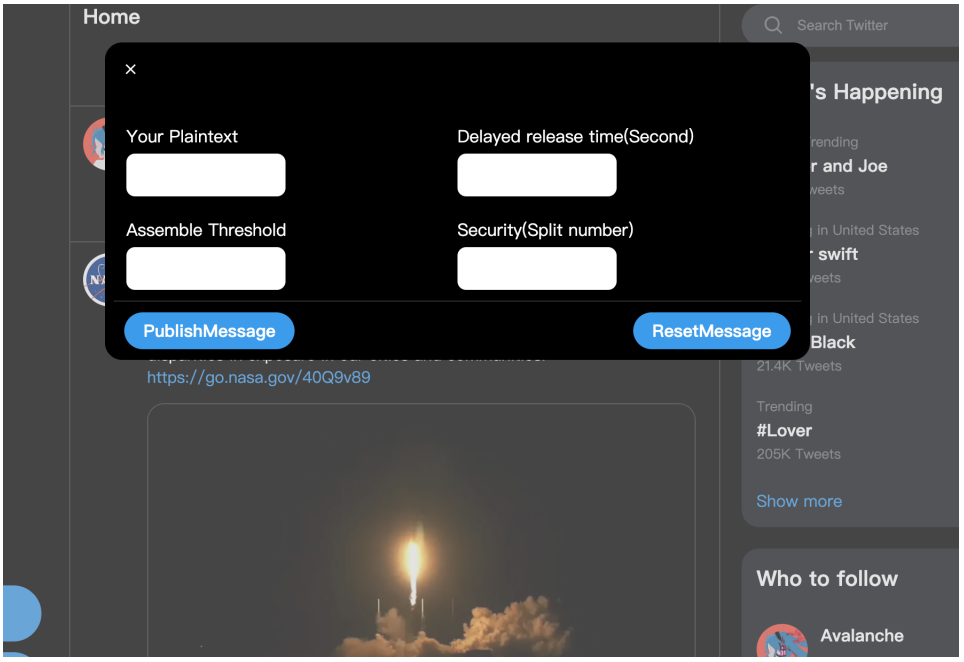

FIGURE 3.3: Tweet UI

Here sender need to give these 4 parameter: plaintext, delay release time, Assemble threshold(secret sharing threshold) and security(secret sharing number). Then sender can publish message, and it will accounting time at this moment. Before the time is up, sender can click 'ResetMessage'. Then the message will cancel publish and clear all sharing from holder.

Here is the 'Get Message' UI:

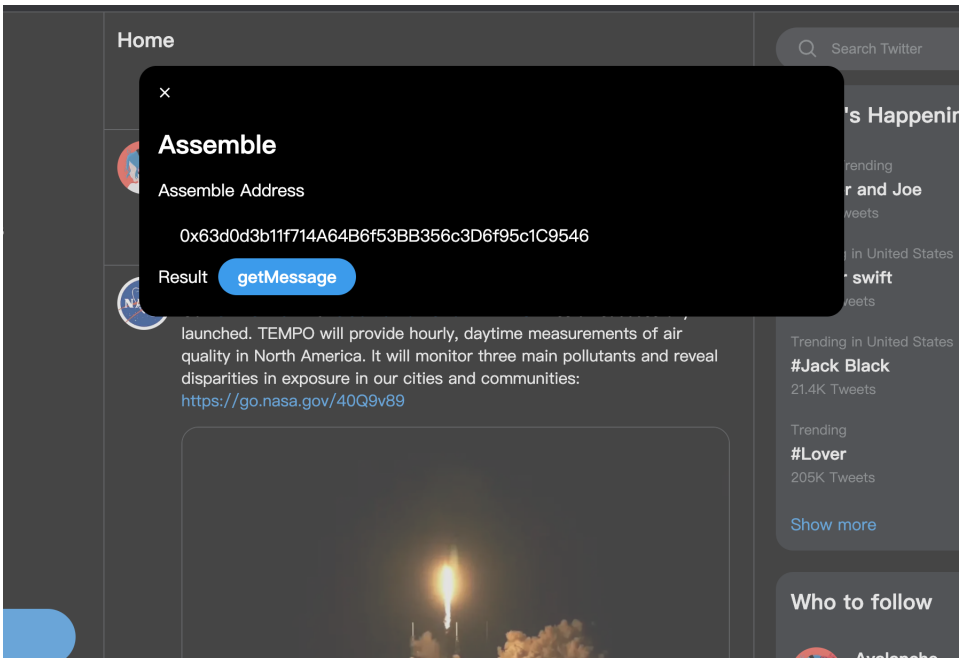

FIGURE 3.4: Get Message UI

Here is the assemble contract address for convenient to check. Anyone can click `getMessage` button and get message about sender published when the time is up.

Finally, here is the 'Verify' UI:

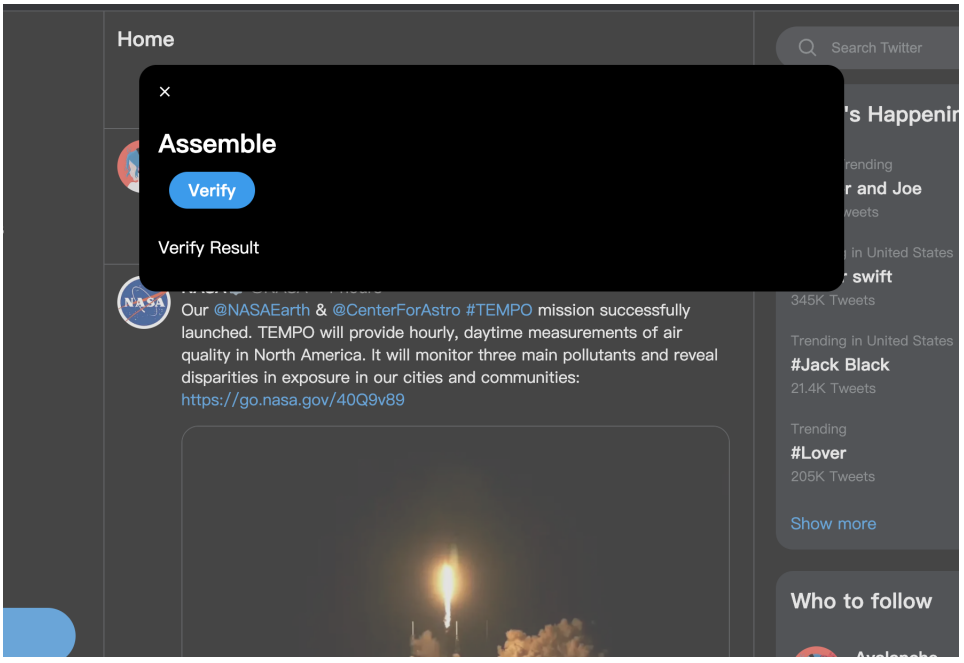

FIGURE 3.5: Verify UI

In this interface, publisher can click 'Verify' the message to verify it is equal to its own publish. I will give more details of the detailed verification step in the following part.

Here is the summary about whole process:

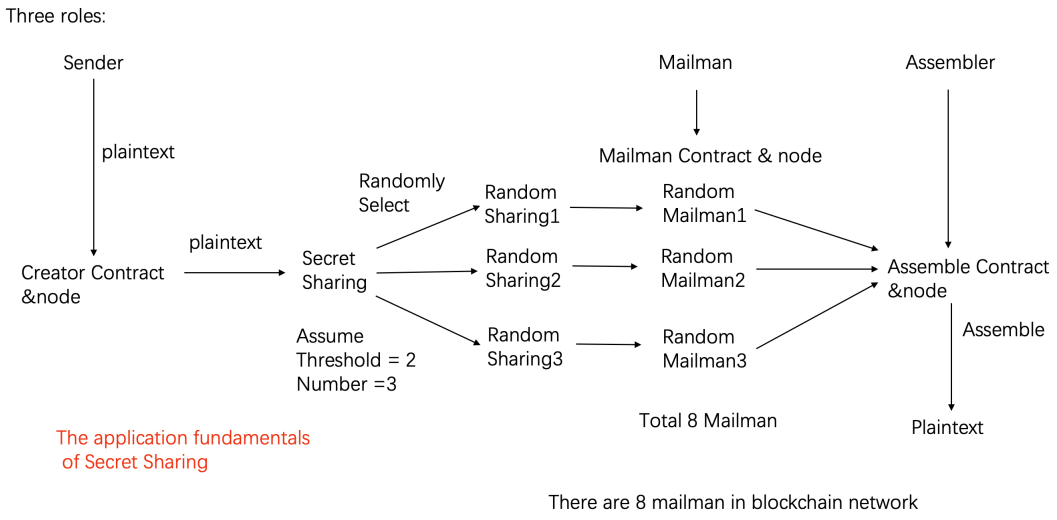

FIGURE 3.6: Secret Sharing

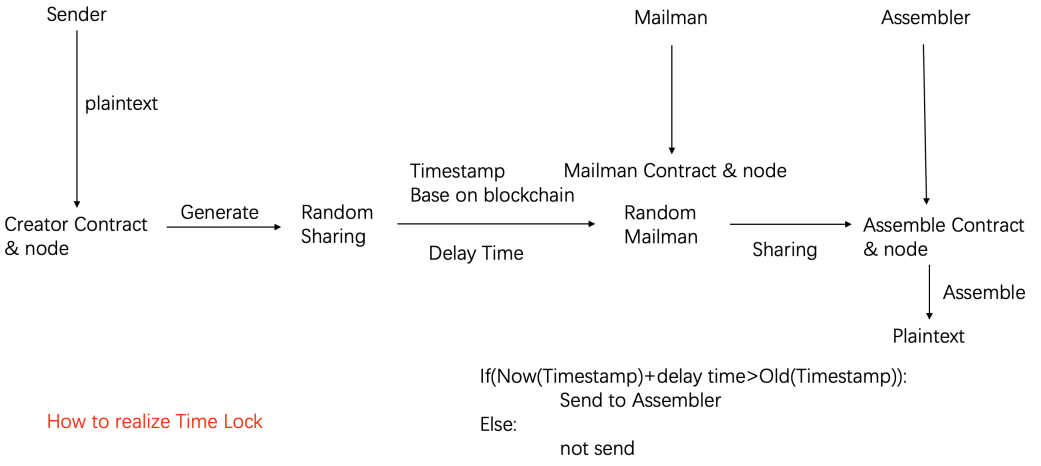

FIGURE 3.7: Time lock

3.2 The interaction between on chain Ethereum and off chain node

This is a distributed system, so there are many user node. Specifically, each user who enroll the system has a node to interaction with on chain Ethereum because only Ethereum can not complete all function.

Because this system deployed on `Ganache` network and all of the node only on one PC, I use the local communication to simulate the part of node communicate remotely in the component of secret sharing, Multi-layer encryption, Signature and Hash calculation authentication mechanism and so on. In this system, I use the method of front-end event listening to simulate Whisper transmission. It means all of the node in one PC and they support the server together. To realize information transmission, node should send message to other node through front-end listening. Here is the image to explain this process:

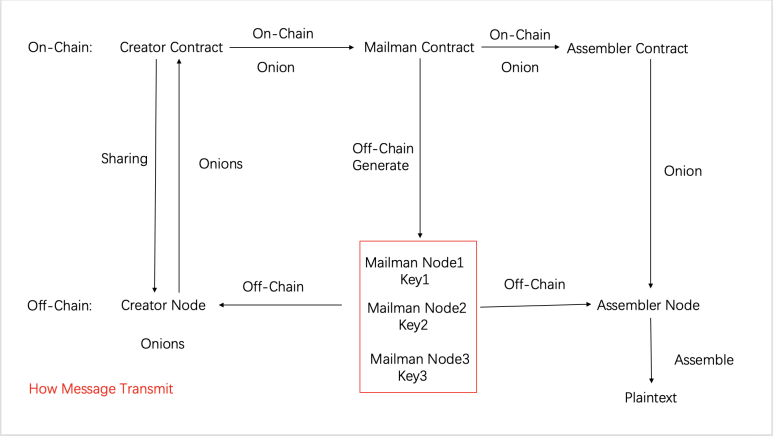

FIGURE 3.8: Interaction between node and Ethereum

Chapter 4

Multi-layer encryption and Verification mechanism.

4.1 Multi-layer encryption

In the step of holder get message and wait for time up, there is a risk of some holder collusion and assemble the plaintext premature. For example, we assume threshold is 2 and number is 3, if one holder who has sharing want to get plaintext in advance, he just need cooperate with another holder because threshold is 2.

In order to avoid this problem, this system use multi-layer encryption. The multi-layer encryption means each selected holder will generate a symmetric encryption key. This key will be sent to both message publisher and assembler. Before the sender give sharing to holder, he will use all of the symmetric key to encrypt each sharing. Each multi-encryption sharing will be send to one random holder. When the time up, the holder will send all of the multi-layer encrypted sharing to assembler. Then the assembler will get the all of the symmetric encryption key and decrypt each layer of sharing encryption. Finally, assembler will get each sharing and assembler them to plaintext.

The advantage we use multi-layer encryption is although some holder want to collusion, they only have the multi-encryption sharing. They can not recover to plaintext because they do not have all of the holder key. Only sender and assembler who have all of the symmetric decryption key can revert to plaintext. Here is the process about how multi-layer encryption:

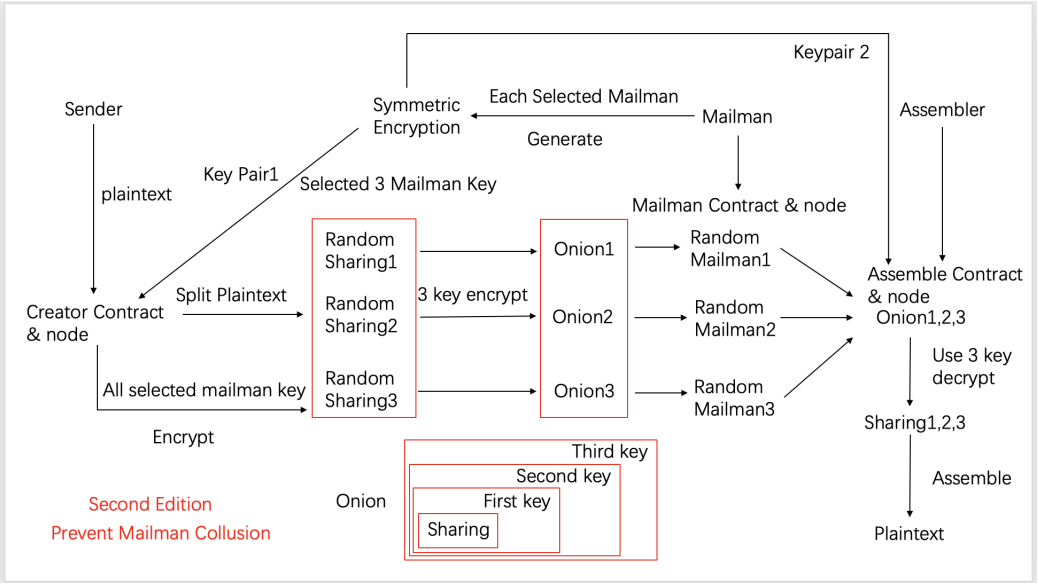

FIGURE 4.1: Multi-layer encryption process

4.2 Signature and Hash calculation authentication mechanism

In this part, I give the signature and hash calculation authentication mechanism to verify the assemble message is same to published message.

Before the plaintext be split to many sharing, sender calculate a hash message from plaintext. Then, the sender use hash message and private key to generate digital signature. The signature and hash message will be send to assembler. Assembler will use the signature and hash message to recover the sender account. Because the sender account is public, assembler will match the real sender account to revealed account. If these two account are same, it means the hash message he received is sent from sender actually. When the assembler reveal the plaintext by assembling sharing, he can calculate hash message from plaintext by himself. Then he can match the hash message which is sent from sender with calculated hash message by himself. Then it is easy to verify is the plaintext which the assembler reveal is same to sender publish or not.

Here is the image of whole process:

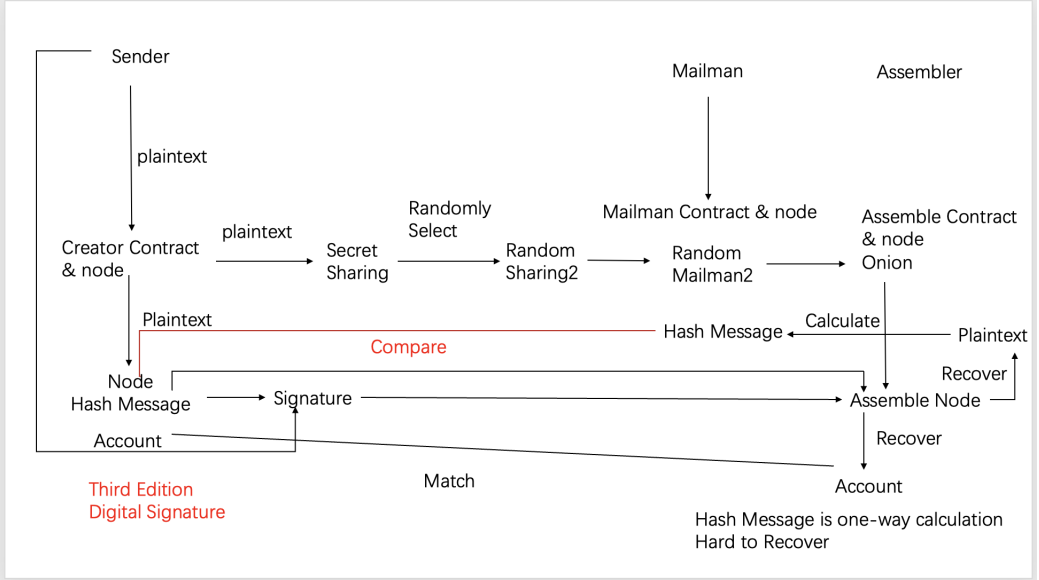

FIGURE 4.2: Authentication mechanism

Chapter 5

Demonstration of the process

5.1 Publish and get message

Firstly, we need to set the publish parameter. Here we set the published message is `hello`. The delayed release time is 3 second. The threshold is 2 and total sharing number is 3:

FIGURE 5.1: Publish demo

After we click publish, we need to verify and confirm by metamask. Then we should wait for 3 more second after we confirm transaction. Following, we can open `Get Message` interface. When we want to get message, we need do more step for transaction and authorization after click `getMessage`. Finally we can get message:

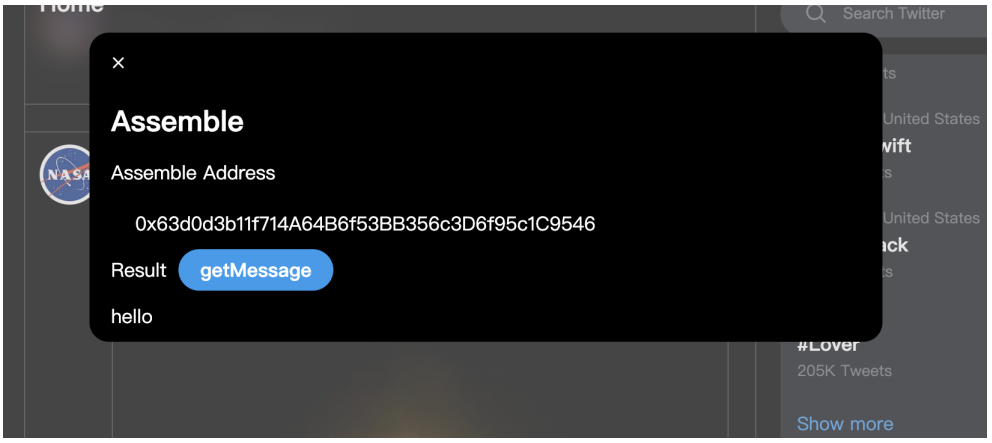

FIGURE 5.2: Get message demo

After message has been revealed, it means there is no message on schedule. In the status of no message on schedule, if we click `getMessage` again, it will prompt:

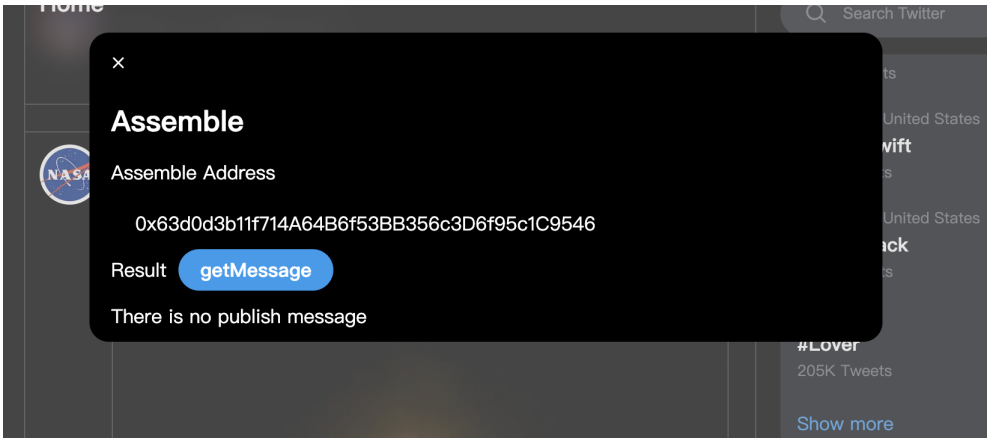

FIGURE 5.3: No publish

If we already publish a message and it still on pending time, user can click `Reset Message`. After confirmation with metamask, it will delete message on pending and stored in assemble node which is not be revealed. After user click `Get message`, it will also show `There is no message`.

Then, if the time of the assembler want to reveal message is not satisfy condition, it will prompt `Time not satisfy`.

Here we set a high delay time:

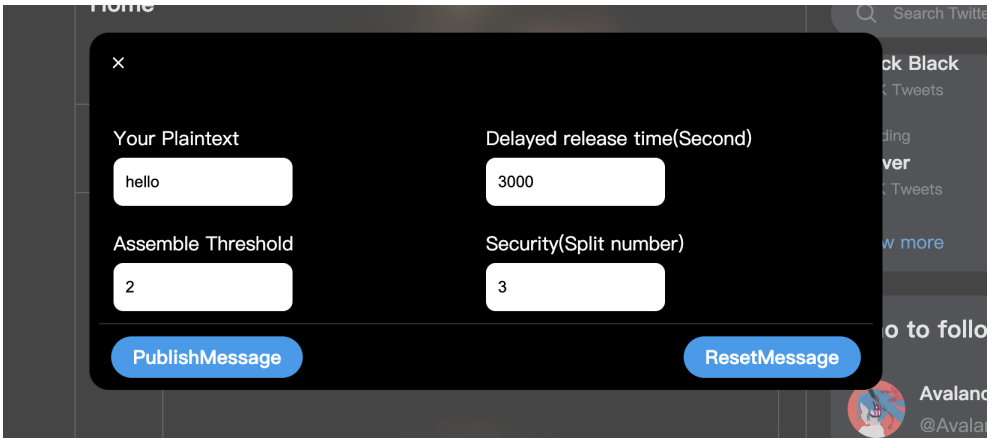

FIGURE 5.4: High time delay

It will prompt 'Time not satisfy':

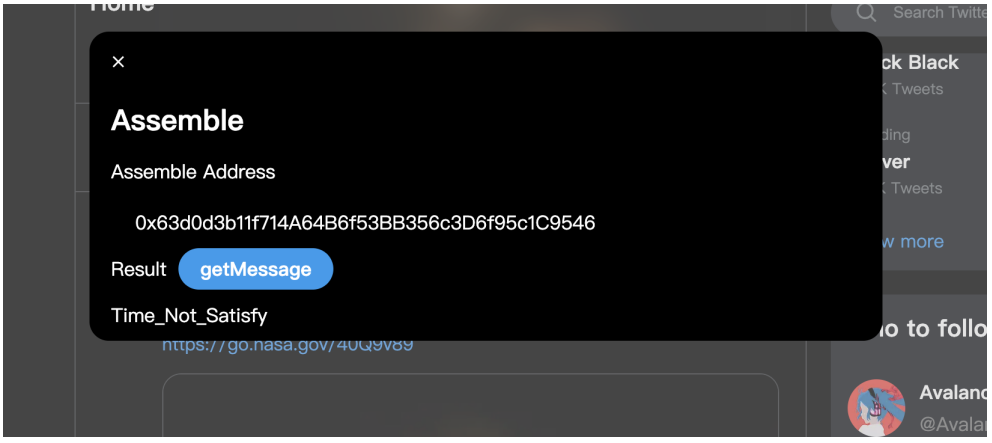

FIGURE 5.5: Time not satisfy

5.2 Verify

Here the assembler can verify the accuracy of message content and sources. I mentioned that assembler can realize it by two step: matching account address and hash message. If these two condition are correct, it will show correct here:

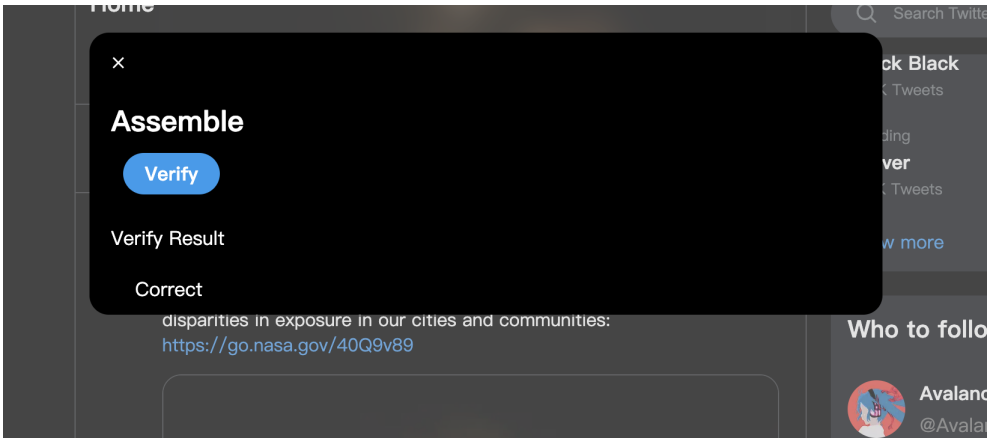

FIGURE 5.6: Verify

Chapter 6

The shortcomings of the system

6.1 Gas consume

This system make each holder have a contract but not makes each holder 'enroll' in one holder contract deployed by sender because of avoiding sharing and contract address leak through Ethereum. It is hard to control access permission precisely. Specifically, it is hard to totally isolate message between different holder. However, if I choose deploy multiple holder contract, it will generate large energy consumption to deploy holder contract. In this test system, the consumption of deploying 10 contract needs 0.17 ETH but what if user want to split to larger number of sharing. It will generate huge consumption.

6.2 Distributed environment

This system uses the way of front-end listening to simulate the message transfer between off chain node. However there is still a big difference between distributed environment such as maximum encrypted string length limit between network communications, browser compatibility issues, security restriction of whisper transmission. In principle, I think these problems can be solved. I already do a lot work but still some implementation difficulties to hard to actualize it out.

6.3 Reward and punishment mechanism

The purpose of any holder and assembler enroll this system is want to get reward. But this system do not given reward and punishment to participant. In the next edtion, I would like to realize it.