Fall 2022

Proxy Re-Encryption in Blockchain-based Application

Wangcheng Yuan

Proxy Re-Encryption in Blockchain-based Application

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Wangcheng Yuan

Nov 2022

The Designated Project Committee Approves the Project Titled

Proxy Re-Encryption in Blockchain-based Application

by

Wangcheng Yuan

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

Nov 2022

Prof. Thomas Austin	Department of Computer Science
Prof. Mike Wu	Department of Computer Science
Prof. Fabio Di Troia	Department of Computer Science

ABSTRACT

Proxy Re-Encryption in Blockchain-based Application

by Wangcheng Yuan

Nowadays, blockchain-based technology has risen to a new dimension. With the advantage of the decentralized identity, data are transferred through decentralized and public ledgers. Those new contracts provide great visibility. However, there is still a need to keep some data private in many cases. Those private data should be encrypted while still benefiting from the decentralized on-chain protocol. Securing those private data in such a decentralized blockchain-based system is thus a critical problem. Our solution provides a decentralized protocol that lets users grant access to their private data with proxy re-encryption in SpartanGold (a blockchain-based cryptocurrency). We implement a third-party storage provider called a proxy to store clients' private data in an encrypted form. Whenever someone wants to access a client's private data, the client uses their private key along with the buyer's public key to generate a re-encryption key. The third-party proxy uses the re-encryption key to re-encrypt the client's encrypted data for the recipient and send the result to the buyer. As a result, only the buyer can decrypt the re-encrypted data by using their private key, without revealing the data owner's private key or the private data to the third-party proxy. Our protocol has secured the private data on the decentralized blockchain-based application without relying on trusted parties. We use medical data as a use case to validate our protocol. In our medical use case, the patient's medical records are stored on the third-party proxy, and when specialists request medical data from the patient, the patient generates the re-encryption key and sends it to the proxy. The proxy re-encrypted the data and sends back to the specialists.

ACKNOWLEDGMENTS

I would like to express my special thanks of gratitude to my advisor Prof. Thomas Austin for his assistance and guidance throughout the journey of my thesis project. I also express my appreciation to my committee members Prof. Mike Wu and Prof. Fabio Di Troia, for their participation and feedback on this thesis project.

TABLE OF CONTENTS

CHAPTER

CHAPTER 1

Introduction

Blockchain technology was popularized by Satoshi Nakamoto in 2008. In Satoshi's paper [1], he introduced the first well-known Blockchain cryptocurrency, Bitcoin. Bitcoin is a decentralized digital currency that allows transfers of "bitcoins" without the need for a trusted third party. Bitcoin has received substantial attention because of its simplicity, transparency, and increasing popularity [2]. One of the most significant advantages of Bitcoin is the decentralized identity along with the opportunity to have privacy. There is no central authority for any transactions [3]. Instead, all clients in the same network share the same public ledger. The public ledger contains shared data and transactions in an encrypted format, and clients do not need to provide their real names or personal information. From Bitcoin, people saw that decentralized identities could bring great flexibility and visibility.

Non-fungible tokens (NFTs) were created in 2010, and the first projects using them were released in 2017 on the platform Ethereum [4]. NFTs are unique cryptographic tokens that exist on a blockchain and cannot be replicated [5]. Both Bitcoin and NFTs reply on blockchain transactions to validate the authenticity and record ownership. However, unlike Bitcoin, NFTs can represent real-world items such as digital art, audio information, or medical records.

NFTs have become popular in the marketplace in 2020 [6] because they provide unique, verifiable, and immutable proof of ownership of digital goods. Each token stored in the blockchain doesn't have the same value. Kus Khalilov and Levi [7] state that more and more companies utilize NFT blockchain technology as their management system to store data and interact with customers.

1.1 Problem Statement - Managing Private Date on the Blockchain

In most of the existing NFT blockchain-based technology, data is in a publicly visible format. Once the information is sent to the network and registered in the ledger, everyone in the blockchain network can see it. This is not ideal because, in many situations, data should be kept in a private format that only authorized identities can access [8]. For example, a client could disclose their medical records from the hospital or personal documents to trusted identities, such as insurance companies or personal doctors; other identities should not see those records [7]. Those private data should be encrypted while still benefiting from the decentralized on-chain protocol. Since all identities in the blockchain are distributed and decentralized, there is no physical machine or comprehensive information for each identity. Thus, securing private data in NFT blockchain-based application becomes a significant problem.

1.2 Previous Solution - Public-key Encryption

Chen et al. [9] propose an approach that uses a combination of cloud storage and public-key encryption (PKE). PKE is the most widely used algorithm to share private data in a 1-to1 manner. Cloud storage is presented as a third party to eliminate the data owner's local storage hardware and management overhead. The data owner encrypts the data using their public key before uploading it to a third party. The encrypted data and other information is shown in the blockchain. The data owner could send the private key to a third party whenever a consumer wants to access the data. The third party would use the private key to decrypt the data and send it to the consumer. However, the third party would then access the data itself in this scenario. What's more, the third party could send the private key to other identities in the blockchain. Since the encrypted information is broadcast in the network, other clients could also use the private key to decrypt the content. Even though enterprise cloud providers such as Amazon Web Services (AWS) and Google Cloud Platform(GCP) are considered “trusted” third parties overall for the blockchain application, there are still potential security and privacy limitations.

1.2.1 Our Solution - Proxy Re-encryption

Our solution provides a decentralized protocol with Proxy Re-Encryption (PRE) in NFT blockchain applications. A client could upload their private data encrypted by its public key to the third-party, called proxies. In practice, proxies often relate to cloud providers. When a client (Bob) wants to access another Client’s (Alice) private data stored in proxies, he could ask Alice for permission and send money to her. Once Alice receives the money and approves Bob’s request, she will use her own private key and Bob’s public key to generate a *re-encryption key* and send it to the proxy. A *re-encryption key* is a special key for the proxy to transfer from one secret key to another without ever accessing or exposing the plaintext or data [10]. The proxy would use the re-encryption key to re-encrypt Alice’s data for Bob’s public key. Finally, the proxy sends the final encrypted data to Bob, and Bob’s private key can decrypt the final encrypted data. In our proposed protocol, the client could upload private and public data into the blockchain without letting third parties, including the cloud provider, view the private data content.

For example, Figure 1 shows how PRE is used in a blockchain-based medical record application. When the patient sees the general doctor, the doctor will send the patient’s health record to the storage provider in encrypted form, which is the proxy in this case. Later on, when the patient is referred to see the specialty doctor, the doctor could ask the patient for permission to access their own previous health record. Once the patient agrees, they could use the specialty doctor’s public key along with their private key to generate the re-encryption key and send it to the

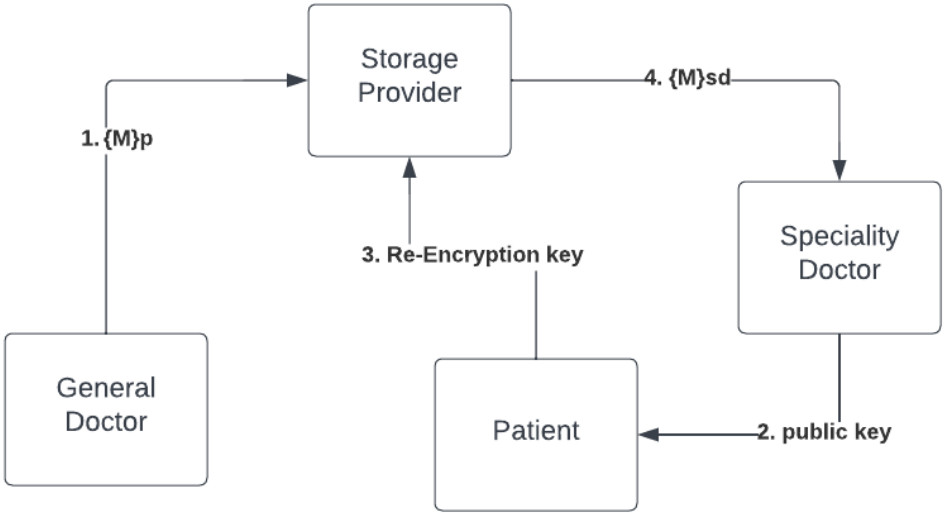

Figure 1: Recording and sharing medical data using PRE

storage provider. The provider storage would encrypt the patient's health record so that only the specialty doctor could decrypt and see the data. The patient's health record is stored distributively, and no party can access the data without the patient's authorization.

1.3 Results

We implement our PRE protocol in Spartan-Gold [11], which is a JavaScript blockchain-based framework. By using our implemented protocol, the user can upload encrypted data to the proxy while broadcasting the encrypted data along with messages, such as listed price and description. The consumer can look at the description and list price to determine whether they want to access that encrypted private data. The proxy stores the data and can re-encrypt it for any user authorized by the data owner. Critically, the proxy has no way to reveal private data to unauthorized clients. All transactions made during the PRE process will be announced in the blockchain.

CHAPTER 2

Background

2.1 Blockchain

Blockchain is a decentralized technology that shares the public ledger in the blockchain network [12]. Each block inside the blockchain contains the data and the previous block's hash value. The data field contains the actual data and both the sender's and receiver's addresses. The hash value of the previous block makes the blockchain achieve strong transparency. It is difficult for attackers to tamper with the data since one single change on the block will modify all data histories stored in the blockchain network. The block also includes a timestamp field to indicate the transaction's date.

2.1.1 Peer-to-Peer (P2P)

Blockchain is a peer-to-peer (P2P) network, which means every single individual identity acts as a peer, and they share the same public ledger [13]. When any identity initiates the transaction, the miner finds the block and broadcasts it to the blockchain network. All the other identities will update their ledger at the same time. This feature of blockchain brings security and transparency because even the attacker can tamper with the data on one identity's side; they have to tamper with the data on all other identity's sides. Since the blockchain stores data in a decentralized way, each individual identity keeps a copy of the transaction history. P2P makes it difficult for attackers to tamper with the data within the blockchain network. The P2P system really limits the central control from the centralized storage system. It also brings the blockchain to achieve high availability since if one identity breaks or loses the data, there are other replicas inside the blockchain network.

2.1.2 Proof-of-Work

In the blockchain network, there are miners to find the block for making transactions. Miners are also considered clients that act as identities in the blockchain. Miner uses proof-of-work (PoW) to verify the transaction and create the block to put into the blockchain. A PoW is a form of a cryptographic algorithm in that the miner has to prove to others that they solve the hashing algorithm, which means a certain amount of computational effort has been made [14]. After the miner finds the PoW, they will broadcast the PoW to the blockchain network and waits for other identities to review. After all identities in the blockchain network agree and verify the PoW, the miners will put the transaction into the block and broadcast it to the blockchain. The miners who found the PoW will be rewarded with a transaction fee.

2.1.3 Drawback on Blockchain

Blockchain achieves high availability and transparency by using the public ledge. This feature of blockchain makes the blockchain-based application more secure than the centralized data storage system. However, transparency sometimes is a two-sided sword. The public shared ledger works great when the transaction is public to the network, but when the transaction is only intended to be visible to an only specific identity in the blockchain network, the confidentiality of the blockchain does not perform that well. For example, if the blockchain-based application is used for sharing medical records, some of the medical records should keep in private, which is only available to the patient and their doctors. Blockchain often inevitably leads to the problem of exposing sensitive information such as critical medical data or personally identifiable user-specific data [15].

2.2 Filecoin (FIL)

Filecoin (FIL) is a decentralized blockchain-based storage that can store users' information other than digital currency. Like other blockchain-based storage applications, FIL also relies on a P2P network, and it uses InterPlanetary File System (IPFS) to store the files and data on unused hard drives in the network. Miners in the FIL get rewards when they store the data and files. The public shared ledger records the transactions and provides proof that miners store the files correctly [16]. FIL provides the ability to store sensitive data by encrypting the data using PKE before sending it to the blockchain. This feature of FIL helps solve the confidentiality problem in blockchain-based applications. However, only the data owner can view and access the FIL in this case, which means the private data can not delegate to other identities even with the data owner's authorization. For example, the miner stores the patient's medical record, and later on, a specialist is referred to the patient and needs to access the data. In this case, the patient has to get their data back from the miner who stored the data and then decrypt and send it to the specialist. Another solution would be for the patient gives their private key to the specialist. This can solve the problem, but it is not a good practice in the long run to expose the private key to other identities.

2.3 StorJ

StorJ is another decentralized blockchain-based cloud storage that can store users' information and data. The StorJ has three major components: Storage nodes, Uplinks, and Satellites [17]. Storage nodes are responsible for storing and retrieving data. Uplinks are applications installed on the storage node to upload and retrieve data. Satellites act as a bridge between the storage nodes and the uplinks. They are responsible for directing the traffic, such as what data is stored on which storage nodes.

Unlike FIL, which is optimized for publicly visible data, StorJ automatically encrypts the data when the client uploads the data. The encrypted is split into 80 pieces, and each piece is stored on 80 different storage nodes using the satellite [17]. When the client wants to access the data, they only need 29 out of 80 pieces to reconstruct the encrypted file. StorJ seems to provide more capability in terms of storing private data, but they had the same problem as the FIL. The private data is only accessible to the data owner. If the data owner wants to share the data with other identities, they have to retrieve it by themselves and then send it to their consumer. In this case, the StorJ can only store the data, not distribute the data.

2.4 Fine-grained Access Control Approach

Sun et al. [18] proposed a blockchain-based data-sharing approach that provides the ability to encrypt the data. Their approach is to first perform a hash calculation on the data and then store the corresponding value on the blockchain to ensure integrity and authenticity. Then, they use PKE to encrypt the data and store the encrypted data in a distributed storage protocol. The distributed storage is isolated from the blockchain, which you can think about the actual data is stored on the cloud provider, and the blockchain only stores the corresponding hash value [18]. When the client tries to retrieve the data, the encrypted keyword index information from the blockchain will help the client to retrieve the data from the corresponding data store from the distributed storage protocol [18]. To delegate the data to another identity, the keyword index can be shared with the identity without retrieving the data first. This approach can not only solve the problem of confidentiality on the blockchain but also solves the problem of delegating the data to other identities without exposing the data to a third party.

2.5 Lightweight Message Sharing Approach

Fu et al. [19] propose an approach to solve both the privacy and data sharing problem in the blockchain-based storage system. First of all, they apply an interleaving encoder to encrypt the original data. Then, they implemented a (t,n) - threshold data sharing scheme [19]. Like StorJ, the encrypted data is mapped to n different short shares. It only needs t pieces to retrieve the data instead of n total shares. The index of those data shares is stored in different blocks, which all blocks chain together to make the complete blockchain. When the data owner or authorized identity wants to access or retrieve the data, they just need to request at least t shares of the encrypted data shares from the blockchain.

2.6 Proxy re-encryption (PRE)

Proxy re-encryption (PRE) is a type of public-key encryption (PKE). In traditional PKE operations, only two parties are involved in encryption and decryption. PRE has an additional semi-trusted third-party proxy [20]. PRE allows a proxy entity to transform or re-encrypt the data from one public key to another without accessing the underlying plaintext or private keys [21]. When Bob wants to access Alice's data, Alice could use Bob's public key along with her private key to generate the re-encryption key. Alice sends the re-encryption, and the proxy uses the re-encryption to encrypt Alice's cipher, which could only be decrypted using Bob's private key. PRE was first formalized by Blaze et al. [22], and after that, PRE was modified and improved in many different schemes such as unidirectional PRE [23], which is another PRE that the re-encryption key only enables delegation in one direction but not the opposite [23]. PRE was utilized in several applications, including e-mail forwarding, law enforcement, cryptographic operations on storage-limited devices, distributed secure file systems, and outsourced filtering of encrypted spam [24].

2.7 Spartan-Gold

SpartanGold (SG) [11] is a simplified blockchain-based cryptocurrency created by Austin [11]. SG is written in JavaScript and runs on the Node.js platform. SG is flexible and available to implement and experiment with different features for educational purposes. SpartanGold's design is loosely based on Bitcoin. SpartanGold uses a proof-of-work (PoW) blockchain. In the initial design of SG, it can only make transactions that are based on currency (spartan gold). The client needs to give the output field, which includes the amount of the gold and the receiver's address, in order to make transactions. Then the transaction is signed with the user's private key and broadcast to the blockchain network. SG comes with two different approaches to running the program: the single-threaded mode and the multi-process mode. We use single-threaded mode for our protocol and focus on how PRE integrates with SG.

CHAPTER 3

Technologies Used

In this chapter, we provide an overview of the technologies used to facilitate better understanding of our implementation.

3.1 SpartanGold (SG)

SpartanGold [11] is a simplified blockchain-based cryptocurrency for education and experimentation. It was created and designed by Professor Thomas Austin. SpartanGold's design is loosely based on Bitcoin. SpartanGold uses a proof-of-work (PoW) blockchain. In the initial design of SG, it can only make transactions that are based on currency (spartan gold).

The below UML sequence diagram shows how Alice sends money to Bob from the initial SG.

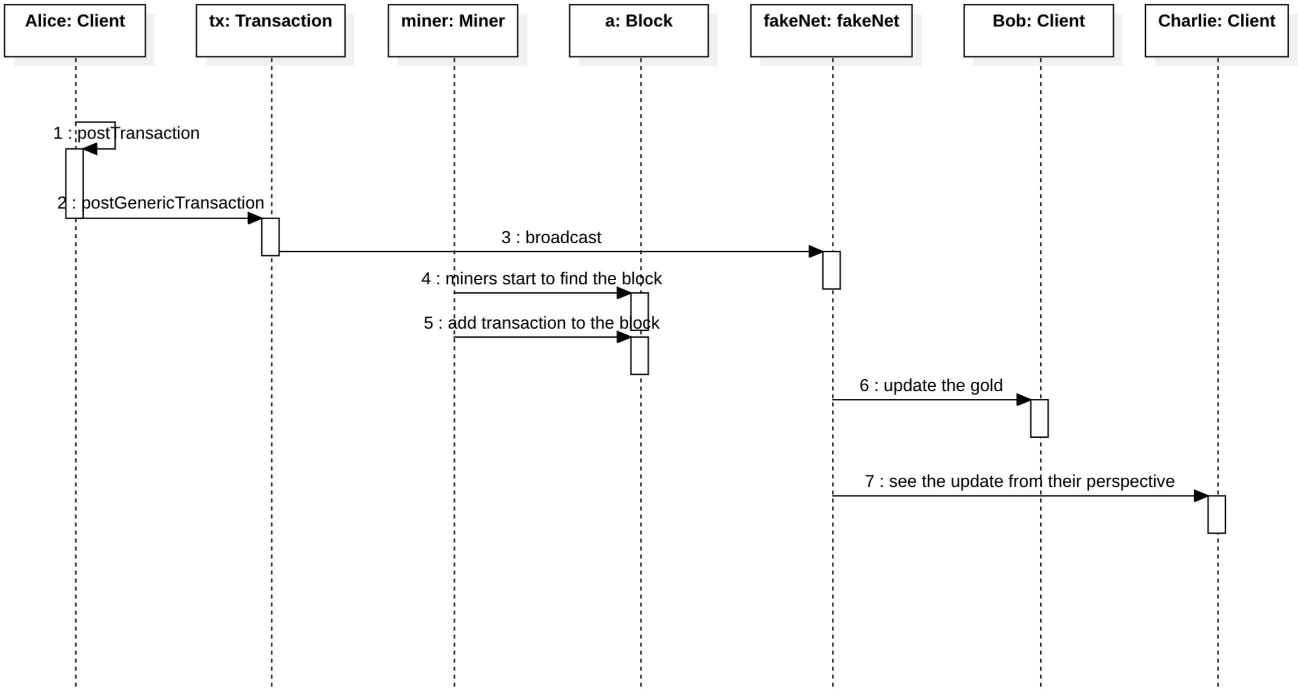

Figure 2: Alice sends money to Bob

1. postTransaction()

   The first step is that Alice calls the `postTransaction` method to initiate the transaction. The `postTransaction` is implemented inside the **Client.js**. It passes an output array, the amount of gold, and the receiver's address, and then returns a post transaction using `postGenericTransaction`. Inside the `postTransaction` method, it will check if the client has sufficient gold to make the transaction and then pass the output array and fee to the `postGenericTransaction`.

   ```
   alice.postTransaction([{ amount: 32, address: bob.address }]);
   ```

   ```
   postTransaction(outputs, fee=Blockchain.DEFAULT_TX_FEE) {
       if (totalPayments > this.availableGold) {
         throw new Error(‘Requested ${totalPayments},
         but account only has ${this.availableGold}.‘);
       }
       return this.postGenericTransaction({
         outputs: outputs,
         fee: fee,
       });
     }
   ```

2. postGenericTransaction()

   The second step is to invoke the `postGenericTransaction` method inside the `postTransaction`. The `postGenericTransaction` handles the transaction from the client and broadcasts it to the network. There is a txData to handle special parameters rather than the gold. In this case, since we only make gold transactions and no other data is passed into the transaction. Inside the `postGenericTransaction`, it first creates a transaction object by using the client's address and SG public key, and then it uses SG private key to sign the transaction. Finally, `postGenericTransaction` will broadcast the transaction to the blockchain network and also adds to the pending transactions, which is ready for miners to mine the block for it.

```
  postGenericTransaction(txData) {
  let tx = Blockchain.makeTransaction(
    Object.assign({
        from: this.address,
        nonce: this.nonce,
        pubKey: this.keyPair.public,
      },
      txData));
  tx.sign(this.keyPair.private);
  return tx;
}
```

3. Broadcast

   The third step is to broadcast the transaction from the `postGenericTransaction` to the blockchain. After broadcasting the transaction, every client who registered in the blockchain network can view this transaction.

```
this.net.broadcast(Blockchain.POST_TRANSACTION, tx);
```

4. Miners start to find the block

The last step inside the `postGenericTransaction` is to add the transaction to the pending transaction map, in which the key is the id of the transaction, and the value is the transaction itself. After miners find a valid block, the transaction is finally processed.

```
this.pendingOutgoingTransactions.set(tx.id, tx);
```

5. Add transactions to the block When miner's find the valid block, they will add the left transactions in the network to the block by using a queue.

```
this.transactions.forEach((tx) => {
  this.currentBlock.addTransaction(tx, this);
});
```

6. Bob's gold balance is updated

   After the transaction is completed, it will automatically update Bob's gold balance and everyone who registered in the network can view this change from the blockchain.
   Before Alice sends Bob 32 dollar:

```
Initial balances:
Alice has 233 gold.
Bob has 99 gold.
Charlie has 67 gold.
Minnie has 400 gold.
Mickey has 300 gold.
Donald has 0 gold.
```

   After the transaction is completed:

```
Final balances (Donald's perspective):
Alice has 265 gold.
Bob has 67 gold.
```

```
Charlie has 67 gold.
Minnie has 850 gold.
Mickey has 679 gold.
Donald has 125 gold.
Proxy has 49 gold.
```

7. Other clients sees the update from their perspective Since Charlie is also registered in the blockchain network, Charlie shares the same public ledger with Bob, and from Charlie's perspective, he will have the same updated information just as Bob does above.

3.2 Recrypt-js

Recrypt-js, designed and implemented by yjjnls [25], is a JavaScript SDK for proxy re-encryption compatible with py_sdk. For our protocol, we use recrypt-js as the framework to work on top of the SG. Below is how this framework works:

1. Generate the public and private keys for both parties (sender and receiver). kp is the keypair for the party, sk is the private key that is generated from kp. pk is the public key generated from kp.

```
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());
```

2. Encrypt the plain text data using the public key of the sender (pk_A).

```
let obj = PRE.encryptData(pk_A, "test data")
```

3. Generate the re-encryption key (rk) using the sender's private key (sk_A) and receiver's public key (pk_B).

```
let rk = PRE.generateReEncrytionKey(sk_A, pk_B);
```

4. Proxy re-encrypt the initial encrypted data (obj) by using the rk

```
PRE.reEncryption(rk, obj)
```

5. Decrypt the re-encrypted data by using the receiver's private key (sk_B)

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

CHAPTER 4

Implementation - Oathkeeper

Our protocol, named Oathkeeper, combines the SG and recrypt-js framework together to achieve the proxy re-encryption in a simplified blockchain-based cryptocurrency. Based on the SG, we created another class named Proxy.js. To illustrate how our protocol works, the sections below show what the proxy re-encryption process looks like. Below UML sequence diagram shows how the proxy re-encryption works in Oathkeeper:

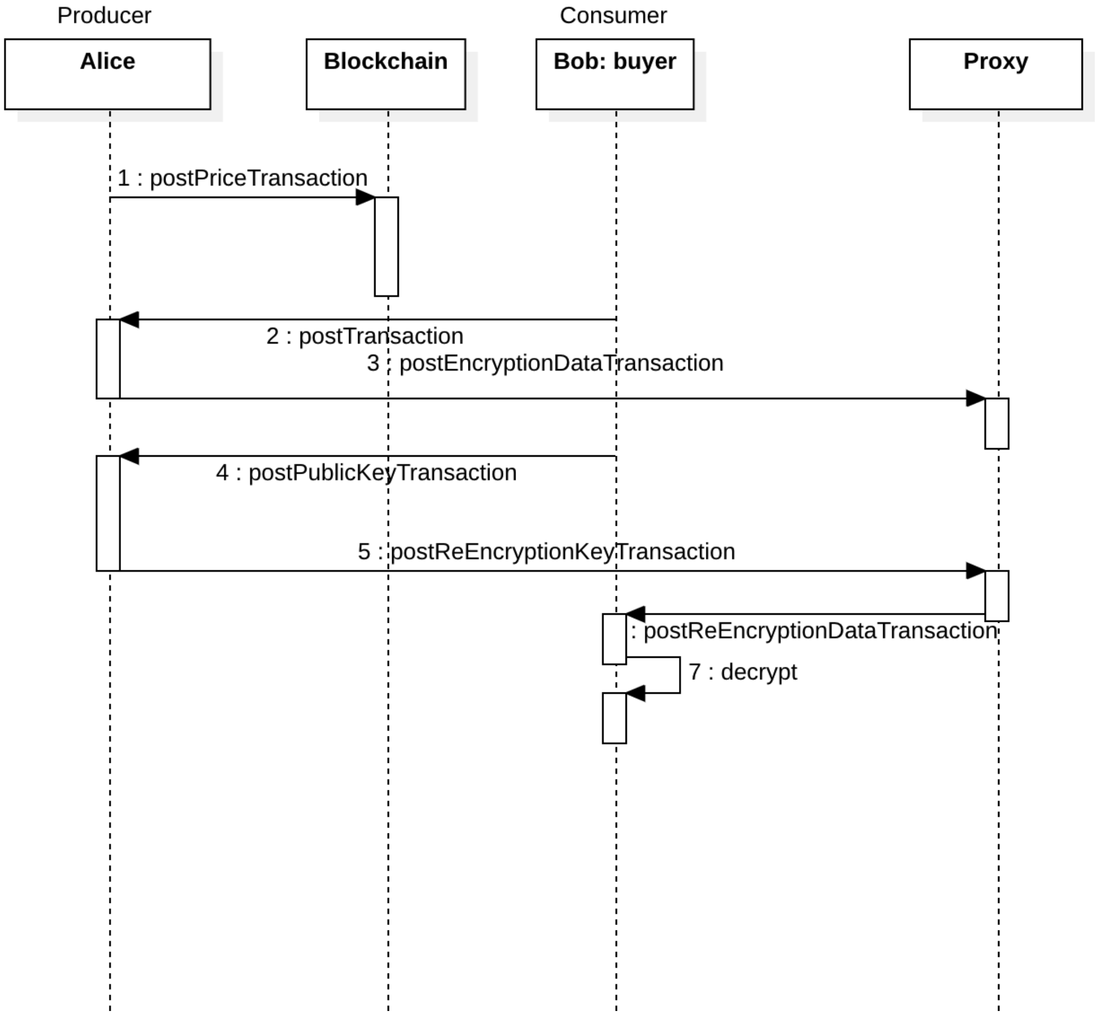
Producer
Consumer
Alice
Blockchain
Bob: buyer
Proxy
1 : postPriceTransaction
2 : postTransaction
3 : postEncryptionDataTransaction
4 : postPublicKeyTransaction
5 : postReEncryptionKeyTransaction
: postReEncryptionDataTransaction
7 : decrypt

Figure 3: Bob requests data from Alice in Oathkeeper

1. postPriceTransaction

   ```
   alice.postPriceTransaction({dataName: "COVID−19 Vaccine",
   description: "The data and deatil about my COVID−19 vaccine
   information ", price: 33.0}, "Pfizer, took on March 2021");
   ```

   The first step is that Alice announces her data into the blockchain. We implement a method called `postPriceTransaction()` in the Client class. This method allows the client to post their content along with its price to the blockchain. The first parameter is the data information in an object format, such as the data name, data description, and the price of the data. The second parameter is the actual content of the data.

   ```
   postPriceTransaction(obj,content) {
       this.uploadPrivateData.set(obj.dataName, content);
       return this.postGenericTransaction({
         data: obj
       });
    }
   ```

   The `postPriceTransaction()` method does two things. The first thing is to save Alice's uploaded data into a map called `uploadPrivateData`. The key of the map is the data name, and the value is the actual content. The second thing is to return a transaction by using the `postGenericTransaction()`. Here the txData field is the uploaded data.

   After some time, miners found the proof and put the transaction into the block. Now everyone in the network can view the previous transaction.

   ```
   setTimeout(() => {
   ```

```
console.log();
console.log("***Starting a late-to-the-party miner***");
console.log();
console.log(bob.lastBlock);
```

```
listData: Map(2) {
  '7Jywq94pQJL8mk87rGZaWU/SYP7FWporblYtXP+SiGE=' => {
    dataName: 'COVID-19 Vaccine',
    description: 'The data and deatil about my COVID-19 vaccine information
    price: 33
  },
  'aDsxAo9i/ST5YxnnfH+Qd4YfqzekauWO5gfYGGHBvyY=' => {
    dataName: 'BR',
    description: 'My recent blood test result',
    price: 180
  }
},
```

As you can see above, the client in the blockchain network can see the data information and Alice's address from the block. If any client is interested in the data, they can also view the description and the data price.

2. Requesting Data

After Alice posts her data into the blockchain, Bob is interested in Alice's data and wants to buy Alice's content. Bob makes a single currency transaction that pays the right amount of money to Alice's address.

```
let obj = bob.lastBlock.secretOf(alice.address);
bob.postTransaction([{ amount: obj.price,
address: alice.address }]);
```

Once Bob pays the money, Alice and Bob now can start the re-encryption process to deliver the data content to the buyer through the proxy. Each time a data content transaction happens, a new public/private key pair is generated for both Alice and Bob. However, the SG address of the public and private keys remains the same.

The below code shows how the public/private key is generated for both Alice and Bob.

```
 alice.generateReEncryptionKeyPair(obj.dataName);
bob.generateReEncryptionKeyPair(obj.dataName);
```

We implement a method inside the Client class called `generateReEncryptionKeyPair()` to generate the proxy public/private key pair for the Client. This method saves the proxy key pair from the mapping of the Client. The key of the map is the data name, and the value of the map is the key pair. The map lets the client know which proxy key pair is used for which data content. From there, all things are ready to go, and we can start the re-encryption transaction.

```
generateReEncryptionKeyPair(dataName) {
      var kp_proxy = PRE.Proxy.generate_key_pair();
      var sk_proxy = PRE.Proxy.to_hex(kp_proxy.get_private_key().
            to_bytes());
      var pk_proxy = PRE.Proxy.to_hex(kp_proxy.get_public_key().
            to_bytes());
      this.proxyKeyPair.set(dataName, {pk:pk_proxy, sk: sk_proxy});
  }
```

3. postEncryptionDataTransaction

```
 alice.postEncryptDataTransaction(obj.dataName, proxy);
```

The first step to start the proxy re-encryption process is to encrypt the initial data by using Alice's public key. In `Client.js`, a method called `postEncryptDataTransaction` is implemented for the data owner to encrypt the data content by their public key. The first parameter of this method is the name of the data since the data owner could find the content by using the `uploadPrivateData` map. The second parameter is the proxy to which the encrypted data is sent. After the data owner uses the recrypt-js library to encrypt the data, the proxy triggers a method called `saveEncryptedData` to save the encrypted data content.

```
postEncryptDataTransaction(dataName, proxy) {
    let content = this.uploadPrivateData.get(dataName);
    let kp = this.proxyKeyPair.get(dataName);
    let initialEncryptedData = PRE.encryptData(kp.pk, content);
    proxy.saveEncryptedData(dataName, initialEncryptedData);
    return this.postGenericTransaction({
        encryptedData: {
          filename: dataName,
          encryptedContent: initialEncryptedData
        }
      });
  }

saveEncryptedData(dataName, initialCipher) {
    this.encryptedData.set(dataName, initialCipher);
  }
```

Inside the `saveEncryptedData`, the key is the name of the data, and the key is the encrypted data content by using the data owner's public key. Finally, the data owner calls the `postGenericTransaction` to put the encrypted data as the data field and broadcast the transaction to the network. Thus, the transaction happens in the blockchain, but clients cannot view the content of the information.

Below is the initial encrypted data from the blockchain:

```
The initial cipher encrypted by Alice public is:

{"key":"048af708a6ec2c7aa55ac8d9250caf067eee4c2131df
247b2665ebbb10206159545efd45a5ab5ec7cea471afcd8b8f6c
81d40249fd9c538876510f19dbdb14835204e206e403c2b015ff
2ae7d78dfd02ce4c34eeabd14692c4914b05768fa4b19dd082aef
8cebf741593dca6ed35ece04b180f80bb1b118c765413ca4702db
e16420ae2ea4239d901ba135887a54890cb182c3e0f67c001265f
1a1383927cce9a6c0","cipher":"MQldn6J/xKrQ99X+IoPzlrv4
JvQI8vccdGd9PH160RI="}
```

4. postPublicKeyTransaction

```
bob.postPublicKeyTransaction(obj.dataName, alice);
alice.postReEncryptionKeyTransaction(obj.dataName, proxy);
```

The next thing is to send the buyer's public key to the data owner so that the data owner cam generate the re-encryption and sends it to the proxy. The public/private key is different than the public/private key of the client's address. It is generated from the recrypt-js library, and the data owner has no information before. Thus we implemented a method named `postPublicKeyTransaction` in the `Client.js`.

```
postPublicKeyTransaction(dataName, dataOwner,
    fee=Blockchain.DEFAULT_TX_FEE) {
```

```
    let kp = this.proxyKeyPair.get(dataName);
    dataOwner.saveReceivedPublicKey(dataName, kp.pk);
    return this.postGenericTransaction({
      cipher: kp.pk,
      fee: fee
    });
  }

saveReceivedPublicKey(dataName, receivedPubKey) {
    this.buyersPubKey.set(dataName,receivedPubKey);
  }
```

The `saveReceivedPublicKey` method is to save the buyer's public key into a map. The key is the name of the data, and the value is the public key.

5. postReEncryptionKeyTransaction

```
 alice.postReEncryptionKeyTransaction(obj.dataName, proxy);
```

This is to generate the re-encryption using the data owner's private and buyer's private keys. After the data owner generates the re-encryption, the data owner sends the re-encryption key to the proxy and saves it on the proxy side. We implement a method called `postReEncryptionKeyTransaction` inside the Client.js. Inside this method, we use the recrypt-js library to generate the re-encryption key.

```
postReEncryptionKeyTransaction(dataName, proxy,
fee=Blockchain.DEFAULT_TX_FEE) {
    let kp = this.proxyKeyPair.get(dataName);
    let rekey = PRE.generateReEncrytionKey(kp.sk,
                this.buyersPubKey.get(dataName));
    proxy.saveRencryptionKey(dataName, rekey);
    return this.postGenericTransaction({
      rekey: rekey,
      fee: fee
    });
  }
saveRencryptionKey(dataName, reEncryptionKey) {
      this.reEncryptionKey.set(dataName, reEncryptionKey);
  }
```

The `saveRencryptionKey` method is to save the re-encryption from the data owner for the proxy. The key is the name of the data, and the value is the re-encryption key. Finally, the `postReEncryptionKeyTransaction` broadcasts the transaction to the blockchain.

Below shows the re-encryption key we can find in the blockchain:

```
The re-encryption key is:

3ab6dd1cfc48655d72e0b221e29b9c0e8ea1bf6337f14
4727257ede422b3630f04e57e0c2c325efa365c2ec05b
28a7d5bd61cf443b27450033c97fa185b3c23a26907aa
b9431958bfa0467a7a267699a720c551d317441a889ea
d2e7fee71d1937
```

6. postReEncryptionDataTransaction

The `postReEncryptedDataTransaction` function is responsible for re=encrypting the initial encrypted data by using the re-encryption key and then sending the result to the buyer. We implement a method inside the Proxy.js called `postReEncryptedDataTransaction`.

```
proxy.postReEncryptedDataTransaction(obj.dataName, bob);
```

```
postReEncryptedDataTransaction(dataName, buyer,
    fee=Blockchain.DEFAULT_TX_FEE) {
    PRE.reEncryption(this.reEncryptionKey.get(dataName),
                this.encryptedData.get(dataName));
    buyer.uploadPrivateData.set(dataName,
                 this.encryptedData.get(dataName));
    let tx = super.postGenericTransaction({
        fee: fee
      });
    return this.addTransaction(tx);
  }
```

The first parameter is the name of the data, which is helped to find the corresponding re-encryption key and initial encrypted data. The second parameter is the buyer that the proxy would send. Inside this method, we use the recrypt-js to re-encrypt the encrypted data with the re-encryption key. Then we call the `uploadPrivateData` method from the buyer's side, which is to save the re-encrypted data for them. Finally, the proxy would broadcast the transaction to the blockchain network.

Below shows the re-encrypted data by using the re-encryption from the blockchain:

```
The encrypted data by using re-encryption key is
```

```
{"key":"04b531b558a978bacd19ba3a89a7436788ccc4b83
8c2ce70498daf8f4095db0b6304dcce74b7c8682ed3dbca3c8
1bc4e14fd175aafc279772784a54512976ed743048e99d77a1
5738e1bdc1306e79982b7417268ba08e957ad080d68749c163
698ca8f323bcbc0b86f35971bad85e668a81af94fdfe86b0d0
8146b8629c643a2fa37ae2ea4239d901ba135887a54890cb18
2c3e0f67c001265f1a1383927cce9a6c004e57e0c2c325efa3
65c2ec05b28a7d5bd61cf443b27450033c97fa185b3c23a269
07aab9431958bfa0467a7a267699a720c551d317441a889ead
2e7fee71d1937","cipher":"MQldn6J/xKrQ99X+IoPzlrv4JvQI8vccdGd9PH160RI="}
```

7. decrypt

   Finally, after Bob receives re-encrypted data on their side, he could use his private key to decrypt the content.

```
decrypt(dataName) {
    let kp = this.proxyKeyPair.get(dataName);
    let decryptData = PRE.decryptData(kp.sk,
          this.uploadPrivateData.get(dataName));
    return decryptData;
  }
```

```
Bob’s decrypt result is: Pfizer, took on March 2021
```

CHAPTER 5

Use Case - Medical Information

Our introductory example shows that our protocol could be used for exchanging and storing medical information. Our protocol brings a few advantages to the medical information system compared to the traditional medical records system. Our protocol provides strong confidentiality with PRE and relies on blockchain storage for distributed perspective. In this way, there is less control of a single party in the system [26]; instead, the user controls their own data. Also, it is easier to transfer records to alternative providers.

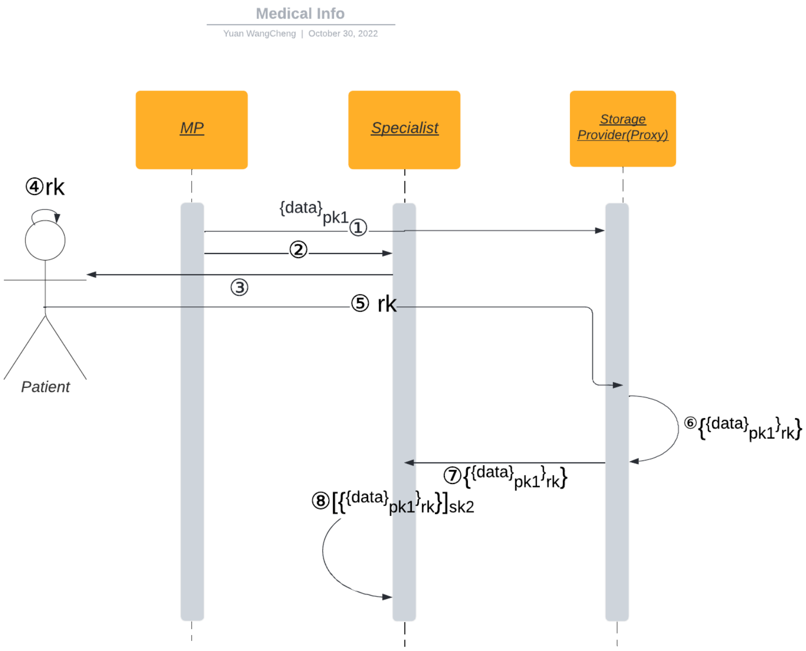

Figure 4: Recording and sharing medical data using PRE

Figure 2 shows the procedures for how data is stored and exchanged using our protocol. Let's assume that the patient goes to see the medical provider (MP):

1. MP writes the patient's information (`data`), encrypted with the patient's public key, pk, and sends the result, M1 to the proxy in the blockchain

2. The MP refers the patient to the specialist

3. The specialist asks the patient for the previous medical case information

4. The patient releases the data and generates a re-encryption proxy by using their private key, sk, and the specialist's public key, pk2.

5. The patient sends the re-encryption key to the proxy

6. The proxy uses the re-encryption to encrypt the patient's medical information for the specialist's public key

7. The proxy sends the result, M2, to the specialist.

8. The specialist uses their private key, sk2, to decrypt M2 to view the patient's medical information

This approach offers the following benefits:

- Patients control their own data

  If the medical practice goes down or is otherwise unavailable, the medical data is still stored on the blockchain. Compared to traditional medical record systems, the data would not go away unless the client requested it. Our protocol can prevent data from being deleted by accidents such as system maintenance or centralized deletion. Our protocol also streamlines the process of exchanging data. There is no need to involve the original medical doctor in decisions about releasing patient data. All the system needs to do is get permission from the patient, and the patient generates the re-encryption key and sends it to the blockchain to release the data.

- Updating medical results

  In addition to easily exchanging data stored on the blockchain, the specialist can further append new results to the blockchain. The client first releases their medical records to the specialist by sending a re-encryption key to the proxy. Then the proxy sends the re-encrypted data to the specialist. The specialist appends new results to the blockchain using the patient's public key. Thus the next specialist who needs the patient's medical information could see the previous history. Critically nothing ties the client to the data on the blockchain when the specialist writes to it.

5.1 Inference Attack Against Anonymity

From our protocol and implementation, each data transaction will initiate different public/private key pairs for the proxy re-encryption process. However, the patient's and doctor's addresses remain the same in the SG network. For example, if the patient sends their information to the PCP doctor, both of them generate the public/private key pair for the proxy re-encryption. If the next time the patient sends information to the same doctor, both of them will have different proxy re-encryption keys, but their spartan gold address does not change.

Even if the encrypted information is still not visible or easy to break in the blockchain, people can gather some information from both the sender's and receiver's addresses. For instance, if people saw in the blockchain that the patient sent multiple times to a specific address, they can know that the patient is connected to that specific address for some reason or purpose, but they can't figure out what exactly the information is. However, if people can figure out the identity is the receiver's address, then things become much worse. For example, if people can figure out the address is an Ear Nose Throat (ENT) doctor, and people know this particular patient has an ENT problem, which breaks the confidentiality in our proposed protocol [27]. The inference attack is a limitation in our proposed Oathkeeper implementation. The solution to the inference attack will be an area for our future work.

CHAPTER 6

Experiemnts

This chapter shows how our implemented protocol, Oathkeeper, works on the medical data. In this case, we will have the patient delegates their medical record to the specialist after they complete their session with Medical Provider (MP). The medical data are stored on the proxy, which in reality, it will be the cloud provider storage.

6.1 Announcing Data to Blockchain

First, we discuss what we do after we view our MP first. Usually, when the patient has some sickness and views the MP. It would be great if the MP could recognize the patient's problem and give a corresponding medical solution. If not, the MP needs to refer the patient to some specialists for further investigation. In order to better help the specialist to give a medical solution, most of the time, they need the medical record data from the MP's session. Each time the patient views the MP, the MP will announce the session data into the blockchain. In the blockchain, all people who are registered in the medical blockchain system can see the attribute information, such as the date, where it is from, and some descriptions. But people can't see the exact medical record data. Below is what the data looks like in the blockchain network. As you can see, people can only see the fields, but the data itself is never exposed.

```
'6EVyFf7Hn5OI/CuecOtktiRH+huCiRUpuBXm8BVB1eA=' => {
      dataName: 'COVID-19 Vaccine',
      date: 'Nov 10, 2022',
      MP: 'Kaiser Permanente South San Francisco Medical Center',
      description: 'COVID-19  vaccine symptoms',
      price: 33
    },
```

The `6EVyFf7Hn5OI/CuecOtktiRH+huCiRUpuBXm8BVB1eA=` is the address of the

MP doctor. The price here does not stand for the gold balance. This could be some medical credential number or the code that can confirm anyone has to use this specific number to prove the right privilege.

6.2 Sharing Data with Proxy

The next step is to share the data with the proxy. As we discussed in Chapter 3, the data is stored on the proxy side, a cloud provider in the real world. Even though the patient has the initial data, it is still a better practice to store the data in a decentralized manner. Using our protocol, the patient can encrypt the data by using their public key and send the encrypted data to the proxy. Both the patient and the protocol are part of the blockchain. You can consider the proxy as another client but save the data as the storage provider. Below is how the initial encrypted data looks like in the blockchain:

```
{"key":"0498e45eb7c77052c660eda23072df0b236
ae61a8dd3b0e658e3355f3e5d22dfd46ec720b4901f
28199a8cffad227df8e47f62a5ccdf321230da6821d
516de3ff30468578c252e97d2446bfc1c3e374f8d62
b72de7c704b8a63be00b283046b5aacb85547da4e3e
5b0c32286587a4e8ab736890f8e31c349d99b282d18
526375c1b6f72e49ab11b39099a40eaea42e3f82c2b
a7a07fb4be94aaffa5a53670948c128","cipher":
LzP2YKHVNhKSozpxAzroZ6xRyngrTfGTO59NYDltnLI
hZsAzCautQulhM+gGZ4yF"}
```

6.3 Requesting Medical Data

When the patient is referred by the MP, the specialist will then start approaching the patient. The specialist first needs to send the patient a message that can confirm they have the right authority to take care of the patient's case and needs the data. Thus, they need to call the `postPriceTransaction`, where the price is the special token representing their privilege. If the specialist can successfully confirm their identity, then the patient and the specialist can start the re-encryption process. The

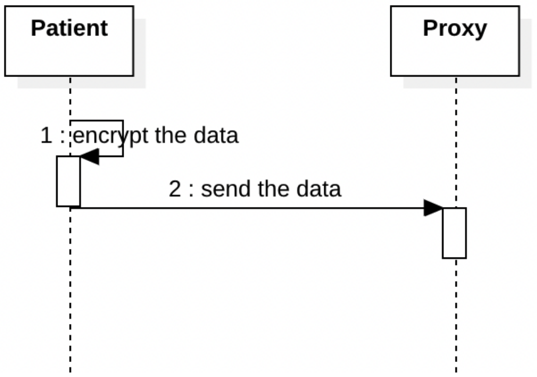

Figure 5: Sharing Data with Proxy

patient and the specialist will generate different public/private re-encryption keys for each data transaction.

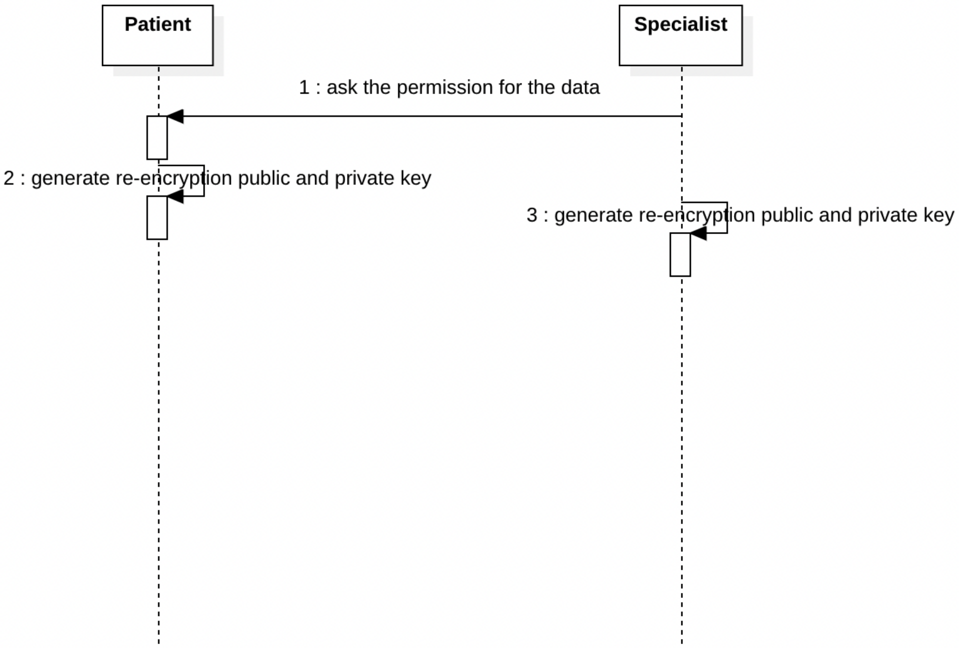
Patient
Specialist
1 : ask the permission for the data
2 : generate re-encryption public and private key
3 : generate re-encryption public and private key

Figure 6: Specialist asks for the permission from the patient

6.3.1 Multiple Specialists

There is a scenario in which the patient is referred to multiple specialists. In this case, multiple specialists need to contact the patient that they need the session data. Our protocol can successfully achieve that by generating different re-encryption public/private keys for both the patient and the specialist. Since the data is stored on the proxy, the patient just needs to use each specialist's public key to generate the unique re-encryption keys for them in this case, and there is no conflict between all specialists.

6.4 Re-encrypting Data

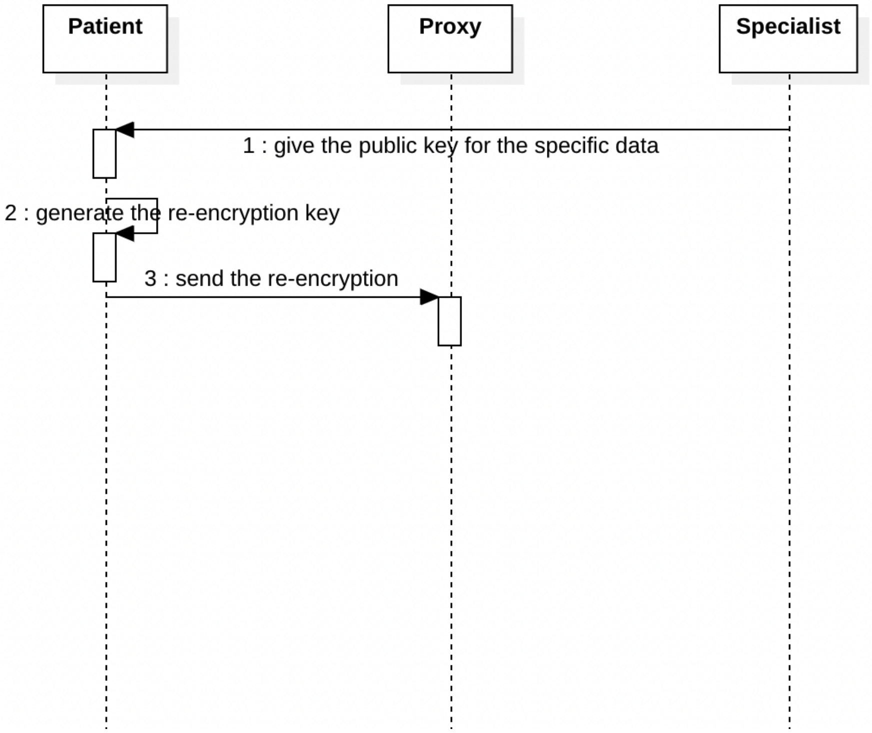
Patient
Proxy
Specialist
1 : give the public key for the specific data
2 : generate the re-encryption key
3 : send the re-encryption

Figure 7: Re-Encryption Process among three identities

Now the patient needs to do two things: (1) generate the re-encryption key and (2) re-encrypt the data and send it to the proxy. To generate the re-encryption key, the patient needs the specialist's public key along with their private key. Below is what the re-encryption key looks like:

```
The re-encryption key is:
```

```
df5a10658d836c1642d29e1268e9133a3d40ae83e6
4ac93a6e86b1a21e80061e04770f9cf2408bfa1bd5
f579280a8aabb5ae8325341dc92e67ed23e148b263
2a9b44ba9a0e4bf6661a23af7ec9cda06b0ca1ca0c
6e89f73657751cea766769f8dd
```

After the patient generates the re-encryption key, they send it to the proxy and the proxy can start the re-encryption process.

6.4.1 Re-encryption By Proxy

The proxy now has the re-encryption key and the initial encrypted data. All they need to do now is to re-encrypt the data and send the final result to the specialist.

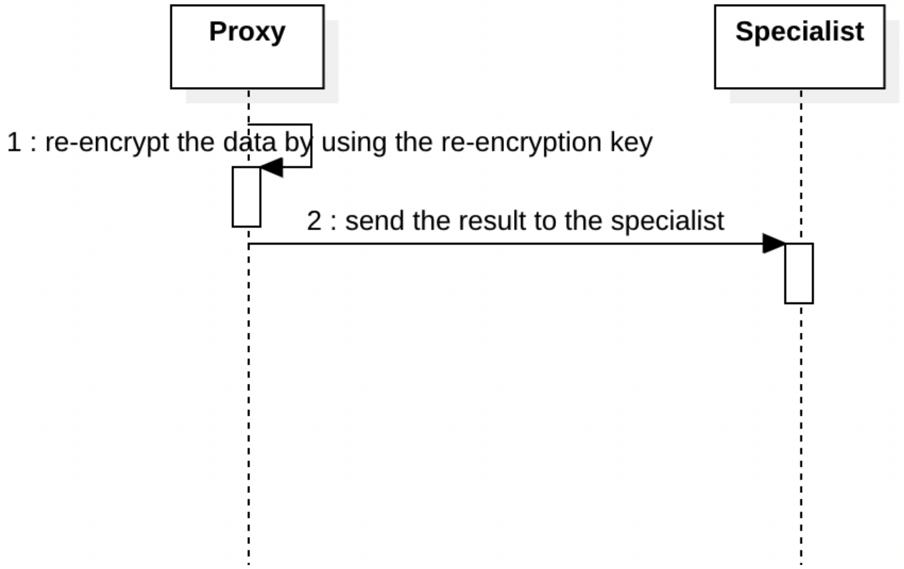
Proxy
Specialist
1 : re-encrypt the data by using the re-encryption key
2 : send the result to the specialist

Figure 8: Proxy re-encrypt the data by using the re-encryption key

Below show the data after the re-encryption process:

```
The encrypted data by using re-encryption key is
{"key":"0464d8f0be30f7fe22599e9a99712b23c8db8c00
d673f37551f1cb7066a1867a7979220075313e99a60ef217
e22957f992b87b2dc11a8ef3ece37a36516d06344b04c3ef
e1cac2cd27d8a0e2de573af5f064404e80198d10bce44ff5
15e5000da8100a1e8cdc9b254e22167a1f7c84487f3605ad
17727d4d01c28966c38c0c34653bf72e49ab11b39099a40e
aea42e3f82c2ba7a07fb4be94aaffa5a53670948c1280477
0f9cf2408bfa1bd5f579280a8aabb5ae8325341dc92e67ed
23e148b2632a9b44ba9a0e4bf6661a23af7ec9cda06b0ca1
ca0c6e89f73657751cea766769f8dd","cipher":"LzP2YKH
VNhKSozpxAzroZ6xRyngrTfGTO59NYDltnLIhZsAzCautQulhM+gGZ4yF"}
```

6.4.2 Decrypting Re-encrypted Data

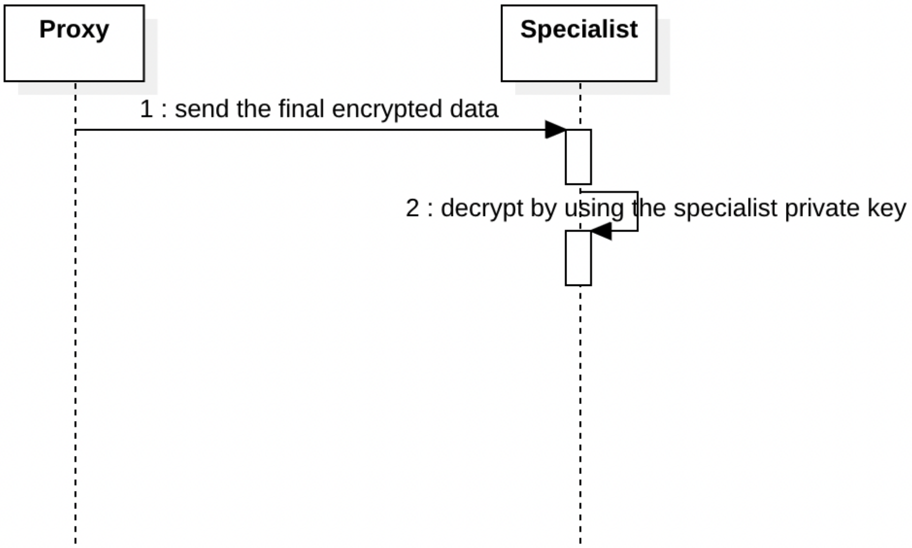
Proxy
Specialist
1 : send the final encrypted data
2 : decrypt by using the specialist private key

Figure 9: Specialist decrypts the data

The last step is to send the final encrypted result to the specialist, and only the specialist can decrypt it by using their specialist's private key.

```
Specialist's decrypt result is: Headache along with muscle pain.
```

CHAPTER 7

Conclusion and Future Work

In conclusion, we implemented a protocol named Oathkeeper that combines PRE with the blockchain providing confidentiality of the data stored on the blockchain. At the same time, Oathkeeper achieved the ability to store data in a decentralized manner. Oathkeeper can be used in many real-world cases, such as medical data exchange. We show that Oathkeeper helps the medical system deliver patient's data without content exposure while the patient's data are stored by a third party. Oathkeeper can also help the patient delegate their data to different specialists at the same time, using different re-encryption keys.

There are a few areas in which the Oathkeeper can be improved in the future. First of all, in Chapter 3, we explained the inference attack against anonymity, which is an attack to gather information from pre-known or fixed addresses on the blockchain. The problem with the above attack is that people can figure out if the patient's address is connected to the doctor's. Future work will seek out a way to hide the SG address in our protocol, this avoiding the problem above. Since the proxy is the only party that needs the receiver's address, the sender can directly send the receiver's SG address to the proxy. Finally, the proxy will encrypt the receiver's SG address and broadcast it to the blockchain.

Another improvement can be integrating Oathkeeper with data storage services on the cloud provider, such as S3 [28] for videos, images, and structural files and DynamoDB [29] for metadata. In our current implementation of OathKeeper, the proxy is implemented as a class object in which the data is stored within the class. Instead of saving the data within the proxy class, we can store the data in the real cloud provider so that the proxy is only responsible for generating the re-encryption key and re-encryption data. We can further improve Oathkeeper by registering the cloud storage provider to our SG network. In this way, we no longer need the proxy as an object in our protocol; instead, we can build serverless services, such as Lambda [30] on the cloud side, to handle the proxy re-encryption process, meanwhile still storing the data.

LIST OF REFERENCES

[1] S. Nakamoto, ‘‘Bitcoin: A peer-to-peer electronic cash system,’’ *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[2] A. Urquhart, ‘‘The inefficiency of bitcoin,’’ *Economics Letters*, vol. 148, p. 80–82, 2016.

[3] V. Malik, ‘‘The history and the future of bitcoin,’’ 2016.

[4] L. V. Astakhova and N. V. Kalyazin, ‘‘Non-fungible tokens (nft) as a means and object of ensuring information security,’’ *Automatic Documentation and Mathematical Linguistics*, vol. 56, no. 3, p. 116–121, 2022.

[5] R. Sharma, ‘‘Non-fungible token (nft): What it means and how it works,’’ Nov 2022, accessed on 12-10-2022. [Online]. Available: https://www.investopedia.com/non-fungible-tokens-nft-5115211

[6] H. Bao and D. Roubaud, ‘‘Non-fungible token: A systematic review and research agenda,’’ *Journal of Risk and Financial Management*, vol. 15, no. 5, 2022. [Online]. Available: https://www.mdpi.com/1911-8074/15/5/215

[7] M. C. Kus Khalilov and A. Levi, ‘‘A survey on anonymity and privacy in bitcoin-like digital cash systems,’’ *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2543--2585, 2018.

[8] A. Urquhart, ‘‘The inefficiency of bitcoin,’’ *Economics Letters*, vol. 148, pp. 80--82, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165176516303640

[9] Z. Chen, A. Wu, Y. Li, Q. Xing, and S. Geng, ‘‘Blockchain-enabled public key encryption with multi-keyword search in cloud computing,’’ *Security and Communication Networks*, vol. 2021, pp. 1--11, 01 2021.

[10] B. Partner, ‘‘Decentralized identitynbsp;: Granting privacy with proxy re-encryption,’’ Jun 2019, accessed on 12-10-2022. [Online]. Available: https://medium.com/@teamtech/decentralized-identity-granting-privacy-with-proxy-re-encryption-e0bf68ad465c

[11] Taustin, ‘‘Taustin/spartan-gold: A simplified blockchain-based cryptocurrency for experimentation,’’ accessed on 12-10-2022. [Online]. Available: https://github.com/taustin/spartan-gold

[12] S. Meunier, ‘‘Chapter 3 - blockchain 101: What is blockchain and how does this revolutionary technology work?’’ in *Transforming Climate Finance and Green Investment with Blockchains*, A. Marke, Ed. Academic Press, 2018, pp. 23--34. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128144473000033

[13] I. Acharjamayum, R. Patgiri, and D. Devi, ‘‘Blockchain: A tale of peer to peer security,’’ in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2018, pp. 609--617.

[14] C. Schinckus, ‘‘Proof-of-work based blockchain technology and anthropocene: An undermined situation?’’ *Renewable and Sustainable Energy Reviews*, vol. 152, p. 111682, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032121009564

[15] J. Sedlmeir, J. Lautenschlager, G. Fridgen, and N. Urbach, ‘‘The transparency challenge of blockchain in organizations,’’ *Electronic Markets*, vol. 32, 03 2022.

[16] A. Ismail, M. Toohey, Y. C. Lee, Z. Dong, and A. Y. Zomaya, ‘‘Cost and performance analysis on decentralized file systems for blockchain-based applications: State-of-the-art report,’’ in *2022 IEEE International Conference on Blockchain (Blockchain)*, 2022, pp. 230--237.

[17] X. Zhang, J. Grannis, I. Baggili, and N. L. Beebe, ‘‘Frameup: An incriminatory attack on storj: A peer to peer blockchain enabled distributed storage system,’’ *Digital Investigation*, vol. 29, pp. 28--42, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287618303438

[18] J. Sun, L. Ren, S. Wang, and X. Yao, ‘‘A blockchain-based framework for electronic medical records sharing with fine-grained access control,’’ *PloS one*, vol. 15, p. e0239946, 10 2020.

[19] J. Fu, N. Wang, and Y. Cai, ‘‘Privacy-preserving in healthcare blockchain systems based on lightweight message sharing,’’ *Sensors*, vol. 20, no. 7, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/7/1898

[20] YouTube, Jul 2018. [Online]. Available: https://www.youtube.com/watch?v=1RuP2HZ6zRc&amp;t=212s&amp;ab_channel=TaariqLewis

[21] B. Partner, ‘‘Decentralized identitynbsp;: Granting privacy with proxy re-encryption,’’ Jun 2019, accessed on 12-10-2022. [Online]. Available: https://medium.com/@teamtech/decentralized-identity-granting-privacy-with-proxy-re-encryption-e0bf68ad465c

[22] B. Rawal and Y. Wang, ‘‘Splitting a pre-scheme on private blockchain,’’ 05 2019.

[23] P. Miao, S. Patranabis, and G. Watson, ‘‘Unidirectional updatable encryption and proxy re-encryption from ddh or lwe,’’ Cryptology ePrint Archive, Paper 2022/311, 2022, https://eprint.iacr.org/2022/311. [Online]. Available: https://eprint.iacr.org/2022/311

[24] Y. Shi, J. Liu, Z. Han, Q. Zheng, R. Zhang, and S. Qiu, ‘‘Attribute-based proxy re-encryption with keyword search,’’ *PLOS ONE*, vol. 9, no. 12, p. e116325, 2014. [Online]. Available: https://app.dimensions.ai/details/publication/pub.1041299693

[25] Yjjnls, ‘‘Yjjnls/recrypt-js: Js sdk for proxy reencryption functionality, support spec256k1,’’ accessed on 12-10-2022. [Online]. Available: https://github.com/yjjnls/recrypt-js

[26] K. Singh, C. Rangan, R. Agrawal, and S. Sheshank, ‘‘Provably secure lattice based identity based unidirectional pre and pre + schemes,’’ *Journal of Information Security and Applications*, vol. 54, p. 102569, 10 2020.

[27] X. Chen, J. Zhang, D. Wu, and R. Han, ‘‘Hippa’s compliant auditing system for medical imaging system,’’ in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, 2005, pp. 562--563.

[28] ‘‘S3,’’ 2002, accessed on 12-10-2022. [Online]. Available: https://aws.amazon.com/s3/

[29] D. Rangel, ‘‘Dynamodb: Everything you need to know about amazon web service’s nosql database,’’ 2015, accessed on 12-10-2022. [Online]. Available: https://aws.amazon.com/dynamodb/

[30] R. W. Hendrix, ‘‘Lambda,’’ 1983, accessed on 12-10-2022. [Online]. Available: https://aws.amazon.com/lambda/