國立陽明交通大學

網 路 工 程 研 究 所

碩 士 論 文

初 稿

基於區塊鏈之身份識別與存取控制框架 - 以開放銀行為例

Blockchain-based Identification and Access Control Framework - A Case Study of Open Banking Ecosystem

研 究 生：鄭人豪

指導教授：袁賢銘 教授

中 華 民 國 110 年 8 月

基於區塊鏈之身份識別與存取控制框架 - 以開放銀行為例
Blockchain-based Identification and Access Control Framework - A Case Study of Open Banking Ecosystem

研 究 生：鄭人豪　　　　　　　　Student: Jen-Hao Cheng

指導教授：袁賢銘　　　　　　　　Advisor: Shyan-Ming Yuan

國 立 陽 明 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文 初 稿

A Thesis Draft

Submitted to Institute of Network Engineering

College of Computer Science

National Yang Ming Chiao Tung University

in partial fulfilment of the requirements

for the Degree of

Master

in

Computer Science

Aug 2021

Hsinchu, Taiwan

中 華 民 國 110 年 8 月

基於區塊鏈之身份識別與存取控制框架 - 以開放銀行為例

學生：鄭人豪　　　　　　　　　　　　　　　指導教授：袁賢銘 教授

國立陽明交通大學 網路工程研究所

摘　　要

　　隨著社群平臺愈來愈普及，多數網站提供第三方登入 (social login)，讓初次使用的用戶可以用第三方平臺現有的帳號完成註冊及登入，幫助用戶免於記下不同網站的帳號及密碼，亦不用填寫繁雜的註冊表單。對於用戶而言，可以達到更好的用戶使用體驗；對於應用程式開發人員而言，不必自行管理個資、建立會員系統，由第三方平臺負責管理，而當需要在提供多種不同的服務時，可以使這些服務支援同一種第三方平臺驗證方式，即可達到單一登入 (SSO) 功能。但第三方登入系統仍屬於中心化系統，用戶的數位身分及資料皆屬於第三方平臺，因此用戶在使用服務的同時，也提供個人資料給服務提供者。

　　然而，以太坊區塊鏈擁有防偽造、防竄改及去中心化的特性，可以安全、有效存放紀錄於鏈上，達到透明性且安全性。透過以太坊虛擬機及以太坊智能合約可以建構去中心化的應用程式，提供更完整、多樣的功能。

　　本篇論文提出使用以太坊區塊鏈技術應用於開放銀行 (Open Banking)，使銀行機構與第三方服務業者 (TSP) 以安全、透明方式合作，透過區塊鏈技術管理並整合用戶身分，方便用戶管理及存取資料，且提供第三方登入的優點，並使得去中心化平臺成為信任的第三方，負責管理對應用戶身分並且提供存取控制之功能，同時兼具區鏈特性，提升安全性。除了驗證用戶身分功能外，亦提供用戶授權功能，用戶可以自行決定授權範圍及對象，授權對象包含第三方服務 (TSP) 業者。

關鍵字: 區塊鏈、智能合約、第三方服務提供者、開放銀行

Blockchain-based Identification and Access Control Framework - A Case Study of Open Banking Ecosystem

Student: Jen-Hao Cheng   Advisor: Dr. Shyan-Ming Yuan

Institute of Network Engineering
National Yang Ming Chiao Tung University

Abstract

Open Banking has become a trend in the financial industries for providing innovative and diverse services. The concept of open banking aims to give third-party service providers (TSPs) the right access to customer's financial data for further analysis uses, thereby helping them to get a better deal and improve the customer experience. Since an open banking ecosystem serves as a platform for various participants and establishes trust through trust third party, there has been increasing attention paid to personal data privacy and identity integration. Previous studies have examined blockchain technology applied to open banking for protecting customer's privacy, but identity verification and integration have been lacking.

The objective of our research was to propose a general framework for blockchain-based identification and account integration. In this framework we proposed, the customers can view their latest complete data and have the capability to authorize particular TSP to access their financial data. We apply programmable smart contracts to our framework to realize these functionalities and build a distributed application (DApp) that gives customers more autonomy over their own data. Moreover, we also evaluate the performance of the proposed framework and compare it with a centralized system.

Keywords: Blockchain, Smart contract, Third-party providers, Open banking

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years social login services have become ever more prevalent such as Google, Facebook, or Twitter. The user can sign into a third party website without creating a new account. Those services provide unified identity management through social login and allow the user to confirm the access permission. Although social login seems to have a lot of benefits, e.g., security, convenience, and ease of use, there has a concern about those service providers may collect sensitive information about the user [1]. Besides, those social login services are mainly based on the centralized systems that enforce data permit across different parties. If one third party application sends a retrieval request for the data, it must obtain permission from the centralized party.

Open Banking is a hot topic in the financial industry nowadays. It aims to share customer's financial data with different organizations and required consent. The banking institutions disclose APIs to third-party service providers for creating new services, analytics, financial products to improve customer experience. It is not only meeting customer needs but also help third-party service provider to create innovate activity for exploring prospective customers and accelerate financial inclusion. In recent years, Open Banking has been adopted in various stages in countries around the world. Three phases have been defined by Open Banking and each stage represents sharing data scope. In Taiwan, Financial Supervisory Commission (FSC) has approved several banks allowing to join in the second phase of Open Bnaking [2].

- **Phase 1 Public information:** interest rates, exchange rates, mortgage rates, foreign currency exchange rates, etc.

- **Phase 2 Customer data:** accounts information, loan, deposit, credit cards, personal information, etc.

- **Phase 3 Transaction information:** account integration, payment, debit authorization, settlement of the loan, etc.

A key issue is the privacy of customer's personal data including customer's deposits, loans, investments, and account information. When financial institutions disclose APIs to TSPs, the system has numerous critical concerns such as malicious attacks, tampering. Once the attacker hacks the system, the customer data will be exposed and may cause enormous losses. Blockchain technology can provide data storage, access control, transaction security, and tamper-proof data so that it can protect customer data privacy [3].

Blockchain technology brings numerous benefits in a variety of industries, providing more security in trustless environments. The blockchain is a distributed digital ledger that storing records or data in blocks and those blocks are linked through cryptographic proofs so that the attacker can not temper any data. In Ethereum blockchain, it provides smart contract to us to deploy autonomous applications without third party and interact with smart contract on Ethereum network.

1.2   Objective

In order to address these problems, we propose blockchain-based identification and access control system for Open Banking ecosystem in this thesis. The system allows organization administrators to interact with blockchain for register the digital identity of the customer by the actual identity, and customer also can manage their digital identity and control their data access by calling smart contract functions directly. If any organization or financial institution wants to participate in existed blockchain-based Open Banking ecosystem, the system platform will create an Ethereum account for them so that they can prove they have ownership for Ethereum address. Customer registers their bank account and then bind their actual identity to their Ethereum address for enabling blockchain-based third-party authentication. After binding, customers have a unique digital identity on blockchain, and they can log in to another financial institution or third-party service provider without filling any form. In addition to binding, customers can also integrate existed accounts into their unique digital identity on blockchain.

With blockchain technology, the system is more secure, traceable, transparent, and tamper-resistant. TSP also does not have to create authentication systems and it can ensure customer's digital identity doesn't have tampered with. TSP is required to invoke smart contract function to confirm their access permission and access scope. After the financial institution successfully authenticating the TSP, it issues access tokens to TSP. TSP can request customer data with access token through API.

This thesis is organized as follows nine chapters. Chapter 1 and 2 introduce the background knowledge of blockchain and related work in Open Banking ecosystem. Chapter 3 describes the system overview, its scenario and explains how it work. Chpater 4 presents the smart contract used in our system and its workflow. Chapter 5 illustrates case studies and experimental validation. Chapter 6 presents demonstration of our system. Chapter 7 describes the evaluation of our system performance, including gas consumption and its throughput. Finally, we summarize our system functions, our finding, and discussion in Chapter 8 and 9.

Chapter 2

Background

2.1 Ethereum

2.1.1 Smart Contract

Ethereum [4] was proposed in 2014 by Vitalik Buterin, it is a decentralized, open-source blockchain, and it also supports smart contracts. Because Ethereum enables smart contracts, the programmers can build their distributed applications by writing smart contract programs, e.g., Solidity. With smart contract technology, they can build self-enforce, self-verify and tamper-proof systems, such as voting systems, healthcare, supply chain, financial service, and so on.

In the Etheruem platform, the block not only stores transaction records, but also stores smart contract so that Ethereum has the capability to compute business logic. The blockchain uses Merkle tree to store the transactions in every block.
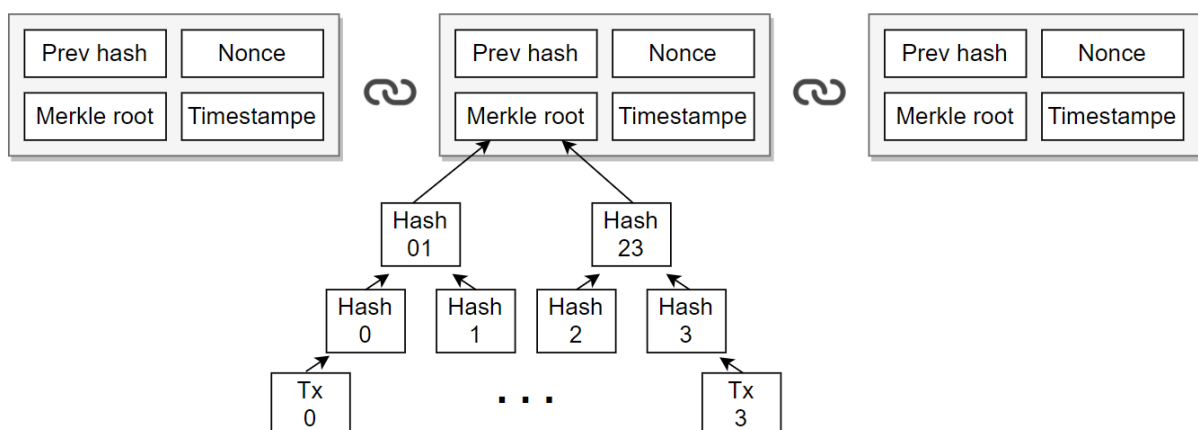
2.1.2 Merkle tree

Figure 2.1: Merkle tree in Bitcoin blockchain

Merkle trees are important in blockchain technology. A Merkle tree is a tree in which every node is stored with the hash of data. Each node is the hash of his leaves. In Bitcoin blockchain,

it uses the Merkle tree as proof to make sure the data block can't be tampered with, it is not possible to modify the data after the data written in the blockchain as shown Figure 2.1.
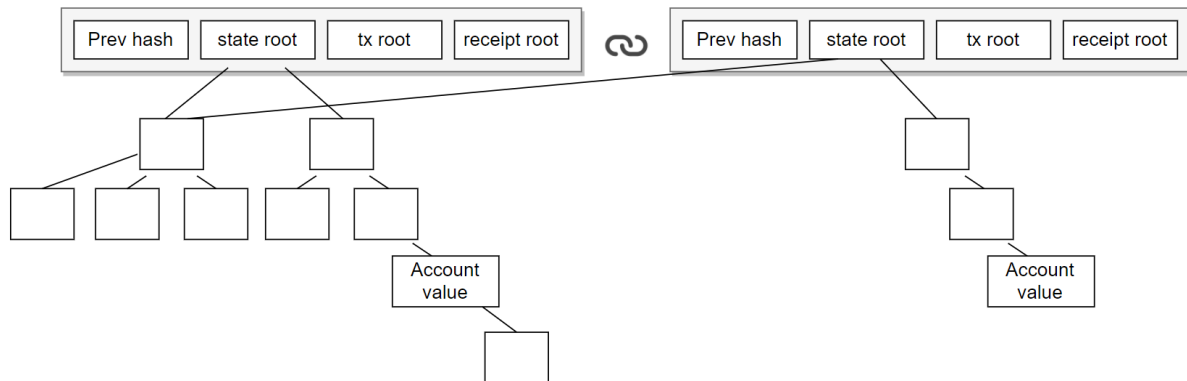
Figure 2.2: Merkle tree in Ethereum

The Merkle tree in Ethereum is as Figure 2.2. Every block header has not only one tree, but three trees for transactions, receipts, and state. The state root is the root hash of the Merkle Patricia tree that used to store the entire state of the Ethereum blockchain, like account balances, contract storage, and contract code. Unlike Bitcoin blockchain, Ethereum uses Merkle Patricia tree as the state tree, it consists of a map struct, the keys are account address and the values are account declarations such as the balance, nonce.

2.1.3 Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography is public key cryptography based on elliptic curves. In Ethereum, they use ECC to generate the key pair. Initially, the base point G on elliptic curve $CURVE$ must be provided, and then randomly generate 256-bits integer m as a private key which is used to sign message or transaction. If given m, it is easy and fast to find 512-bits public key P. But if given P, it is impossible to find m because of Elliptic Curve Logarithm Problem. The Ethereum address is the last 20-bytes of Keccak-256 hash of the public key.

$$P = [m]G \tag{2.1}$$

Elliptic Curve Digital Signature Algorithm (ECDSA) is used to create a digital signature of data that uses elliptic curve cryptography. ECDSA isn't used to encrypt any data, it makes sure

that the data was not tampered with. In the Ethereum blockchain, it used to prove ownership of an address without revealing private key.

After signing message using ECDSA through private key, the digital signature consists of three values: r, s, v. r and s are the values used in standard ECDSA signatures, and v means recover id that used to recover signed message. In blockchain (e.g., Bitcoin, Ethereum), we called it public key recovery. If we get (r, s) and message, we can compute the public key to verify message.

2.2 MetaMask

Figure 2.3: Architecture of DApp

MetaMask [5] is the most popular crypto wallet for accessing Ethereum distributed application (DApp). This tool can enable web3 API in website so that users can interact with various Etehereum blockchain from Javascript [6], e.g., Mainnet, Testnet. It also creates accounts by the user themself. The user of MetaMask can create and manage their accounts; moreover, Meta-Mask provides an interface that user can perform a transaction to the connected blockchain.

Because the user securely manages owned Ethereum account through MetaMask, the user can use their private key to sign a transaction or sign data to prove ownership of an account. Using this crypto wallet, the developers of decentralized applications can build their own cryptocurrencies and focus on designing and implementing functions of smart contracts. There are several different architectures for distributed apps [7], the most common way is the developer design frontend that can allow the users to interact with the business logic by MetaMask. Or, the user can send a transaction to blockchain directly.

In the MetaMask wallet, it seems to have a lot of benefits. Firstly, the keys are stored in the user's browser and it doesn't store on wallet provider's server, so the user can manage his/her private key and public key without server. Secondly, it provides an easy to use interface, every user can send and receive cryptocurrency or token.

Regarding the architecture of Dapp, Figure 2.3 shows that the web3.js libraries can enable user's browser to interact with blockchain so that users can read and write data from smart contracts, send transactions between accounts.

2.3 Open Banking
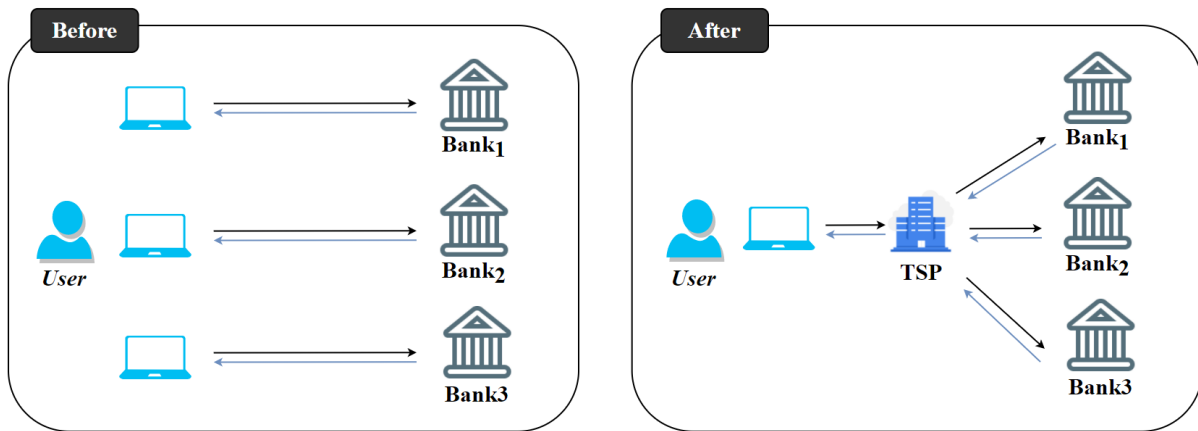
Figure 2.4: Relaptionship between users and banks

In the traditional banking system, the transaction data of a user's account is widely regarded as bank assets. Open Banking aims to return the control of bank account information to users so that users have the right to allow other TSPs to access their account information using API. Sharing financial data can bring more diverse financial services to users and it can also promote innovation and cooperation between banks and TSP. The relationship between users and banks is shown in Figure 2.4.

The Open banking is governed by UK and European regulations including Revised Payment Services Directive (PSD2), General Data Protection Regulation(GDPR), etc. In the United Kingdom, Open Banking Implementation Entity (OBIE) was created to develop API standards for open banking. It also requires domestic banks to allow verified TSPs to access user data with the user's consent to provide customized services. Currently, each bank or TSP is required to renew users' approval for data sharing once every 90 days [8].

2.4   Related works

In this section, we will present an overview of the related literature on blockchain-based access control, identity management, data sharing and the Open Banking ecosystem. We don't just point out the differences between previous studies and our research, but we also look at the tradeoff between the access control models.

2.4.1   Identity integration

Mudliar *et al.* [9] proposed an integration solution of national identity using blockchain technology. Each entity (regime officials and citizens) has a unique blockchain address for identification and integrates it with a unique identity number issued by the government. As the authors write: "With the possibilities of the blockchain technology, all such identities can be consolidated and only one identification can be used for the endless applications".

2.4.2   Access control

In traditional access control management, the most common solution is PKIs, but it has some concerns about scalability and granularity. Paillisse *et al.* [10] presented a blockchain-based approach to address these problems. They took advantage of blockchain to record and distribute access control policies. Firstly, they selected Locator/ID Separation Protocol (LISP) to perform access control and modified this protocol for communication with the blockchain. Secondly, an open-source blockchain, the Hyperledger Fabric, was used to refresh access policies in the control plane. Thirdly, they constructed a prototype system and used Group-Based Policy (GBP) for network administrators to specify network configuration.

Daraghmi *et al.* [11, 12] described a blockchain-based system for electronic medical records and academic records, using blockchain smart contracts to provide secure and efficient data access while protecting the patients' privacy. These systems enabled the data owner to securely control accesses their records.

Fu *et al.* [13] proposed a user rights management system that aims to protect user privacy through enforcing executable sharing agreement. They also adopted multi-layer blockchain

architecture to satisfy CAP (consistency, availability, and partition tolerance) theorem.

Rouhani *et al.* [14] proposed a distributed Attributed-Base Access Control (ABAC) system that can provide auditing of access attempts. This work has focused on addressing audibility and scalability. They also applied the blockchain-based solution to the digital library for validation.

Guo *et al.* [15] introduced the smart contract technology for multi-authority ABAC. In their proposed architecture, the interactions among data users, data owners, and authorities are built on the blockchain smart contract.

In many cases, users must be both authenticated and authorized prior to entering the system. Traditional access control models have resolved the issues. For example, Attributed-Base Access Control (ABAC) utilizes a set of attributes (e.g., user attributes, environmental attributes, and resource attributes) to set permissions; Role-Based Access Control (RBAC) granted access permissions depending on users' role in the organization. Generally, ABAC is a flexible and fine-grained mechanism, but it is more complex and requires well-designed management.

In view of all the access control mechanisms mentioned so far, the focus of our research is users' consent and secure authentication and authorization. In the current research, we investigate blockchain technology, access control mechanisms. The challenge arises when users have multiple digital identities and need to monitor the use of their personal information by another organization.

2.4.3 Open banking

Open banking brings not only opportunities but also challenges all over the world, such as malicious attacks, identity theft, data leakage, and privacy disclosure, etc. Fortunately, blockchain, with the properties of tampering resistance, transparency, and verifiability, perfectly matches the requirements of open banking services. We will review four types of blockchain research in the open banking area with different aspects; they are data privacy protection, access with customers consent, provenance data for tracking TSPs accessing behavior, and open API management.

First, Hao wang et al. [3] proposes a data privacy classification method and disclosure scheme for achieving customer privacy protection. Their proposed idea works well in dealing with deficiencies when implementing blockchain in open banking, such as the granularity

of privacy-preserving, various financial data, the complexity of banking subsystems, and hierarchical data management. Second, Mukhopadhyay et al. [16] systematically illustrate the importance of customer consent management. This study also proposes a TSP rating design for customer's reference before making consent. Third, Zhiyu Xu et al. [17] indicate that sensitive financial data requires higher authentication and provenance for participants. Thus they develop a provenance-provided data-sharing model for customers and banks to audit and track the TSPs' historical access behavior. Fourth, Q. Zhang et al. [18] discuss the security risks when financial institutes provide application programming interfaces (APIs) to TSPs. And they conceptualize a data-sharing scheme and an API consensus mechanism that aims to prevent the open APIs from being malicious forged.

Chapter 3

System Design

3.1 Overview

Our proposed blockchain-based access control framework mainly consists of two components: blockchain network, DApp. The blockchain network mainly helps organizations (banks) to build a tamper-proof, traceability, and decentralized ecosystem. It also allows customers to manage their data access. The DApp was developed by programmers of the organization allows customers to directly interact with the Ethereum node through RPC. It means that customers perform access control without organizations (banks), but creating digital identity is only conducted by organizations (banks). For the convenience of description, the notations of this research are illustrated in Table 3.1.

Table 3.1: Notations

| Notation	Description
C_i	Customer / User
Org_i	Organization
RA	Regulatory Authority
D_{ij}	Data of C_i in Org_j
TSP_i	Third-party Service Provider
Acc_{ij}	Org_j Account of C_i
ID_i	Identification card number of C_i
Add_i	Ethereum address of i
Pri_i	Private key of Add_i
DI_i	Digital Identity of C_i
R_{ijk}	Access right of TSP_i to access Org_j with C_k consent
$ACMgr_i$	Access Control Manager contract of C_i
$OMgr$	Organization contract
N	Nonce
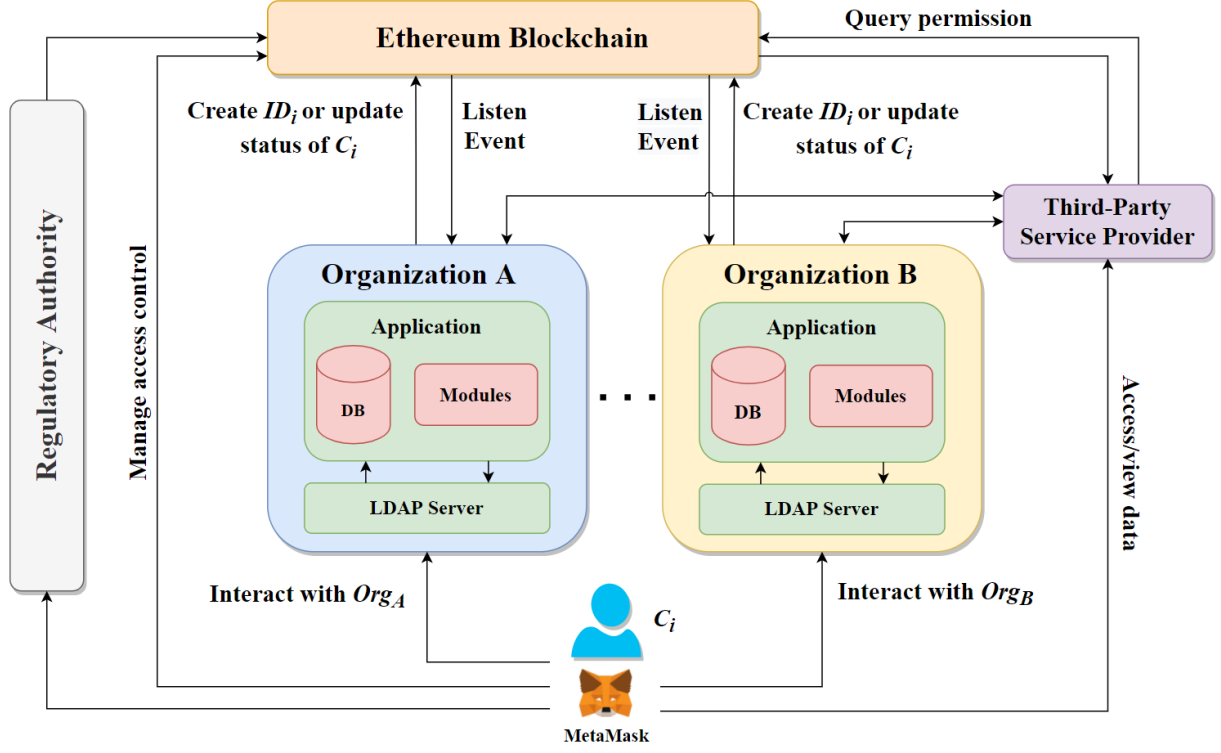
Figure 3.1: System Architecture

The architecture of our system is given in Figure 3.1. Our proposed system solved the account integration and identity verification. And it can apply to various scenarios which need data sharing and distributed access control management such as open banking, medical records, and academic records. At the initial stage, each Org or TSP gets the exclusive Add and the corresponding Pri to take part in our open banking ecosystem. They must put the safety of Pri first and be careful; otherwise, identity theft may occur. Our system employs the hash algorithm Keccak-256 to hash C_i's identity information, i.e., ID_i. This system records the hash value in the blockchain to represent C_i's identity instead of storing ID_i. There are several parties in our system: blockchain network, organization, third-party service provider, and user. The interpretation of these parties is stated as following:

- **Blockchain network:** The Ethereum blockchain network is used to store the user's DI and data access rights and deploy smart contracts, including identity creation and storage. Every node on blockchain has a copy of the ledger and ensures data on the blockchain cannot be tampered with. With decentralized technology, the management of data access rights doesn't have any governing authority to monitor. There have two kinds of contracts

to realize that account integration and access manager, detailed in Sect. 4.

- **Organization:** A set of organizations within an ecosystem, identified by the Add and IP addresses. Each organization represents an independent system, it owns membership software that provides an organization with functionality such as storing and editing member information. The organization not only provides services to customers depend on applications, but also collects user data by using a database. In this paper, organizations provide an interface that users decide to share data.

    - $DB_j = \{D_{1j}, D_{2j}, ..., D_{nj}\}$: Database of Org_j is used to store customer data and only owner of data can decide which TSP can access.
    - Because the organization Org_j maintains membership software, it will provide the user C_i a regular account Acc_{ij}. In the appilcation of Org_j, the user C_i can access their data D_{ij}.

- **Third-party service provider:** The third-party service provider can be any organization or entity that performs financial services to customer. In this paper, we assume that TSP doesn't manage membership software and they retrieve customer data through blockchain technology.

    - The aim of the TSP is for collect data of C_i from organizations $\{Org_1, Org_2, ..., Org_n\}$ if the organization owns the data of C_i and TSP gets the permission.
    - When the permission is revoked by triggering smart contract, the access data request is regarded as invalid even if the token exists and it is not out of date.

- **User:** A user owns an organization account and wants to participate in our proposed system. After registering an organization account, the user first needs to pass identity card authentication. Due to the digital identity is unique and important on the smart contract, the organization takes responsibility for ensuring the correct identification card number and the safety of user's personal data. Besides, the user should generate the Ethereum account themself through Metamask.
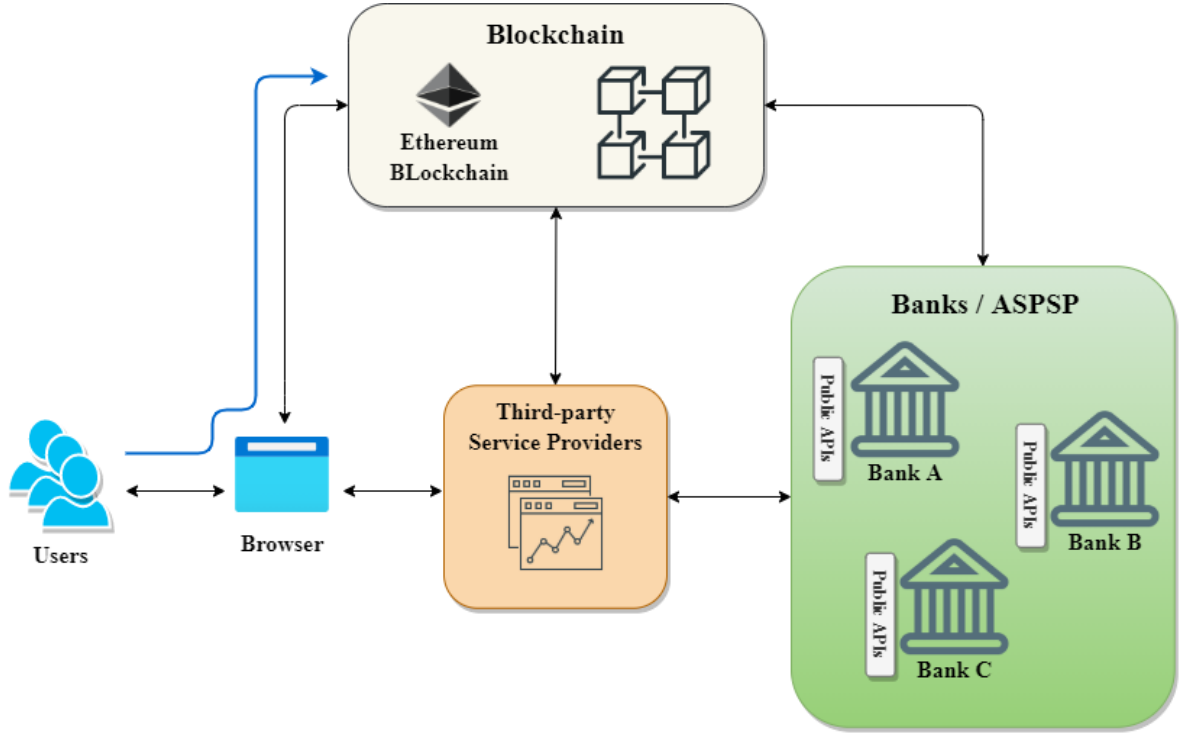
3.2   Scenario

Figure 3.2: Relation between open banking roles

From the user perspective, our proposed system provides a single digital identity DI_i and access control. The access control for data sharing is constructed using smart contracts. So when banks disclose user's personal data to TSP, banks must have the user's consent through call specific user's smart contract.

In order to apply our proposed system to open banking ecosystem, we have three clearly defined roles: Customer (User), Financial institution, and Third-party services provider. Figure 3.2 gives an overview of our proposed system. It shows the relation between these roles and includes workflows, detailed in Sect. 3.3 Each customer interacts with Blockchain by using MetaMask, they not only login with MetaMask but also manage their own access manager contract. That's why customers can allow the specific party to access their data.

The user can specify the data attribute, source, destination through the smart contract to decentralized access control. And the user also can revoke access rights while he/she doesn't need the service that TSP provides.

15

3.3 Workflow

3.3.1 Identity verification

Figure 3.3: Account creation flow

User registration is commonly used by many companies or any online service which record user data. Although in recent years there has been an increase in social login, the sign-up for social media is still necessary. Figure 3.3 shows the flow for the user who visits the application (e.g., bank's website) and completes a signup form as shown in Table 3.2. The identification card number of the form is especially important for identity because it can be used to establish a digital identity on the blockchain. In this paper, we assume every organization should have its own membership software such as LDAP . And every user has multiple accounts, their personal information and data generated through organization service are belong to the organization themself.

Table 3.2: Sign-up form used in organization

Fields	Description	Required
username	Username as login account	Yes
password	The password of the account	Yes
email	User's email	No
phone	User's phone	No
id card number	Identification card number	No

Customers should remember their username and password pairs for different sites. The username of the form is the account Acc for log in to Org_j, many real-world applications already adopt this scheme to build their membership system. An individual customer C_i of the real world usually has many accounts $\{Acc_{i1}, Acc_{i2}, ..., Acc_{in}\}$, those accounts are independent of each other under the current ecosystem.

Figure 3.4: Identity verification flow

Figure 3.4 shows the flow that an individual customer applies for real-name authentication. Participating in existing blockchain-based open banking ecosystems requires real-name authentication in order to prevent identity theft and avoid duplicate data.

In the first step, every user who wants to enable blockchain-based functionality carries their own personal identification card. After verifying by the staff of the bank, the ID_i will be submitted to blockchain by invoking $OMgr$'s function, and then update the user's account status for pass authentication.

In this paper, we assume that organizations or banks are trustworthy, they won't submit fake information. But if they are the malicious attacker, they only submit the wrong ID_i and create an empty DI_i on blockchain without harming the user's right or revealing personal data.

3.3.2 Account binding & unbinding

Figure 3.5: Account binding flow

In the account binding process, the user must log in with user's Acc first and apply for identity verification in person. If users intend to enable third party service, they are required to get a private key and its corresponding Ethereum address. Users can generate their private keys by using MetaMask [5] and store recovery key (seed) in safe location. MetaMask guarantees user's information safety, it won't collect any private keys, addresses, balance and personal information. After retrieving the private key and Ethereum address, users can submit its Ethereum address to the Org_i server, then the Org_i check if the address is valid using the smart contract function. Finally, Org_i triggers the binding process and creates a new job to subscribe to the specific event with a transaction hash. After adding a block of transactions to blockchain, this job will add hashed ID_i to the status of Acc_i.

Through executing the above process flow, the user identity verification will be confirmed by the bank and the DI_i will be created by blockchain. After binding Acc with the Ethereum address, users can manage all data access right themselves. This process not only enables distributed access control but also enables third-party login.
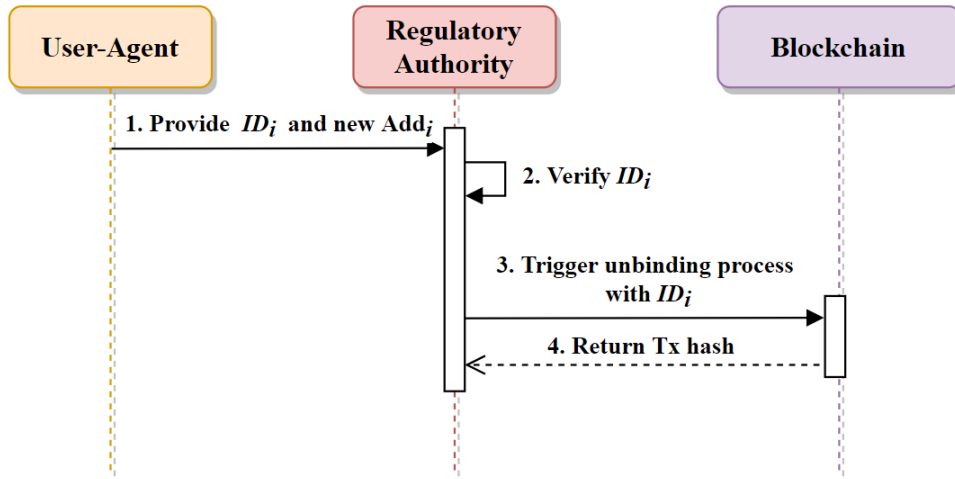
Figure 3.6: Account unbinding flow

Figure 3.6 presents that an individual user triggers unbinding account process. To disconnecting a users' DI from their Add, the unbinding request will be sent to the RA that is recognized as an impartial third party. If users lose Add's private key or intend to bind another Add, they should prepare a new Add in advance and bring their identity document to RA to change their binding status.

With this unbinding mechanism, it brings a convenient way to reset and prevent unauthorized transmission of data. Once this process executes successfully and the new mapping is strictly written into the block, the previous address will become invalid for login or any operations. The $ACMgr$ that the previous address managed stored in the blockchain permanently, and no one can remove this contract except it implements self-destruct function. Even though we can't remove this contract from the blockchain, the attacker is nothing to do. This contract only stores access rights and data attribute names.

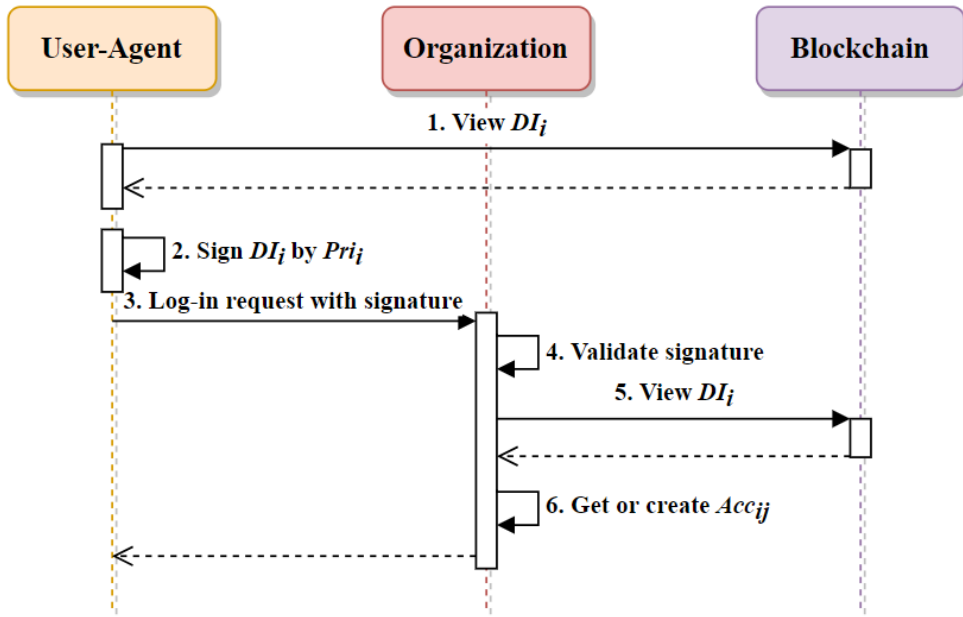### 3.3.3 Third-party login with Ethereum account

Figure 3.7: Third-party login flow

In the frontend user-agent, the user utilizes an Ethereum account to retrieve their DI_i from the blockchain without through a centralized server such as an organization, bank. Then the user signs this DI_i with MetaMask to generate a digital signature for proof of identity. After that, the server-side validate the authenticity and integrity of this digital signature. Additionally, the server-side also retrieve DI_i from the blockchain to confirm that the mapping of the Ethereum account and the specific DI_i.

The organization in our framework is regarded as service provider with and without the functionality of membership software. In our proposed ecosystem, every organization should support blockchain-based third-party login. For example, if users have enabled the functionality of blockchain, users can use their Ethereum account to log in to any service providers instead of Acc_i account and password.

The proposed framework supports two kinds of login mechanisms: (1) Regular account login: The user enters the account and password to log in system. The user behavior will not be affected if the user doesn't have Ethereum account. (2) Blockchain-based third-party login: The user can install MetaMask extension then choose third-party login with MetaMask, the server-side will retrieve the corresponding Acc or create a new one if the user never ever log in.

3.3.4 Data sharing

Figure 3.8: Date Sharing flow

The first step of this Figure 3.8 shows that C_i authorize Org_A the access right to access the personal information stored in Org_B. Any organizations are not involved in this authorization process, because web3.js library allows us to interact with the blockchain through the user-agent browser. The remaining steps are challenge-response authentication for obtaining an access token, one organization presents N and another organization must provide a valid signature, i.e., using private key to hash the specific N. Thus, the organization which obtains access token can access user's data without authenticating again.

Chapter 4

Implementation

This chapter describes the design of smart contracts and provides the detailed implementation of each function that is adopted in this paper. The smart contract diagram as shown in Figure 4.1, all of the organizations have common $OMgr$ for managing users' information and storing users' status. The ecosystem initiator enables blockchain-based functionality by deploying a $OMgr$. It offers application binary interface (ABI) files for all participants to easily call smart contract functions.

Figure 4.1: Smart Contract Diagram

4.1 Smart contract design

Organization Manager

The $OMgr$ is used to manage information that is related to users and organizations, i.e., it records data attributes, users' identity hash value. In Table 4.1, the *UserInfo* structure represents an independent DI. After the identity verification process is done as shown in Figure 3.4, the DI will be created for users automatically.

Because every organization has one key pair for determined their identity, we can retrieve their address from the private key and then append these addresses to the variable $orgs$ in $OMgr$.

Table 4.1: Table of *UserInfo* structure

Variable	Type	Description
lastModify	address	Address of last organization that verifies the user
orgs	mapping(address==>bool)	Organization's mapping
userAddress	address	Corresponding Ethereum address
accessManagerAddress	address	Address of $ACMgr_i$ contract

This smart contract provides functions to create user identity, bind account with Ethereum address, and create exclusive $ACMgr$ for the user as shown in Figure 4.2. And it also defines events in order that the smart contract can emit events to record logs on the block. That enables traceability of identity creation processes. The most important functions we used in $OMgr$ are *addUser*, *bindAccount* and only the legitimate organizations are able to trigger these functions.

The function *addUser* is invoked after finishing identity verification, the $OMgr$ check whether the ID exist on the blockchain. Then if the ID is new one, the $OMgr$ will create *UserInfo* for users. Otherwise, the $OMgr$ will append the address of Org who triggers this function to the variable *orgs* in *UserInfo*.

The function *bindAccount* is used to bind user's DI to Ethereum address. So that every contract $ACMgr$ will record organizations which the user has registered. One DI only corresponds to one Ethereum address, to establish a contract for storing access right.

Access Manager

After binding to address, contract $ACMgr$ will be created and required to transfer ownership of this contract, i.e., the user has full right to manage its contract and authorize access right to any legitimate organization. And users have a full right to make their choice as to whether or not to allow personal data opened. This contract uses two mappings to store access right in the form of key and value pairs where every key is unique. These keys represent attribute names that RA approves and submits. It is necessary to formulate data attributes for users and organizations

by RA because financial data attributes will update dynamically and only RA can update these attributes. For example, the Financial Supervisory Commission (FSC) was in charge of the development, supervision, regulation, and examination of financial services in Taiwan.
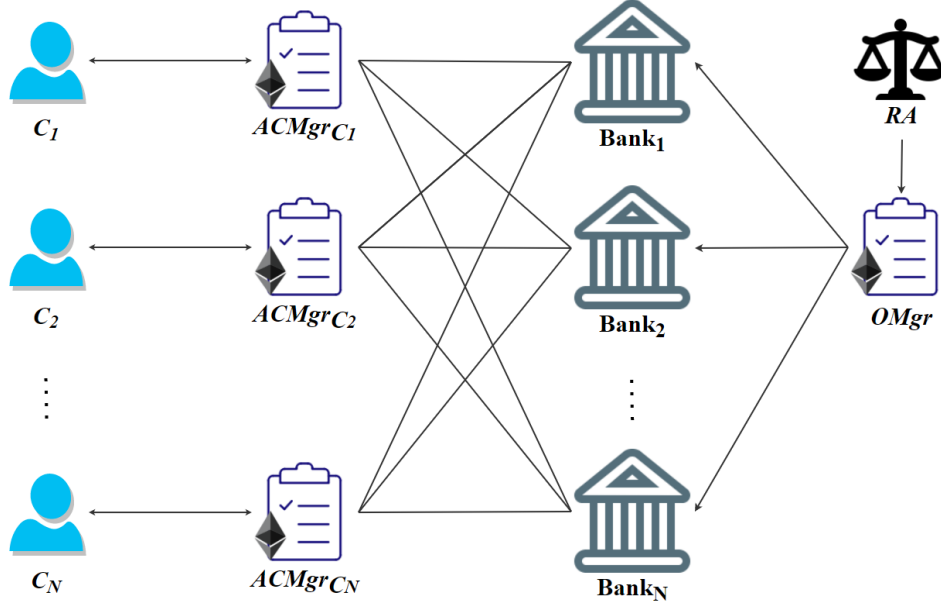

Figure 4.2: The relation between smart contracts

For the purpose of delivering a better user experience, we designed two access authorization modes for access personal data. If the C_N want to allow TSP to access the data from all Org ($Bank$), the C_N needs to authorize access many times due to one on one authorization, it is called Mode 1 as shown in Table 4.2. Another mode is one-to-many, the user only needs to select the attribute name that the user wants to share. Afterward, all the legitimate banks will accept any data request from TSP against the selected attribute.

Table 4.2: Table of access right structure

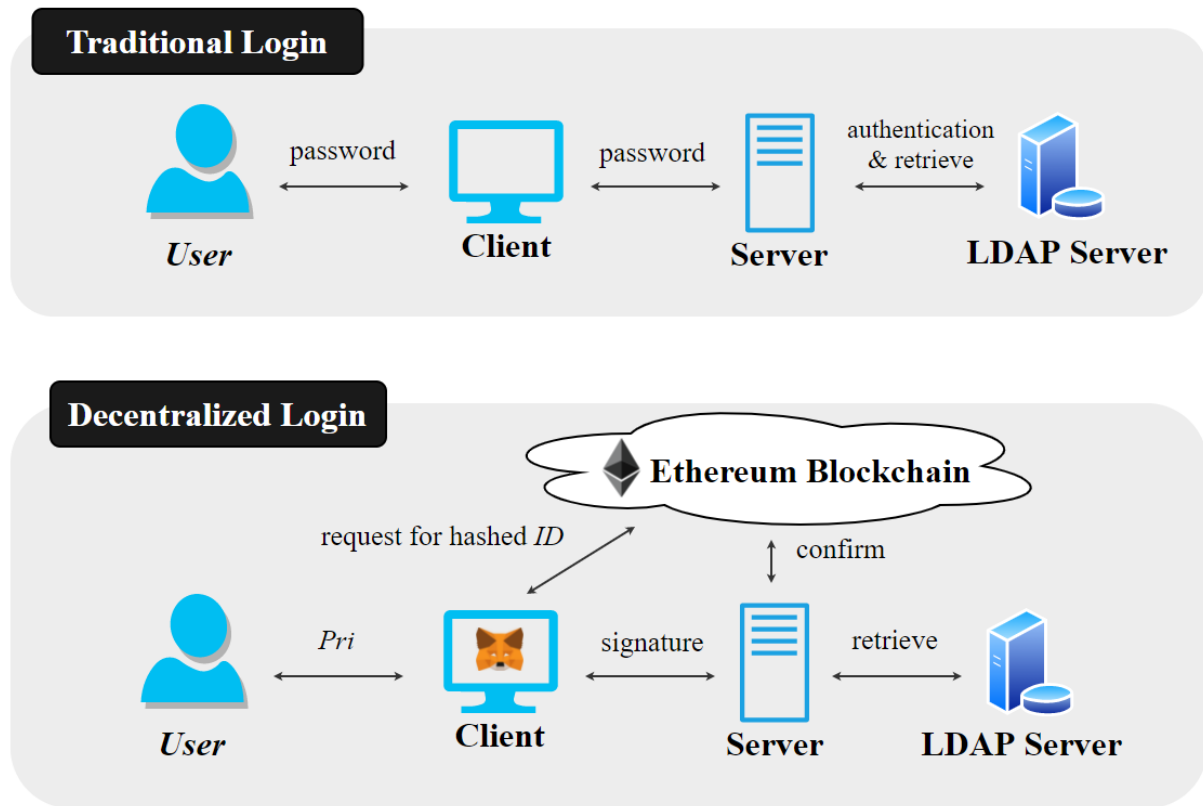| Variable	Type	Mode
accessAuthority	mapping(string=>mapping(address=>mapping(address=>bool)))	Mode 1
oneApproved	mapping(string=>bool)	Mode 2

4.2 Third Party Login

Figure 4.3: Traditional vs. Decentralized login

In contrast to traditional login as shown in Figure 4.3, the user can log in to other organizations with decentralized login and it doesn't need to remember any password, because the unique DI store on blockchain and the server can verify the user's identity by validating the signature that the user sign its hashed ID.

Our framework supports both of login mechanisms and utilizes hashed ID as the primary key to restoring user account information that the user stores on the LDAP server. Before binding account, the hashed ID doesn't establish yet and leave the hashed field empty.

To begin this decentralized login process, the client should install the MetaMask extension on Chrome browser and import existed private key or register a new one. The first step in this process was to use *call* function to view the user's hashed ID from blockchain. The second step was to sign the hashed ID by the user's private key using ECDSA. The final step was confirm with blockchain by server and then retrieve user account information.

4.3 Account Integration

In existing systems, users have an obligation to remember their own username/password pairs as users register a new one. Furthermore, because these pairs are following different rules, such as at least upper and lower case letters, character, symbol. It would be difficult for users to manage these pairs that are distributed and messy. There are several tangible solutions for storing these pairs nowadays. For instance, 1password and LastPass are password managers that save your username and password to cloud storage. But they cost money and users are required to trust a third party that controls their personal information. Besides, some services and platforms have already had their extreme vulnerability. Instead, we propose an account integration solution, which makes use of blockchain technology to generate tamper-proof record and realize access control.

In order to effectively perform account integration without a single authority entity, we took advantage of a secure hash algorithm and blockchain smart contracts that execute when the specific conditions are met. In the proposed architecture, the user has a secure and convenient way to build their own account integration via smart contracts on the blockchain, as depicted below.

(1) A C_i registers organization accounts respectively, these accounts are relatively independent of each other.

(2) One of the organizations creates DI using a hash algorithm Keccak-256 on blockchain as C_i passes real-name authentication, the others append their organization address to the same DI.

(3) After establishing an identity, the C_i will propose an account binding request when the C_i wants to participant in this ecosystem.

(4) One organization that the C_i registers with receives this request and then creates a transaction signed by its organization's private key to bind these accounts to the Ethereum address that the C_i specify.

(5) Afterward, the C_i can use the bound address to sign in or sign up for organizations within this ecosystem.
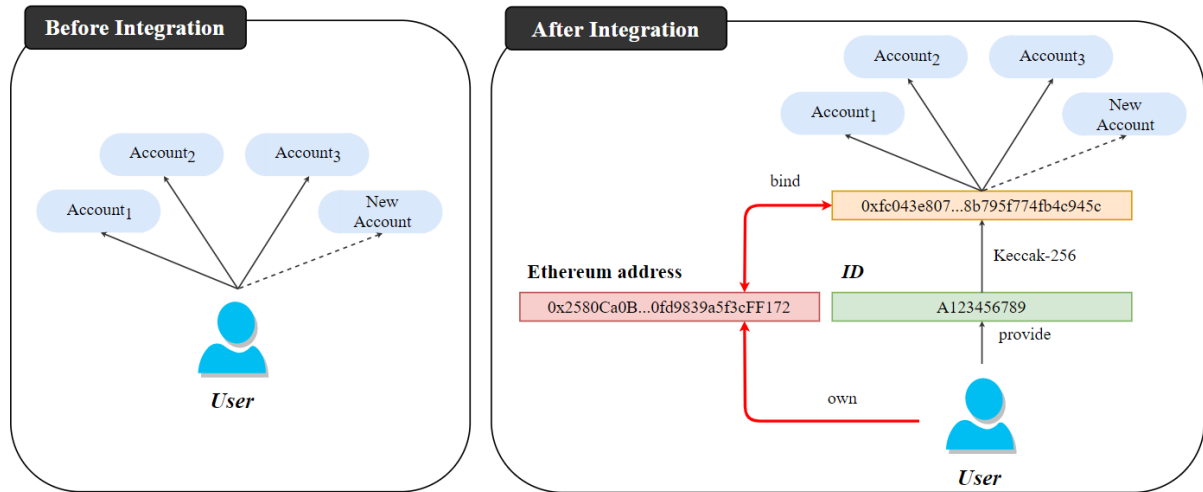
Figure 4.4: Account integration

As the Figure 4.4 shows a scenario, where a user uses the bound address to find an organization account for sign-in organization's service. This account integration has brought benefits to users such as ease of management, sign-up without additional registration, as well as no more information to be revealed to the public.

4.4 Data Sharing

Since our proposed system needs a secure transmitting mechanism for transmitting users' personal information such as financial data and better performance to reduce network latency that transmission need, the JSON Web Token (JWT) was adopted to verify the owner of JSON data. In our proposed system, the authentication module will allow TSPs to perform challenge response authentication in order to obtain a JWT which allows TSPs to access a specific resource from the organization without authenticating again. Once JWT has been obtained from the organization, the TSP can send a request using the JWT to fetch data for a period of time.

Conventionally, a solution was used to do authentication between client and server, called a session. It is used by the server to maintain the status of the client and the session is stored on the server. Although it may cause performance overhead in the case of large volumes of users by storing much session data on the server's memory, it still works for most applications. Our application has made use of sessions once the user has authenticated, i.e., the user login service.

Although session technology solves authentication issues well, we have different requirements for our data sharing schema. In our scenarios, one application is required to communicate with multiple authentication servers to get corresponding tokens as shown Figure 4.7. Specifically, in a single user's data request, the TSP's server should send access token requests many times and keep them in the database for later use (see sample in Figure 4.5).

identity	org	jwt	createdAt	updatedAt
0xfc043e8076...795f774fb4c945c	0xA3E898C280...EB4F3A6BFA67163	eyJhbGciOiJI...4DEKSaFE99JNXTU	2021-05-03 06:08:14.048 +00:00	2021-05-03 06:08:14.048 +00:00
0xfc043e8076...795f774fb4c946c	0x18DB34BAE0...9FF380D29BFFED7	eyJhbGciOiJI...8q_Pxx0f3ket0EM	2021-05-03 06:08:14.048 +00:00	2021-05-03 06:08:14.048 +00:00
0xfc043e8076...795f774fb4c947c	0xF799B44624...8C03051A2BE7359	eyJhbGciOiJI...1Ncyhx0Vo7UesBU	2021-05-03 06:08:14.048 +00:00	2021-05-03 06:08:14.048 +00:00
0xfc043e8076...795f774fb4c948c	0x8423B74781...A6AD4B77DFB7015	eyJhbGciOiJI...G6qEYx1wrxajffI	2021-05-03 06:08:14.048 +00:00	2021-05-03 06:08:14.048 +00:00

Figure 4.5: Screenshot of token database in TSP

Therefore, instead of using the session for authentication of data sharing, we take advantage of JWT to issue access tokens. The JWT is a self-contained token that has authentication information, expiration time, and other user-defined claims digitally signed. As illustrated in Table 4.3, each JWT contains these claims for specifying the entity which the TSP wants to query. The *hashed* value is used to identify the data owner thereby the TSP can access the user's data without more detailed information about the user.

Table 4.3: JWT claims used in our system

Name	Type	Semantics
hashed	byte32	The hash value of user's ID. For example, 0xfc043e80768cb3034a508ca5e0e256c5c72aad2642771f18b795f774fb4c945c
iat	timestamp	The time at which the JWT was issued.
exp	timestamp	The expiration time of the JWT
iss	address	The issuer of the JWT. For example, 0x1F7F0F7BE634D340EB070F3F3C21B6CE4AB857BD
sub	address	The subject of the JWT, i.e., the TSP that will use the JWT. For example, 0xA3E898C280220BF5FAE9E7E6CEB4F3A6BFA67163

After the user agrees to grant access permission for the TSP via *ACMgr* contract, the user will inform the TSP to send requests to all organizations via the states of the user's *ACMgr* contract and configuration file. The mapping of address of the organization and IP address as shown in Figure 4.6 to specify the corresponding IP address of the organization. Currently, these configuration files are stored on each organization's servers.

When the TSP is required to access the user's data, it just reuses this token to fetch data if the JWT doesn't expire. The issuer of the JWT will give a specific expiration instant. In most applications, it is recommended to set it smaller (a few minutes). It will need to balance tradeoffs with performance and security. Although JWT brings us a lot of benefits such as superior performance, portable, decentralized, the drawback of JWT is token revocation. This implies that a JWT that organization issued can be valid even if the user takes back the access permission. In order to overcome this drawback, we enable blockchain technology to verify the token. In addition to verifying tokens by the authentication server, our system also verifies tokens with *ACMgr* contract via a function call. The overhead of a function call of the smart contract will be compared in a later chapter.

```
"org_mapping": {
    "0x1F7F0F7BE634D340EB070F3F3C21B6CE4AB857BD": "172.19.0.1:3001",
    "0xA3E898C280220BF5FAE9E7E6CEB4F3A6BFA67163": "172.20.0.1:3002",
    "0x18DB34BAE039C54AA2B56BDF59FF380D29BFFED7": "172.21.0.1:3003",
    "0xF799B4462423B551CF404A3688C03051A2BE7359": "172.23.0.1:3004",
    "0x8423B7478160163C6E3319E64A6AD4B77DFB7015": "172.24.0.1:3005",
    "<Ethereum address of organization>"         : "<ip:port>"
}
```

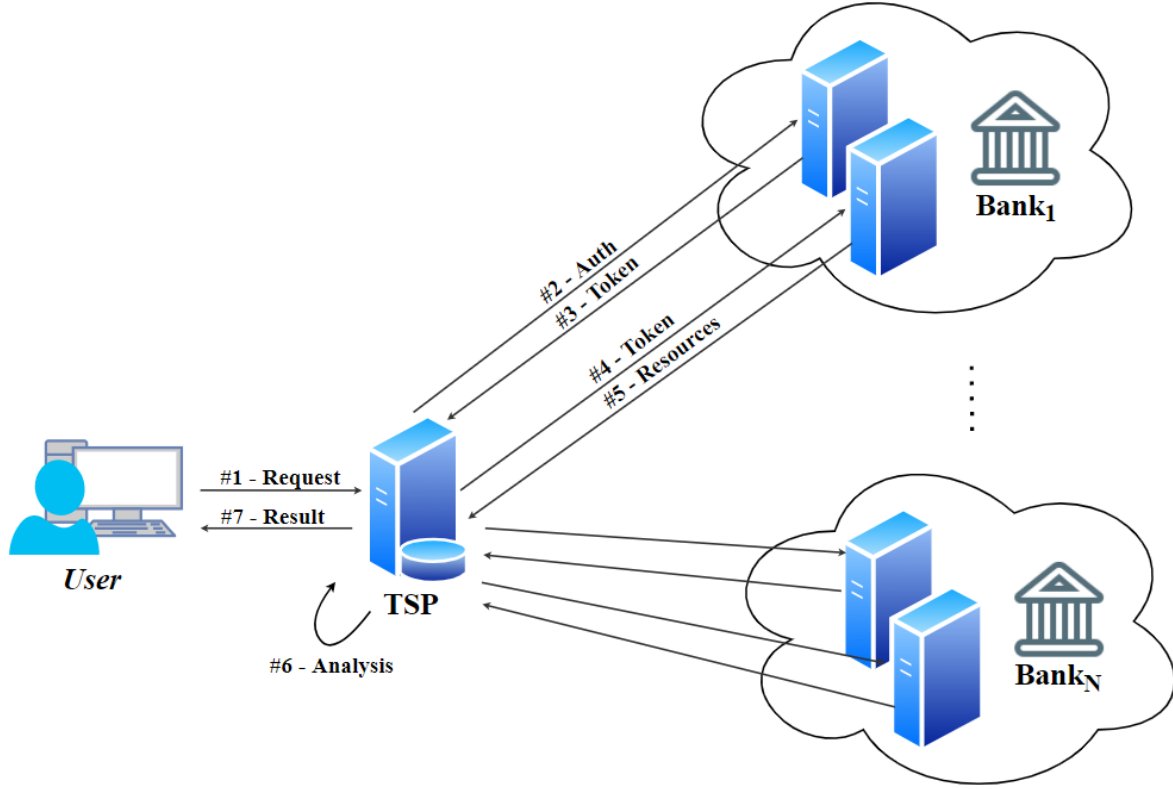Figure 4.6: Example of organization mappiing

Figure 4.7: A user send a request for data

The Figure 4.7 shows that a user sends a request to the TSP server, which will initiate subsequent requests to retrieve JWTs. Firstly, the server has to perform authentication until it collects all JWTs. If one of the requests is failed, just ignore the request failure and report it to the user. It won't affect other normal requests. Secondly, the JWTs can be utilized in two modes. In other words, we do not have to create two kinds of JWT for different modes. Because the JWT doesn't contain access right scope or other credential information. To perform secure data sharing, we have chosen the API to share data between parties. For more flexibility in this framework, we support three ways to let TSPs attach a JWT to the calling API: (1) as a query string parameter. (2) a form body parameter. (3) *x-access-token* header. Table 4.4 shows the APIs used in our framework, the query string parameter *ACMgr* stands for the specific user's contract address.

Table 4.4: APIs used in data sharing

Name	Method	Description	Mode
/users/protected?acc={*ACMgr*}	GET	Get the user's deposit value	Mode 1
/users/protectedInvoice?acc={*ACMgr*}	GET	Get the user's bill	Mode 2

Chapter 5

DApp design and implementation

5.1 User Registration & Sign in

Figure 5.1: Screenshot of sign in page

A user in our proposed system must register as a member of any $Orgs$ and provider his identification number as indicated in the Figure 5.2. Afterwards, if the user wants to activate the third-party login, he submits a binding request after the login. As shown in the Figure 5.3 below, the user clicks on the "Login with Ethereum account" button and the MetaMask signature request will appear to ask the user for confirmation. The purpose of this step is to ensure that the user has ownership of the Ethereum account.

Figure 5.2: Screenshot of sign up window

Figure 5.3: Screenshot of sign in with blockchain

5.2   User's Profile

Figure 5.4: Screenshot of user's profile

After the user log in to the Org's service, the user can view their own full profile. Since our implementation adopted the Lightweight Directory Access Protocol (LDAP) to authenticate users, it has stored and retrieved user information from a hierarchical directory structure. As shown in Figure 5.4, these attributes are parts of the X.500 directory specification, which defines nodes in an LDAP directory. E.g., *dn* (distinguished name), *cn* (common name). In this paper, we also define custom attributes such as *balance*, *hashed*.
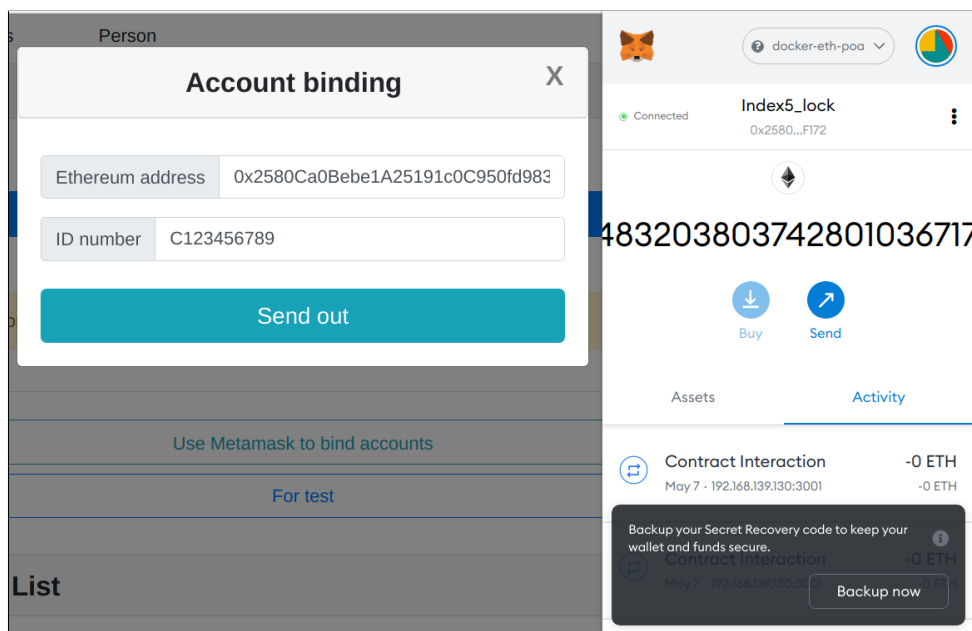
33

Figure 5.5: Screenshot of user's invoice list

Figure 5.6: Screenshot of binding process

Besides, we define the user's bill information, the primary key of this information is his name, each user can view their own bill information as shown in Figure 5.5. So far, we have known two kinds of user information (balance, bill). Even if the user is not involved in a blockchain ecosystem, the user can still see their information.
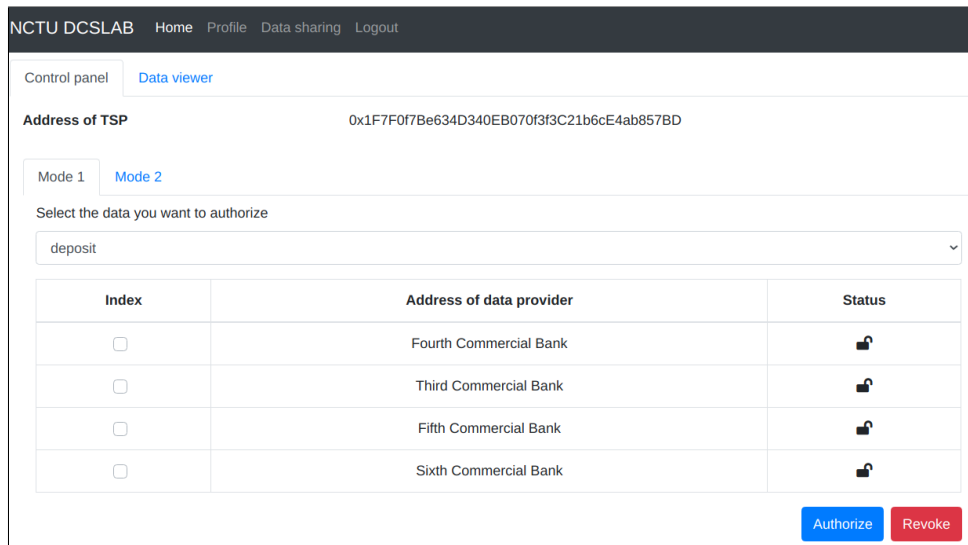
34

5.3   Data sharing

Figure 5.7: Screenshot of user's control panel with mode 1
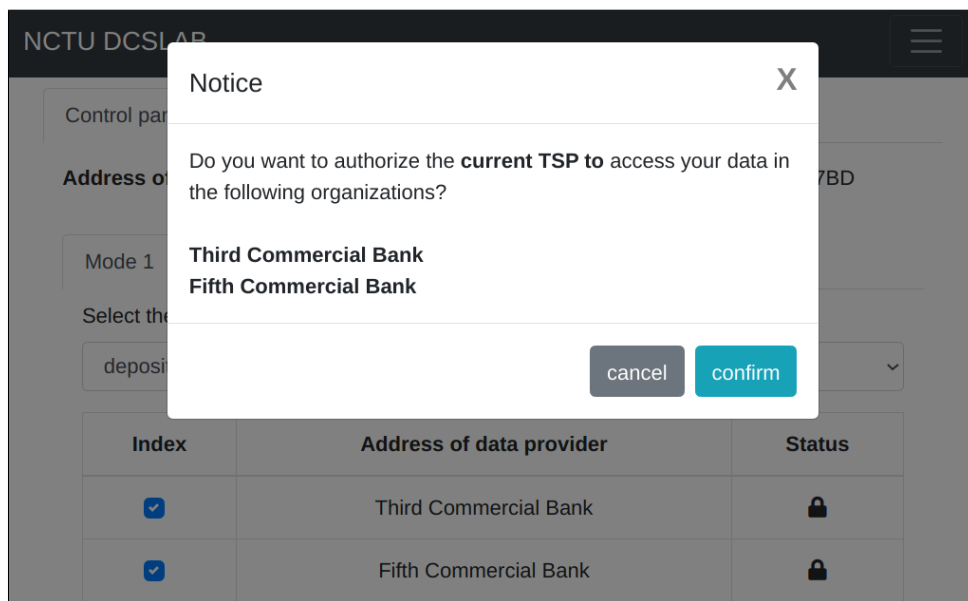
Figure 5.8: Screenshot of user's confirmation with mode 1

We provide a web-based user interface (see Figure 5.7, 5.9) for users to control their data access. A user can authorize or revoke access permission at any time through the control panel of this website (see Figure 5.8). Although this control panel is provided by the TSP, the TSP will not make access decisions on behalf of the user and the user can interact with web3 on the front-end side via MetaMask to connect to the specific Ethereum node.
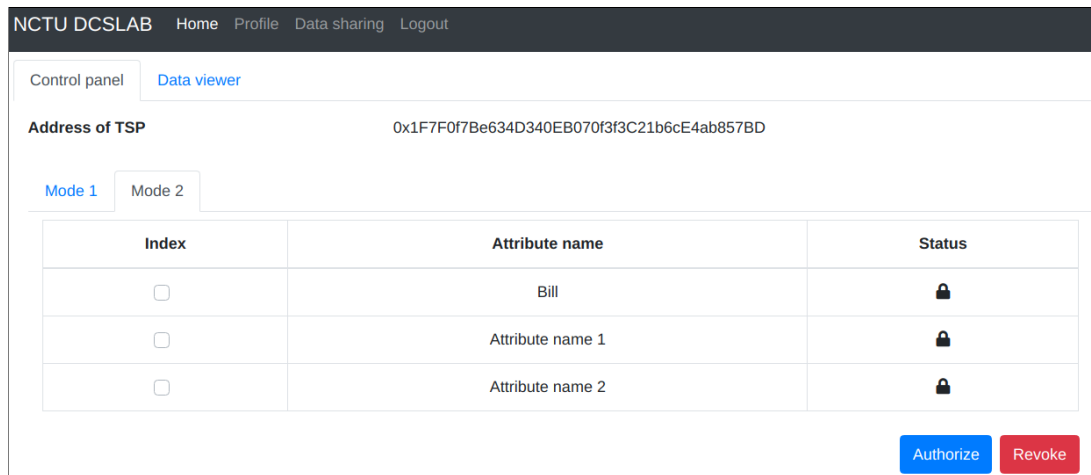
Figure 5.9: Screenshot of user's control panel with mode 2
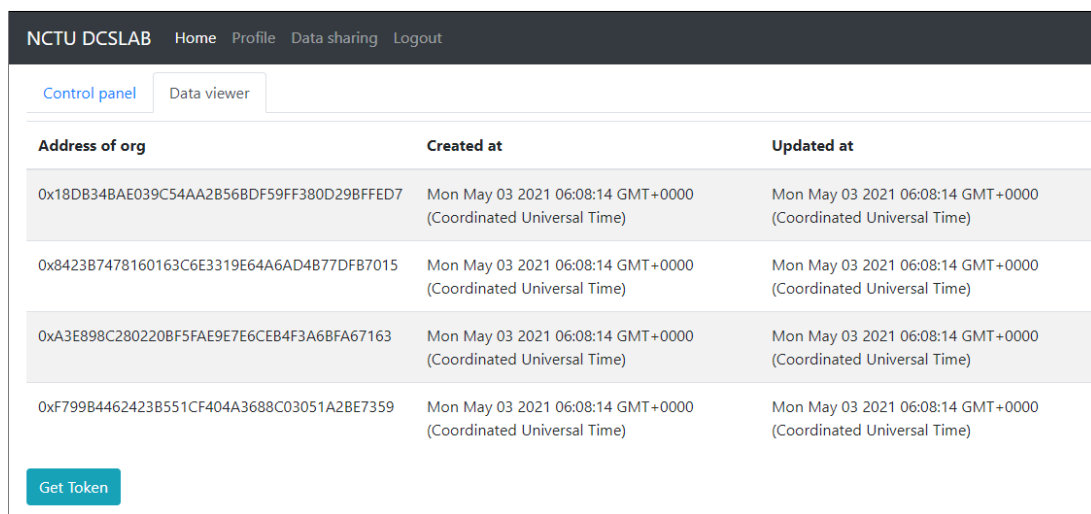
Figure 5.10: Screenshot of JWTs that already stored in TSP

Once the access decision had been made, the user clicks on the "Get token" button to inform the TSP to request for JWTs, and then the user can view the number of JWTs that the TSP got. So that the user can trace how many JWTs are issued by banks.

5.4 Org & TSP

In our implementation, an employee of the TSP and Org have to verify users' identity, so they need a web-based user interface to interact with web3 as well. The main difference lies in the architecture. As shown in the Figure 5.11, the control panel only for employees of TSP or Org.

Figure 5.11: Screenshot of organization's administrator

Figure 5.12: Screenshot of result that TSP generated

Chapter 6

Experimental Evaluation

6.1 Environment setup

Table 6.1: Experimental setup

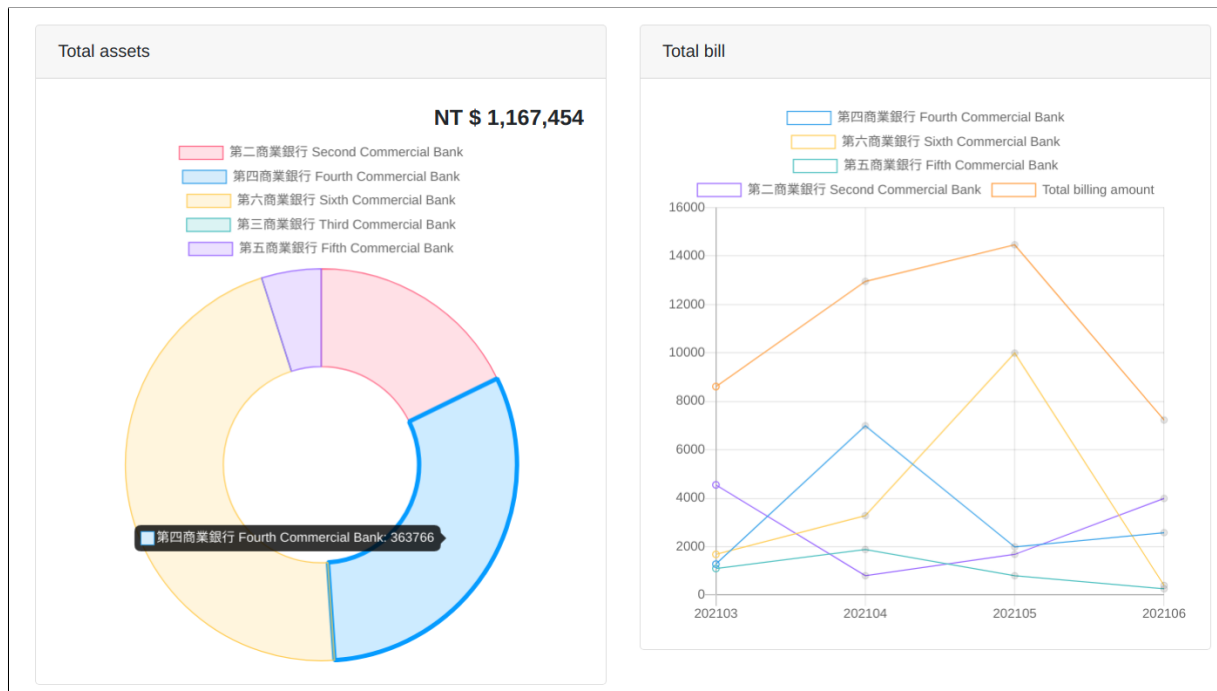|	
Number of organizations	5
Number of nodes	2
Number of sealers	1
State database	LevelDB, key-value storage
Consensus mechanism	Proof of authority (PoA)
Block time	1s
Difficulty	0x1

We set up an experimental environment on a PC running Ubuntu 18.04 and Intel i7-9700 CPU @ 3.00GHz. We deploy the blockchain network in Docker host with Geth tools. Table 6.1 summarizes Ethereum blockchain parameters in our experiment. The TSP's server and Org's server run within a Docker container for local testing. For example, a bank system we used for evaluation includes the following servers: LDAP server, web server. And each web server will interact with a remote Ethereum node through web3js. Although the TSP's system maintains an LDAP server, it is only for record user's information and doesn't responsible for registration. It only allows blockchain-based login without providing username and password.

In our proposed ecosystem, we use consortium blockchain as our blockchain network. The consortium blockchain is a "semi-private" system and works across different organizations. This blockchain is only used internally for a certain organization or TSP. It needs to pre-allocate several nodes as sealers that are responsible for mining blocks. Other nodes can send transactions, call smart contract, but they don't have right to mine blocks. In tern of consensus algorithm, Proof of Work (PoW) consensus algorithm is the most reliable and secure, it can provide a trust-

less economic system. Although it secure and trustless, PoW has the worst performance and has lower tps. In our experiment, we adopt Proof of Authority (PoA) that relies on a limited number of sealers, and it is regarded as an effective and reasonable solution for our scenario.

6.2 Gas consumption

Table 6.2: Gas consumption

Function/Contract	Gas used
addUser (create)	97574
addUser (append)	36435
bindAccount	1390534
authorize	49263
revoke	19284
authorizeAll	46580
revokeAll	16625
New $OMgr$	4494629
New $ACMgr$	1660077

The Ethereum gas refers to the computational expense on the Ethereum network. The amount of gas consumption is determined by the difficulty of calculation. Table 6.2 shows the gas consumption of each smart contract function and contract deployment fee. There is a significant difference between the two type function calls. In the case of the deployment contract, it will spend more gas such *bindAccount*. In other words, if the function does not involve complex logic or deployment, it will consume less gas such as *addUser*, *authorize*, *revoke*.

6.3 Performance evaluation

To analyze and measure the performance of various system services, particularly our web applications. We used the Apache JMeter tools to perform stress testing on system functions and modules. Meanwhile, in order to be closer to the reality of the situation, we also simulated the functions of the general system for comparison. We have divided the evaluation results into

three parts: User Login System, Account Integration, and Data Sharing. The number of users for our throughput performance testing is 1,000, 2,000, 3,000, and 5,000. The send rate (transaction per second) varies from 10 to 500, depending on the intention of each function.

Table 6.3: Performance metrics

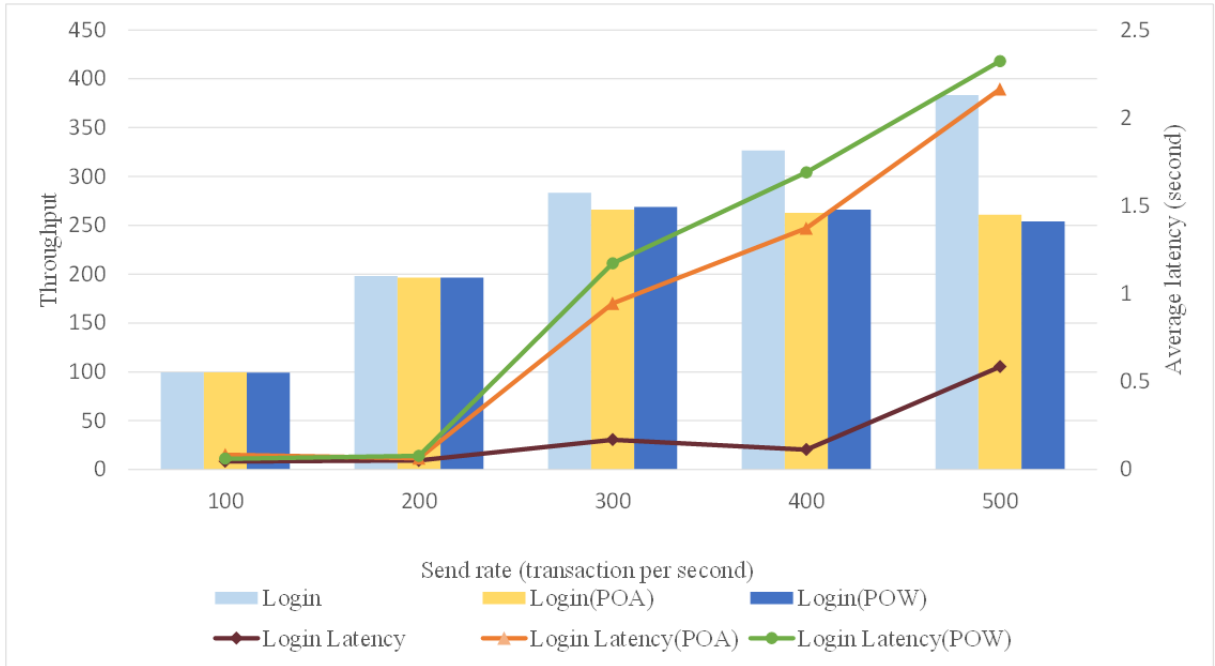| Metrics	Description
Latency	The difference between the time when request was sent and time when response has be received.
Throughput	The number of transactions processed per unit time

User Login System

Figure 6.1: Throughput and latency of various login systems under different send rate.

The results of the performance analysis are shown in Figure 6.1. The *Login* means that the user sent a login request with username and password to the server for authentication. We also tested blockchain-base login with two different consensus algorithms (*Login(POA)*, *Login(POW)*). As far as *Login* function is concerned, there is not much difference between these two consensus algorithms, because the login process is a read-operation. Although login with blockchain has lower performance than general login, it can still meet the login needs.
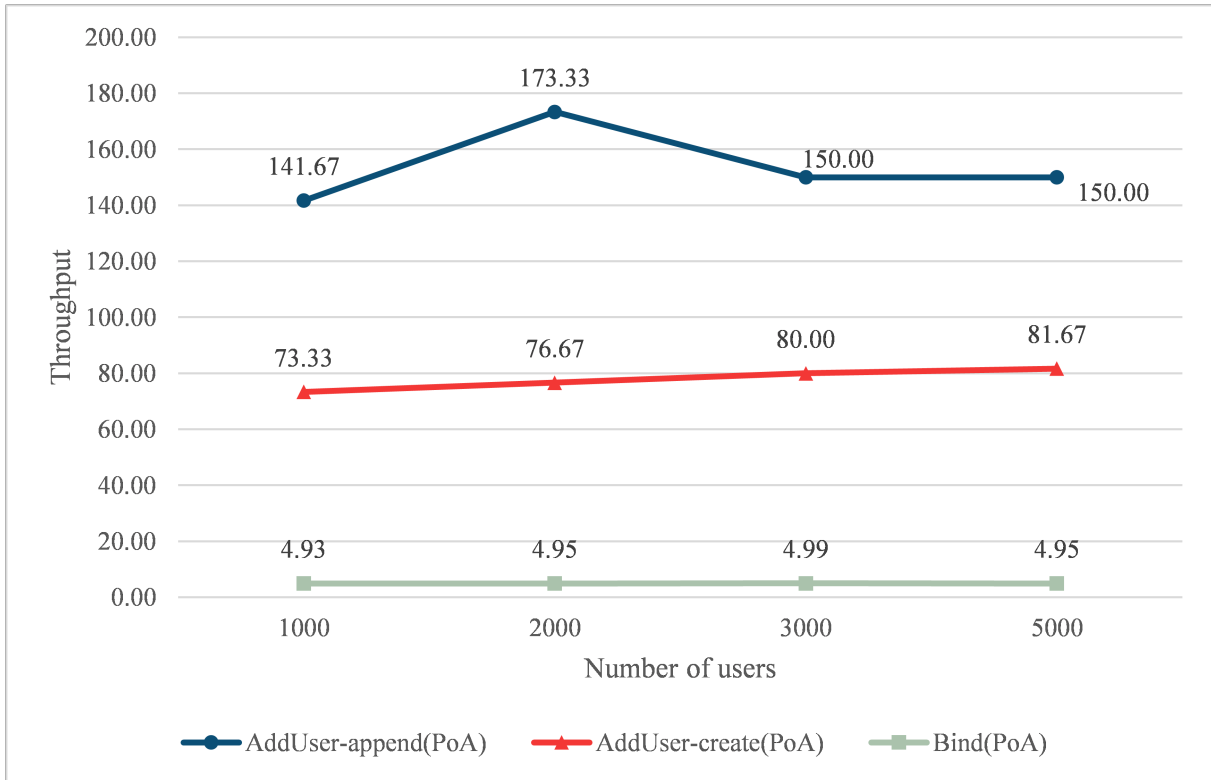
Account Integration

Figure 6.2: Throughput of smart contract functions under the different number of users.

As shown in Figure 6.2, the throughput for different functions of the smart contract. These functions above are write-operations that will update the state of the blockchain. A transaction is broadcasted to the entire blockchain network, processed transactions by miners, and then is published to the blockchain. The throughput will vary according to the complexity of the state change. As shown in the line chart above, the simpler the function, the higher the throughput, which also means that more transactions can be accommodated in the same block. However, the observed difference between *Bind* and others in this result was significant, *Bind* have only five transactions per second. In other words, the blockchain network can only process five bindings of user account per second. Because the binding process involves creating new contracts (*ACMgr*), it will cost more gas. Although the throughput of the binding process is low, each user account only needs to execute once so that the performance is acceptable.
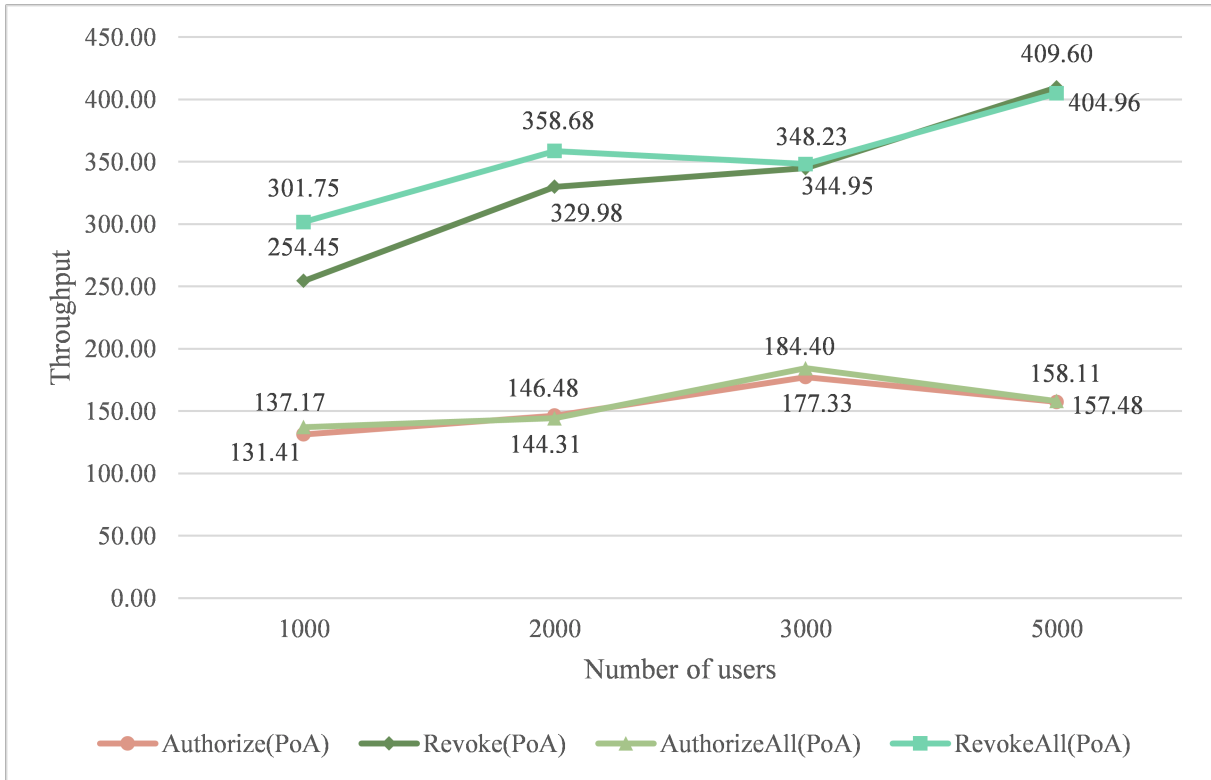
Data Sharing

Figure 6.3: Throughput of mode 1 and mode 2 under the different number of users.

To evaluate the performance of data sharing, we simulated user behavior of authorizing and revoking access rights. We created at least 5,000 Ethereum accounts and used these accounts to bind with the user identification separately. Then, each account sent a request to the blockchain to change the access permission. Because it will change the state of blockchain, these operations are write-operations.

Figure 6.3 shows the throughput of two kinds of different access authorization modes under the different number of users. Authorize/Revoke and AuthorizeAll/RevokeAll represent mode1 and mode2, respectively. With successive increases in the number of users, the throughput of authorization and revocation remained roughly at 150 and 350. Our blockchain-based ecosystem has the ability to process these requests that update the state of the smart contract in acceptable throughput.
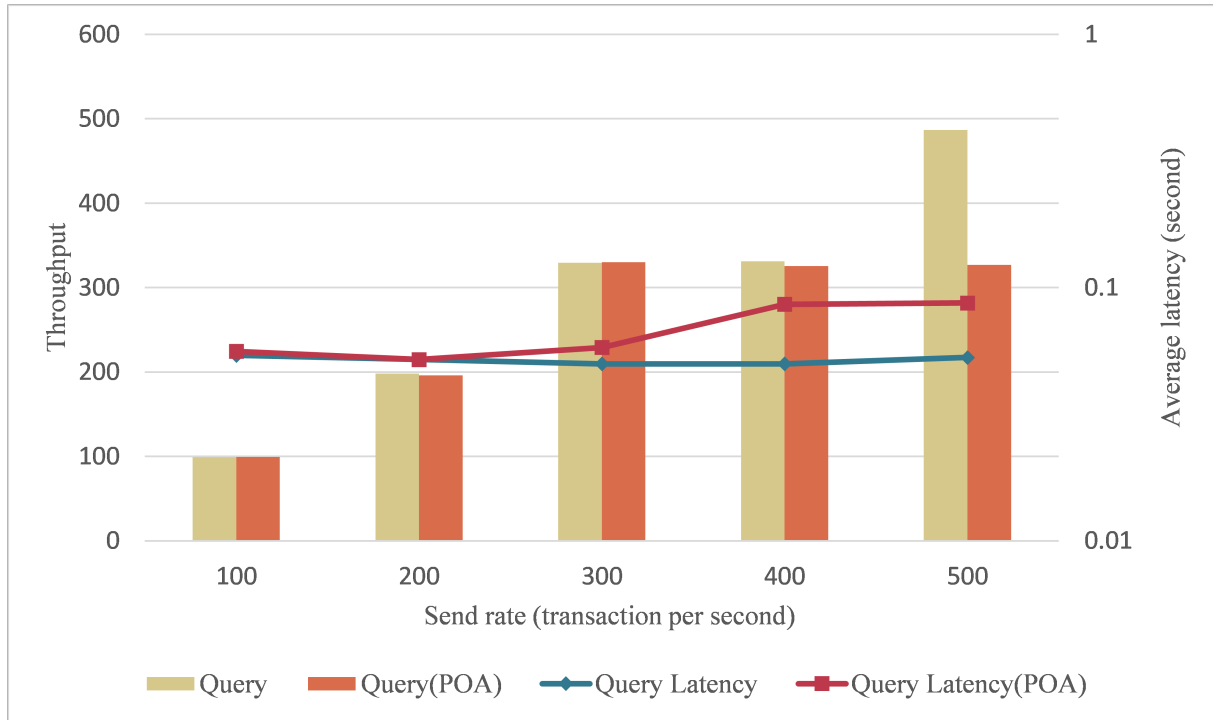
Figure 6.4: Throughput and latency of normal query and blockchain-based query under different send rate

To obtain customer information from the bank, we implement JWT authentication for TSP. If TSP sends a data query request to the bank, TSP puts its JWT into the header of the request. The results of throughput and latency of the data query are shown in Figure 6.4. It indicates the comparison between normal query and blockchain authenticated query. In a normal query, the bank's server only authenticates the JWT. In blockchain authenticated query (*Query(POA)*), the bank's server authenticates the JWT as well as access rights on the user's *ACMgr*.

In summary, we applied blockchain technology and a stateless authentication mechanism to our proposed system. Identity creation and authentication are constructed using the smart contract that can read and write the blockchain ledger. Since the blockchain enables every bank to have the same ledger using a consensus mechanism, the ability for TSPs or users to view the status of smart contracts becomes easier. With blockchain, it achieves transparency, tamper-resistant, and better Security.

6.4 Discussion

Firstly, the users and organization administrators can create Ethereum accounts as digital identities by using crypto wallets or tools. The private key of this account can be used in any blockchain network to initiate transactions or triggered contracts, but balances and smart contracts are independent in different chains. Secondly, each organization provides a DApp that improves the user experience. For example, users can access their financial data from different banks through a unique digital identity stored on the blockchain without remembering bank accounts and passwords. Thirdly, since our system focuses on user consent, any data access must obtain permission from the user. We implemented user consent by restricting that only the owner of this *ACMgr* could call functions. Therefore, banks and TSPs did not interfere with deciding of the user.

Our decentralized access control systems bring more opportunities and challenges: (1) **Innovative service**: This framework provides organizations with the ability to improve current services. They can extend existing products and develop new services through our data sharing model and identity integration. Therefore, more traditional banks and Fintech companies are willing to join this ecosystem to provide more convenient services. (2) **Reducing the risk**: Due to the characteristics of blockchain technology, we can trust the validity of digital identity. Hence, such a digital identity cannot be corrupted and promotes transparency and tamper-proof. Unique user identity and data sharing between banks facilitate KYC (Know Your Customer) procedures and help AML (Anti-Money Laundering). If one of the banks verified the user identity through real-name authentication, it writes data relating to the customer information into the block.

Chapter 7

Conclusion

Our research was motivated by identity authentication and data sharing, that open banking needed a secure manner to decentralized access right and provided user data API under protecting uses' privacy. In this research, we presented the design of an access control framework based on blockchain smart contracts. In order to perform self-sovereign identity, we presented identity integration that can be used for participating in this ecosystem and controlling data access rights. We described the system architecture, workflow of each functionality, and interactions among users, organizations, and TSPs. Ethereum smart contract can distribute data access rights to the whole network and provide tamper-proof identity and audit logs. The prototype of our proposed system was implemented to validate our design on a private network of multiple Geth nodes. Our proposed system solved the shortcomings of the conventional centralized system and enabled distributed access control functionality. In this framework, users can authorize/ revoke data access right directly without any organizations. The organization allows external data access requests from legitimate TSPs if TSPs obtain the consent of the user.

We apply public key cryptography (ECC) and digital signature algorithm (ECDSA) to DApp for user authentication and compare it with the general system (username and password). Although the general system has better performance, the blockchain-based login provides a more secure and convenient way, i.e., account integration, decentralization system.

Our experimental evaluation shows that our system can handle at least 5,000 user operations (identity creation, login requests, binding requests, and access right authorization/revocation) in a particular response time and compatible with general systems.

In this research, we focused on the design of user authentication and data sharing for construct a decentralized system. However, we did not consider user data synchronization in different organizations. If the user updates the user profile in an organization, other organizations will not update simultaneously. It is crucial to think of whether the user's data is stored in the blockchain.

References

[1] R. Gafni and D. Nissim, "To social login or not login? exploring factors affecting the decision," *Issues in Informing Science and Information Technology*, vol. 11, no. 1, pp. 57–72, 2014.

[2] "Taiwan regulator allows banks to enter second stage of open banking," *The Paypers*, Jan 2021. [Online]. Available: https://thepaypers.com/online-mobile-banking/taiwan-regulator-allows-banks-to-enter-second-stage-of-open-banking--1246579#

[3] H. Wang, S. Ma, H.-N. Dai, M. Imran, and T. Wang, "Blockchain-based data privacy management with nudge theory in open banking," *Future Generation Computer Systems*, vol. 110, pp. 812–823, 2020.

[4] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[5] "Metamask." [Online]. Available: https://metamask.io/

[6] "web3.js - ethereum javascript api." [Online]. Available: https://web3js.readthedocs.io/en/v1.3.0/

[7] F. Wessling and V. Gruhn, "Engineering software architectures of blockchain-oriented applications," in *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2018, pp. 45–46.

[8] M. O. I. humble travel enthusiast. Extreme food scholar. Writer. Communicator." , "The head of the open bank of england says that brazil seems to be on the right track finance," May 2021. [Online]. Available: https://www.mediarunsearch.co.uk/the-head-of-the-open-bank-of-england-says-that-brazil-seems-to-be-on-the-right-track-finance/

[9] K. Mudliar, H. Parekh, and P. Bhavathankar, "A comprehensive integration of national identity with blockchain technology," in *2018 International Conference on Communication information and Computing Technology (ICCICT)*. IEEE, 2018, pp. 1–6.

[10] J. Paillisse, J. Subira, A. Lopez, A. Rodriguez-Natal, V. Ermagan, F. Maino, and A. Cabellos, "Distributed access control with blockchain," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[11] E.-Y. Daraghmi, Y.-A. Daraghmi, and S.-M. Yuan, "Medchain: a design of blockchain-based system for medical records access and permissions management," *IEEE Access*, vol. 7, pp. 164 595–164 613, 2019.

[12] E.-Y. Daraghmi, Y.-A. Daraghmi, and S.-M. Yuan, "Unichain: A design of blockchain-based system for electronic academic records access and permissions management," *Applied Sciences*, vol. 9, no. 22, p. 4966, 2019.

[13] W.-K. Fu, Y.-S. Lin, G. Campagna, D.-Y. Tsai, C.-T. Liu, C.-H. Mei, E. Y. Chang, M. S. Lam, and S.-W. Liao, "Soteria: A provably compliant user right manager using a novel two-layer blockchain technology," *arXiv preprint arXiv:2003.10128*, 2020.

[14] S. Rouhani, R. Belchior, R. S. Cruz, and R. Deters, "Distributed attribute-based access control system using a permissioned blockchain," *arXiv preprint arXiv:2006.04384*, 2020.

[15] H. Guo, E. Meamari, and C.-C. Shen, "Multi-authority attribute-based access control with smart contract," in *Proceedings of the 2019 International Conference on Blockchain Technology*, 2019, pp. 6–11.

[16] I. Mukhopadhyay and A. Ghosh, "Blockchain-based framework for managing customer consent in open banking," in *The" Essence" of Network Security: An End-to-End Panorama*. Springer, 2021, pp. 77–90.

[17] Z. Xu, Q. Wang, Z. Wang, D. Liu, Y. Xiang, and S. Wen, "Ppm: a provenance-provided data sharing model for open banking via blockchain," in *Proceedings of the Australasian Computer Science Week Multiconference*, 2020, pp. 1–8.

[18] Q. Zhang, J. Zhu, and Q. Ding, "Obbc: A blockchain-based data sharing scheme for open banking," in *CCF China Blockchain Conference*. Springer, 2019, pp. 1–16.