

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 博士

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  
指導教授：袁賢銘

Student: Jen-Hao Cheng  
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 博士

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，同時兼具區塊鏈特性，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

摘要 . . . . .	ii
Abstract . . . . .	iii
Table of Contents . . . . .	iv
List of Figures . . . . .	vi
List of Tables . . . . .	vii
<b>1 Introduction . . . . .</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Objective . . . . .	2
<b>2 Background . . . . .</b>	<b>4</b>
2.1 Ethereum . . . . .	4
2.1.1 Smart Contract . . . . .	4
2.1.2 Merkle tree . . . . .	4
2.1.3 Elliptic Curve Cryptography (ECC) . . . . .	5
2.2 MetaMask . . . . .	6
2.3 OAuth . . . . .	8
2.4 Trust Service Provider (TSP) . . . . .	9
2.5 Related works . . . . .	10
2.5.1 Identity integration . . . . .	10
2.5.2 Access control . . . . .	10
2.5.3 Open banking . . . . .	10
<b>3 System Design . . . . .</b>	<b>11</b>
3.1 Overview . . . . .	11
3.2 Scenario . . . . .	14
3.3 Workflow . . . . .	15
3.3.1 Identity verification . . . . .	15
3.3.2 Account binding & unbinding . . . . .	17

3.3.3	Third-party login with Ethereum account . . . . .	19
3.3.4	Data sharing . . . . .	20
<b>4</b>	<b>Implementation . . . . .</b>	<b>21</b>
4.1	Smart contract design . . . . .	21
4.2	Third Party Login . . . . .	24
4.3	Account Integration . . . . .	25
4.4	Data Sharing . . . . .	27
4.5	Token design . . . . .	28
<b>5</b>	<b>Experimental Case Study . . . . .</b>	<b>29</b>
<b>6</b>	<b>Demonstration . . . . .</b>	<b>30</b>
<b>7</b>	<b>Experimental Evaluation . . . . .</b>	<b>31</b>
7.1	Gas consumption . . . . .	31
7.2	Throughput . . . . .	31
<b>8</b>	<b>Discussion . . . . .</b>	<b>33</b>
<b>9</b>	<b>Conclusion . . . . .</b>	<b>34</b>
	<b>References . . . . .</b>	<b>35</b>
	<b>Appendix A 附錄標題 . . . . .</b>	<b>35</b>

# List of Figures

2.1	Merkle tree in Bitcoin blockchain . . . . .	4
2.2	Merkle tree in Ethereum . . . . .	5
2.3	Architecture of DApp . . . . .	6
3.1	System Architecture . . . . .	12
3.2	Relation between open banking roles . . . . .	14
3.3	Account creation flow . . . . .	15
3.4	Identity verification flow . . . . .	16
3.5	Account binding flow . . . . .	17
3.6	Account unbinding flow . . . . .	18
3.7	Third-party login flow . . . . .	19
3.8	Date Sharing flow . . . . .	20
4.1	Smart Contract Diagram . . . . .	21
4.2	The relation between smart contracts . . . . .	23
4.3	Traditional vs. Decentralized login . . . . .	24
4.4	Account integration . . . . .	26
7.1	Number of users vs throughput . . . . .	32

# List of Tables

3.1	Notations . . . . .	11
3.2	Sign-up form used in organization . . . . .	15
4.1	Table of UserInfo structure . . . . .	22
4.2	Table of access right structure . . . . .	23
4.3	JWT claims used in our system . . . . .	28
7.1	Gas consumption . . . . .	31



# 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 [?]. 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 [?].

- **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 [?].

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. Chapter 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 [?] 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

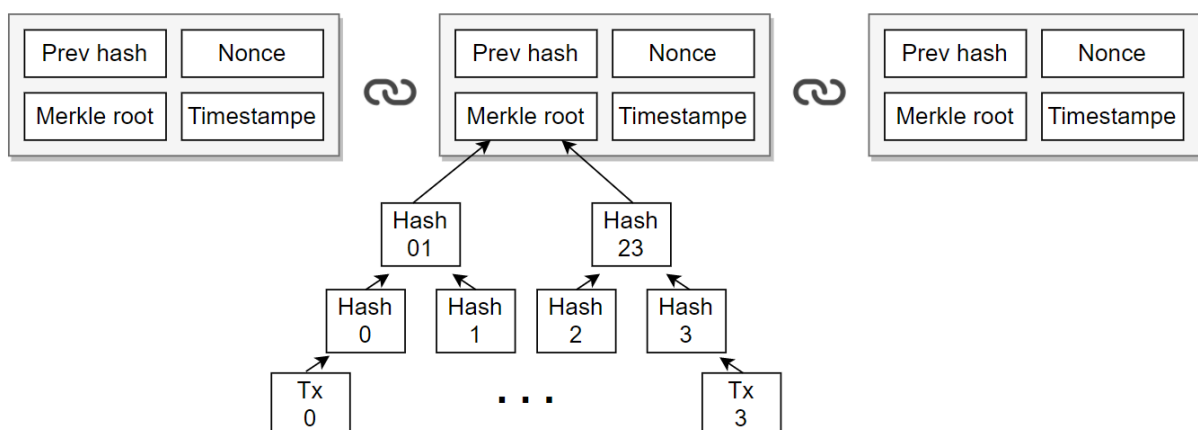


Figure 2.1: Merkle tree in Bitcoin blockchain

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.

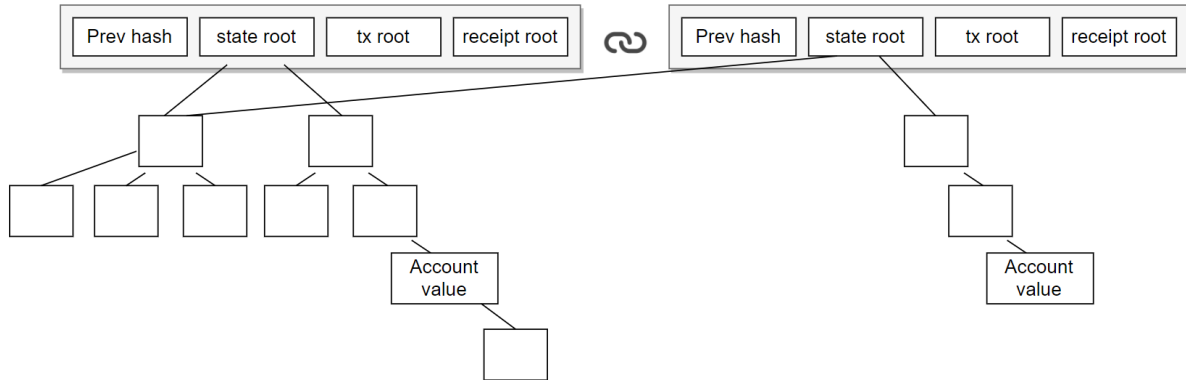


Figure 2.2: Merkle tree in Ethereum

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.

$$P = [m]G \quad (2.1)$$

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

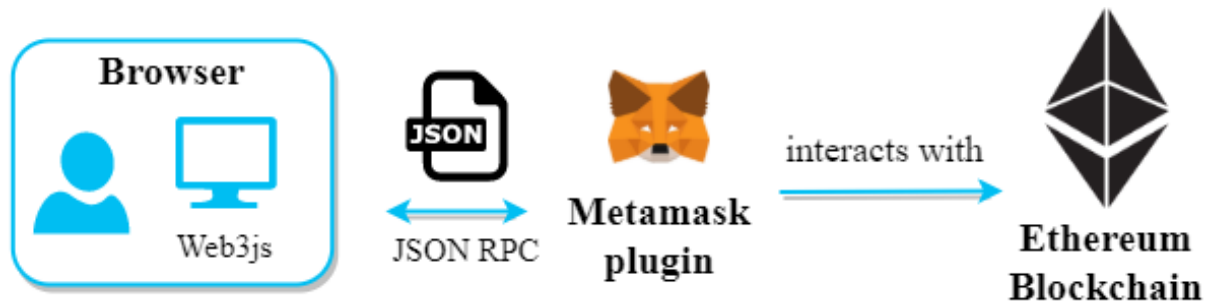


Figure 2.3: Architecture of DApp

MetaMask [?] 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 [?], e.g., Mainnet, Testnet. It also creates accounts by the user themselves. The user of MetaMask can create and manage their accounts; moreover, MetaMask 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 [?], 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 OAuth

- OAuth 2.0 flow and explain



## **2.4 Trust Service Provider (TSP)**

- Open Banking flow - Current TSP in Taiwan - Three stage table

## **2.5 Related works**

### **2.5.1 Identity integration**

### **2.5.2 Access control**

### **2.5.3 Open banking**

In this section, we will provide an overview of related literature about blockchain-based access control, identity management, data sharing, and Open Banking ecosystem.

In traditional access control management, the most common solution is PKIs, but it has some concerns about scalability and granularity. Paillisse *et al.* [?] presented a blockchain-based approach to address these problems. They take advantage of blockchain to record and distribute access control policies. Daraghmi *et al.* [?, ?] described a blockchain-based system for electronic medical records and academic records, using blockchain smart contract to manage the data access permission securely and effectively. It also utilizes advanced encryption techniques to protect user's privacy. Rouhani *et al.* [?] proposed a distributed Attributed-Base Access Control (ABAC) system that can provide auditing of access attempts. This work has focused on addressing audit and scalability, moreover, they apply the solution to the digital library and improve a lot. Fu *et al.* [?] 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.

More recent attention has focused on data sharing. The common scenario is Open Banking system since it needs secure identity authentication and the perfect mechanism of user data privacy. With blockchain technology, it can solve the shortcoming of the centralized system and prevent user's data from being breached.

# Chapter 3

## System Design

### 3.1 Overview

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 $i$
$Pr_i$	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

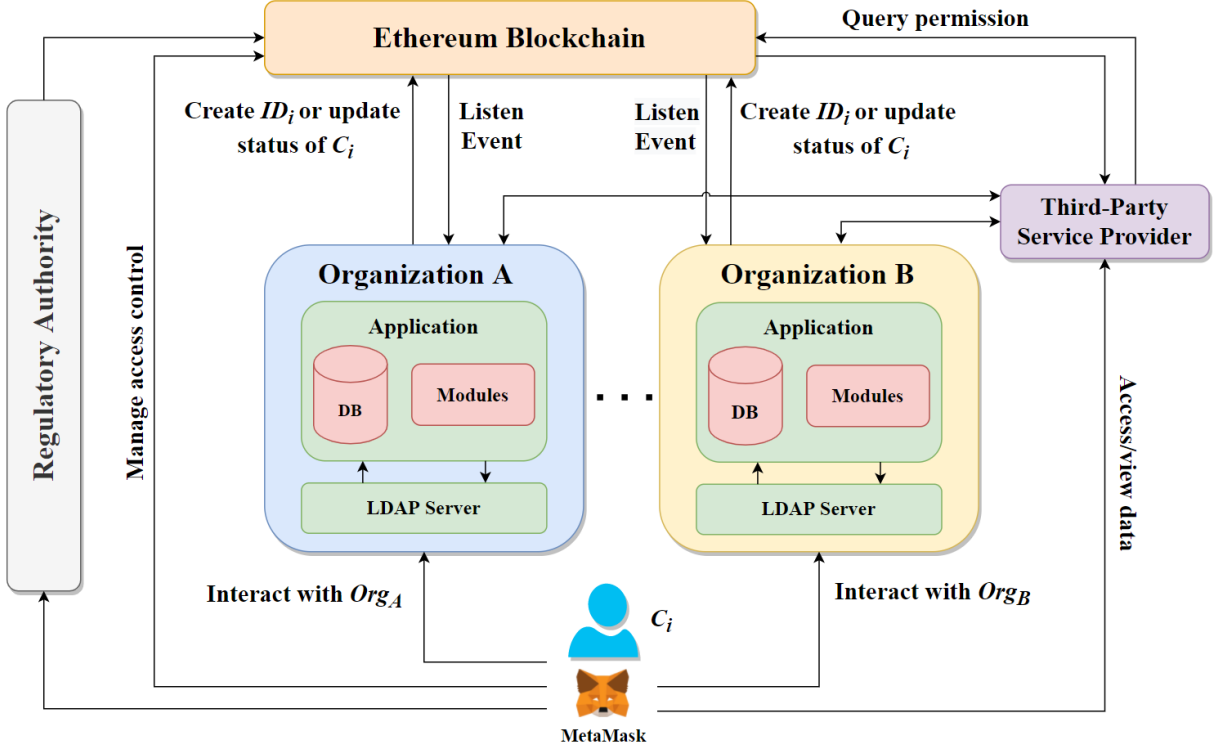


Figure 3.1: System Architecture

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.
  - $DB_j = \{D_{1j}, D_{2j}, \dots, D_{nj}\}$ : 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 application 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  $\{Org_1, Org_2, \dots, Org_n\}$  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.

## 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 themselves.

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  $\{Acc_{i1}, Acc_{i2}, \dots, Acc_{in}\}$ , 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 [?] 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<sub>i</sub>* server, then the *Org<sub>i</sub>* check if the address is valid using the smart contract function. Finally, *Org<sub>i</sub>* 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<sub>i</sub>* to the status of *Acc<sub>i</sub>*.

Through executing the above process flow, the user identity verification will be confirmed by the bank and the *DI<sub>i</sub>* will be created by blockchain. After binding *Acc* with the Ethererum address, users can manage all data access right themself. This process not only enables distributed access control but also enables third-party login.

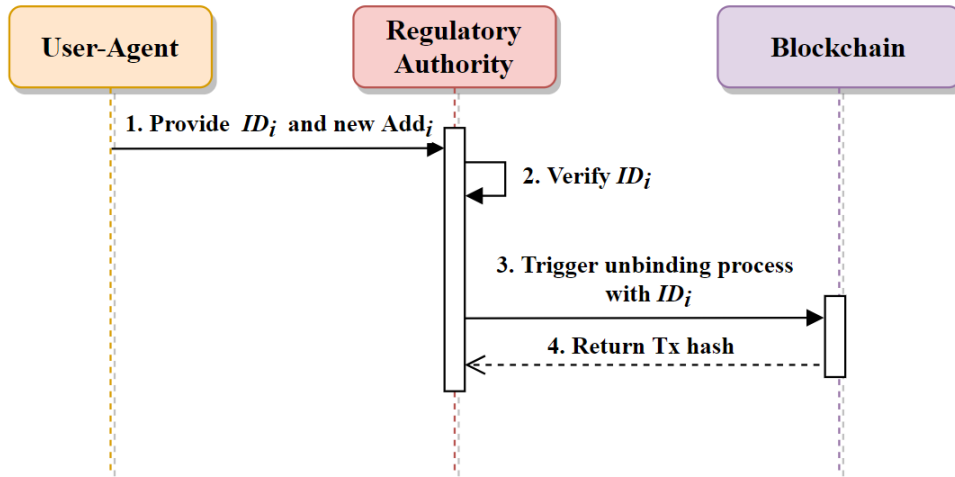


Figure 3.6: Account unbinding flow

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, it is always public.

### 3.3.3 Third-party login with Ethereum account

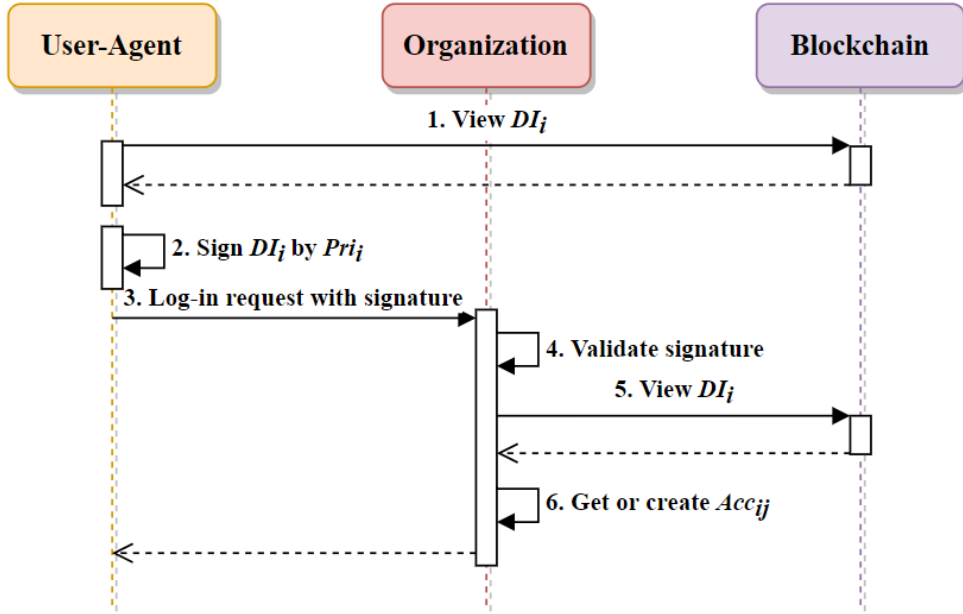


Figure 3.7: Third-party login flow

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

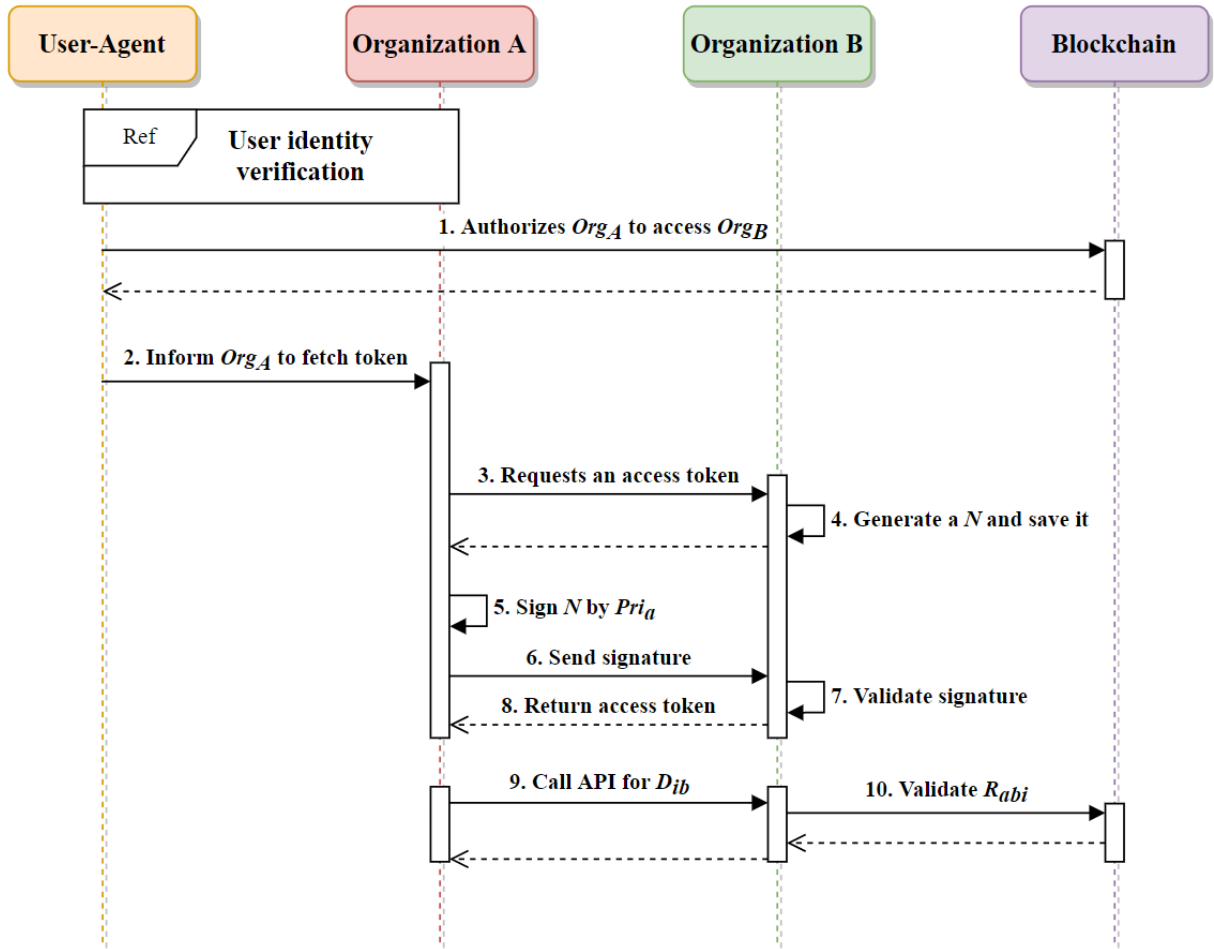


Figure 3.8: Data Sharing flow

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.

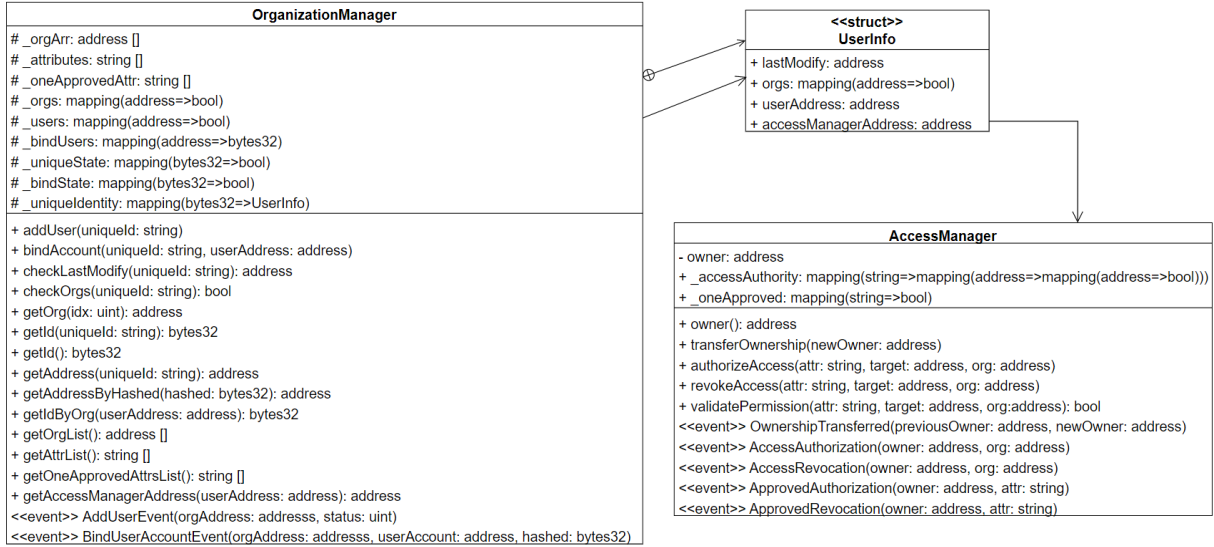


Figure 4.1: Smart Contract Diagram

### 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 *org* in *OMgr*.

Table 4.1: Table of *UserInfo* structure

Variable	Type	Description
<i>lastModify</i>	address	Address of last organization that verifies the user
<i>orgs</i>	mapping(address==>bool)	Organization's mapping
<i>userAddress</i>	address	Corresponding Ethereum address
<i>accessManagerAddress</i>	address	Address of <i>ACMgr<sub>i</sub></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 its owner belongs to users, 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.

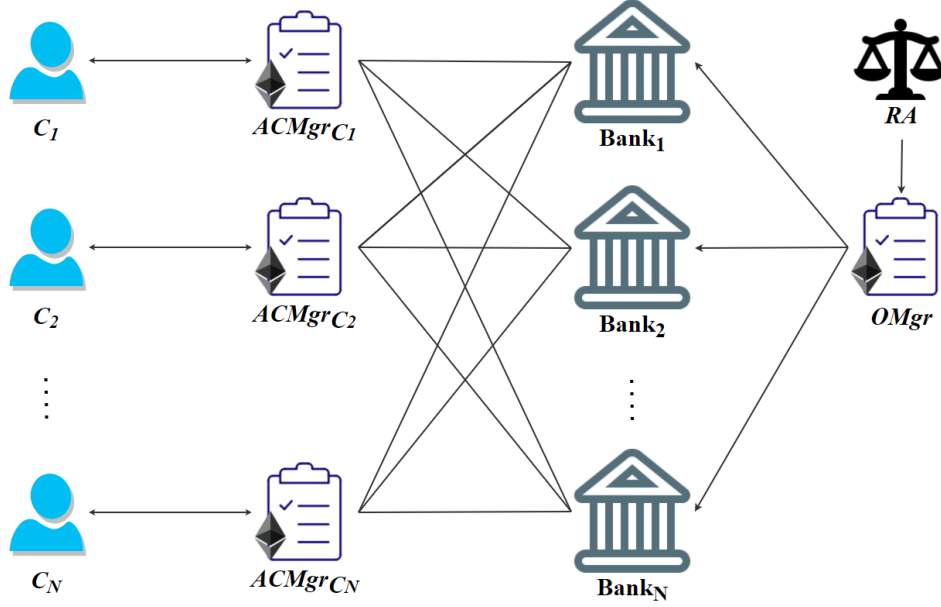


Figure 4.2: The relation between smart contracts

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
<i>accessAuthority</i>	mapping(string=>mapping(address=>mapping(address=>bool)))	Mode 1
<i>oneApproved</i>	mapping(string=>bool)	Mode 2

## 4.2 Third Party Login

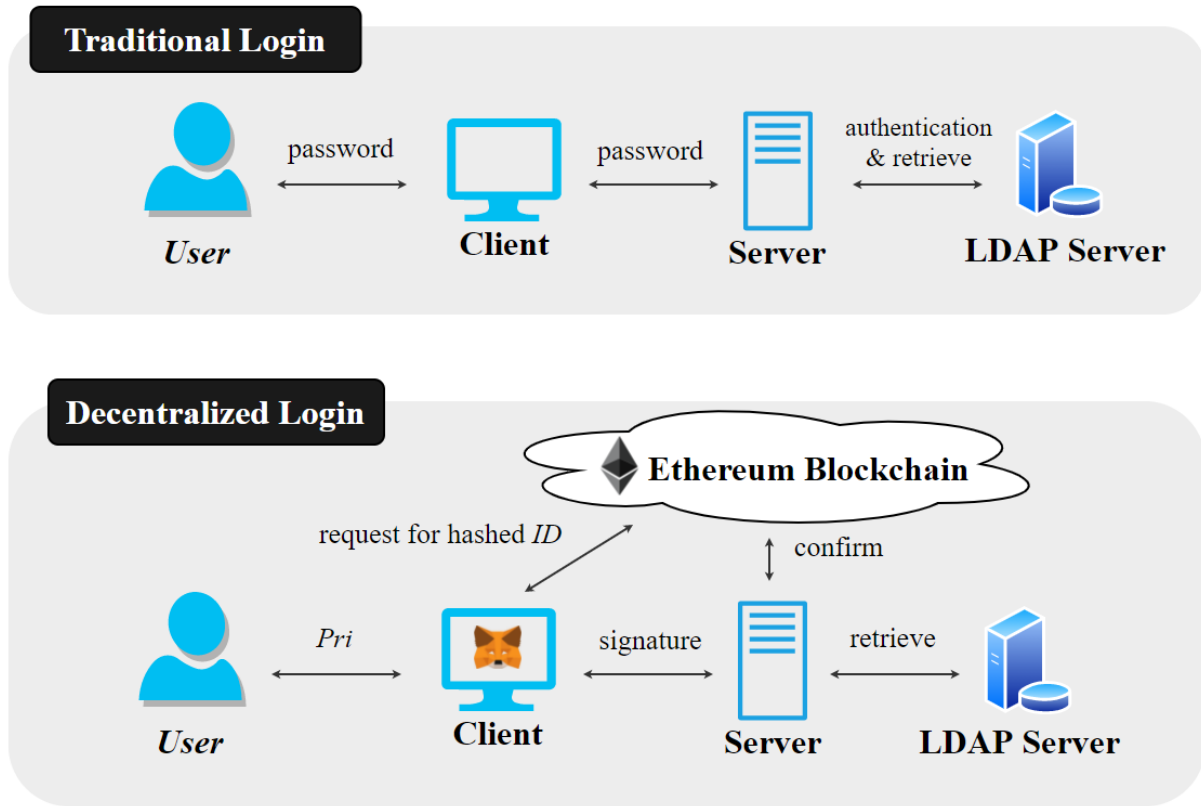


Figure 4.3: Traditional vs. Decentralized login

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.

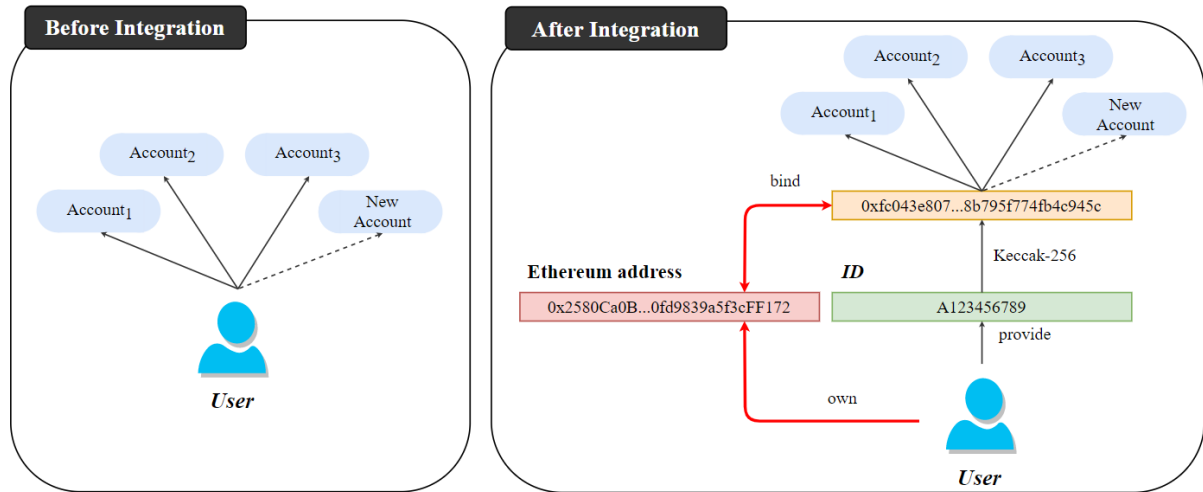


Figure 4.4: Account integration

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.

## 4.5 Token design

Table 4.3: JWT claims used in our system

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

# **Chapter 5**

## **Experimental Case Study**

Here are the experimental case study

# **Chapter 6**

## **Demonstration**

Here is the demonstration.

# Chapter 7

## Experimental Evaluation

Here is the evaluation.

### 7.1 Gas consumption

Table 7.1: Gas consumption

Function/Contract	Gas used
AddUser-create	97574
AddUser-append	36435
Bind	1390534
Authorize	49263
Revoke	19284
Authorize-all	46580
Revoke-all	16625
New <i>Omgr</i>	4494629
New <i>ACMgr</i>	1660077

### 7.2 Throughput

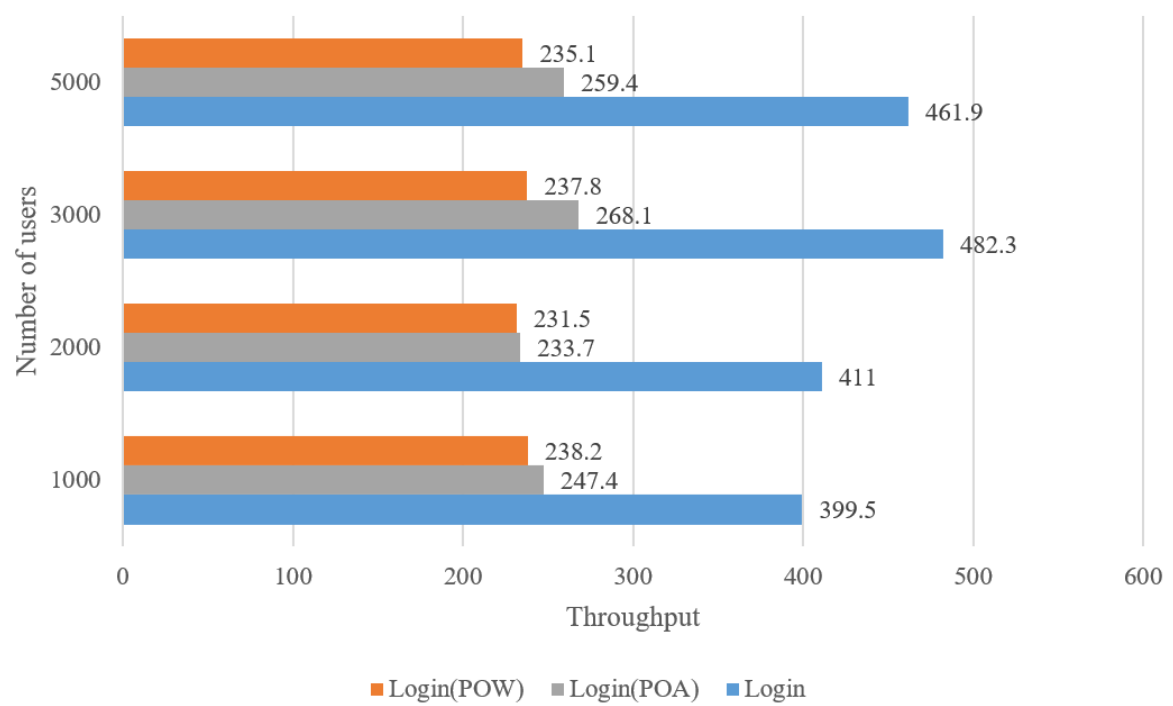


Figure 7.1: Number of users vs throughput



# **Chapter 8**

## **Discussion**

future work: User data synchronization. How/Why? (KYC)

# **Chapter 9**

## **Conclusion**

Here is the conclusion.

# Appendix A

## 附錄標題

### A.1 Testing