

Investigate the possibility of using smart contracts and digital signatures to create a legally binding contract, and to create a prototype opensource web application as a proof of concept

Mr. Yosnai Chanatrutipan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science

Department of Computer Engineering

FACULTY OF ENGINEERING

Chulalongkorn University

Academic Year 2022

Copyright of Chulalongkorn University

ศึกษาความเป็นไปได้ในการใช้สัญญาอัจฉริยะและลายเซ็นดิจิทัลเพื่อทำการเซ็นสัญญาผ่านช่องทาง
ดิจิทัลให้มีผลในโลกจริง และทำการพัฒนาเว็บแอปพลิเคชันต้นแบบเพื่อการเผยแพร่ต่อไปแบบโอเพน

ซอร์ส

นายยศนัย ชนัศรุติพันธุ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2565

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

Thesis Title	Investigate the possibility of using smart contracts and digital signatures to create a legally binding contract, and to create a prototype opensource web application as a proof of concept
By	Mr. Yosnai Chanatrutipan
Field of Study	Computer Science
Thesis Advisor	Associate Professor NUTTAPONG CHENTANEZ

Accepted by the FACULTY OF ENGINEERING, Chulalongkorn University in
Partial Fulfillment of the Requirement for the Master of Science

..... Dean of the FACULTY OF
ENGINEERING
()

THESIS COMMITTEE

..... Chairman
(Assistant Professor SUKREE SINTHUPINYO, Ph.D.)
..... Thesis Advisor
(Associate Professor NUTTAPONG CHENTANEZ)
..... External Examiner
(Assistant Professor Peerapat Chokesuwattanaskul)

ยศนัย ชนัศรุติพันธุ์ : ศึกษาความเป็นไปได้ในการใช้สัญญาอัจฉริยะและลายเซ็นดิจิทัลเพื่อทำการเซ็นสัญญาผ่านช่องทางดิจิทัลให้มีผลในโลกจริง และทำการพัฒนาเว็บแอปพลิเคชันต้นแบบเพื่อการเผยแพร่ต่อไปแบบโอเพนซอร์ส. (Investigate the possibility of using smart contracts and digital signatures to create a legally binding contract, and to create a prototype opensource web application as a proof of concept) อ.ที่ปรึกษาหลัก : รศ.ดร.ณัฐพงศ์ ชินธเนศ

เราได้ศึกษาความเป็นไปได้ในการนำเทคโนโลยีบล็อกเชนมาใช้ในการเซ็นสัญญาซื้อขายที่มีผลในโลกจริง จากผลการศึกษาเราพบว่าเราสามารถนำบล็อกเชนมาใช้ในการยืนยันตัวตนของคู่เซ็นสัญญา ใช้ในการสร้างลายเซ็นอิเล็กทรอนิกส์ที่สามารถนำมาใช้เซ็นสัญญาได้ และนำมาใช้ในการบังคับใช้เงื่อนไขสัญญาหลังจากการเซ็นสัญญาสิ้นสุดลง

การยืนยันตัวตนและการเซ็นสัญญานั้นทำได้โดยการใช้หนึ่งในเทคโนโลยีของบล็อกเชนที่เรียกว่า decentralized identity และ verifiable credential. ส่วนการบังคับใช้สัญญานั้นสามารถทำได้ผ่านสัญญาอัจฉริยะที่ทำงานบนระบบบล็อกเชน สัญญาอัจฉริยะนั้นมีข้อจำกัด การบังคับใช้สัญญาจะถูกจำกัดอยู่ภายใต้กรณีที่เงื่อนไขของสัญญานั้นไม่ซับซ้อนมากนัก ซึ่งขอบเขตของงานวิจัยเราก็คือสัญญาเช่าแบบง่าย เนื่องจากเทคโนโลยีบล็อกเชนนี้นั้นมีความปลอดภัยสูง มีลักษณะที่ไม่สามารถแก้ไขสิ่งที่ทำไปแล้วได้ เก็บบันทึกข้อมูลทุกอย่างที่เกิดขึ้น และยังเข้าถึงได้ทุกคนอย่างเท่าเทียมแบบกระจายศูนย์โดยที่ไม่ได้มีได้บุคคลใดควบคุม ลักษณะเหล่านี้ทำให้สัญญาที่เซ็นโดยวิธีของเรามีความปลอดภัยและโปร่งใสกว่าวิธีธรรมดา สุดท้ายนี้เราได้ทำการพัฒนาเว็บแอปพลิเคชันต้นแบบเพื่อเป็นแนวทางในการสร้างเว็บเซ็นสัญญาด้านบล็อกเชนเพื่อการเผยแพร่ต่อไปแบบโอเพนซอร์ส

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์
ปีการศึกษา 2565

ลายมือชื่อนิสิต
ลายมือชื่อ อ.ที่ปรึกษาหลัก

6372096621 : MAJOR COMPUTER SCIENCE

KEYWORD:

Yosnai Chanatrutipan : Investigate the possibility of using smart contracts and digital signatures to create a legally binding contract, and to create a prototype opensource web application as a proof of concept. Advisor: Assoc. Prof. NUTTAPONG CHENTANEZ

We investigated the possibility of using blockchain to create a legally binding contract. According to our study, blockchain can be used to authenticate the identities of the involved parties, to provide a cryptographically generated electronic signature used to sign a contract, and to automate and enforce the term of an agreement. The authentication and the signing are done using a blockchain-based self-sovereign identity framework called a decentralized identity and verifiable credentials. The term can be enforced by using a smart contract, a program that runs on the blockchain. Some contracts terms are too complex to be translated into a smart contract code. Therefore, we limit the scope of our thesis to enforce just a simple lease agreement between a tenant and a landlord.

Blockchain is publicly accessible, immutable, verifiable, and timestamped by design. Therefore, the contracts signed this way are more secure and more transparent compared to the ones signed with the traditional method. The main drawback of this method is user experience. Interacting with the blockchain required a certain level of computer literacy which made it not suitable for everyone.

We built a proof-of-concept web application. We design it in such a way that it can be used by anyone, sacrificing some decentralized aspects for ease of use. The users can sign and enforce a lease agreement just by using their phone, without having to interact with the blockchain themselves at all. The code is open source, published on GitHub.

Field of Study: Computer Science

Student's Signature

Academic Year: 2022

Advisor's Signature

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor for their invaluable guidance and support throughout the completion of this thesis.

Yosnai Chanatrutipan

TABLE OF CONTENTS

	Page
.....	iii
ABSTRACT (THAI)	iii
.....	iv
ABSTRACT (ENGLISH)	iv
ACKNOWLEDGEMENTS	v
TABLE OF CONTENTS	vi
Abstract	1
INTRODUCTION	2
Chapter 1 signing a contract.....	7
Signing on paper.....	7
Signing on computer	9
Electronic signature	9
Problem with an electronic signature	10
Digital signature	11
Certificate authority and Digital certificate	13
A review of an existing document signing web application	14
Chapter 2 BLOCKCHAIN	16
Blockchain.....	16
Smart Contract.....	17
The blockchain used in this thesis.....	19
Chapter 3 Decentralized identity.....	20

A decentralized identity identifier (DID).....	22
A decentralized identity document. (DID document).....	24
Problem with DID and DID document.	28
Verifiable credential.....	30
Chapter 4 Method	36
Signing a contract using a verifiable credential as an electronic signature	39
Enforcing a lease agreement with smart contract	44
Deploying a smart contract	46
Smart contract automation on Ethereum.....	47
Advantage of rent payment with smart contracts.....	48
Related work	49
DIAGRAM (OVERVIEW OF THE THESIS).....	50
Chapter 5 Opensource Web application.....	57
How to create a decentralized identity.....	57
How to issue and obtain a verifiable credentials: part 1	63
How to issue and obtain a verifiable credentials: part 2	65
The web application	69
SIGN PAGE.....	70
Document authentication Page.....	77
Lease Page	78
Chapter 6 Conclusion and Future work	80
REFERENCES.....	82
VITA.....	84

Abstract

We investigated the possibility of using blockchain to create a legally binding contract. According to our study, blockchain can be used to authenticate the identities of the involved parties, to provide a cryptographically generated electronic signature used to sign a contract, and to automate and enforce the term of an agreement. The authentication and the signing are done using a blockchain-based self-sovereign identity framework called a decentralized identity and verifiable credentials. The term can be enforced by using a smart contract, a program that runs on the blockchain. Some contracts terms are too complex to be translated into a smart contract code. Therefore, we limit the scope of our thesis to enforce just a simple lease agreement between a tenant and a landlord.

Blockchain is publicly accessible, immutable, verifiable, and timestamped by design. Therefore, the contracts signed this way are more secure and more transparent compared to the ones signed with the traditional method. The main drawback of this method is user experience. Interacting with the blockchain required a certain level of computer literacy which made it not suitable for everyone.

We built a proof-of-concept web application. We design it in such a way that it can be used by anyone, sacrificing some decentralized aspects for ease of use. The users can sign and enforce a lease agreement just by using their phone, without having to interact with the blockchain themselves at all. The code is open source, published on GitHub.

INTRODUCTION

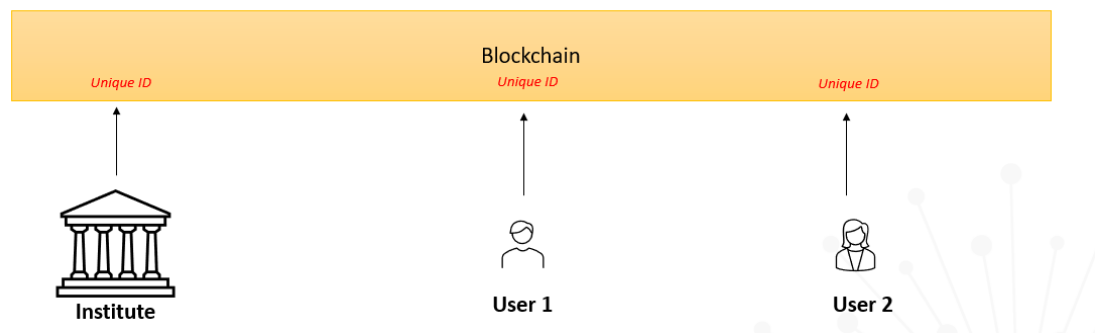


Figure 1 , A decentralized identity is a combination of digital ID and blockchain. The identity of an individual or organization can be stored, authenticated, verified, and viewed directly on the blockchain without requiring a third-party intervention.

A contract needs to be signed to be legally valid. The most common way is to sign a handwritten signature on a paper, but in this thesis, we will focus on how to digitally sign a document on a computer.

Unlike on paper, a signature on computer can be generated in many ways, the quality of signatures varies based on the method used to create them, some are more secure than another. What signature we use depends on the level of security required for the contract in question. The contract between schools and parents about the small deposit fee would not need the same level of integrity as the contract between two companies dealing with millions of dollars. In this paper, we will focus on the contracts that are highly staked, meaning that the contract should be signed in a secure manner with a good quality signature.

A signature create on a computer is call an electronic signature. An electronic signature can be anything as long as it can be used to represent a person, for example, a digital hand-drawing signature, a name in a plaintext format typed in Microsoft word, a signature created from the secure cryptographic method, or a signature issued by a trusted authority, and etc. In legal setting, all signatures are

equally valid and legally equivalence to the handwritten signature done on paper[1]. However, using a too-simple signature can lead to a problem later. It would be best to use a signature that is cryptographically generated from a well-known algorithm and that is bound to the real-world identity of the owner. We will discuss this in more detail in Chapter 1.

In this thesis, we use a feature on a blockchain, called decentralized identity to create an electronic signature used to sign a contract. A decentralized identity is a digital identity management system operated on the blockchain, in which everyone owns a unique identification number. To be able to claim and operate this identity, the cryptographic key is required; These keys are given only once when the users create their decentralized identity. We will use these keys to sign a contract. These keys are our electronic signature. The difference between these blockchain keys and a normal key will be discussed in Chapter 2. The contract signed with these keys can be authenticated and verified by anyone directly from the blockchain, and it can be done in total privacy.

We can go even further with the blockchain using verifiable credentials. This verifiable credential is one feature of decentralized identity. It is a digital credentials issued by a trusted institute. It exists to bind the real-world identity and credentials of its owner to an online identity, in this case a decentralized identity. For example, this guy, who has decentralized identity ID 1234, wants everyone to know that he can drive a car. He can ask a driving license issuer to issue him driving license in the form of a verifiable credentials addressing to a decentralized identity ID 1234, he can use this verifiable credential to proof to anyone that he can drive as long as he has the private key corresponding to that ID to back up his claim.

In our thesis, we use this verifiable credential for signers to authenticate each other. Verifiable credentials are built on top of decentralized identity, which are built on top of blockchain. Therefore, the verifiable credentials are cryptographically secure and immutable by design. It can be used as an electronic signature too. The guy

from the previous example could use that driving license verifiable credentials to fill in the form of a contract or simply attach it along with the contract document as an electronic signature. Using these methods, we can have more than one electronic signature per contract. Normally, we sign the contract only once, but in our thesis, we can sign the contract multiple times, not in the form of handwritten signature, but in the form of credentials from verifiable credentials. It's like attaching your digital ID to the contract documents. It acts like an electronic signature.

Anyone who reads the signed contract will see which part of the contract is data from a verifiable credential and which one is not. In our demo application, the signer can force the other party to fill in a specific form of the contract using only data from a verifiable credential. Therefore, the contract cannot be finalized or signed unless the signer has verifiable credentials to back up his claim.

While the decentralized identity is publicly searchable like searching a name on google, these verifiable credentials are kept privately even though its underlying technology is based on decentralized identity which is public. More detail on decentralized identity and verifiable credentials in Chapter 3.

The method we use can sign any type of contract, however, enforcing them with a smart contract is a different matter. Smart contracts are currently not developed enough to enforce just any type of contract; Some contracts terms are too complex to be translated into a smart contract code. Transferring real money or setting up an automated recurring payment system is also not possible due to a smart contract limitation. We offer a solution to some of these problems in this thesis.

Smart contracts are pieces of code with predefined action sitting quietly on the blockchain waiting for someone to execute them. The landlord and the tenant must define what action it should do in advance, since it cannot be changed after its creation. For example, landlord can set a smart contract like this "If this guy does

not pay rent by 20 August, then set this lease agreement status to TERMINATED FOR A BREACH OF CONTRACT”.

The key point is that smart contracts are immutable. It cannot be modified. It will do exactly what it was program to do. It records everything it has ever done. The code is on the public blockchain so anyone can verify it. This is perfect to enforce a legal contract. However Smart contracts are not easy to construct, it must be written in a programming language. They are not suitable to be used by everyone. This thesis will show how an application can provide the tools for anyone to create smart contracts without coding.

This thesis is organized in the following ways. Chapter 1 is an overview of signing a contract in general; we will go over all the tools that are needed to sign a contract, e.g., an electronic signature, public key infrastructure, certificate authority, and digital signature. Chapter 2 and Chapter 3 will provide the reader with the essential background knowledge on blockchain, decentralized identity, and verifiable credentials. Chapter 4 explains how to use them to sign and enforce a contract. At the end of chapter 4, there is an easy-to-follow diagram overviewing the whole operation. Chapter 5 will show how to implement the web application outlined in chapter 4. The web application functionalities are demonstrated in a series of figures in detail from start to finish.

The concept of decentralized identity, verifiable credential and smart contract is not yet a mature concept. Documentation on how to build it in detail is rare, and in some cases, nonexistent. There is no contract signing service on the market that uses verifiable credentials or decentralized identity to sign a contract just like us yet. There is also no service that allows a user to enforce the term of lease agreement with a smart contract either despite this idea being proposed in many research papers. This is due to some smart contract limitation that we will discuss later.

In this thesis, we create a demo web application to demonstrate the whole process in more detail and to serve as an implementation reference for any future work.

Chapter 1 signing a contract.

Signing on paper

Signing a contract on paper typically involves the following steps:

1. **Drafting the Contract:** The first step is to create the contract itself. This is typically done by a party or their legal representative, outlining the terms and conditions of the agreement. The contract should include all relevant details such as names and addresses of the parties involved, the purpose of the agreement, obligations and rights of each party, payment terms, duration, and any other specific terms and conditions.
2. **Reviewing the Contract:** Once the contract is drafted, it is essential for all parties involved to carefully review the document. The party should ensure they understand the terms and conditions and seek legal advice if necessary. This step helps avoid misunderstandings and disputes later.
3. **Making Amendments:** If any party wishes to make changes to the contract, they should communicate their proposed amendments to the other party. Both parties can negotiate and reach an agreement on any modifications before proceeding to the next step.
4. **Signing the contract:** Once all parties are satisfied with the contract and any amendments, the contract can be signed.
5. **Retaining Copies:** It is crucial to keep the signed contract in a safe and easily accessible place. Parties should store the contract along with any supporting documents related to the agreement. This ensures that all parties can refer to the contract if disputes arise or if they need to enforce or comply with its terms in the future.

It's important to note that the specific process may vary depending on the jurisdiction and the type of contract involved. Consulting with legal professionals is always recommended to ensure compliance with local laws and regulations. We will not be focusing on these matters in this thesis.

Although this thesis builds a web application that can sign any type of contract, we will focus on the leasing agreements. The steps of signing a leasing contract are listed below.

1. **Initial Agreement:** The prospective tenant and landlord first agree on the terms of the lease, including the rental price, lease duration, security deposit amount, maintenance responsibilities, and any other specific terms and conditions. This negotiation phase is crucial to ensure both parties agree before proceeding.
2. **Drafting the Lease Agreement:** Once the terms are agreed upon, the landlord prepares the lease agreement. The document outlines all the terms discussed in the initial agreement, as well as any additional clauses or provisions required by local laws.
3. **Reviewing the Lease Agreement:** Both the tenant and the landlord should carefully review the lease agreement to ensure they understand all the terms and conditions. It is advisable for each party to seek legal advice if needed, especially if there are any complex provisions or unfamiliar clauses.
4. **Amendments and Negotiations:** If either party wishes to make amendments to the lease agreement, they should discuss the proposed changes with the other party. Negotiations may take place to reach a mutually acceptable version of the agreement. It is important to document any agreed-upon modifications and ensure that both parties are satisfied with the final terms.

5. **Execution of the Lease Agreement:** Once the lease agreement is finalized, it is time to sign the document.
6. **Retaining Copies:** It is important to keep the signed lease agreement in a safe place, along with any supporting documents related to the tenancy. Both the tenant and the landlord should retain copies for reference, enforcement of terms, or resolving any potential disputes that may arise during the lease period.

Signing on computer

Signing on a computer is like signing done on paper, but the handwritten signature is replaced by an electronic signature.

Electronic signature

ขอแสดงความนับถือ

 (ยศนาย ชนัสสุตพันธ์)
 4 / 1 / 66

Figure 2 , an example of an electronic signature.

The easiest way to sign a contract digitally is to use applications such as Microsoft word or Adobe acrobat. In those applications, users simply type in or draw their signature on top of the document. The signature made this way is called an electronic signature. For a non-serious contract, signing a document like this is

perfectly fine. However, this type of signature is not secured enough in a professional setting. Before we talk about the downside of this type of signature, let's formally define what an electronic signature is.

An electronic signature is the signature that is made on a computer. It can be any data that is associated with its owner [1]. For example, your voice recording can be used as an electronic signature too. If it can be used to represent the person, then it counts as an electronic signature. In Thailand, it is legally recognized as an equal to the signature done on paper.

Problem with an electronic signature

1. It can be forged, and impersonated.

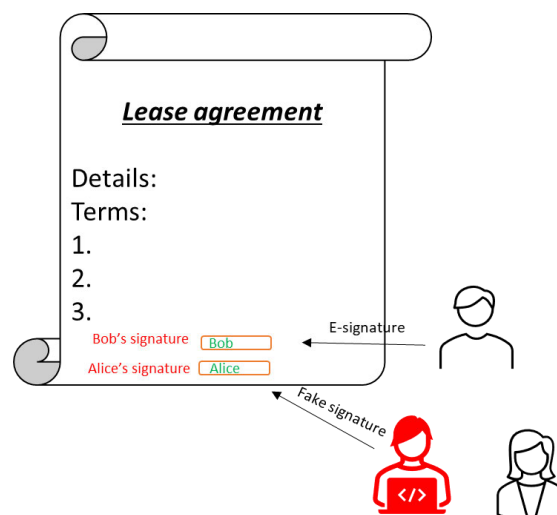


Figure 3, signature forging; a malicious actor type in a name that is not his. There is no way to prove who made the signature without external factor. This is a bad signature. A good signature should be able to be verified cryptographically by itself.

2. Not non-repudiation

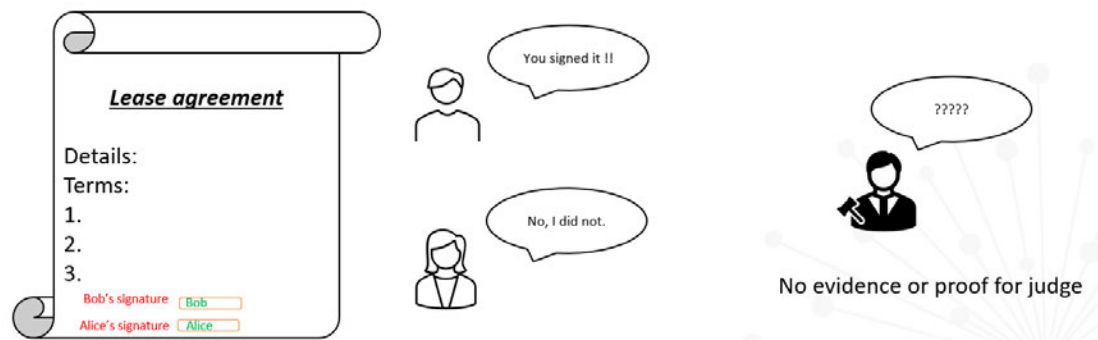


Figure 4 Not non-repudiation; the signer can deny any involvement with the contract. A good signature should be undeniable once signed.

Digital signature

The previous figures explained how a simple electronic signature made by yourself is not good enough in a high staked contract. The de facto standard is to sign a document under a public key infrastructure. The signature is created from a secured cryptographic algorithm to ensure the integrity and authenticity of the signature and its operation.

In public key infrastructure, each person carried 2 keys, a public key, and a private key (also known as a secret key). Both keys are created from a cryptographic algorithm. The public key can be shared with anyone, but the private key should remain a secret only to the owner. The most common metaphor is that the private key is the key to your house, and the public key is the address of your house.

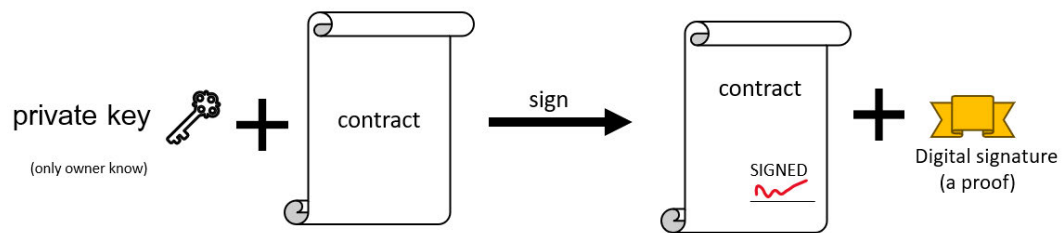


Figure 5, A private key can be used to sign a document. The result of the signing is a signed document and a piece of data called digital signature. In other words, a digital signature is a code, containing proof that you signed *THIS* document with *THIS* private key.

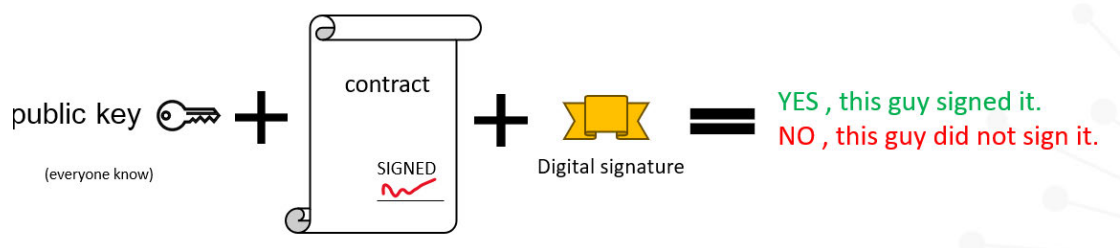


Figure 6, A digital signature is a non-repudiation proof that the signer did indeed sign a contract. Anyone can use the author's public key, a signed document, and a digital signature to verify the integrity and authenticity of the signed document.

These key pairs can be made offline by anyone; Nevertheless, one can also obtain these keys from a third-party identity provider such as a certificate authority. In addition to just a key pair, it comes in bundle with your identity attached to the keys, this is called a digital certificate.

Certificate authority and Digital certificate

Keys pair and digital signature alone cannot be used to identify the real-world identity of the signer. The use of third-party identity provider such as certificate authority is needed to link these cryptographic keys pair, to the real-world identity of the key owner.

A certificate authority (CA) is a trusted third party that issues digital certificates, which are used to establish the identity of individuals, organizations, and devices in a secure manner. Digital certificates contain information about the identity of the certificate owner, as well as an owner's public key.

CAs are responsible for verifying the identity of individuals and organizations before issuing digital certificates to them. They use a variety of methods to verify identity, including checking government-issued identification documents, verifying business licenses, and performing background checks. Once an individual or organization's identity has been verified, the CA issues a digital certificate that can be used to establish their identity and sign the document.

CAs play a crucial role in the operation of many online systems, including e-commerce, online banking, and other applications that require secure communication and authentication. They are typically highly trusted entities, and their digital certificates are widely accepted as evidence of identity in online contexts. Many document signing service providers are partners with the certificate authority, to offer an all in one, ease of use, centralized document signing platform.

A review of an existing document signing web application

Non-blockchain based online signing service.

There are many trusted signing services in the market, such as Adobe sign, Dropbox sign, DocuSign etc. The advantage of using these trusted services is that they provide everything the signer need to sign a contract in an all-in-one package such as creating and editing the document functionality, issue a digital certificate, secure and reliable signing process, data storage to store the document, an ability to verify the signed document, API request for developer, customer support, and more. The downside is that these services are centralized, not opensource, charge a subscription fee, some offer a free service but start costing money if the user goes above the free tier limit. Most importantly, all services rely on the central server of the host. If the website becomes unavailable, due to any reason, for example, a failure due to the cloud provider that the website is operated on, then the whole system will stop functioning. Although, this shouldn't be that much of a problem in modern times due to the advancement of clouds technology. Another downside is that all user activity is tracked by the service provider since everything is happening on their site.

Blockchain-based online signing services.

The centralized issue can be solved using a blockchain. The blockchain will be discussed in more detail in chapter 2, for now we will focus on the website that uses blockchain in their document signing service.

Blockchain is typically used to record the evidence of the signing, for example, storing the hash of a signed document on the blockchain, leaving a permanent immutable record. In some websites users can sign a document with a private key of their blockchain account, just like how we use a normal private key to sign a contract in a traditional setting. These websites all fail in one aspect. The blockchain key pairs

that the users use to sign a contract have no link to their real-world identity, unlike a digital certificate and a private key obtained from a certificate authority. We will solve this issue with a decentralized identity and a verifiable credential in the next chapter. We will bind the identity of an individual to their blockchain key pairs.

The signing services built on top of the blockchain technology are divided into 2 categories based on the type of blockchain these services operate on. The services built on the private or consortium blockchain require users to pay for the services. Meanwhile, services operating on the public blockchain are technically free to use, but the users need to pay the transaction fee themselves. Blockchain technology is inherently tamper proof, signing a contract on the blockchain guarantees the safety of the record.

Here is the (non-inclusive) list of signing services utilizing blockchain technology.

Sign.co- a paid signing service operated on IBM Hyperledger.

OpenLaw – an opensource document signing service operated on Ethereum.

EthSign – a signing service that operates on polygons. Currently in beta.

Our thesis goes further than these websites by adding in a decentralized identity and verifiable credential to bridge the gap between online identity and the real-world identity. Note that these websites only sign a document, there is no service that I know of that use smart contract to enforce the term after signing. The reason none exists is due to a certain problem with a smart contract that will be discussed later.

Chapter 2 BLOCKCHAIN

In this chapter, I will review all the decentralized technologies that we used to build a web application. We will start with blockchain and a smart contract, and then move on to the topics of decentralized identity and verifiable credentials, which is a technology that enables self-sovereign identity governance on the blockchain that we will use it as a replacement for a digital certificate.

Blockchain

Blockchain is a type of distributed ledger. The simplest way to understand blockchain is to think of it as a network of a synced-digital ledger. Back in the old days, a ledger was a book where all transactions are recorded. Now, instead of one book, there are many identical copies of that book held by different people. All of them keep checking each other's books to make sure that all their books are identical. Whenever a new record is to be added, there must be a consensus among all the book holders. This is called a consensus mechanism.

The blockchain works in the same way. Instead of multiple books, we store the same data on different computers, or in other words, a network. Transactions are stored in blocks. Each block represents a group of transactions, a new block is appended to the previous block forming a chain of blocks, hence the name Blockchain. Note that there exist hundreds of unique blockchain networks. The most well-known ones being Bitcoin and Ethereum.

One benefit of the blockchain is that it is decentralized. No single entity has control over it. It is also protected by cryptographic algorithms making it less vulnerable to hacking. To use the blockchain, you need a private key and a public key just like in a

public key infrastructure. You will obtain these keys when you create a blockchain account.

Blockchain can be used more than just for financial transactions. For example, it can be used to store information or create a new digital currency.

In conclusion, blockchain technology offers the following benefits.

1. Decentralization: Its decentralized nature enhances transparency, security, and resilience since no single entity has control over the network.
2. Security and Immutability: Blockchain is incorporated with a cryptographic technique. Once a transaction is recorded on the blockchain, it is nearly impossible to alter or tamper with, providing data integrity and preventing fraudulent activities.
3. Efficiency and Cost Reduction: Blockchain enables faster and more efficient transactions by eliminating the need for intermediaries.
4. Smart Contracts and Automation: some blockchain platforms support smart contracts, which are self-executing agreements with predefined rules and conditions. It is an ideal tool to enforce the terms of the contract.

Smart Contract

A smart contract is pieces of code reside on the blockchain. These codes can be called by a user or other smart contract to automatically perform a certain action on a blockchain, e.g., transferring assets between accounts, check user balances, and etc. Not all blockchains support the use of smart contracts. The most widely used blockchain for smart contracts is Ethereum. We will only refer to the Ethereum

smart contract from now on. Smart contracts from other blockchain will not be discussed.

Smart contracts are inactive by default. It lays dormant until someone executes it and goes back to sleep after its job is done. This is done to reduce the workload of the node running the blockchain. To prevent spamming, the user must pay a fee each time they want to use a smart contract to do something. The fee also increases based on the amount of work the smart contract does. The smart contract should be designed in such a way that it consumes as little fee as possible. In a private blockchain or enterprise blockchain, this fee can be set to zero. A service deployed on such a blockchain can be built without this restriction. However, the cost and the manpower need to set up and maintain the private blockchain make this option unrealistic for a small business or an individual developer. In this thesis, we will deploy the smart contract on the public blockchain (Ethereum) as it offers better security, longevity and requires no maintenance.

The biggest problem with the smart contract on the public blockchain is that it must be executed by someone. Technically, you cannot schedule it either since you must be present with the private key at all interaction, and you don't want to put your private key up online to automate some tasks.

A smart contract only supports cryptocurrency transactions, so you cannot transfer real money without some outside shenanigans. With these restrictions, it explains why there is no website that uses smart contracts to enforce a legal agreement. The problems with the automation are solved if the users do not mind using an external program to automate his private key usage. For example, the user can use a third-party service provider that lets him set up his private key to interact with the smart contract periodically and automatically. We will propose a possible solution for some of these problems; more detail in chapter 4.

The term “smart legal contract” can be used to refer to a legal contract incorporated with the smart contract. Legal contracts are typically written in a natural language. Some people have been working on writing contracts in term of pure code[2]. However, it is beyond the scope of this thesis. For this thesis, instead of writing a contract in code, we write a contract in natural language and then enforce the terms with code written in the smart contract.

The blockchain used in this thesis.

There are many blockchain available to choose from with different advantages and disadvantages. It doesn't really matter which one we pick since our work can be implemented in pretty much every blockchain. Nevertheless, we will use the two most popular blockchain, Bitcoin and Ethereum, in our demo opensource web application. Bitcoin is used to manage users' keys pair, identity, and their credentials (technically, we use ION, a blockchain that runs on top of Bitcoin). As for the smart contract, we use Ethereum.

Chapter 3 Decentralized identity

Blockchain or not, when we sign a contract, we must leave proof behind so that other people can verify it later, i.e., a digital signature. Normally, we store this digital signature on a private computer device or on an online server. However, these are susceptible to loss and fraud. If we store that part of the proof on the blockchain instead, then that proof stays on the blockchain as a permanent record forever, and even better, verifiable and timestamped too.

By the same logic, we can also store our identity or our credentials on the blockchain too. In this way, anyone can look at your information directly from the blockchain without needing to contact any organization or a government. The data on the blockchain cannot be tampered with, providing the best security that protects the integrity of your credentials and is also publicly verifiable available to everyone.

Blockchain is decentralized by design, storing your data or proof on the blockchain is an easy thing to do. You don't have to rely on anyone. You don't have to follow any guidelines on the best practice on how to store your data in it. It is an inherent nature of the blockchain that allows you to do these things by yourself. However, to keep things interoperable between different blockchain, to make other people search for you easily, we need to adhere to the community standards. These concepts of storing your information on the blockchain are called a *decentralized identity*. The most widely used standard and specification are the one published by World Wide Web Consortium (W3C)[3] and Decentralized Identity Foundation (DIF). All works related to decentralized identity in this thesis are made under these two specifications. If I were to summarize decentralized identity in one short sentence, then it would be “a decentralized identity is a combination of digital ID and blockchain”.

Decentralized identity is a system of identity management that uses blockchain technology to enable individuals and organizations to have more control over their own digital identities and personal data. In this system, identity is stored in a decentralized manner (on the blockchain), rather than being controlled by a single centralized authority. This means that individuals and organizations can use their own digital identities to access online services and resources, without having to wait for an online service to authenticate your identity with the authoritative source or identity provider that issue your identity. An online service, or anyone else, can verify your identity instantly, directly, and privately from the blockchain.

Decentralized identity is currently being developed by multiple organizations. The leading organization are World Wide Web Consortium (W3C) and Decentralized Identities Foundation (DIF), whose goal are both to create an open ecosystem of decentralized identity that is accessible to anyone and interoperable across different organization and across different blockchain.

Let's see how decentralized identity can be created. We will show just the concept in this section. In Chapter 5, we will show the technical details and the code required to create them. Just like any secure transaction, whenever we want to do something that others can verify later, we must *create a proof (verifiable evidence)*. The blockchain does this inherently.

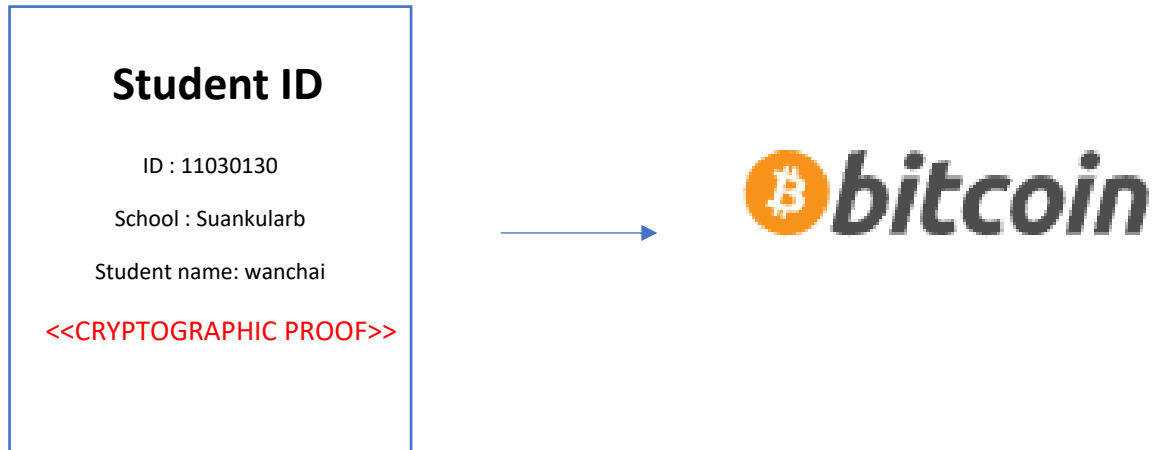


Figure 7, an example of a decentralized identity, you can make a document containing your credentials, signed it, and then publish them to the blockchain of your choice. You can always claim ownership of this document if you have a private key used to create this document. (Note that most blockchain will insert this cryptographic proof automatically). The private key in this context is the private key of your blockchain account, not the traditional one.

In the next section, I will show you how to create a proper decentralized identity based on a community standard, published by W3C.

A decentralized identity identifier (DID)

Under the decentralized identity system published by W3C, each person is uniquely identified by a string (stored on the blockchain), called Decentralized Identity Identifier (*DID*). We will refer to this name many times in this thesis. It can be used to refer to any person or an organization. In short, a *DID* is your digital ID on the blockchain.

A DID must follow the format below.

did : (blockchain) : (unique id)

For example, a person with a DID, *did:btc:11111111111111111111* has their identity published on the Bitcoin blockchain with a unique ID “1111111111111111”

A person with a DID, *did:eth:ABCDEFGHUIJK* has their identity published on the Ethereum blockchain with a unique ID “ABCDEFGHUIJK” .

A DID represents your unique ID on the blockchain. You can share it with anyone. Unlike a national ID, you don’t have to worry about its falling into the wrong hand. What is important and should be kept a secret is the private key associated with the DID, not the DID itself.

did:ion:EiAmFosP8PQIpI4PftKVt5fZaC_gbcNg8xM6nDAQf4I4FA

Here is an example of an actual DID I created and published on a blockchain called ION. This DID represent the web application that I created for this thesis. When someone creates a DID, they will receive a cryptographic key pair, a private key and a public key just like in a public key infrastructure. The private key can be used to claim ownership of this DID. The public key can be used to verify the integrity of the document signed by this private key. In summary, anyone can have a DID, a unique ID on the blockchain. It can be operated and controlled with cryptographic key pairs.

DID is not limited to an individual. An organization can create a DID to represent themselves on the blockchain too. We will show how to read and write a DID in Chapter 5.

A decentralized identity document. (DID document)

A decentralized identity identifier (DID) is just an ID in string format. We need a way to store data. A decentralized identity document, or in short, a DID document, is a file published on the blockchain in which anyone can read. It contains your data. The DID document is bound to the DID of its owner.

A DID document is just a file stored on the blockchain. You can insert anything that you want inside it. Whenever someone wants to see your information, you give them your DID. They can use that DID to search for a DID document on the blockchain by themselves at any time without needing to rely on anyone. You can prove that DID document on the blockchain belongs to you by using a private key. A DID document content is divided into two parts, your information, and the cryptographic proof. This cryptographic proof can be viewed as a digital signature so that a DID document can be claimed later, just like the role of a digital signature in a traditionally signed document. The cryptographic proof, used in W3C standards, is just a public key of the owner. Only the private key associated with that public key can claim this document ownership. From this point onward, the term cryptographic proof and public key have the same meaning and may be used interchangeably. They are proof, or a digital signature, that says this data belongs to you.


```

{
  "id": "did:ion:EiAmFosP8PQIpI4PftKVt5fZaC_gbcNg8xM6nDAQf4I4FA",
  "@context": [
    "https://www.w3.org/ns/did/v1",
    {
      "@base": "did:ion:EiAmFosP8PQIpI4PftKVt5fZaC_gbcNg8xM6nDAQf4I4FA"
    }
  ],
  "service": [
    {
      "id": "#logo",
      "type": "photo",
      "serviceEndpoint": "https://imgur.com/a/yjkaNov"
    },
    {
      "id": "#WebApplicationURL",
      "type": "linked",
      "serviceEndpoint": "https://www.laknainam.works/"
    },
    {
      "id": "#Contact",
      "type": "linked",
      "serviceEndpoint": "VCsignMasterThesis@gmail.com"
    }
  ],
  "verificationMethod": [
    {
      "id": "#key-1",
      "controller": "did:ion:EiAmFosP8PQIpI4PftKVt5fZaC_gbcNg8xM6nDAQf4I4FA",
      "type": "EcdsaSecp256k1VerificationKey2019",
      "publicKeyJwk": {
        "kty": "EC",
        "crv": "secp256k1",
        "x": "s9jij62nXBxFWNdUVZAMHcS0q91SL9Ypi9U0BT3Crwk",
        "y": "T3QcokSg0zZ7GdextrKcMoYH50Wy9zTYuWDioFRZKbQ"
      }
    }
  ]
},
1,

```

Figure 8, an actual DID document published on the blockchain ION, associated with our web application (did:ion:EiAmFosP8PQIpI4PftKVt5fZaC_gbcNg8xM6nDAQf4I4FA). You can see that you can insert anything inside it. In this case, I insert the web URL, photo, and a contact address inside it. A verification method is automatically generated so it is easier to make than its look. You can see that the verification method is actually a public key. (The easiest way to search for someone else DID document is to use a website <https://dev.uniresolver.io/>, enter a DID and the website will automatically query a corresponding DID document for you)

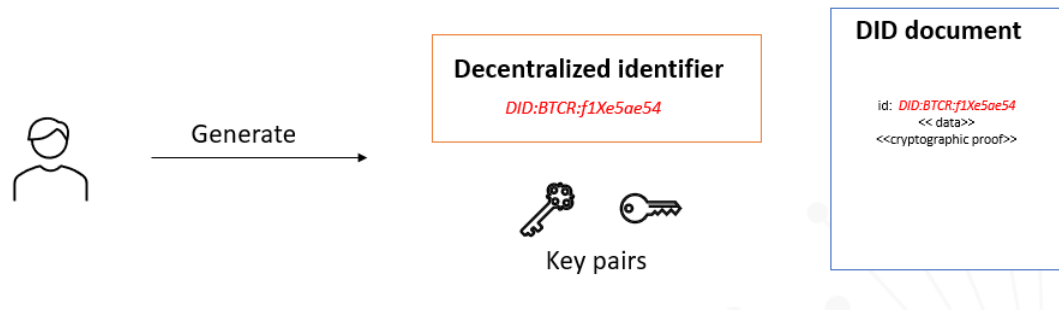


Figure 9, Creating a DID document; A person generates a key pair, a decentralized identifier, and a DID document. This process can be done offline. We will show how to do this in code in a later chapter. Due to the probability involving large numbers, it is almost guaranteed that you will never create the same DID with someone else.

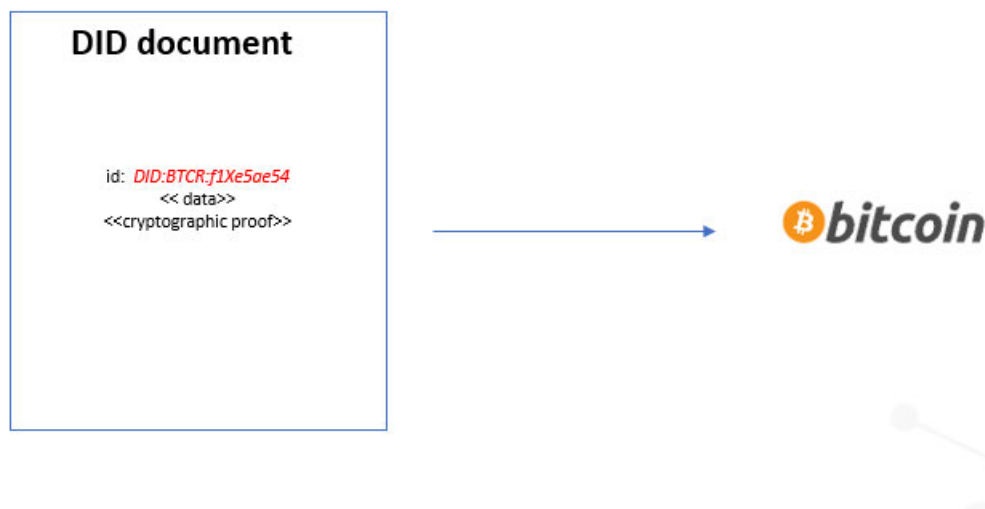


Figure 10, You can self-publish your DID document on the blockchain of your choice (Doesn't have to be bitcoin)

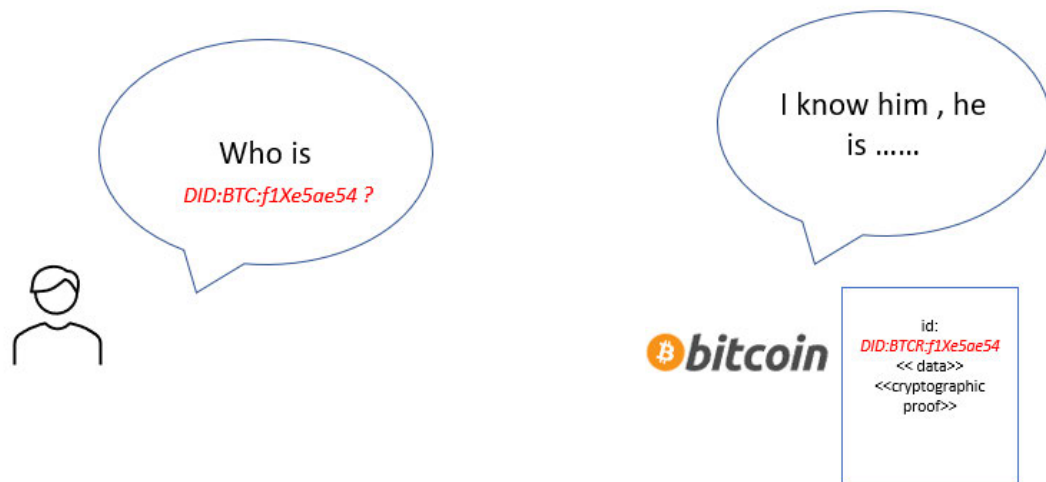


Figure 11 anyone can search for you directly from the blockchain.

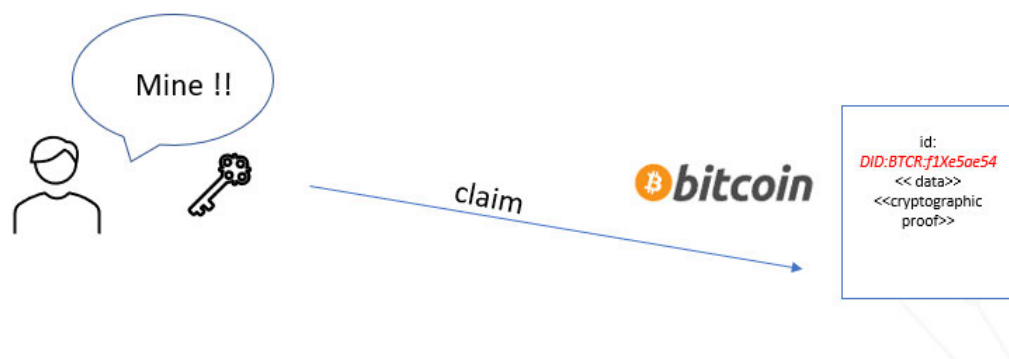


Figure 12, but only you can claim it ownership.

Problem with DID and DID document.

We can store our information on the blockchain using DID and DID document.

However, we don't want to store our sensitive information inside it. For example, storing user credentials like national ID, diploma, birth certificate on the blockchain is a terrible idea.

There must be a way to store our credentials privately, but somehow still be able to utilize the power of the blockchain to keep the data immutable, contain proof of ownership, publicly searchable, and cryptographically verifiable. Luckily, we already have a technology for that, it is called a *Verifiable credential (VC)*. Before we talk about the Verifiable credentials, let define what a credential is.

A credential is a claim about someone about something. For example, a diploma is a claim a person graduated from a university. A driving license is a claim that you are allowed to drive. A digital credential is a credential store on computers, for example, a digital professional certificate from an online learning platform.

The main issue with a traditional digital credential is that it can be inconvenient to verify since it relies on the issuer responsibility, it also notified the issuer if we try to verify it, exposing a private interaction between two persons.

For example, if a student wants to apply for a job, he need to use his diploma; the employer may want to check whether this student submits a legit diploma or not, the employer must contact the school and ask them to verify the integrity of the diploma. Even if the diploma file is signed with a digital signature attached, they still must go through the trouble of validating it, the school might store their digital certificate on third party centralized server, which can be compromised. You may grab the wrong one from a scam website, etc. In conclusion, you must check both student credentials and school credentials to verify the diploma. Decentralized

identity can solve this problem due to its immutable nature. You don't have to worry whether the file content is modified, or changed location, or whatever.

It is also easy to forge or copy digital credentials. Just because someone has a copy of your diploma doesn't mean they are the real owner of the credential despite passing the document authentication test. These digital credentials may be signed with a school signature, it can be authenticated, but it does not offer a way to claim it. The employer may be able to proof that the diploma is real, but that doesn't tell them that the student is the original owner or not. Note that the school will be aware of some of these interactions between the student and the employer too.

A good digital credential should be able to be verified by everyone securely and publicly yet should not reveal an attempt to do so, to the other party, even to the issuer themselves. It should only be claimable by its owner too.

Verifiable credentials are introduced to solve all these problems.

Verifiable credential

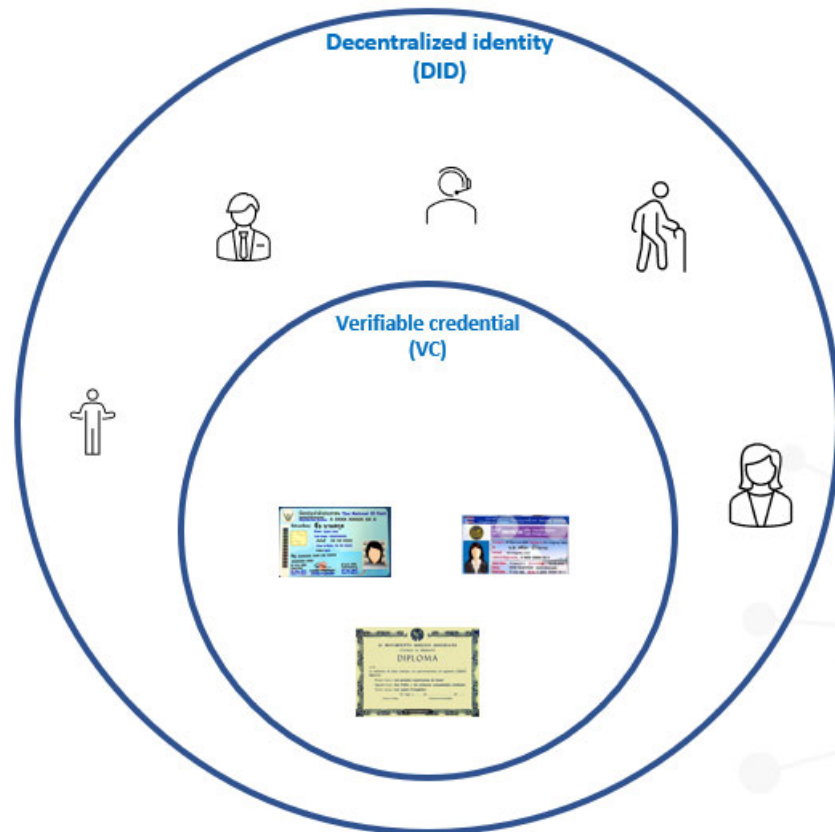


Figure 13, a verifiable credential is a credential that is linked to a DID

A verifiable credential is a digital credential combined with a digital signature, made specifically to support a decentralized identity system. Its ownership and integrity can be verified through a cryptographic algorithm hence the name “verifiable”. The simplest way to think about a verifiable credential is that, it is a digital ID that has a cryptographic proof attached to it. It can be verified by anyone, but only the owner can claim it. The verification and the proof of ownership does not require any trusted third party to perform unlike their traditional counterpart. The trick is to store some of it on the blockchain, enough to be verified and harness the power of the blockchain, but not too much that it reveals the sensitive information. Once issued, it stays on the owner device, binds to their DID, and does not exist anywhere else. The owner is the sole controller of their data. Later in this paper, we will propose a new

way to sign a contract using this verifiable credential as a signature instead of a digital certificate issue by the certificate authority.

An individual can have multiple verifiable credentials issues by different organizations. To share a verifiable credential with others, the user can select the information inside each verifiable credential they want to partially share and wrap them up in a “verifiable presentation”. A verifiable presentation is a wrapper around one or more verifiable credential. Think of a wallet with a credit card and national ID inside it. You can hand over your wallet to the police to identify yourself. In this case, the wallet is a presentation that contains the credentials, credit card and a national ID. In our demo web application, the user can share the verifiable credentials using a QR code from their mobile phone.

Perhaps the most important feature of the verifiable credentials is that no one will be able to track your verifiable credentials usage. You can send verifiable credentials to anyone. Anyone can verify it privately from the public blockchain. Reading from the blockchain does not leave any record, so no one will ever know who just checked on you. The issuer will not know how, where, and when you used the verifiable credentials that they issued to you. Once issued, the verifiable credentials belong solely to you. No one can control it except you.

The picture below shows an example of how the verifiable credentials work technically. For the full technical explanation of verifiable credentials and decentralized identity, the reader is advised to read from the official documentation.

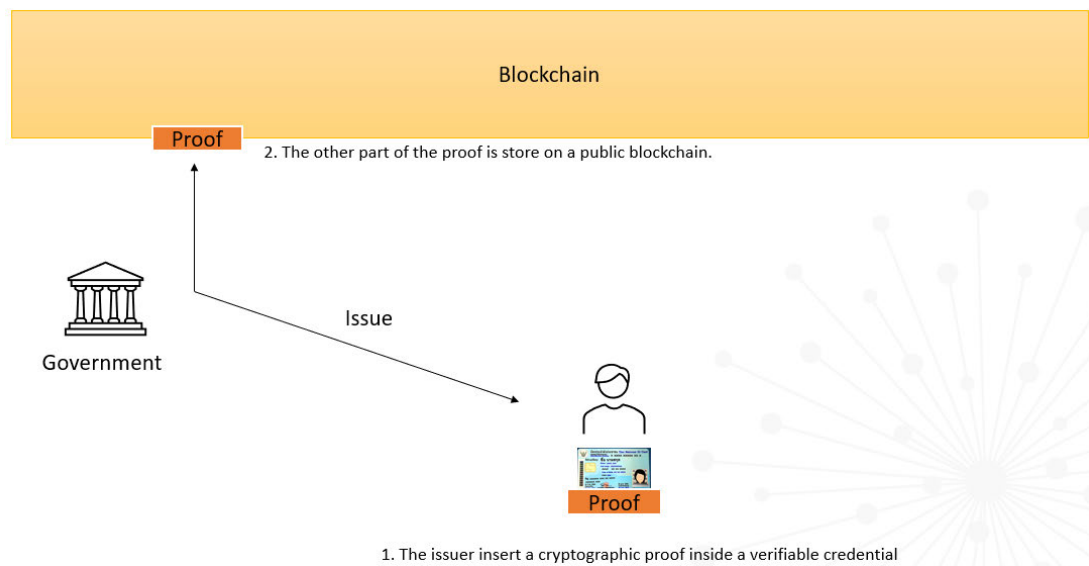


Figure 14, this is the simplified view of how a VC works. In this figure, the government (the issuer) wants to issue a digital national ID to an individual. The issuer creates two cryptographic proofs, one attached to the issued credentials, and another one stored on the blockchain. The government must have their decentralized identity published on the blockchain to represent themselves on the blockchain, so that anyone knows where to look for the proof. Whatever data signed by a government; anyone can verify it by grabbing cryptographic proof the government store on the blockchain. (The government store a proof inside their DID document, if the verifier knows the government's DID, he can retrieve a corresponding DID document from the blockchain and use a proof stored inside it to authenticate the signed data)

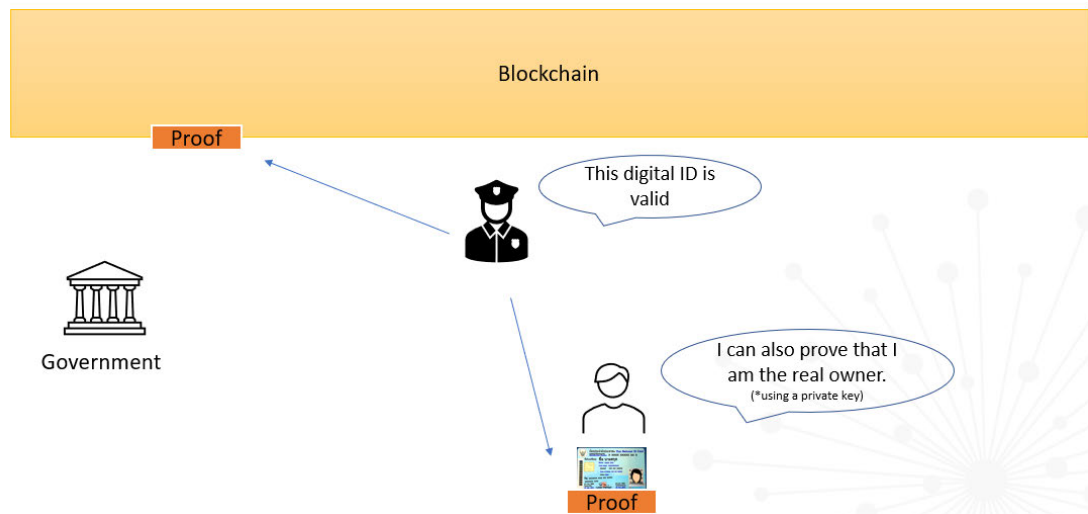


Figure 15, the verifiable credentials can be verified by comparing the proof on the blockchain with the proof on the credentials. If it matches, then the verifiable credential is validated. Note that the verification steps is more complex than what is shown here. The proof stored on the blockchain is the public key of the issuer inside a DID document. And the proof inside the issued credential is a digital signature signed by the issuer. The policeman must use the government public key store inside the Government's DID document to cryptographically validate the verifiable credential. What made this better than the traditional credential is that the proof inside the blockchain is immutable, timestamped, and easily searchable due to its being bound to DID. Moreover, even if a malicious actor stole the phone and the verifiable credentials, he cannot use them since he does not have the private key to claim the ownership of the verifiable credentials (assuming that the owner does not store the private key in the same device). Another key point is that, even if the government private key becomes compromised later, the verifiable credential issue before the compromised date is unaffected, since the verifiable credentials are timestamped. One can easily prove that it was issued before the compromised date.

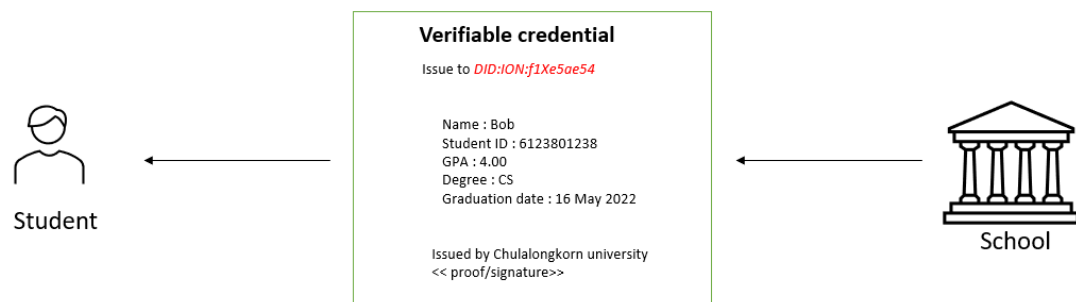


Figure 16, A simplify view of a verifiable credentials, the school issues a diploma in the form of a verifiable credential. The school issued this verifiable credential to the student with the DID of *DID:ION:f1Xe5ae54*. The student can claim this verifiable credential as long as he has the private key associated with that DID.

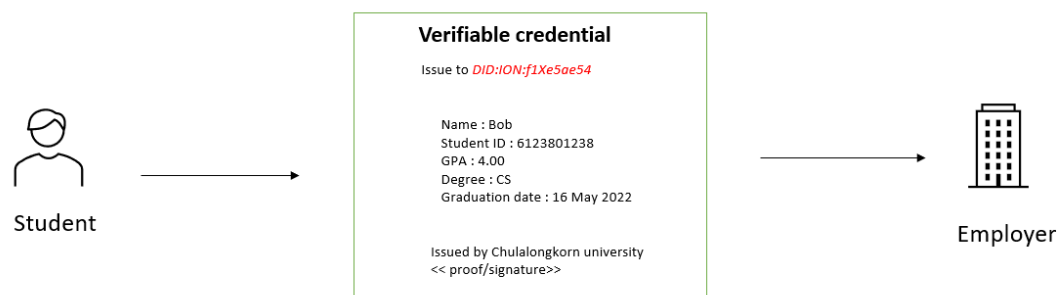


Figure 17 An employer verifies the authenticity of this diploma by using a public key of the school. The student also has to perform the proof of ownership to show the employer that he is the owner of the verifiable credential. Note that the school is not aware of this interaction and never will.

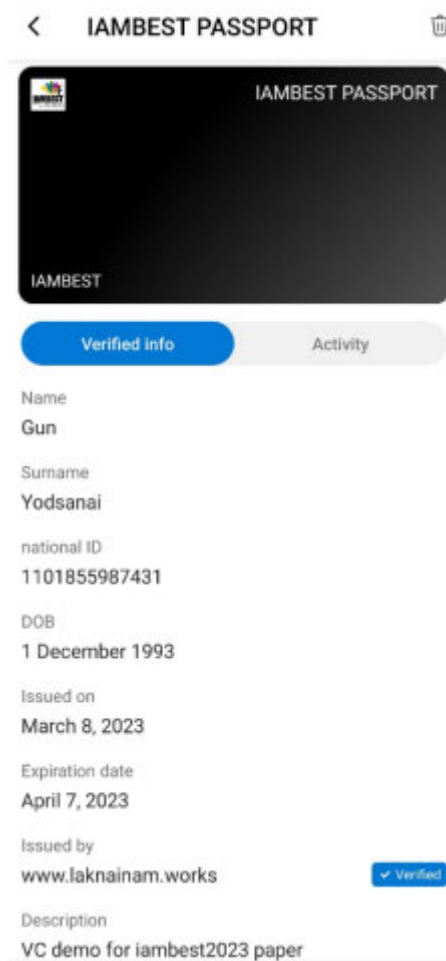


Figure 18, An example of a verifiable credential store on my phone.; A Verifiable credentials is just a text file like a DID document, but store privately on phone. Meaning that a developer can made it look however they want. This representation style is used by Microsoft.

In conclusion, the verifiable credentials are digital credentials, cryptographically tied to a decentralized identity identifier, that can be verified by anyone without leaving any record and can only be claimed by their owner.

Chapter 4 Method

In this chapter, we will show how to use a decentralized identity, verifiable credentials, and smart contracts to sign and enforce the term of the contract.

Both verifiable credentials and smart contracts are inherently designed to be part of blockchain technology. The contracts signed this way should have the following properties:

1. **Authenticity:** The contract is signed with DID private key. Anyone can use the signer's DID public key to verify the authenticity of the contract. The signer's real-world identity can be obtained from verifiable credentials associated with that DID private key.
2. **Integrity:** Any modification or alteration of the contract, even a minor change, would invalidate the cryptographic proof.
3. **Non-Repudiation:** Cryptographic proof prevents signers from denying their involvement or disowning the contract. Due to the unique link between the private key and the signer's identity, it becomes extremely difficult for the signer to repudiate their signature. If a dispute arises, the recipient can provide the digital signature from a signed contract as evidence of the signer's intent and involvement. This strengthens the legal enforceability of the contract and ensures accountability among the parties involved.
4. **Autonomy and Automation:** Smart contracts remove the need for intermediaries and manual intervention. Once deployed on the blockchain, they automatically execute predefined actions when specific conditions are met. The term in a smart contract cannot be denied. Once deployed on the blockchain, it cannot be changed and will only do its job in that way.

Before we talk about how to create such a web application, let's review the current industry standard method of signing a contract and see how we can improve upon it.

1. The users must obtain the digital certificate from a certificate authority or from a trusted identity provider. This step includes performing identity checks with the issuer.
2. The user can then sign the contract with the private key associated with that digital certificate.
3. Once signed, anyone can verify the integrity and the authenticity of the signed contract using the digital certificate of the author.
4. The tenant and the landlord fulfil the term of the contract as written in an agreement.

On step 1, a digital certificate can be replaced with a verifiable credential. They technically serve the same purpose, connecting a real-world identity of the owner to the key pairs, but the verifiable credential is a more advanced one due to it being a newer technology, i.e., built with blockchain integration in mind. Verifiable credentials are more secure and easier to verify and authenticate than a digital certificate.

Unfortunately, an identity check with the issuer in step 1 is unavoidable. Blockchain or not, the users must perform an identity check at least once, otherwise the verifiable credential will be meaningless since it does not bind to the real-world identity.

On step 2, we replace a digital certificate private key with a decentralized identity private key.

On step 3, the problem with the digital certificate approach is that the verifier must use the signer's digital certificate to verify the integrity of the signed contract, but how can one obtain signer's digital certificate in the first place? The signer can share

it with the verifier, or it can be obtained from the certificate authority who issued it. The 3 ways communication channel between signer, verifier, and the certificate authority can be eavesdropped on or can be intercepted by the malicious actor. All it takes is for one party to become compromised, and the whole thing will collapse.

This issue can be solved by using verifiable credentials and decentralized identity. Instead of exchanging confidential data with each other directly, we verify and share each other's identity through the blockchain. The data in the blockchain is immutable and innately cryptographically verifiable by anyone.

Verifiable credentials also have another advantage over digital certificates, it can be seamlessly integrated with other blockchain functionality and utilized its full power. For example, we could store the hash of the signed contract inside a DID document corresponding to the private key that was used to sign it. Leaving permanent proof of signing on the blockchain. (User does not have to do this though, they can sign a contract and save the document and the proof offline). The point is that it can be integrated with any blockchain features, unlike a digital certificate that does not have native blockchain support.

Signing a contract using a verifiable credential as an electronic signature

In this section we will list the whole procedure of the signing process between web application (service provider) and the users. An easy-to-follow diagram is attached at the end of this chapter.

A. Obtain a digital identity (DID/VC)

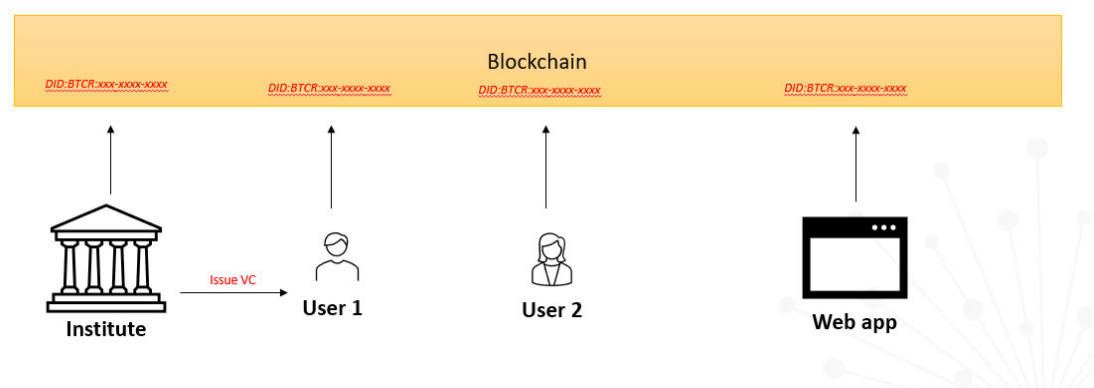


Figure 19, the issuer, users, and web application all have to create the decentralized identity to represent themselves to the other party.

Users who wish to sign a contract must create their decentralized identity by publishing a DID document onto the blockchain. The user then performs an identity check with an issuer of their choices to obtain a verifiable credential. The users must reveal their DID to an issuer so that an issuer can issue verifiable credentials to a correct DID address. Note that an issuer must verify that the user is the real owner of that DID before handling out verifiable credentials or else it will be issued to the wrong person.

The verifiable credentials should contain the information necessary to sign a contract, such as name and birthdate, or national ID. Extra information

can be added depending on the user's need. For example, one can request a university to issue a verifiable credential that contains the diploma as proof of graduation. Note that, if the user has already had a DID or a verifiable credential for other used before, he can reuse them without having to create or request for a new one. For example, the user may reuse his identity issued by the school to apply for an online shopping service rather than making a new one. Different services may require different levels of trust. For example, applying for a banking service may require a verifiable credential officially issued by your government, but applying for an online shopping service may accept a verifiable credential issue by any private company. It all depends on the service provider whether they trust the organization that issued you the verifiable credential or not. Once issued, a verifiable credential can be reused again and again without being tracked. Note that the issuer can set the expiration date for the verifiable credential, so you may have to renew a verifiable credential occasionally.

At this point, the user should have the following pieces:

DID – a unique digital ID

DID private key – for signing contract, for claiming DID and VC.

DID public key – for others to authenticate a contract you signed.

DID document – store user info and public key.

Verifiable credentials – a digital credential issued by a trusted source.

B. User authentication

Prior to the signing, both parties select the credential that they want to share from the verifiable credential and turn it into a verifiable presentation. Both parties share each other's credentials by exchanging the verifiable presentation. There are two verification steps that the users must perform here. First, the users must prove that they are the legit owner of the verifiable credential. This can be done using DIDs private key. Second, they must check that the verifiable credential of the other party is issued by a trusted issuer not some random untrusted organization. After this step, both parties are aware of each other's real-world identity and personal information. The web application should provide a necessary tool to assist this user credential exchange. In addition, the web application may also participate in these credentials exchange by presenting their verifiable credentials to the user. The web application could create their own decentralized identity and authenticate themselves with the identity provider so that the user can trust them.

C. Drafting and signing a contract

The web application should provide a platform where both parties can draft and sign a contract simultaneously to encourage fair signing. To make the signing fair, the web application must ensure that both users agree on the state of the contract, whether to keep on editing the document or whether both are ready to sign. This can be done by opening a secure communication channel between both parties. Communication between two parties must also be encrypted in such a way that the web application themselves cannot read them to ensure the privacy of the users. This is a typical scenario that can easily be

implemented using a standard encryption protocol. One options, the most extreme one, is to use a decentralized messenger; A messaging method that does not rely on the central server for a communication between users, but the encrypted message is broadcast through the blockchain node, ensuring the integrity while maintaining full privacy. We will not focus on this encryption part or use it in our web application.

The web application must provide the user with an ability to verify, authenticate, and claim both the verifiable credential and the decentralized identity. The web application must allow users to extract the credentials from the verifiable credentials and insert it into the contract document. This is what I mean when I say using the verifiable credential as an electronic signature. An electronic signature can be anything that can represent a person. The information inside a verifiable credential can be used as an electronic signature since it can represent the person's identity or credential. It's like attaching your digital ID with the contract document. Anyone who reads the document will know it's you who signed the contract since no one else has the private key to use your verifiable credential like that. It is a good quality signature, even more secure than a signature from a digital certificate.

For example, instead of typing in your date of birth in the contract, you can grab your birth date from a birth certificate verifiable credentials issued by the hospital and insert it on to the document. The web application should make it clear that this part of the document is verified data, for example, highlighting the text in color to differentiate it from a normal text. The key points here is that a signature generated from verifiable credentials is not limited to a typical handwritten signature, it can be any information like date of birth, diploma, or anything.

Once a contract document is finalized. A contract is signed once again with the decentralized identity private key, to leave proof that the information inside a document, including the part filled with the data from verified credentials, is done and is authorized by its owner. The users will obtain the signed document in pdf format and a digital signature as proof of signing.

We mentioned earlier that we signed a finalized contract with the decentralized identity private key, but whose keys should we use? We proposed 2 signing methods. The first method uses all signatures of the involved party, the signers, and the web application. For example, a contract between 2 individuals will be signed by three decentralized identity private keys, one from each signer and one from the web application. The second method, less decentralized but more convenient, uses only the web application's DIDs private key. As long as the web application is not compromised, then the signed document is as secured as done by the first method.

First method: Every participant signs a contract.

This approach is the most secure and decentralized way of signing a contract. Note that the user is required to have a digital wallet that can sign a transaction with decentralized identity private key.

Second method: Only Web application sign a contract.

In this method, users do not sign the contract themselves, rather, they give permission to the web application, to sign the contract for them. This can be done by the following step,

1. Users reveal their verifiable credentials to the web application.
2. The web application authenticates user identity and allows users to insert their credentials or attach verifiable credential data into a contract.
3. The web application then signed the finalized contract with its private key.

We will use this second method for our demo web application.

Enforcing a lease agreement with smart contract

The web application should provide the users with lists of lease agreement templates to choose from. The smart contract associated with these templates will be generated automatically when the users fill in the form. This should be enough for most users where an agreement is straightforward and can be constructed from the provided template.

For a custom contract that the user drafts from scratch, the web application will ask landlord on what the payment detail, or a termination condition should be before the contract is signed. The landlord can do this by filling in the form provided by the web application.


```
function pay_rent() public payable {
    require(msg.value >= rent);
    uint excess = msg.value - rent;
    if (excess > 0)
        payable(msg.sender).transfer(excess);
}
```

Figure 20; Smart contract in Ethereum is written in solidity, a programming language designed for developing a smart contract. The figure above shows a code that allows user to pay rents in an exact amount. If the user pays more than needed, then the smart contract will only retrieve a necessary amount and send the rest back.

กำหนดเงื่อนไขสัญญาอัจฉริยะ ✕

วันเริ่มต้นสัญญา	วันเริ่ม 12/11/2023
รายวัน / รายสัปดาห์ / รายเดือน / รายปี	ทุกๆ 1 เดือน
ระยะเวลาสัญญา (จำนวนครั้งที่ต้องจ่าย)	งวด 5 เดือน
ค่าเช่างวดละ	บาท 15,000 บาท
ค่าแรกเข้า	บาท 500 บาท
จำนวนครั้งที่ขาดจ่ายได้สูงสุด	ครั้ง 1 ครั้ง
จ่ายล่วงหน้าก่อนเข้าพัก	ครั้ง 2 เดือน

หมายเลขอ้างอิง
0

SUBMIT 

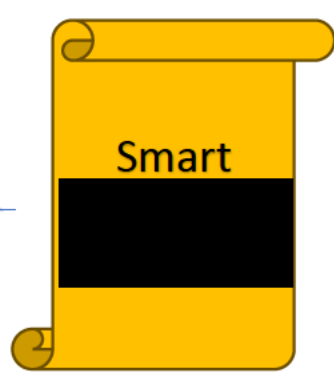


Figure 21; The web application should provide a user with an easy-to-use interface that can automatically generate a smart contract based on the user input.

Our method is very simple, the web application simply replaces the variable in a prewritten smart contract with a value that the user specified. For example, if the user wants to create a smart contract to enforce the term specified in the form in the figure above, that rent is equal to 15000 baht per month, we can just replace “rent” variable in figure 20, with the value “15000 baht”.

Note that the smart contract has not yet supported real money transfer. So, the above example is hypothetical, baht should be replaced with crypto currency. For Ethereum, the smart contract used Ether as a currency.

Deploying a smart contract

The web application generates the smart contract based on the condition specified in the previous section. Each lease agreement has its own smart contract. In this thesis, we deployed a smart contract on Ethereum blockchain. The web application must inform the user with the smart contract address right after it is deployed, so that the user will know where to look in case the web application might no longer be accessible in the future.

Interacting with a smart contract requires the user to talk to the blockchain, which is not an easy thing to do. Therefore, the web application should provide a place where users can interact with the smart contract from within the web application. Note that we want to remember which users belong to which contract, so that we can grab the correct one for them when they request it. To achieve this, we created another smart contract, “the factory”, as a state keeper. This factory keeps track of all the smart contracts that have been deployed by the web application. It will record which smart contract belongs to which users and which lease agreement. With this factory contract, the users can always retrieve their smart contract from Ethereum using the web application. The web application will read this retrieved contract and provide an easy-to-use interface for a user to interact with it. Note that

this factory smart contract is not necessary, the tenant and the landlord can access a deployed smart contract by themselves any time on Ethereum (Assuming that they remember its address).

Smart contract automation on Ethereum

As mentioned before in Chapter 2, the smart contract on Ethereum can only be executed by a user command. The user needs a private key to initiate a command. This is the only way to execute a smart contract.

The tenants must execute a smart contract with his private key to pay rent each month. This action cannot be automated. It cannot be scheduled. It is how smart contracts are designed. If you want to transfer money, you must use the private key of the money owners with no exceptions.

This weakness of Ethereum is one of the most discussed problems on Ethereum community. Fortunately, the solutions are underway. The Ethereum community is working on a feature called “**An account abstraction**”[4] This account abstraction allows users to have more control over their own Ethereum account. For example, you can have someone else pay for you or you can pay for someone else. This was previously impossible since the private key of the owner is required to transfer money from the owner’s account.

Account abstraction allows users to delegate their keys usage to someone else without actually giving them a key. For example, you can delegate the ability to transfer funds out of your account, to someone else[5], and let him execute a smart contract to pay rent for you in your name, using money from your account. Such a services provider will exist in the future. Enabling us to enforce and automate the terms of the contract without requiring user input periodically. The delegatee doesn’t have to be a person. For example, you can delegate your right to transfer funds to a smart contract. The smart contract will then use your keys to do whatever you program it to do.

Visas are working on a similar tech called “a delegable account”[6, 7], based on the same Account Abstraction method we mention above. Their goal is to allow users to be able to set up a recurring payment on Ethereum. The tenants may go on a vacation without having to worry about bringing his private key along to pay rent with a smart contract. The recurring payment system will automatically pay for him as scheduled. This is a work in progress, it may take a few years before we can use it.

Smart contracts are easier to automate if the tenant and landlord seek help from a service provider. For example, our web application can help users to automate the lease agreement. The web application acts as a server that runs 24/7. Therefore, we can program it to interact with the smart contract at the specific time in advance. For example, we can tell our web application to talk to smart contract at each midnight to check whether the terms are fulfilled yet or not. If is not fulfilled, then alert the landlord.

Let’s see another example, the smart contract can also act like an escrow, it holds a tenant money; the tenant can pay rent in advance to this smart contract, the web application can then schedule a command to tell a smart contract to release this fund to a landlord. The landlord does not have to do anything in this case. Everything is automated for him.

In general, smart contracts can be automated by a service provider if it does not involve transferring funds **from** someone else’s account. The service provider can schedule the smart contract to pay someone, but it cannot force someone to pay, since the user private key is needed in this case. Account abstraction will solve these problems.

Advantage of rent payment with smart contracts

Paying rent via smart contract offers many benefits compared to the traditional payment. First, the record of payment is safely stored on the blockchain which is

immutable and can be verified by anyone at any time. Second, it ensures that the correct amount of rent is paid, no more no less.

Paying rent in cryptocurrency has a low transaction fee no matter how large the amount is. Chen Qi-Long et al. [8], calculated the cost of using smart contract in lease agreement on Ethereum. Simple transactions like paying rent cost 0.000062 Ether or 0.01 USD at the time of the writing. In other blockchain such as Polygon, this fee can go as low as 0.0001 USD if the smart contract is well optimized.

Related work

The idea of using verifiable credential as a signature to sign a contract has not been proposed before but using smart contract to automate the terms of the contract has been done by many authors. Shanker [9] design a smart contract for lease agreement. In his work, the landlord deployed smart contract on Ethereum blockchain and notified the tenant. The tenant must pay rent to that smart contract. Shanker smart contract does not automate any process. The landlord still must check the money in the smart contract manually and decide whether tenants follow the terms properly or not. The smart contract simply acts as a payment system. His smart contract is premade with fixed condition unlike our approach that generates based on user input.

Smart contracts are not limited to lease agreements. Hasan and Salah [10] use smart contract to build a decentralized proof of delivery system that enable merchant and buyer to trade securely without middleman. It is designed in such a way that all entities involved are incentive to act honestly.

At the time of the writing, there is no research paper that discusses using verifiable credentials as a means to sign a contract like us. Constructing a custom smart contract for each lease agreement based on the contract document and user input are also not found in any literature.

DIAGRAM (OVERVIEW OF THE THESIS)

A visual example of the signing process from start to finish.

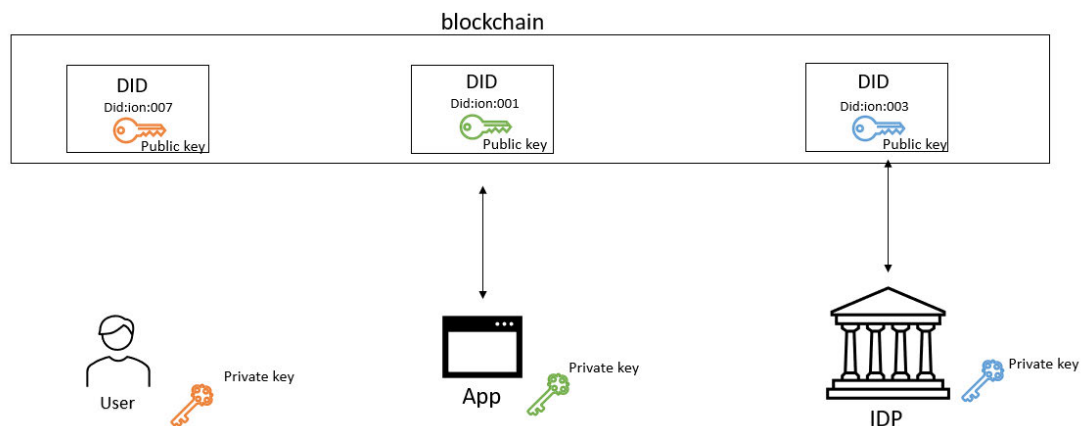


Figure 22, All participants, the users, the web application, the issuer (identity provider), create a decentralized identity to represent themselves. The public key is stored inside a DID document published on the blockchain. The private key is kept secret.

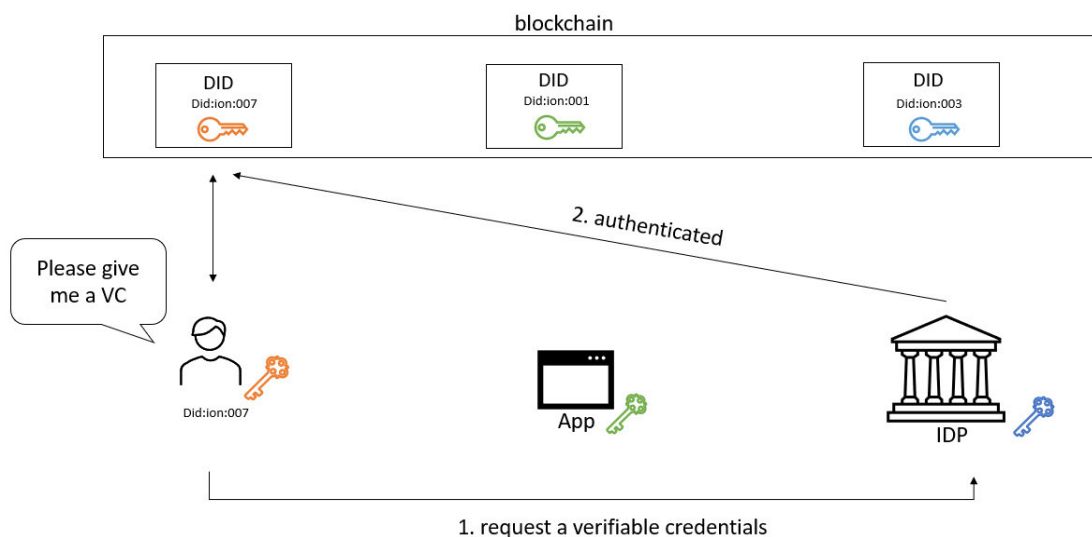


Figure 23, User request a verifiable credential from an issuer of his choice. Identity verification is the responsibility of the issuer. The issuer may refuse to issue verifiable credentials if the user does not have enough evidence to authenticate himself.

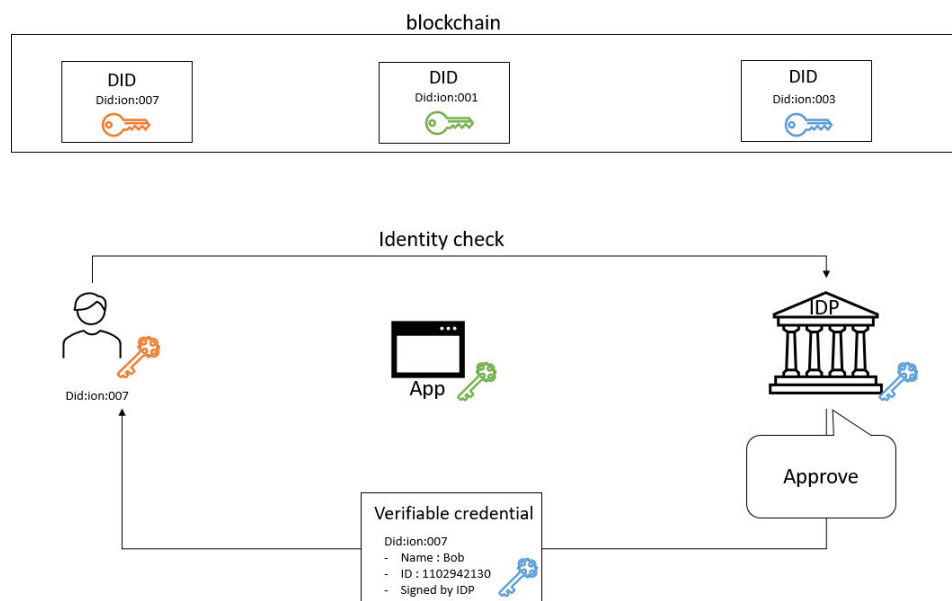


Figure 24, If the user passes an identity check, the issuer will issue a verifiable credential, binding user real world identity to his DID.

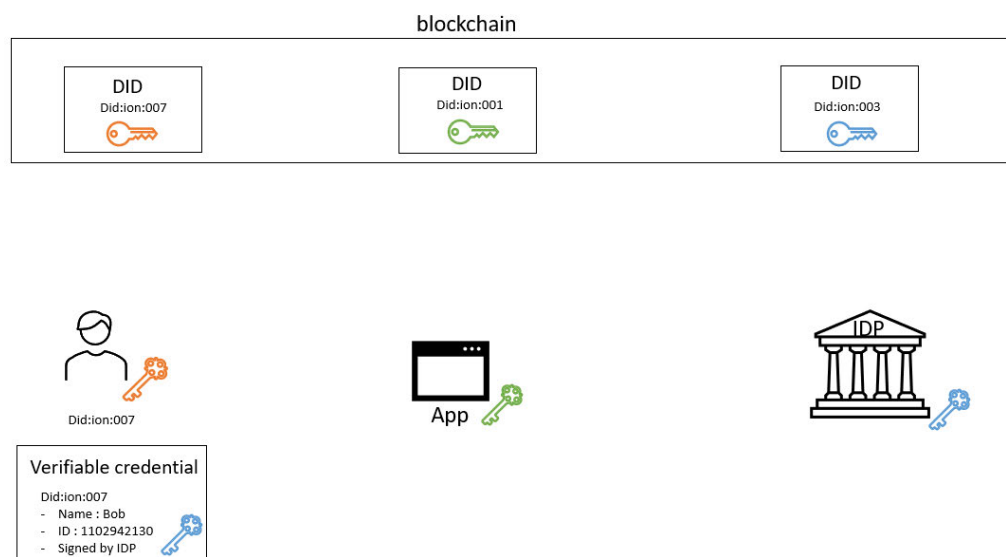


Figure 25, The user is now ready to sign a contract. Note that if the user already has a verifiable credential from other use before, he can skip all the steps above and reuse the old one.

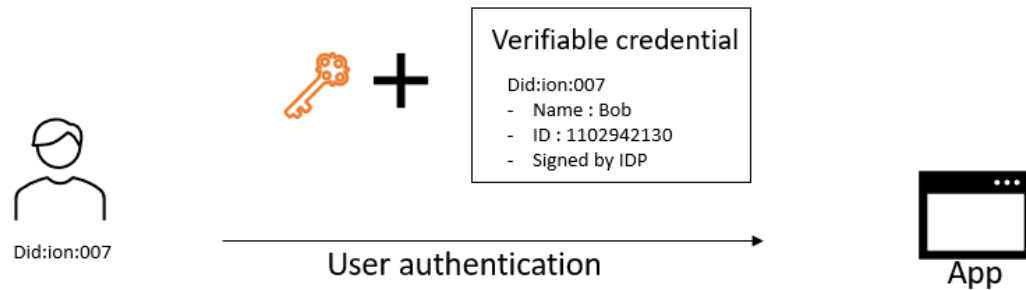


Figure 26, the user can authenticate himself with the web application by using a verifiable credential. The user must use his private key to claim its ownership, showing that this verifiable credential is not a stolen one.

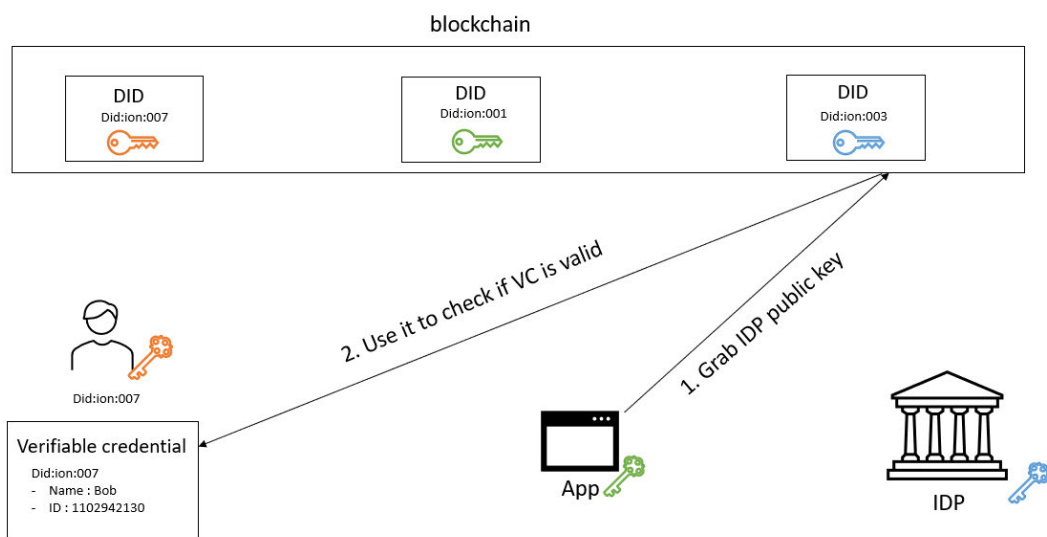


Figure 27, the web application verifies the integrity and the authenticity of this verifiable credential with proof store inside an issuer DID. This step will not be notified to the issuer, keeping an interaction between users and web applications a secret to everyone else.

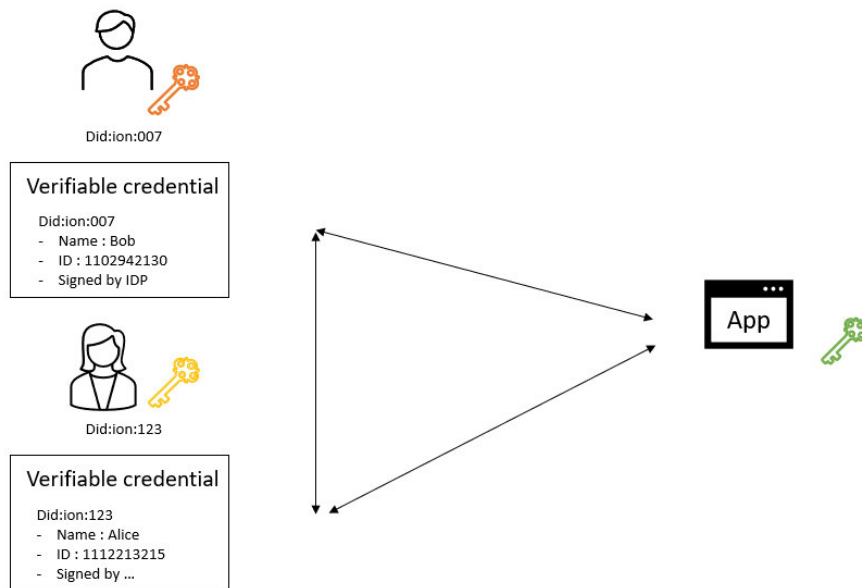


Figure 28, if there are multiple signers involved, the web application will share these verifiable credentials with all involved parties. At this point, all participants learn of each other's identity. Ready to draft and sign a contract.

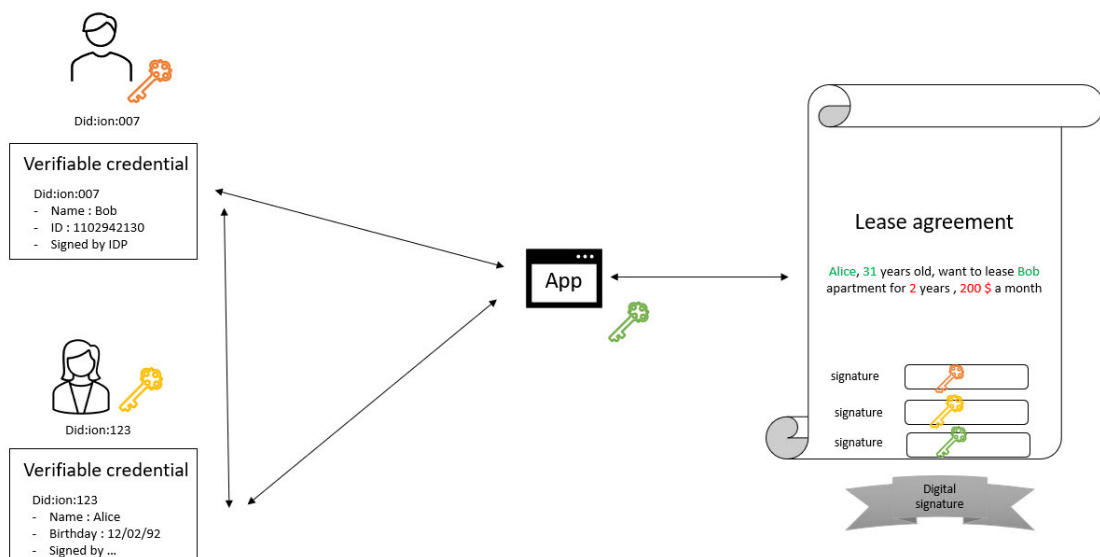


Figure 29, the web application allows user to fill in the form of the contract with an information inside a verifiable credentials. In other words, the users sign a document

with verifiable credentials. This is what it means to use verifiable credentials as an electronic signature. It's like attaching the digital ID with the contract document. The web application allows the landlord to force the user to fill in the contract with only a verified data, forcing the tenant to go and obtain the verifiable credentials from a trusted issuer first.

The last step involves each party signing a document with their private keys, including the web application. The reason the web application key is needed is to add extra security, to ensure that the document creates these ways (green text=verified data), is not arbitrarily forge to look like one. (Anyone can create a fake green text, but if the web application is overseeing the whole operation, then this cannot be done).

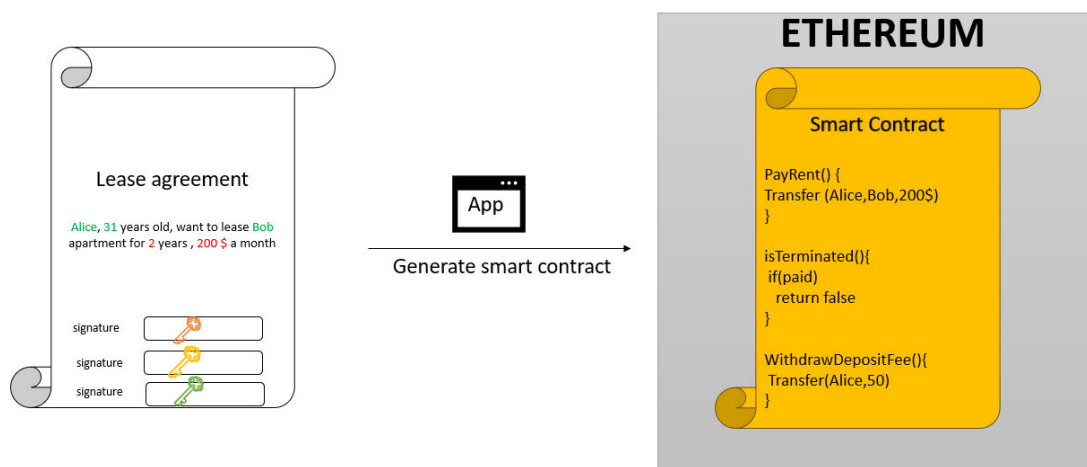


Figure 30, The web application generates a smart contract and publishes it on an Ethereum blockchain. If the user uses a contract template provided by the web application, then this smart contract is automatically generated. For a custom contract, or if the landlord wants to add in an extra term, the landlord can use a tool provided by a web application to manually add these enforcing conditions to a smart contract. In the future, the web application might use AI to determine the term of the contract just from a text document.

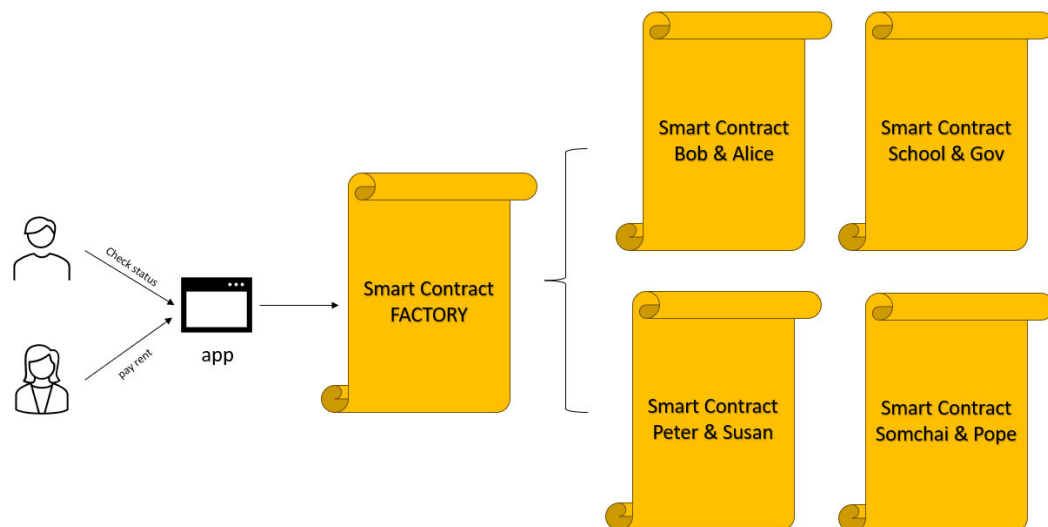


Figure 31, While not necessary, the web application should build a smart contract factory to record all the smart contracts that has been deployed. The smart contract factory records which smart contract belongs to which users. This way, the users can interact with their smart contracts from the web application at any time, i.e., The web application acts as an intermediary between user and blockchain. Alternatively, if a user wants to do things in the most decentralized way possible, the user can choose to interact with the smart contract directly from Ethereum by himself. Note that he must remember the smart contract address to locate it on the blockchain.

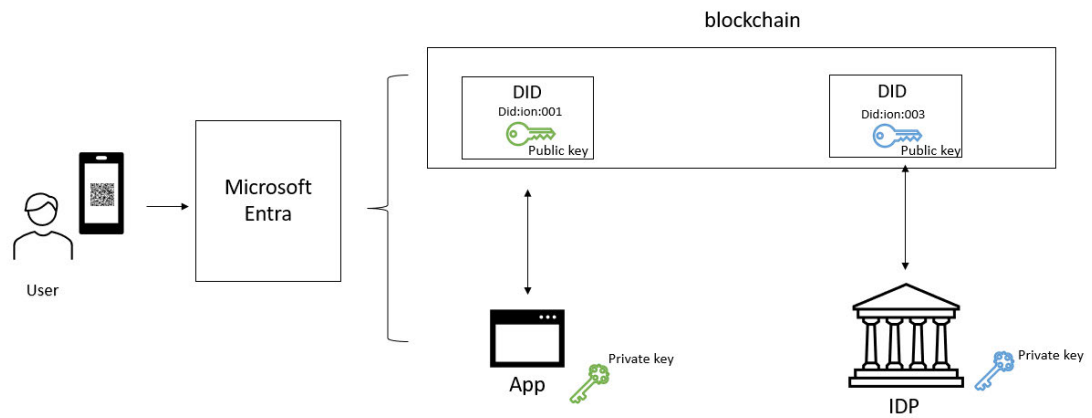


Figure 32, All the steps mentioned above required the user to be competent enough to manipulate their own private key. We use a different approach in our web application. We abstract away all the operation on the user's side, by relying on a decentralized identity as a service by Microsoft Entra. The user can obtain and share their verifiable credentials simply by using a QR code on a phone.

Chapter 5 Opensource Web application

This chapter starts with how to create a decentralized identity and verifiable credentials. After that, we will showcase the web application functionality.

The source code is available at: <https://github.com/TaroAndMulan/VSignAlpha>.

The web application that we created has the following functionality.

1. Allow users to authenticate each other using verifiable credentials.
2. Allow users to draft a contract from scratch or from a provided template.
3. Allow users to sign a contract.
4. Allow users to save a signed contract in a pdf format.
5. Allow users to verify and authenticate a signed contract.
6. Allow users to enforce the term of the lease agreement with smart contracts.
7. Allow users to view and interact with the deployed smart contracts.
8. Allow tenants to pay rent and landlords to receive money in cryptocurrency.
9. Allow tenant and landlord to use a smart contract as an escrow.

How to create a decentralized identity

We want our web application to sign a contract in such a way that anyone can verify it directly from the blockchain without relying on any third party. To achieve that, the web application must first create a decentralized identity to represent themselves on the blockchain first. Decentralized identity is supported in all major blockchain. It doesn't really matter which one the developer chooses, since it is designed to be interoperable across different blockchain.



Figure 33, ION; a layer 2 blockchain built on top of Bitcoin, built specifically for a decentralized identity system. An identity can be created and managed through an open-source tool.

In this paper, we create a decentralized identity on the blockchain name “ION”. It was created by Decentralized identity foundation (DIF), one of the two biggest names in decentralized identity community. ION is an open, public, permissionless layer 2 decentralized identity networks built on top of the Bitcoin, offering the same level of trust and security as Bitcoin. It can support thousands of DID operations per second across the network.

Ion is like any other blockchain, if you want to participate in it and use it to its full potential, you must download the whole blockchain to your computer, becoming a new node in the network. This can be troublesome; Ion uses Bitcoin as a backbone. Its total size is huge. In this paper, we will sacrifice some of the decentralized aspects for ease of use. We will rely on other people’s node and communicate to the ION blockchain via this node instead of dealing with the blockchain directly ourselves, saving tons of disk space. Many organizations provide an easy shortcut to read and write from ION network. For example, Microsoft allows any organization to set up decentralized identity on ION using a “Microsoft Entra” service. The decentralized identity foundation (DIF) provides several opensource codes on GitHub for developers to interact with ION.

The developer or user can create a decentralized identity by the following steps.

1. Clone an ion-tools repository[11]. ION tools repository contains all the tools and utilities you need to work with DID.
<https://github.com/decentralized-identity/ion-tools>
2. Follow the instructions on the GitHub page. Create a key pair, use it to create a DID, then publish it on the blockchain. I will highlight some of the steps in a list figure below.

```
let authnKeys = await generateKeyPair();
console.log(authnKeys.privateJwk)
console.log(authnKeys.publicJwk)
```

Figure 34. key pair can be created in single line of code. (Caution: You must write down or save your private key now. You will not be able to recover it later after this point.

```
let did = new DID({
  content: {
    publicKeys: [
      {
        id: 'key-1',
        type: 'EcdsaSecp256k1VerificationKey2019',
        publicKeyJwk: authnKeys.publicJwk,
        purposes: [ 'authentication' ]
      }
    ],
    services: [
      {
        id: 'picture',
        type: 'photo',
        serviceEndpoint: 'https://imgur.com/AIkpvFz'
      },
      {
        id: 'contact',
        type: 'linked',
        serviceEndpoint: 'Bob@gmail.com'
      },
      {
```

```

        id: 'Nationality',
        type: 'text',
        serviceEndpoint: 'Italian'
      },
      {
        id: 'DateOfBirth',
        type: 'number',
        serviceEndpoint: '02/12/93'
      }
    ]
  }
});

```

Figure 35 With a key pairs ready, DID document can be created using a constructor `new DID()`. Note that you can insert whatever data you want in the DID document. In this figure I create a DID for a person named Bob with his photo and contact address.

```
id:ion:EiBAhyd-INT04D9oa_IE040HqgS66-7nb3oxDZyLAjwSJw
```

Figure 36, You cannot select your own DID since it is automatically generated from the key pairs, which is randomly generated.

```
let anchorResponse = await anchor(createRequest);
```





Figure 37, everything we do so far is offline. We can play around with this local DID without affecting the blockchain. The DID can be published to the blockchain using “anchor” function. After this line of code, the decentralized identity you created earlier will be published on the ION blockchain. (Note: it may take a couple of minutes for DID to be published on the blockchain).

```
const DigitalSignature = await sign({ contract, privatekey });
const isLegit = await verify({ DigitalSignature, publickey });
```

Figure 38, sign and verify any payload with DID key pairs; This is how our web application signs and verifies the contract document.

did	method	method-specific-id	path=abempty	query	fragment
did:ion:EiBAhyd-INT0409oe_IE040HgS66-7nb3oxDZyLAjwS3w	ion	EiBAhyd-INT0409oe_IE040HgS66-7nb3oxDZyLAjwS3w			

Services

 photo #picture https://imgur.com/AIkpvFz
 linked #contact Bob@gmail.com
 text #Nationality Italian
 number #DateOfBirth 02/12/93

Verification Methods


 EcdsaSecp256k1VerificationKey2019 #key-1 {"kty":"EC","crv":"secp256k1","x":"s9t92Y_vxVA80prA0VzyNwVfS8a4u9hp7rXQuUaa5s","y":"3FZ-jzyYC3V7sGmuUj013DT68UIB363NkoubRyWPE1A"}
--

Figure 41, As show in the figure above, this DID link to Bob. The verification methods section contains proof (a public key). Anyone can grab this public key and verify any contract that Bob signed. Note that the private key is not online. The owner keeps private keys to themselves and publishes only the public key to the blockchain.

```
{
  "WebappPrivateKey": {
    "kty": "EC",
    "crv": "secp256k1",
    "x": "s9jij62nXBxFWNduvZAMHcS0q91SL9Ypi9UOBT3Crwk",
    "y": "T3QcokSgOzZ7GdexnxKcMoYH50Wy9zTYuWDioFRZKbQ",
    "d": "MzsGLadxCqpnZtyNmOQ1LKb1JCa6PDJSq_1-M0HEJ1w"
  },
  "WebappPublicKey":{
    "kty": "EC",
    "crv": "secp256k1",
    "x": "s9jij62nXBxFWNduvZAMHcS0q91SL9Ypi9UOBT3Crwk",
    "y": "T3QcokSgOzZ7GdexnxKcMoYH50Wy9zTYuWDioFRZKbQ"
  },
  "webappION":
  "did:ion:EiAmFosP8PQIpI4PftKVt5fZaC_gbcNg8xM6nDAQf4I4FA",
}
```

Figure 42 The web application identity is published on the blockchain as “did:ion:EiAmFosP8PQlpl4PftKVt5fZaC_gbcNg8xM6nDAQf4I4FA”. The public keys and the private keys are also shown. The web application will sign the contract with this private key.

Note there are other methods and approaches in how to create a DID too. Some companies provide a service for users to easily create a DID in a few clicks without coding like Microsoft. However, these companies are centralized providers, so choose carefully. In the next section we will move onto how to create verifiable credentials.

How to issue and obtain a verifiable credentials: part 1

Just like DID, there is an opensource code to issue and obtain a verifiable credentials in a few lines of codes at <https://github.com/decentralized-identity/did-jwt-vc>.

Let’s take a different approach here, I will not use this opensource code tool to create or issue verifiable credentials. Verifiable credentials are hard to use since the owner’s private key is required. We want to abstract away this difficulty and remove the burden from the users. Therefore, we will use a third-party service that can issue and verify the verifiable credentials in such a way that the users don’t have to manage private keys on their own. They can use their phone to receive or request verifiable credential by QR code.

In this thesis, we use Microsoft Entra as a backend to issue and verify the verifiable credentials. Note that we don’t have to use any third-party service for this. but I chose this approach since it is the most convenient way for end users.

Microsoft Entra is a service that allows users or organizations to manage a decentralized identity and verifiable credentials without having to talk to the

blockchain directly. Microsoft Entra operates on the same ION blockchain that we used in the previous section. Microsoft role is to help us communicate with the ION blockchain.



Figure 43 While it is possible to do everything in a decentralized way. Using private keys is too overwhelming for a general user. It can be dangerous if the user doesn't know how to keep the private keys safe.

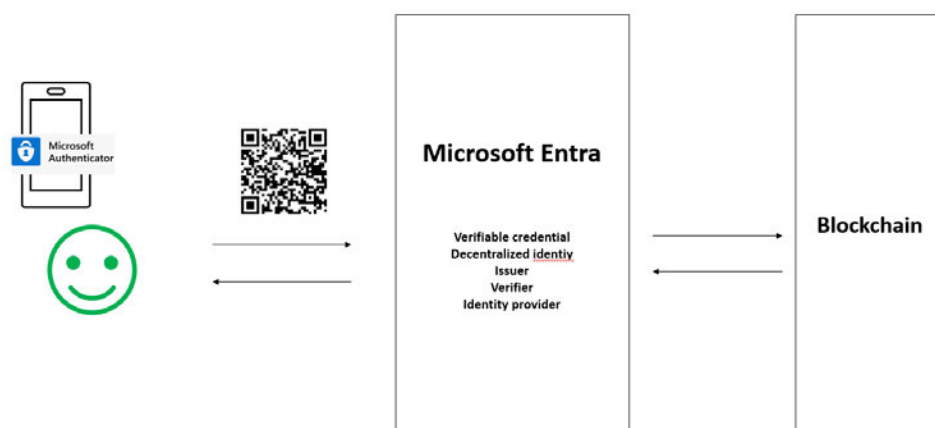


Figure 44; We use Microsoft Entra service to allow users to retrieve, store, and share a verifiable credential via QR code on their mobile phone.



Figure 45 For an overview of how decentralized identity work on Microsoft Entra, please visit <https://www.microsoft.com/en/security/business/identity-access/microsoft-entra-verified-id>

How to issue and obtain a verifiable credentials: part 2

Issuing verifiable credentials to the user is not a part of web application jobs. The users should obtain verifiable credentials from a trusted issuer instead; Nevertheless, the detail on how any organization can become an issuer, and how the user can obtain a verifiable credential from them is shown in detail in the following figures.

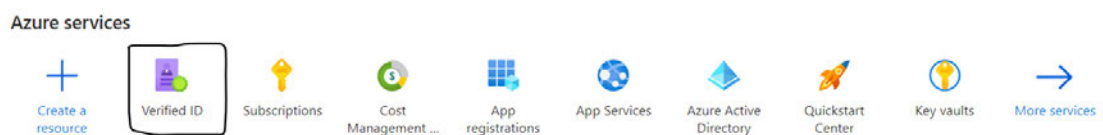


Figure 46; Set up an azure account for your organization. Then follow instructions from Microsoft Entra document. Programming knowledge is not required here. You will be asked to create a DID, linking a DID to the azure account, storing a private and public key inside azure storage, publishing organization identity on the ION blockchain, registering your domain, etc. After that, the organization can design, create, issue, revoke a verifiable credential to an individual from a “verified id” tab.

[Home](#) > [Verified ID | Credentials](#) > [Create credential](#) >



THESIS EXAM PASSPORT ...

Verified ID

Details

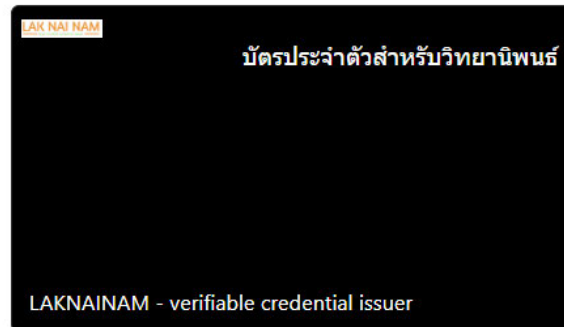
Manage

Properties

Issue credential

Revoke a credential

Credential preview



[Change style](#)

Credential details

Directory ⓘ

Default Directory

Credential type(s) ⓘ

VerifiedCredentialExpert

Claims ⓘ

4 claims

Figure 47; In this figure, I use Microsoft Entra to create verifiable credentials. As you can see on the left side of the screen, you can issue and revoke credentials within a few clicks. Microsoft also provides an opensource code on GitHub for an organization to set up a web server that can issue and verify verifiable credentials as an alternative of doing thing from the web browser.

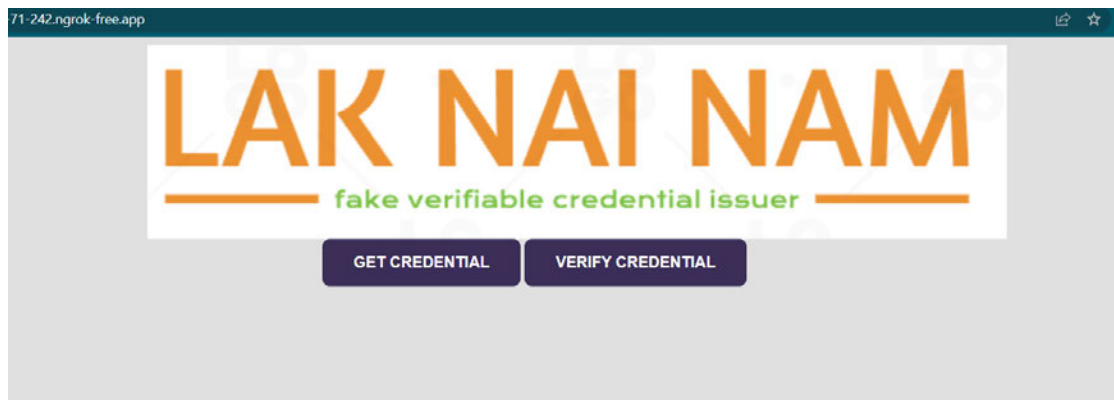


Figure 48; For this thesis, I created a dummy organization called “lak nai nam” to issue a verifiable credential to myself.

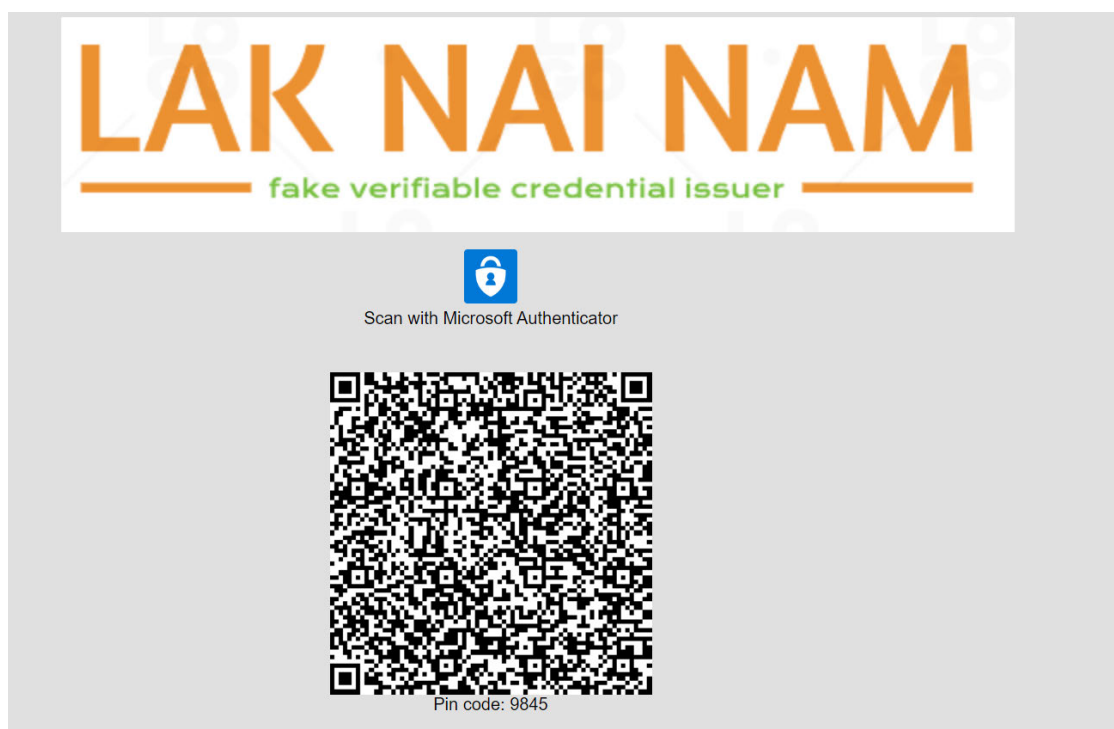


Figure 49; On the client side, the user simply has to download a Microsoft authenticator app and Scan a QR code to receive a verifiable credentials.

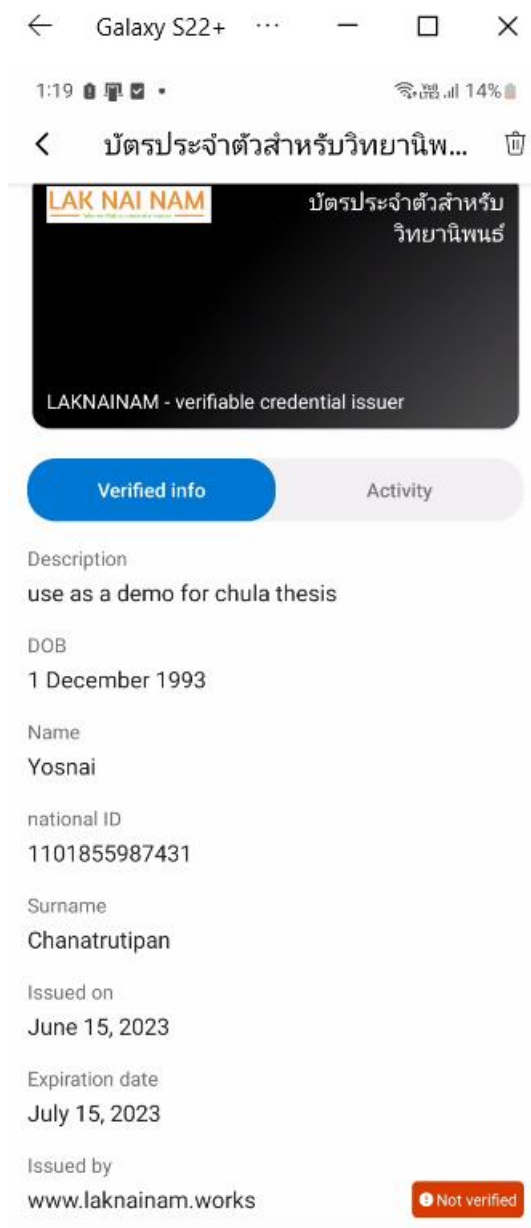


Figure 50, After the user scans the QR code, the verifiable credentials will be stored privately on their phone. This figure is an example of how a verifiable credential looks like on a phone (Microsoft authenticator app). Users can share this verifiable credential with anyone via a QR code.

The web application

The website is built using the popular web framework, React and NextJS. As a proof-of-concept web application, we will focus on the functionality of the website not how these libraries are used.

The website is divided into three main pages. The first page *SIGN* is where the users draft and sign a contract. The second page *VERIFY*, is where anyone can verify the integrity and authenticity of a signed contract. The third page *LEASE* is where landlord and tenant can see the status of the deployed smart contract.

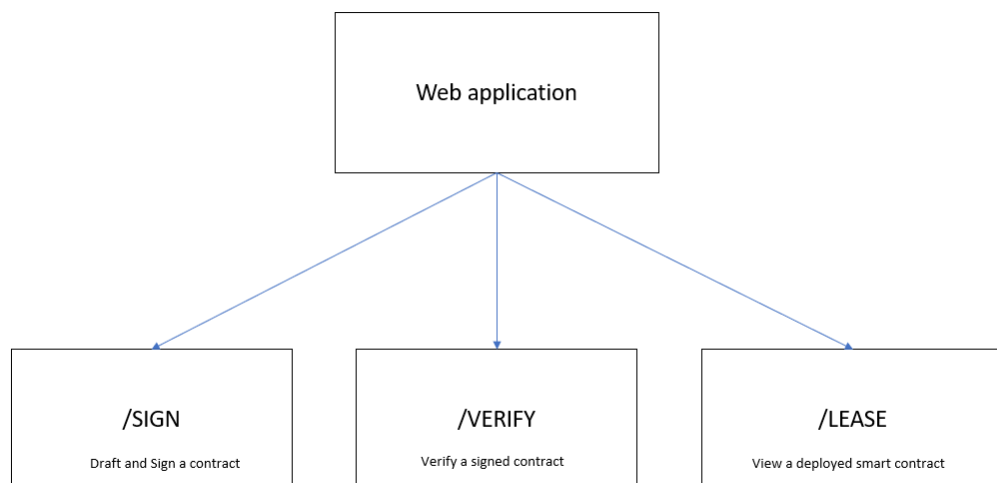


Figure 51, web application structure;

SIGN PAGE

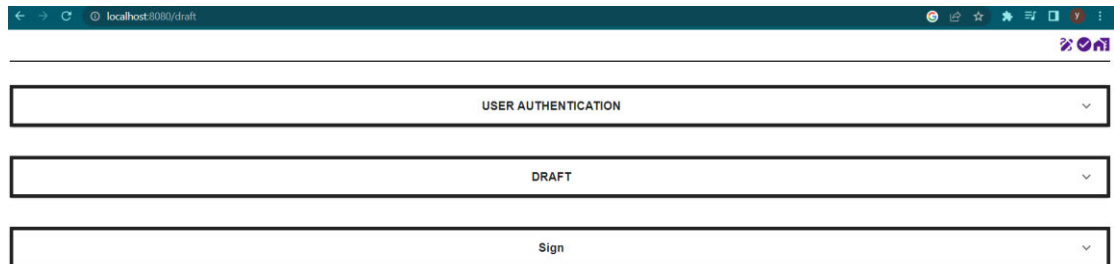


Figure 52 SIGN page; Users can authenticate each other in a user authentication tab. Draft tab allow user to draft a contract template. The sign tab is for users to fill in the contract and sign it.

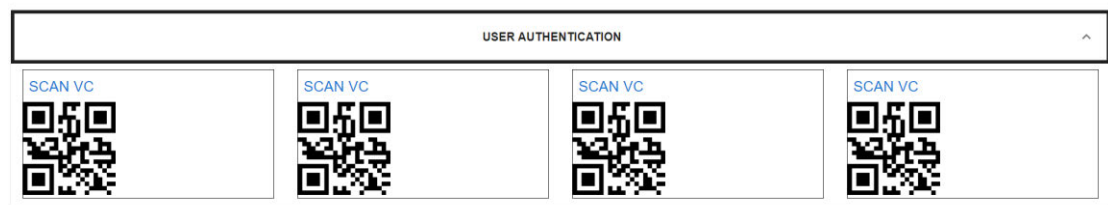


Figure 53, USER AUTHENTICATION TAB; allow user to scan a QR code to share a verifiable credentials, (the website will send a verifiable credential request to the user phone)

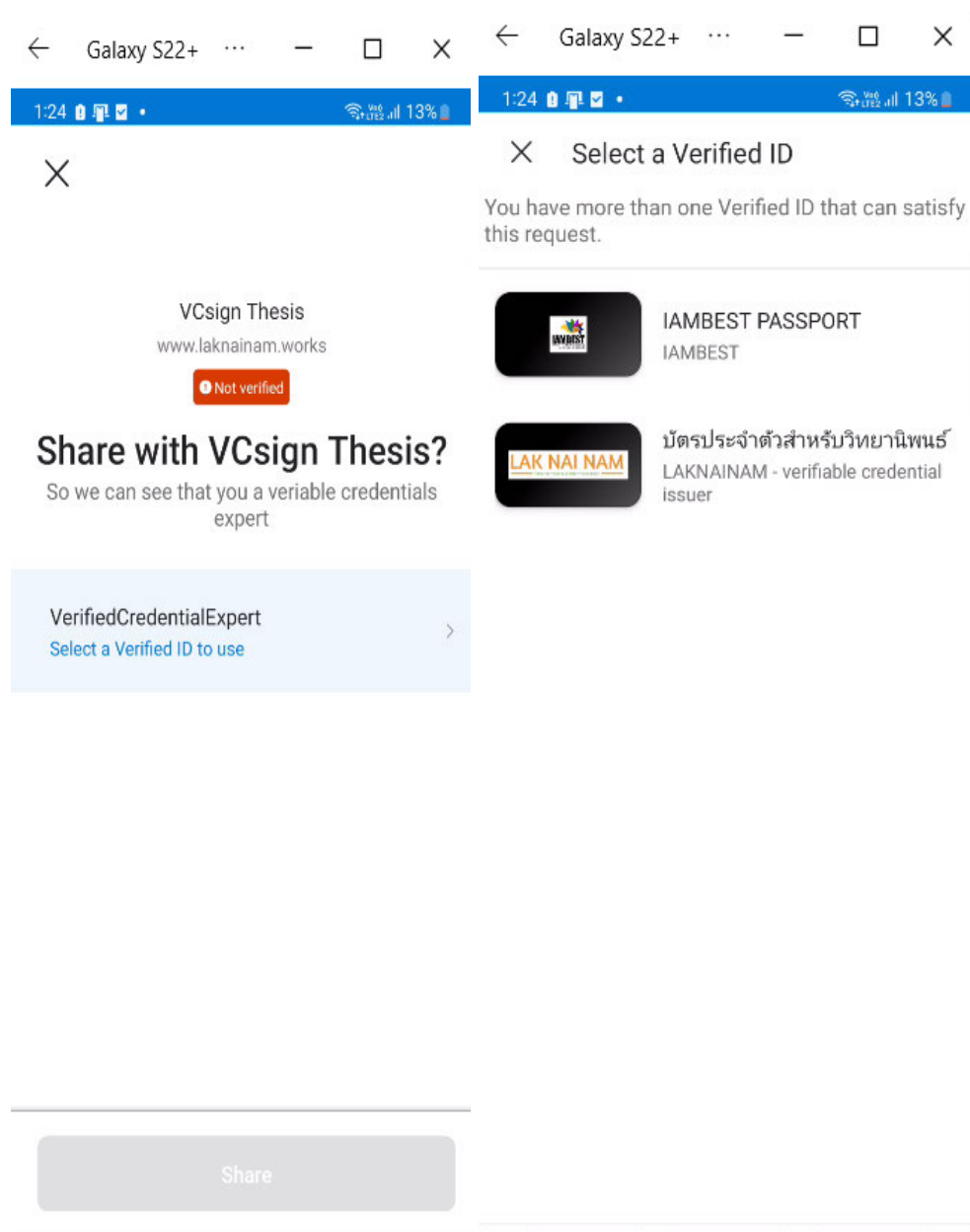


Figure 54, USER AUTHENTICATION; a mobile phone screen from a user perspective. The left picture show a credential request. If the users have more than 1 verifiable credentials, they have to pick one that fit the requirement. The web application is the one who decide which type of verifiable credentials is accepted.

Figure 55, USER AUTHENTICATION; The user successfully authenticated himself. If there are multiple signer, they can scan the other QR code. Once verified, the user can use these credentials obtained from a verifiable credentials to sign the contract.

Figure 56, DRAFT; allows users to draft any type of contract. The square bracket denotes a form that required a user input in a contract. There are two types of square brackets. The one ends with “:VC” and the one that doesn’t. VC symbol indicates that this data must be filled with verified information obtained from a verifiable credential. For example, $[ผู้ให้เช่า:VC]$ is the name of the landlord. The landlord must authenticate himself using a verifiable credentials first, only then he can fill in this information using a data from a verifiable credential he used to authenticate himself. For the square bracket that ends without VC, the website does not limit what the signer can fill in the form there.

หน้าชื่อสัญญาเช่า

โดยหนังสือฉบับนี้ ข้าพเจ้า **Yosnai** ซึ่งต่อไปในสัญญานี้เรียกว่า "ผู้ให้เช่า"

กับข้าพเจ้า **Chanatrutipan** ซึ่งมีบัตรประชาชนเลขที่ **1101855987431** เกิดวันที่ **1 December 1993** ซึ่งต่อไปในสัญญานี้เรียกว่า "ผู้เช่า"

ทั้งสองฝ่ายตกลงทำสัญญากันโดยมีข้อความต่อไปนี้ คือ

ข้อ 1. ผู้ให้เช่าตกลงให้เช่าและผู้เช่าตกลงเช่าบ้านโดยมีวัตถุประสงค์เพื่อการเป็นกรออยู่อาศัย มีกำหนดเวลาเช่า **15 ปี** เริ่มตั้งแต่วันที่ **9 สิงหาคม 2566**

โดยผู้เช่าตกลงให้ค่าเช่าแก่ผู้ให้เช่าเป็นรายเดือนละ **7000** บาท ส่วนเงินค่าน้ำหนักจากทรัพย์สินที่เช่านี้ ให้ผู้ให้เช่าเป็นผู้เสีย

ข้อ 2. ผู้เช่าได้ตรวจสอบทรัพย์สินที่เช่าแล้ว เห็นว่าทุกสิ่งอยู่ในสภาพเรียบร้อยใช้การได้อย่างสมบูรณ์จะดูแลทรัพย์สิน ที่เช่ามิได้ให้สูญหาย และบำรุงรักษาให้อยู่ในสภาพดีอยู่เสมอทรัพย์สินจะเสื่อมสลายตามสภาพเป็นไปตามการและตกลงยอมให้ผู้ให้เช่าหรือตัวแทนข้าพเจ้าตรวจสอบทรัพย์สินที่เช่าได้ทุกเวลาภายหลังที่ได้แจ้งความประสงค์ให้ผู้เช่าทราบแล้ว

ข้อ 3.

ผู้เช่าไม่มีสิทธินำทรัพย์สินที่เช่าออกให้ผู้เช่าหรือผู้เช่าจ้างทำนิติกรรมใดๆกับผู้เช่าในอันที่จะเป็นผลก่อให้เกิดความผูกพันกับผู้เช่าผู้เช่าไม่อาจโดยตรหรือโดยวิധาน ละจะไม่ทำการคิดแปลงหรือต่อเติมทรัพย์สินที่เช่าไม่ว่าทั้งหมดหรือบางส่วน เว้นแต่จะได้รับอนุญาตเป็นหนังสือจากผู้ให้เช่าและหากผู้เช่าได้ทำการคิดแปลงหรือต่อเติมสิ่งใดโดยที่ไม่ได้รับความยินยอมเมื่อใดแล้ว ผู้เช่ายอมยกกรรมสิทธิ์ในทรัพย์สินสิ่งนั้นให้ตกเป็นของผู้ให้เช่านับแต่เมื่อวันตัวหนังสือ

ข้อ 4. เมื่อผู้เช่าจะทำนิติสัญญาหรือหนังสือใด ผู้ให้เช่ามีสิทธิออกเลิกสัญญาได้ทันทีและผู้เช่าจะขอใช้ค่าธรรมเนียมกับค่าทนายความ ค่าทนายและค่าใช้จ่ายในการติดตามทวงถามให้แก่ผู้ให้เช่าจนครบถ้วนหากมีความเสียหายดังกล่าวเกิดขึ้นเพราะผู้เช่าเป็นฝ่ายผิดสัญญา

ผู้สัญญาได้อ่านและเข้าใจข้อความแล้วจึงลงลายมือชื่อไว้เป็นสำคัญต่อหน้าพยาน

signed by VCsingThesis
did:ion:EiAmFosp8PQlp4PftKVt5ZaC_gbcNg8tM6nDAQf4I4FA

Figure 59, A signed contract in PDF format. The green text represents an electronic signature extract from verifiable credentials. The red text is where the users just type in the data normally. The pdf is watermarked with the decentralized identity identifier (DID) of a web application that signed this contract.

ENFORCE WITH SMART CONTRACT ➤

Figure 60; The users can click “Enforce with smart contract” button to generate a smart contract that they wish to deployed on the blockchain.

กำหนดเงื่อนไขสัญญาอัจฉริยะ

×

วันเริ่มต้นสัญญา	<div style="border: 1px solid #ccc; padding: 5px;">วันเริ่ม</div> <div style="border: 1px solid #ccc; padding: 5px;">6/30/2023</div>
รายวัน / รายสัปดาห์/ รายเดือน / รายปี	<div style="border: 1px solid #ccc; padding: 5px;">ทุกๆ</div> <div style="border: 1px solid #ccc; padding: 5px;">30 days</div>
ระยะเวลาสัญญา (จำนวนครั้งที่ต้องจ่าย)	<div style="border: 1px solid #ccc; padding: 5px;">งวด</div> <div style="border: 1px solid #ccc; padding: 5px;">3</div>
ค่าเช่างวดละ	<div style="border: 1px solid #ccc; padding: 5px;">บาท</div> <div style="border: 1px solid #ccc; padding: 5px;">10000</div>
ค่าแรกเข้า	<div style="border: 1px solid #ccc; padding: 5px;">บาท</div> <div style="border: 1px solid #ccc; padding: 5px;">500</div>
จำนวนครั้งที่ขาดจ่ายได้สูงสุด	<div style="border: 1px solid #ccc; padding: 5px;">ครั้ง</div> <div style="border: 1px solid #ccc; padding: 5px;">0</div>
จ่ายล่วงหน้าก่อนเช่าพัก	<div style="border: 1px solid #ccc; padding: 5px;">ครั้ง</div> <div style="border: 1px solid #ccc; padding: 5px;">1</div>

หมายเลขอ้างอิง

thesis

SUBMIT

Figure 61; Customizing smart contract; The landlord can specify the term of the contract manually; the web application will use these data to automatically generate a smart contract and deploy it on the blockchain.

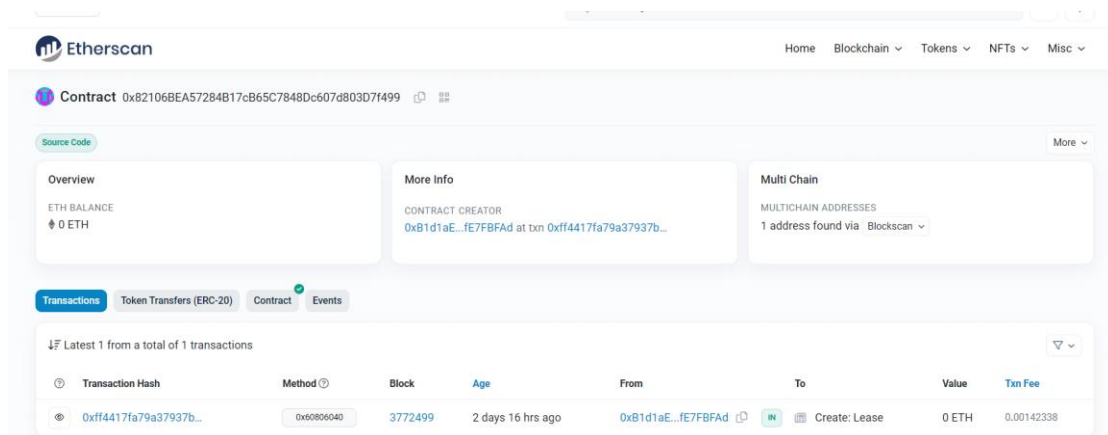


Figure 62; The deployed contract is publicly available on blockchain. In this figure, we use Etherscan, a website that allows users to query the Ethereum blockchain, to view the deployed contract details, information, and its history.

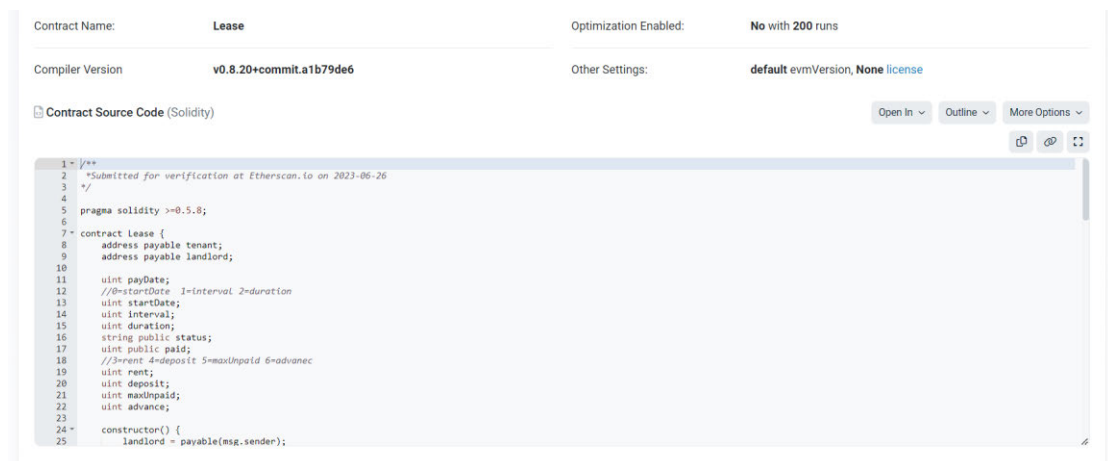
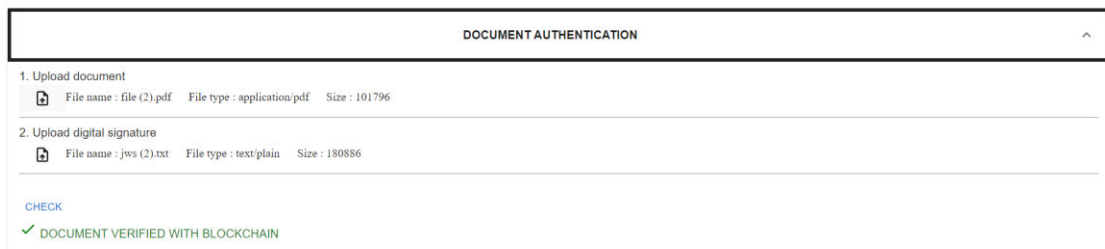


Figure 63, The smart contract code deployed on the blockchain as view from Etherscan; The code resides on the blockchain, being immutable and timestamped, offers a great way to make sure that the term of the contract can be enforce in a transparent and undeniable ways.

Document authentication Page

The signed contract can be verified and authenticated on this page. Any change or modification to the files will be detected by the website.



The screenshot shows a web interface titled "DOCUMENT AUTHENTICATION". It contains two sections for file uploads:

- 1. Upload document**: A file named "file (2).pdf" with a size of 101796 and file type "application/pdf" is shown.
- 2. Upload digital signature**: A file named "jws (2).txt" with a size of 180886 and file type "text/plain" is shown.

Below the uploads, there is a "CHECK" button. Below the button, a green checkmark and the text "DOCUMENT VERIFIED WITH BLOCKCHAIN" indicate a successful verification.

Figure 64, DOCUMENT AUTHENTICATION; user can upload the signed document and a digital signature to verify the integrity and authenticity of the signed document. Note that the users can verify it themselves with the blockchain without relying on the web application.

Lease Page

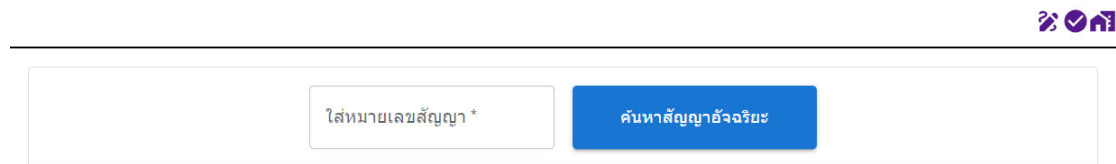


Figure 65; Lease Page, allows tenant and landlord to view and interact with the deployed smart contract.

Lease Agreement

Smart Contract Address : 0x4Abb7b161A8Bf1EA05033027F26e1dB698e1FFb0 [🔗](#)

INFO

Landlord = 0xB1d1aE061e066AE70E2cC794938a619fE7FBFAAd
 Tenant = 0x5095c1faA5CE2B8FE3615c05Bc475F0fCf5b80C9
 เริ่มต้นที่ = "2023-12-11T17:00:00.000Z"
 จ่ายทุกๆ 30 วัน เป็นจำนวน 2 ครั้ง
 ค่ามัดจำ 1 บาท (คืนให้หลังจบสัญญา)
 จ่ายล่วงหน้า 1 งวดก่อนเข้าพัก
 เงินไขผิดสัญญา : ค่าค่าเช่า 1 งวด

STATUS

สถานะ = complete ✓
 ระยะเวลาตามสัญญา : 2 รอบ
 จ่ายแล้ว : 2 รอบ

เริ่มสัญญา จ่ายค่าเช่า ตรวจสอบสถานะสัญญา ถอนเงินออกจากสัญญาอัจฉริยะ

Figure 66; The web application will read the state of that deployed smart contract, including all the variables inside it, and show it to the user as shown in the figure above.

The tenant can pay rent from this page. Payment with cryptocurrency is supported through a MetaMask, a web browser crypto wallet. Note that we do not have technology to pay with real money yet due to the reason stated in Chapter 4. The payment can be automated if the user transfer fund to the smart contract in advance.

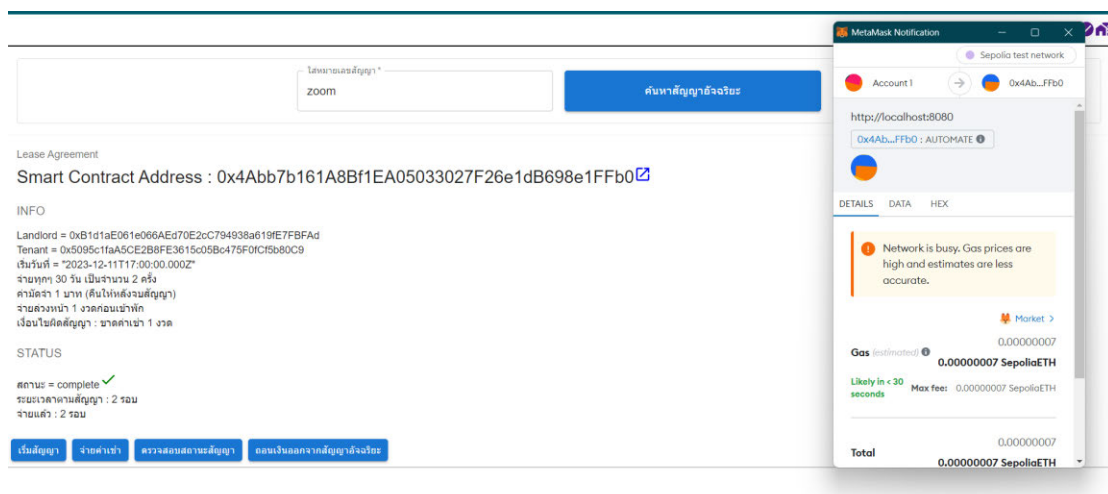


Figure 67, MetaMask is shown on the right-hand side of the figure. The tenants can pay rent directly from the web browser with a crypto wallet (not necessarily have to be MetaMask, there are many available crypto wallets to choose from)

The screenshot shows the Etherscan interface for a smart contract. The 'Overview' section displays the ETH balance as 0.000000007 ETH. The 'More Info' section shows the contract creator as 0xB1d1aE...fE7FBFAAd. The 'Multi Chain' section shows 1 address found via Blockscan. The 'Transactions' section is active, showing a list of 10 transactions. The table below summarizes these transactions:

Transaction Hash	Method	Block	Age	From	To	Value	Txn Fee
0x83483b8004f82285...	Pay_rent	3784923	1 day 1 hr ago	0xB1d1aE...fE7FBFAAd	0x817274...705aF93a	0 ETH	0.00004693
0x67b5665494b05b14...	Pay_rent	3784889	1 day 1 hr ago	0xB1d1aE...fE7FBFAAd	0x817274...705aF93a	0 ETH	0.00004699
0xadb71a87b1a801323...	Pay_rent	3777686	2 days 4 hrs ago	0x5095c1...Cf5b80C9	0x817274...705aF93a	0 ETH	0.00061341
0x4aa8f19d4934a6fe2...	Pay_rent	3777682	2 days 4 hrs ago	0x5095c1...Cf5b80C9	0x817274...705aF93a	0 ETH	0.0005093
0x2ea159b1673fa7406...	Pay_rent	3777682	2 days 4 hrs ago	0x5095c1...Cf5b80C9	0x817274...705aF93a	0 ETH	0.0005093
0xa9ff85c1ffd747ec6a...	LandlordWithdraw	3777679	2 days 4 hrs ago	0xB1d1aE...fE7FBFAAd	0x817274...705aF93a	0 ETH	0.0006124
0x4c5a8eba6d221efe8...	Pay_rent	3777628	2 days 4 hrs ago	0x5095c1...Cf5b80C9	0x817274...705aF93a	0 ETH	0.00043456
0x003ab60b9c616367...	Pay_deposit	3777618	2 days 4 hrs ago	0x5095c1...Cf5b80C9	0x817274...705aF93a	0 ETH	0.00044783

Figure 68, Interacting with the smart contract leave an immutable recorded on the blockchain that can be seen by anyone in the world. In the figure above we used Etherscan to track the smart contract usages, we can see a tenant paying rent and deposit with the smart contract method "Pay_rent" and "Pay_deposit", and see a landlord withdraw the money with "LandlordWithdraw" method.

Chapter 6 Conclusion and Future work

Signing a contract requires an electronic signature; The electronic signature can be created in many ways with different strength and weakness. In this thesis, we decide to use a blockchain to create these signatures. The signature created this way covers many weaknesses of the traditional signature. The blockchain also offers an identity management system. We use this to let users authenticate each other before they signed the contract. The term of the contract can be enforced by using a smart contract. All these features work seamlessly well with each other because they are all based on blockchain technology. The decentralized nature of the blockchain makes communication between entities more convenient and safer compared to the traditional way, where each entity must find the way to contact each other by their own means. In our method, the blockchain acts as a fair intermediary between each party.

Transferring real money with smart contracts is currently not possible and will continue to be so for a while. Most banks and financial institutes have been researching into blockchain for quite some time now. For example, Visa is developing an automated payment system on Ethereum. Mastercard has a crypto card program, that allows users to easily use their crypto. The gap between crypto and fiat money is becoming closer each year, soon it will be possible to enforce the term of an agreement through smart contract using real money instead of crypto currency.

From a technical viewpoint, our method can be seen as an upgrade in every way compared to the traditional document signing services. However, in terms of user experience, it has a major problem. The blockchain technology has not yet developed to the point where it is accessible to all demographics yet. The general population are not yet ready to use blockchain in their daily life. Therefore, we downgraded some of the blockchain's features to make it easier to use when we

designed a proof-of-concept web application. In our case, the users simply need a phone, to scan a QR code and sign a contract.

Here is the list of the possible future work that can improve this thesis.

- 1. Research into smart contracts from a non-Ethereum based blockchain.**

Other blockchain might offer smart contract functionality that is more suitable for our work than Ethereum.

- 2. Research into blockchain related projects from a financial institute**

Recurring payment or payment with a non-cryptocurrency are not natively supported by smart contract. Financial institutes are trying to incorporate blockchain into their services. These projects will contain a clue on how to solve these smart contract limitations.

- 3. Research into National Digital ID (NDID)**

NDID is a Thailand only version of a decentralized identity system. We should be able to use it to sign the contract in the same way too. NDID and DID implementation are completely different and are not compatible in any way but are based on the same principles.

NDID is connected to the Thai citizen database containing banking information and personal information. This can be used as an identity check before signing a contract without needing a verifiable credential. It might even be possible to build a smart contract on it that use Thai baht as a currency where the user can transfer fund to the smart contract with a mobile banking application.