Department of Computer Science

BACHELOR THESIS

Blockchain Identity Management for Cloud Service Providers

Nguyen Doan Anh Khoa, 9249

Assessor: Dr. Michel Toulouse

Co-Assessor: Dr. Lam-Son Le

January 20, 2020

Contents

List of Abbreviations

CA	Certificate Authority
DDoS	Distributed Denial of Service
DNS	Domain Name System
IAM	Identity and Access Management
IdP	Identity Provider
IPFS	InterPlanetary File System
OTP	One-Time Password
PIN	Personal Identification Number
PKI	Public Key Infrastructure
SP	Service Provider
ZKP	Zero-Knowledge Proof
Zk-SNARKs	Zero-Knowledge Succinct Non-interactive ARguments of Knowledge

1 Abstract

Identity management is and will be always a hot topic since the dawn of the Internet. Whereas traditional identity and access management systems are tested and in production for a long time, there are still security holes and inconvenience. On that account, this thesis tried to studied a new design which is more convenient for everyday use. I propose a new authentication method using zero-knowledge proof, together with the help of smart contracts in a blockchain. This new design keeps the same security standards as conventional identity management systems, while also provides transparency, and decentralization and especially more privacy for users' data. The implementation described in this paper also proved that my design is totally feasible, and although being just a proof-of-concept, the end result is very promising for future usage.

2 Introduction

We are living in a digital world, where digital technologies are fundamental for our everyday lives. With new rules, and new possibilities, they change the way we live, the way we work in a way that mankind has never experienced before. Therefore, in the era where everything is digitized and kept online, it is easy to understand that digital online services such as social media, healthcare, delivery, etc., all depend on user identities. The world needs something more sophisticated to identify a person, something more comprehensive than just a combination of username and password. Hence, the aim of this thesis is to study a new identity management system which offers a way to define an identity online based on attributes revolving them, called digital identity. The goal is to help users have an online identity that can act like a passport in the digital world. Unlike other identity management systems nowadays, my new proposed system enables service providers to make use of users' attributes, without them being revealed, i.e. users can totally keep their data secret. At the same time, all users' attributes need to be stamped by a trustworthy validator and stored in blockchain in form of hash and signature, preventing users from adding false claims. The whole process are executed in the main Ethereum network with the help of smart contracts. This way, along with the intrinsic properties of the blockchain, the system appears to be a secure and trustworthy identity management which also offers integrity, transparency, and decentralization.

The structure of this paper is organized as follows. Chapter 3 introduces some concepts and technologies that all should know in advance to understand the sys-

tem. Chapter 4 presents my proposed system in depth: how I came up with the idea as well as an illustration model for it. Chapter 5 describes a prototype and what I have achieved to make this design realistic. Chapter 6 evaluates the system in terms of security and performance, and makes a comparison with some other identity managements nowadays. Finally, chapter 7 concludes my work and lists down next steps for future development.

3 Background

3.1 Digital Identity

Digital identity is like our passport for the digital world. It can hold basic information portraying a person such as name, age, profession, health condition and can be used to identify a specific user. ISO/IEC 24760-1 defines identity as "set of attributes related to an entity"[2]. In that sense, digital identity is interpreted as information used by computer systems to express those set of attributes[39].

3.2 Identity Management

Identity management, also known as identity and access management (IAM), is a discipline for verifying the identity of a user and their level of access to a specific

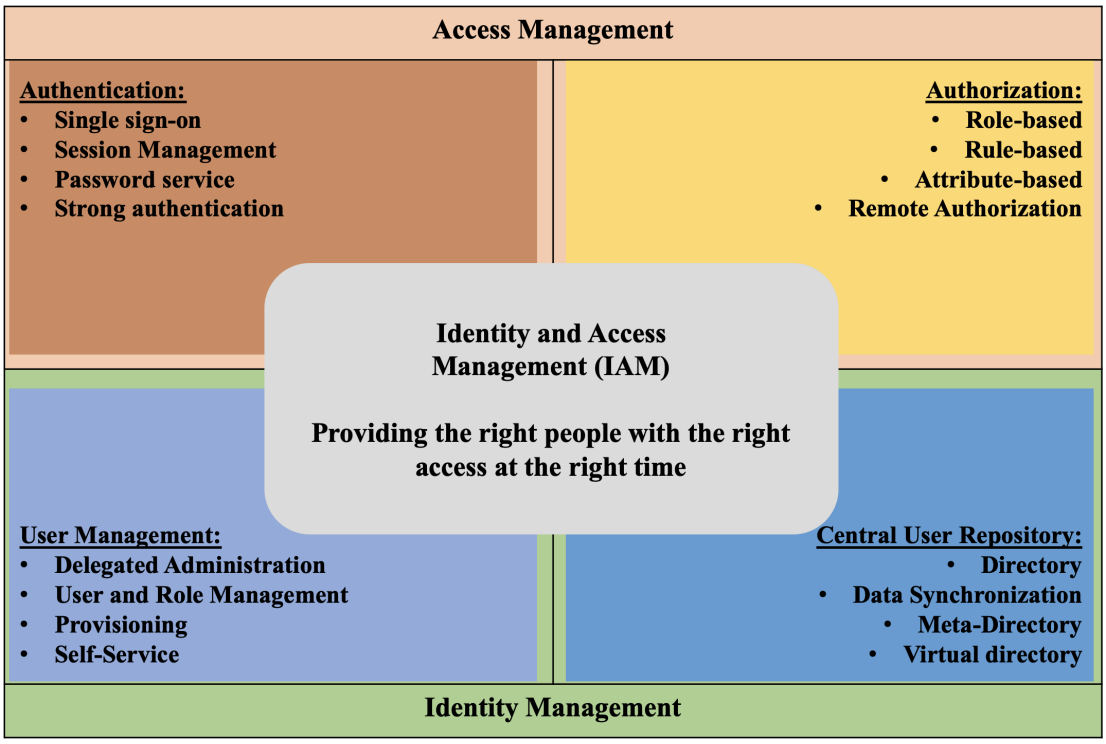

Figure 3.1: Identity and Access Management[33]

system. Both authentication and access management, which controls each user's level of access to a given system, play important roles in protecting user data[30]. The ultimate goal of an IAM is "to provide the right people with the right access at the right time". An IAM framework consist of four major sections: authentication, authorization, user management and central user repository (figure 3.1). Nowadays, IAM is still a hot topic to research on, and people are still finding for the best solution. However, how we define *best* is just really a trade-off between user experience and security.

3.3 User Authentication

User authenticate is a process of proving one's identity to another, who in our context could be an operating system, an application, or a third-party service in general. Not to be confused with authorization, authentication just care about "who you are?", or more precisely only to confirm that "you are really who you claim to be". Whereas, authorization is a process to give you roles and permissions, depending on your identity.

The new system I am going to propose only refers to authentication, and is a gateway to authenticate users for third-party services. How users are authorized, i.e. getting roles and permissions, we let each of services decide for their own. For this reason, from now on whenever I mention IAM, I only consider the authentication aspect of IAM.

There are several methods for one to prove his/her identity, which could be roughly categorized into three factors: authenticate with something you know, something you have, or something you are.

3.3.1 Something You Know

This is the most common and also the simplest method in our everyday use. *Something you know* could be a password, a personal identification number (PIN) or an answer to a secret question that only you supposed to know. However along with the simplicity, it is also the easiest method to crack.

3.3.2 Something You Have

It could be your smart phone, your email, or a smart card. Obviously you cannot prove to a third-party service your possession of a device by showing it to them every time. Instead, a token or an one-time password (OTP) is sent to the device whenever you need to login, and you prove your possession by send back that token or OTP. *Something you have* is more secure than using password, but it is also considered to be more inconvenient. Smart phone or smart card can easily get lost, and with OTP, you have to check your email every time.

3.3.3 Something You Are

This is recognized as the most secure and convenient way to authenticate when you do not have to remember anything or keep anything. *Some thing you are* use things that bio-metrically unique to every other person on this planet. These could include fingerprints, eyes, face, voice, or even the way one walks. However, this method still not common nowadays, because of its requirement of external hardware. You need a scanner to scan your fingerprint, and people will not just buy a fingerprint scanner for logging into Facebook.

3.4 Hash Function

Hash function is a math function used to generate dynamically data input into fixed-size values. Those values are called hash values, hash codes, digests, or simply hashes[41]. Hashes have some characteristics:

- A hash function always generates hashes of the same length. No matter what

size your input is, it always return a fixed-size hash. This is really important in terms of memory saving.

- A small change even in just a single bit of data will result in a whole different hash value. Therefore, you will be informed when a change in the original message happens.

- All hash functions are one-way functions, meaning that given a hash value, it is very difficult to reverse it back to the original message.

Also note that hash functions are not one-to-one functions. When two input strings generate the same hash results, it is called a *collision*. However, the odds of a collision are extremely low. Especially for a more stronger, and has larger output size like *SHA-256*[12], collision is not very a big problem for the foreseeable future.

3.5 Blockchain

Blockchain, which can be understood as digital ledger, was firstly designed by Stuart Haber and W. Scott Stornetta in 1991 and was actual conceptualized by Satoshi Nakamoto in 2008, whose identity is still a mystery[19]. To understand blockchain, we have to break down a few things:

- Transaction: this could hold any information, e.g. a record that says "Alice sent Bob 50 dollars"

- Block: a bundle of transactions

- Chain: blocks that are linked together

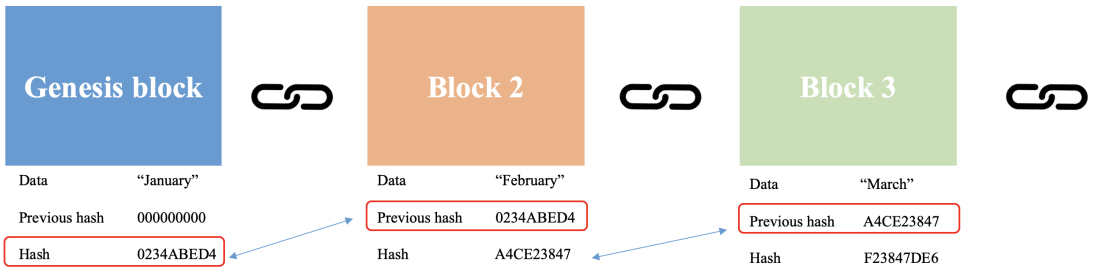

Figure 3.2: Blocks in a blockchain linked together through hash

Now let's take an use case where Alice sent 50 dollars to Bob and she wants to save that information into the chain. *Maryanne Murray* has an amazing explanation for how it works, described as follow[28]:

1. A trade is recorded as a transaction.

2. The transaction is checked by the network. The computers in the network, called 'nodes', validate the transaction by checking its details.

3. If accepted, the transaction is added to a block. Each block contains a hash value, and also a hash value of the previous block in the chain.

4. The hash values connect blocks together in a particular order, making the chain (figure 3.2).

This design makes blockchain hold these properties:

- Tamper-evidence: the way blockchain linked together via hash codes makes it almost impossible to modify. For example, a change in a single bit of data in block 2 leads to a different hash value. Block 3 still has the old hash of block 2, so it need to recalculate as well (as shown in figure 3.3 on the following page). That makes block 4 also need to be edited, and so on.

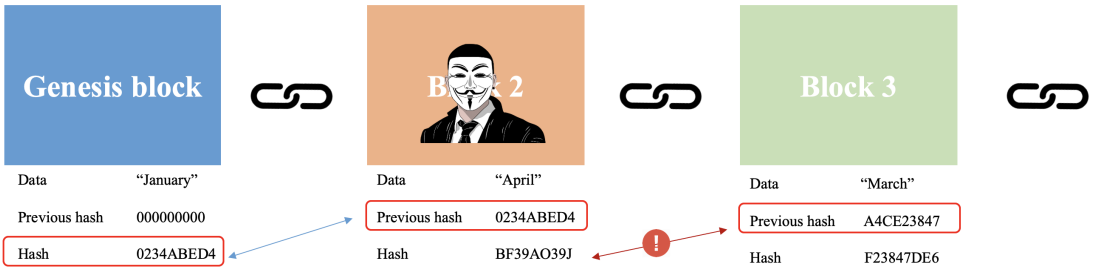

Figure 3.3: The modified block breaks the chain

Recalculating all those hashes in the chain requires an excessive amount of computing power and would be extremely difficult.

- Decentralization: blockchain operates on top of the Internet, on a peer-to-peer network of computers that all run the protocol. Therefore unlike traditional ledgers, blockchain does not need any central authority to keep the database, instead all nodes in the network hold an identical copy of the ledger of transactions. This is why blockchain is so hard to attack, as one has to manipulate 51% of nodes in the network in order to actually own the ledger.

- Transparency: in blockchain, the ledger is public and visible for everyone who can get access to the chain. This property makes blockchain a perfect solution for some fields such as tracing product origin in a supply chain.

3.6 Wallet

A crypto wallet is a computer program allowing to monitor and manage cryptocurrency. It is based on asymmetric cryptography which uses two types of keys: public key and private key. When you want to make a transaction, it is signed

with your private key, and then assigned to the address of receiver's wallet. The wallet address, which is generated from the public key, can also be used to uniquely identify users[21]. Lastly, the account balance of an address is calculated based on all transactions that it involved in in the ledger[8].

3.7 Ethereum

Ethereum is an open-source protocol of a distributed network based on blockchain technology. The intent of Ethereum is to build decentralized applications[9]. While *Bitcoin*[29] only aims to challenge online banking or PayPal alike, the goal of Ethereum is to replace services which is centralized, and make it decentralized. The cryptocurrency used in Ethereum is called *Ether* (ETH).

3.7.1 Smart Contract

Ethereum does this by supporting *smart contracts*, in which service providers can write code for their own set of rules, i.e. smart contracts can be seen as the brain, the logic of decentralized applications. Ethereum showed the ability of smart contracts by stating that a bare-bones version of Namecoin can be written in two lines of code, and other protocols like currencies and reputation systems can be built in under twenty[9]. Once a smart contract is deployed, it cannot be modified anymore and is stuck on the chain forever. The only way to make a change in your code is to deploy a new version of the contract, and the only way to delete it is to set a self-destruct condition beforehand[16]. Hence, we can see that smart contract is very transparent and secure, making it extremely trustworthy in business.

3.7.2 Solidity

Solidity is an object-oriented, high-level language for implementing smart contract, created and developed by Ethereum project. It is influenced by C++, Python, and JavaScripts[13], which makes learning solidity not a big problem if you have already been familiar with those languages. To compile solidity into bytecode, a compiler *solc* need to be installed. For the start, Solidity supports these basic data types[14]:

- Boolean: either true or false
- Integer: signed and unsigned integers of various sizes
- Address: used to store Ethereum address and hold a 20 byte value
- Fixed-size byte arrays: heximal value of various size, e.g. 32 bits or 256 bits
- Bytes: dynamically-sized byte array
- String: dynamically-sized UTF-8-encoded string
- Enum: used to create pre-defined values
- Mapping: basically a dictionary with a pair of key and value. It is declared as `mapping(_KeyType => _ValueType)`

Furthermore, these types can also be used in combination with C-like *Structs* to create more complex data structures. There are also three kinds of storage that Solidity has[15]:

- Storage: used to store contract state variables. It is persistent and quite expensive to use.

- Memory: used to hold temporary values. It only holds data until the execution of the function, and is cheaper to use.
- Stack: used to hold small local variables. It is almost free to use.

3.7.3 Data Storing Problem

One of the limitation of Ethereum and blockchain in general is the amount of data you can store. Most of the chains only allow you to store data in the range of kilobytes or less. Ethereum, which claims not to have a direct or fixed limit for transaction size nor for block sizes, still cannot send more than 1 megabyte data per transaction[4]. The reason is every transaction cost a certain amount of *gas* to execute, and the gas limit makes it impossible to store large data in Ethereum. One way to overcome this issue is that people only store a subset of data, a hash, or a file path link to the raw data on the chain. The raw data itself is stored off-chain by mechanisms such as traditional database, distributed database or distributed file system.

3.8 Distributed File System

Distributed file systems store their files across many machine. Together with traditional database and distributed database, distributed file system is an option for off-chain mechanism to store large amounts of data. The benefits it bringing is[25]:

- Be capable of storing large amount of data
- Make several copies of data, creating redundancy in case of a failure

- Transparency
- Decentralization

3.9 Digital Signature

3.9.1 Definition

Digital signature is a mathematical technique used to certify the authenticity and integrity of a message, software, email, text file,... or basically any electronic document[40], which is based on *Public Key Infrastructure* (PKI).

3.9.2 How It Works

In order to understand digital signature, we need to understand PKI first. It works by using two different cryptographic keys:

- Public key: openly visible for anyone
- Private key: known only to the owner

One interesting thing about PKI is that you can encrypt a message with public key and then use private key for decryption, or vice versa. Now take a use case when Alice want to sign a message and send it to Bob, the steps are described as follow:

1. Alice computes the hash value for her message
2. Alice uses her private key to encrypt the hash value of the message, creating a digital signature. This step is called *signing*

3. Alice send the signature along with the original message to Bob

4. *Verifying* steps are performed by Bob:

    - With the received message, Bob calculates the hash value. He get a hash value A
    - With the received signature, Bob decrypts it with Alice's public key. He now get a hash value B
    - Bob compares hash value A versus hash value B. If they match, the message is guaranteed to be untouched and evidently came from Alice.

3.10 Zero-knowledge Proof

3.10.1 Definition and Concept

Zero-knowledge proof (ZKP) is a cryptographic method allowing one party (*the prover*) to prove to another party (*the verifier*) that they know a secret parameter, called a *witness*, without exposing the *witness* itself to anyone. For example, ZKP can help one to prove that they are above 18 or not, without actually reveal their age. There are two types of ZKPs:

- Interactive: the prover need to perform a certain process of actions to convince the verifier, and this process need to be repeated for every other verifier.
- Non-interactive: the prover can generate a proof, and any verifier can verify this proof for themselves without the need of performing the process repeatedly.

A true ZKP need to have three properties[20][22]:

- Completeness: an honest prover can convince an honest verifier if the statement is true.
- Soundness: an dishonest prover cannot convince an honest verifier if the statement is false.
- Zero-knowledge: if the statement is true, nothing else is revealed to an dishonest verifier, except the fact that the statement is true.

To make it more concrete, Kostas Chalkias[10] has a really simple demonstration of how ZKP works. Imagine Alice has two balls, except for the fact that one is red and one is green, the balls are exactly identical. Alice (*the verifier*), who is color-blinded, is still not convinced that they are actually distinguishable and Bob (*the prover*)'s job is to convince Alice without revealing anything. Here is how to. First, she holds the balls in two hands, shows them to Bob, put the balls behind her back, and shows them to Bob again. Alice asks Bob: "Did I switch the ball?". Now if the balls are identical, and say after one hundred tries, Bob only has approximately 50 percent of guessing right. However, Bob, who is not color-blinded, indeed can determine if the balls are switched or not. By seeing that Bob can answer the question a hundred times correctly, Alice can rest assure that the two balls are really different in color.

3.10.2 Zk-SNARKs

Zk-SNARKs stands for *Zero-Knowledge Succinct Non-interactive ARguments of Knowledge*, which is a type of non-interactive ZKP. It is among the most popular zero-knowledge protocols that have been used. The idea of Zk-SNARKs is behind three algorithms[22]:

- Key generator G: takes a secret *lambda* and a *program* C as input. Then it returns a proving key pk and a verification key vk.
- Prover P: takes a public parameter x, a secret witness w, and the proving key pk as input. It will then return a *proof*, which should only be generated if the witness w satisfies the *program* C (by satisfying the proving key pk).
- Verifier V: takes the *proof* and the verification key vk as input. It will return true if the *proof* is valid and return false otherwise.

4 Proposed Architecture

4.1 Idea and Design

This new architecture based on the work of Jamila Alsayed Kassem et al. in *DNS-IdM: A Blockchain Identity Management System to Secure Personal Data Sharing in a Network*[6], but with some adjustments from my side for how data is stored. The DNS-IdM is a blockchain-based identity management system. With properties inherited from blockchain such as decentralization, transparency, it guaranteed to overcome some concerns in today's IAM solutions. E.g., the single point of failure or third party services abuse the trust granted by users. It emphasizes on attributes revolving around a person, and such information has the right to be controlled by

its owner. That includes the right to read, the right to write, and the right to revoke data off the database. However, data is only useful when a user is able to show it to an intended party. That is why the system allows service providers to read data off the database, but with minimal disclosure. DNS-IdM also offers a domain name system-like experience for identity management, hence the name. Because users have total power over their data, a validation process is needed to prevent false attributes and spam. Each new attribute to be added into the ledger, has first to be validated by an validator. What validator to be called is the job of the domain name system.

When I took a look at this system, my opinion is that while transparency bring benefits, it also has downside. Everyone who get accessed to the ledger can read all the information in it, i.e. the database using to store users' data is an open book. They had addressed this issue by using two separate ledgers: a permissionless ledger and a permissioned ledger, which can add an extra access control layer to only allow authorized users to do certain actions. However, I found a simpler approach to this by using zero-knowledge proof (ZKP). Figure 4.1 on page 23 illustrates my design. With ZKP, users do not need to reveal their data to prove their possession of data, so the raw data does not have to be stored in the blockchain. Therefore, instead of using two separate blockchain networks, my system needs only the main Ethereum network to operate. The ledger now only stores hash of the data, and its digital signature signed by the validator. Whenever a service provider (SP) wants to use any attributes from a user, the SP only need to prepare a program and wait for that user to compute a proof. The valid proof, the hash and the digital signature guarantees three things:

1. The hash guarantees data's integrity.

2. The proof guarantees user's knowledge of the hash, i.e. they know the pre-image of the hash, so they must be the data's owner.

3. The digital signature guarantees that data has already been validated by a validator.

Thus, not only properties like decentralization, transparency, and tamper-evident were kept, but my proposed system also avoided the hassle of building and maintaining a complex permissioned ledger.

4.2 Entities

- User: the actor, the one who has attributes.

- Identity Provider (IdP): a smart contract used to set and get users' attributes (in form of hash and signature) from blockchain.

- Domain Name System (DNS): a smart contract specifically designed for storing validators-related information. It helps users to know where to validate attributes as well as to get validators' public keys.

- Validator: is the one who will verify each attribute before it is actually saved in blockchain.

- Service Provider (SP): the third party application that users wish to authenticated with.

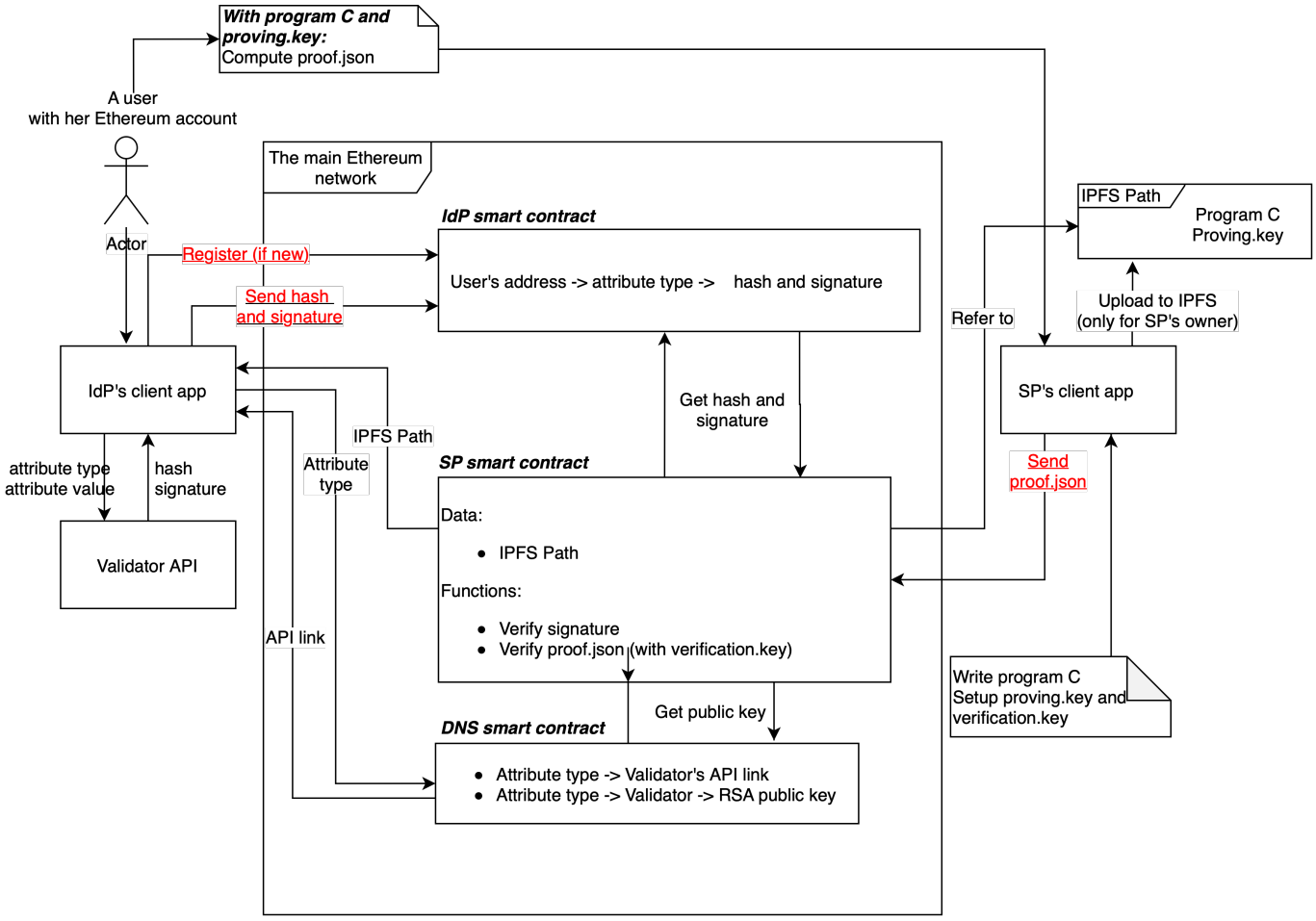

Figure 4.1: New proposed architecture model

As a side note, an *attribute* is understood as a property belong to a user. It consists of attribute type and attribute value. For example, *university* = *VGU* and *age* = *22* is two attributes belong to Alice.

5 Implementation

This section here is to describe how I implemented a prototype for my new proposed design. The prototype consists of an IdP, a DNS, two validators, and two SPs. Each validator has its own key pair and API link for validation. Meanwhile, each SP has its own ZKP program and smart contract.

Two validators include:

- E-Government: used to validate type *name* and *age* attributes
- Education Ministry: used to validate type *university* attributes

Two services include:

- Beer club: the beer club requires one to be over 18 to join the club. Therefore, a user need to have an attribute of type *age* and it also has to be over 18.
- Library: the library requires one to be a student to receive a discount. This can be verified by having an not null attribute of type *name* and type *university*.

5.1 Tools and Technologies

5.1.1 Ganache

Ganache is an application to simulate an Ethereum blockchain network, and offers an explorer for your ledger. It helps to deploy contracts, develop your applications, and run tests[17]. It is mainly used for development purpose, because using the main Ethereum network costs money to deploy contracts or make a transaction. Even when Ethereum offers a test network, it is also very slow and need a lot of effort to work with. With Ganache, you can have a number of accounts with a predefined amount of Ether for the start. You can also control the mining difficulty as you will to have faster transactions, making development become an ease of mind.

5.1.2 MetaMask

MetaMask is a cryptocurrency wallet, which can help to interact with any Ethereum network via your web browser[26], so I used it to interact with mine that created by Ganache. Like I mentioned before, my system works based on using cryptocurrency wallet to identify user, i.e. different user will have different wallet address (section 3.6). Therefore, by managing Ethereum private keys, MetaMask acts as a secure identity vault. This is considered to be very secure because the key storage is encrypted and stored locally on user's own browser[7], and it sign every transaction before added to the chain.

5.1.3 OpenSSL

OpenSSL is general-purpose cryptography library, and helps to secure communications over computer networks[44]. I need to generate a public key and a private key for each validator, so I used OpenSSL for this purpose. More specifically, the steps to generate a new key pair using *openssl* command line would be found in section 5.2.1.

5.1.4 InterPlanetary File System

InterPlanetary File System (IPFS) is a protocol and peer-to-peer network for storing and sharing data in a distributed file system[42], which is the reason why IPFS has no single point of failure. Similar to blockchain, there is a public network where everyone can store their files, but then unlike blockchain, not everyone has to store a copy of it. Thus, redundancy property of IPFS is considered enough[25]. IPFS files are content-addressed and can be uniquely identified by their hashes[5], which can then be easily stored within the blockchain. This makes IPFS a perfect solution for storing data when dealing with decentralized applications. After installed, IPFS needs to be run by fowlling commands:

```
# Run a network-connected IPFS node
$ ipfs daemon

# List peers with open connections
$ ipfs swarm peers
```

5.1.5 ZoKrates

ZoKrates is a toolbox for Zk-SNARKs on Ethereum[47]. It helps to define a *program* C in a high level language, compute a *proof*, and especially it also helps to generate a smart contract to verify the *proof* in Solidity. Basically, with ZoKrates you can run the whole process of Zk-SNARKs out of the box, without worrying about the actual algorithms laying within.

5.1.6 web3.js

web3.js is a collection of Ethereum JavaScript API which allow you to interact with a local or remote Ethereum node, using a HTTP or IPC connection[36]. I used web3.js to develop graphical interfaces to interact with my smart contracts. It helps to build a more friendly user experience as well as make it easier for testing.

5.2 Validators

5.2.1 Generating Key Pair

You would first need to install *openssl* for your favorite operating system, if you have not done so. The following are the command lines in MacOS for a key pair:

```
# Generate 512 bit Private key
$ openssl genrsa -out private.key 512

# Separate the public part from the private key file.
$ openssl rsa -in private.key -pubout > public.key
```

After generating a key pair, the public key should be registered into our *Domain Name System* list. More details will be discussed in section 5.4.2.

5.2.2 Signing

The signing process used the private key, and could only be provided by a validator itself (because only the validator knows its private key). In real life, the process should involve contacting an authority, and this authority should demand corresponding documents before signing. What documents are they totally depends on the purpose. For example, to verify your name and age, it may just be sufficient enough to scan your ID or your passport; while the education ministry need to scan your student ID as well. For simplicity, I assumed that the data provided are all scanned and verified by an authority beforehand. Therefore, my signing process just need an attribute value as input. I made two APIs using ASP.NET Core[27] to simulate the signing process, which can be found at `https://validators-thesis.azurewebsites.net/api/sign/egov?data=attribute-value` or `https://validators-thesis.azurewebsites.net/api/sign/ministry?data=attribute-value`. The IdP would call the former for *name* and *age* attribute type, while the latter is for *university* attribute type. One more thing to note is that the signing process used SHA-256 digest, and PKCS1 padding scheme[45].

5.2.3 Verifying

Given a digital signature and a public key, anyone can verify the signature for themselves. All you need to do is using the exact same hash digest function and padding scheme as signing, which in my case is SHA-256 and PKCS1. Thanks

to the work of *adria0*[3], the verifying process can be done by creating a smart contract in Solidity.

5.3 ZKP Programs for Service Providers

These steps are for each service provider:

1. Write a *program C* for ZKP. For example, here is a program written in ZoKrates to verify if user is over 18 or not:

```
// above18.zok
import "hashes/sha256/512bitPacked" as sha256packed

def main(private field ageDec, field h1, field h2) -> (bool):
    h = sha256packed([0, 0, 0, ageDec])
    h[0] == h1
    h[1] == h2
    bool flag = if ageDec >= 12600 then true else false fi //
        "18" => 0x3138 => 12600 in decimal
    return flag
```

2. Compile the program file, and setup a *proving key* and a *verification key*:

```
$ zokrates compile -i above18.zok # return file "out"

# with file "out"
$ zokrates setup # return a proving.key and a verification.
   key
```

3. Add the program file and the proving key onto IPFS, whose path will be saved in the service's smart contract (see section 5.4.3). Later on, users will

Figure 5.1: The IPFS path for above18.zok program and its proving key

access these files via an IPFS path (figure 5.1) to compute a *proof*:

```
# save all files to be added into a folder name 'dir'
$ ipfs add -r dir/
$ ipfs name publish $HASH_NAME_OF_THE_DIRECTORY
```

4. Auto generate a smart contract in Solidity to verify the proof:

```
# with verification.key
$ zokrates export-verifier # return verifier.sol
```

5.4 Public Ledger

5.4.1 Identity Provider Smart Contract

Data Storage

Users' data is stored by a mapping storage variable in the IdP contract. It is mapped from user's wallet address to a custom type in the form of `structs`, as shown in listing 5.1.

Registration

In order to join in and use the system, a user first need to be registered. This can be done by the IdP keep a list of users and turn `registered` variable into true if the function is invoked, as shown in the `register()` function (listing 5.2).

Read/Write Attributes

Only the hash and the signature of a attribute are stored in the system, so outsiders would not get any information out of the data anyway. That is why I make it in a way that everyone in the network can call the IdP contract to get data of any address (listing 5.4). In order to add a new attribute, however, a user has to be registered beforehand. This will limit unwanted contract call into our system. I did it by defining a modifier to terminate the contract immediately if the caller has not registered (listing 5.3). If the condition is met, the user can add a new attribute in form of hash and signature into our system, as shown in the listing 5.5.

```
1  mapping(address => Data) private clients;
2
3  struct Data {
4      mapping(string => Attribute) data;
5      bool registered;
6  }
7
8  struct Attribute {
9      bytes hash;
10     bytes signature;
```

```
11 }
```

Listing 5.1: Data structure to store users' data

```
1 function register() public {
2     require(!clients[msg.sender].registered, "Already registered")
          ;
3     clients[msg.sender].registered = true;
4 }
```

Listing 5.2: Register function

```
1 modifier onlyClient() {
2     require(clients[msg.sender].registered, "Not register");
3     _;
4 }
```

Listing 5.3: Contract will throw errors if a user has not registered

```
1 function getData(address addr, string memory attributeType) public
      view returns(bytes memory, bytes memory) {
2     return ((clients[addr].data)[attributeType].hash, (clients[
          addr].data)[attributeType].signature);
3 }
```

Listing 5.4: Get data from a specific address and attribute type

```
1 function add(string memory attributeType, string memory hash,
     string memory signature) public onlyClient {
```

```
2     (clients[msg.sender].data)[attributeType] = Attribute(fromHex(
          hash), fromHex(signature));
3 }
```

Listing 5.5: Add a new attribute

5.4.2 Domain Name System Smart Contract

I talked about how every attribute need to be validated before actually saved in the blockchain, but I have not mentioned about how the system knows which validator to contact for validation. We also need a place to store validators' public keys, so it can act as a trustworthy *Certificate Authority* (CA)[37]. Therefore, a smart contract named *DNS* was created for those purposes, having two main methods:

1. Input: an attribute type; output: validator's API link
2. Input: an attribute type; output: validator's public key

The smart contract was also designed in a way that only the owner of a public key can change its value. More specifically, if the Beer Club service was deployed by an account address say `0x123456789`, then only the account of address `0x123456789` can set and change Beer Club's public key in my DNS contract.

5.4.3 Service Provider Smart Contract

In order to successfully authenticated to a service, a user need to pass a two-step authentication. Therefore, a smart contract is created for each service:

- A storage variable to store the IPFS path of the ZoKrates program and its proving key. This will ensure that the program is truly written by the service

provider who user wishes to be authenticated with.

- A method to validate the digital signature: implemented by using a verifying method written by *adria0* which I have mentioned in section 5.2.3.
- A method to validate the proof: implemented by using the method in the smart contract generated in step 4 of section 5.3.

Only and only if both the digital signature and the proof are verified, the user will be added to a authenticated list. Next time she uses the service, the contract will recognize her as being authenticated, and she does not have to repeat the whole authentication steps again.

5.5 Client Apps by React

React is a JavaScript library for building graphical user interfaces. I used it together with web3.js to make client apps for the IdP as well as service providers. Basically it works like this: React is used to build the view and web3.js helps the view to communicate with the smart contracts in Ethereum. Figure 5.2 on the next page shows the interface for the IdP which has these functionalities:

- Show what address is currently using and check *registered* status
- Find a validator and validate a new attribute
- Add the new attribute into the IdP
- Get data back from the IdP

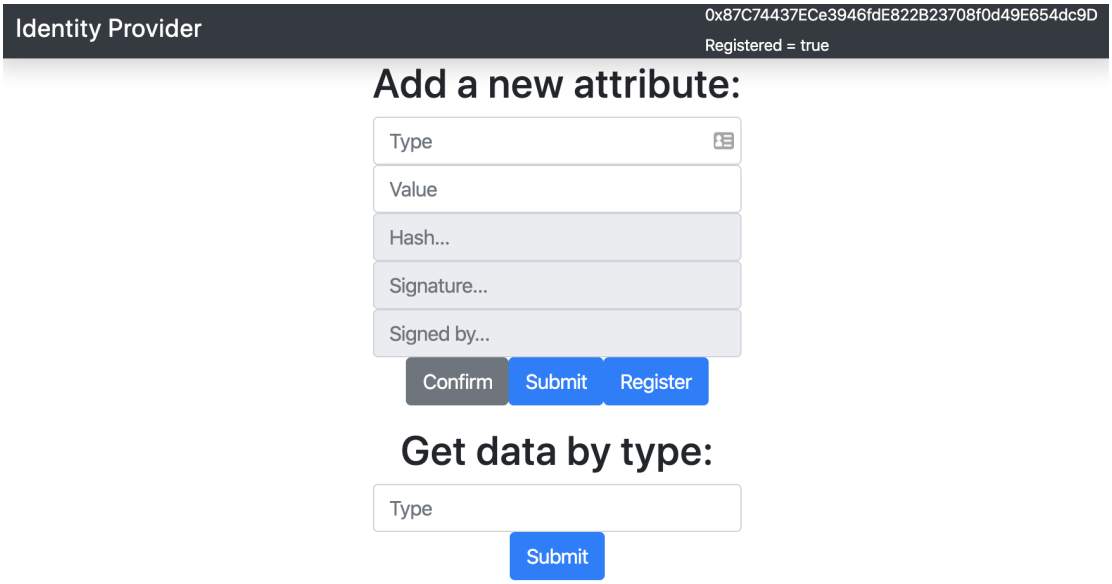

Figure 5.2: Client app for the IdP

Each service provider should also implement their own client app, so user can authenticate to such service. Essentially, the app should allow user to get the IPFS hash (for the ZoKrates program and proving key) and allow user to submit the proof (figure 5.3 on the following page).

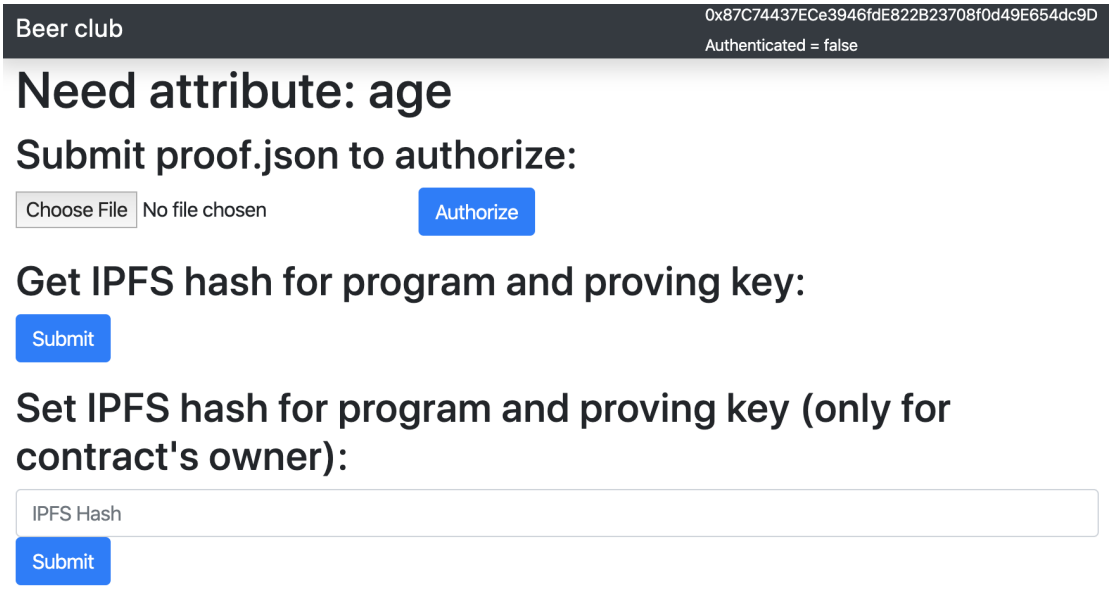

Figure 5.3: Client app for the Beer Club service

6 Evaluation

6.1 Security and Privacy

No Single Point of Failure

The system is a blockchain-based identity management, so identities are maintained on various number of nodes. Each node holds an exact copy of the ledger, helping the system to move away from centralization. Attacks which rely on a centralized server would no longer exist under my system. Take *Distributed Denial of Service* (DDoS) for an example. DDoS attack is a way for hackers to continuously send requests to an online service by making fake traffic from multiple sources[38]. The traffics eat up bandwidth and resources, eventually causing the service to shut

down. Now since blockchain is decentralized, and my system is blockchain-based, it can theoretically absorb DDoS attacks as they happen[24].

Preventing Phishing, Pharming, MITM and Malicious Third Party

In traditional systems, users has no way to tell if they are contacting to the right IdP or SP or not. That is why before blockchain technology, phishing[11] and pharming[18] are so easy to happen. People often tricked to enter their personal information to a fake website. By using smart contracts, this is no longer an issue, because each contract has a unique address and once deployed, the address is immutable. Therefore, every time users communicate with the IdP or any SP, they can know for sure who are they talking to by looking at the address of the receiver. Furthermore, my system does not include HTTP redirection, so phishing, pharming and man-in-the-middle attacks[23] are no longer exist. Even if users talk to a malicious party, they will still not get any meaningful information because of the ZKP. They would just get the hash and signature of the data, which users were willing to show publicly in the first place anyway.

Rightful Ownership

The proposed design truly makes users achieve a hundred percent of data ownership. In fact, the system does not restrict users from choosing a way to store their data. They can store it online, store it offline, or even they can just remember it. The point is, they only need to be able to prove their knowledge of the data when asked. This is what makes my design different from the initial idea, the DNS-IdM. The latter relies on a permissioned ledger to add and maintain attributes. They did not clarify exactly how they create that ledger, but from my experience, any-

thing dealing with many access control layers always exposes some security holes. Moreover, we all know that once something is posted online, you basically do not have control over it anymore. Therefore, the simplest solution is not to post it online and that is exactly what we can achieve with ZKP and the system.

Preventing False Claim

Learning from the DNS-IdM, my system also has a mechanism to validate attributes. It is a new feature that hardly be seen in any traditional systems nor in any blockchain-based identity management that has been developed such as uPort[34], or Sovrin[31]. This feature is extremely important for enhancing security, maybe not for users but for service providers. For example, if Alice, who is only fifteen years old, wants to join the Beer Club service. Knowing that the lower limit for joining a beer club is eighteen, Alice tried to make false claim about herself, and telling the system that she is above eighteen. This is impossible, because thanks to attribute validation, Alice will not provide sufficient documents to prove as such. Also, public keys of trustworthy validators are stored inside the DNS contract, and SPs only trust these validators. Attributes signed by other unverified parties would be simply rejected.

6.2 Performance

One thing need to be considered when using blockchain technology is pricing, so I will evaluate the performance by analyzing the cost of the system. First of all, it is necessary to introduce the concept of *mining* and *gas*. In blockchain, like I have mentioned in section 3.7.3, every transaction need to be validated

and confirmed by special nodes in order to be added to a block. This process is called mining, and those special nodes are called miners[8]. For each successfully confirmed transaction, the miner would get a reward for completing the work by the transaction sender. The reward is calculated based on *gas* and *gas price*. Gas is a unit to measure the effort done by a miner[32], and gas price is the cost for a unit of gas. The higher the gas price, the higher the reward, which means the higher its chance of getting processed by the network faster. Let evaluate some gas price during normal times[1]:

- 40 GWEI will almost always get a transaction into the next block
- 20 GWEI will usually get a transaction within the next few blocks
- 2 GWEI will usually get a transaction within the next few minutes

For example, if a transaction needs 100,000 unit of gas and the sender chooses 20 GWEI for gas price. It would cost 20 GWEI * 100,000 = 2,000,000 GWEI = 0.002 ETH, or by the time of this paper, approximately 0.29 USD.

In my system, there are only three contract calls that involving in creating a new transaction, which costs Ether. Three of such includes: register, add a new attribute, and send a proof (underlined texts in figure 4.1 on page 23). The other contract calls do not alter contract's state, so it do not cost you any money. For the exchange rate of 1 ETH = 143.53 USD (12/01/2019), I analyzed the price for the three contract calls when using gas price of 20 GWEI (table 6.1) as well as other gas prices (table 6.2). As you can see for under 20 GWEI, registration and adding a new attribute only cost under a dollar per a call. By using hash and signature, which sizes are the same for whatever the data is, we can rest assure that the

Table 6.1: The cost to create a new transaction when using 20 GWEI gas price

	Register (Gas: 63,186)	**Add a new attribute (Gas: 327,912)**	**Send a proof for above18.zok (Gas: 6,385,876)**
ETH	0.001264	0.006558	0.127718
USD	0.18142192	0.94126974	18.33136454

Table 6.2: The cost to create a new transaction with different gas prices

	Register (Gas: 63,186)	**Add a new attribute (Gas: 327,912)**	**Send a proof for above18.zok (Gas: 6,385,876)**
40 GWEI	0.36284384	1.88253948	36.66272908
20 GWEI	0.18142192	0.94126974	18.33136454
2 GWEI	0.018142192	0.094126974	1.833136454

cost is always the same when adding a new attribute. Besides, users need only to register once. The only problem is the price to send a proof, and it totally depends on the designed programs and the amount of information a service provider need in order to authenticate users. When using the Beer Club's program (section 5.3), the cost to send a proof is around 18 USD, which is way too high for the gas price of 20 GWEI. However, if we sacrifice some time by using 2 GWEI instead, the cost would reduce to approximately 1.8 USD. It is somehow acceptable when considering users only need to authenticate once for every service provider.

6.3 Comparison

In the end, we can see that all identity and access management systems nowadays have some vulnerabilities, either this or that. Let's take a look at some of existing categories of IAM, and we can see that my proposed system has overcame all of their problems.

Centralized IAM

Centralized IAM relies on a single entity to manage identities:

- It deprives users of their rightful ownership of their identity online: users need to trust the IdP completely to keep their data.
- IdP can act as a single point of failure: data and the whole process of authentication rely heavily on the IdP. This could make it an vulnerable target for hackers to exploit. Once the IdP is defected, the whole system is also down.
- SP abuses the trust granted by users: SP can only retrieve attributes of users from the IdP with their consent. However, mischievous SP could manipulate this process and make users gain permission to get some additional data without their knowing.

Decentralized IAM

Decentralized IAM such as OpenID[43][35] does not rely on a single entity to handle identities, but it often includes HTTP redirection in order to exchange credentials. This action opens up many security flags such as phishing, pharming, or man-in-the-middle attacks, like I have mentioned in section 6.1.

7 Conclusion and Future Work

7.1 Conclusion

Identity management has come a long way for serving as a secure gateway for digital services, and when our lives nowadays more and more depend on these, the need of a sophisticated and trustworthy IAM is more crucial than ever. A combination of username and password are simply not enough to describe one's identity anymore, making conventional IAM systems lose their efficiency and convenience.

For that reason, this thesis has proposed a new way for user authentication that relies on identity attributes. By introducing the DNS system, these attributes are all validated by corresponding authorities before added. This way, they are guaranteed to be one-hundred percent authentic and secure. I also introduced the concept of ZKP, which in my opinion is what makes my design better and different from existing identity management systems. ZKP is the only solution for service providers to make use of users' attributes, without users revealing them. Furthermore, to overcome the drawback brought by centralization, my system used blockchain to manage users' identity attributes and deployed it in a public blockchain network. With the help of smart contracts, the system provides a seamless user experience, while still hands over top-most security and privacy for users at the same time.

In this paper, I also evaluated the new system in terms of security, privacy, performance, as well as made some comparison with existing IAM systems. It still needs a lot of steps forward of researching before this design reaches a certain

level of usability. However, along with the development of blockchain, I believe a mechanism which represents one's identity by their attributes like this will soon become a standard in just near future.

7.2 Future Work

In the end, this is just a prototype and there are still a lot of other tools and technologies to be considered.

Firstly, I chose ZoKrates because of its simplicity, but for real production purpose, it is maybe not a good solution for Zk-SNARKs. The size of the proof generated by ZoKrates programs is somewhat still a little to big, even for a simple program to check users' age like the Beer Club's. It makes the cost for transaction mining be acceptable for this demo, but a little inefficient for production.

Secondly, one problem for my system's design is that hash result will be exactly the same for users sharing the same attribute's values. For instance, Alice and Bob, who are both 22 years old, would get the same hash value for their `age` attribute type. Hence, the next time Alice see Bob's hash value on blockchain, she will be immediately able to realize Bob's real age. One obvious solution to this is using *salt*, which is a non-secret, random value that is added together with the original input[46]. This way we can ensure that that the same input will not consistently hash to the same result.

In addition to this, the system has not yet been tested with the main Ethereum main network, or any real cloud infrastructure. Since the objective of this thesis was just to develop and test basic proof-of-concept, all I did was merely on local. Therefore, the price listed in section 6.2 are also just for reference only.

After all, blockchain has not been here for so long. It is still a new technology that needs to be tested and analyzed properly.

Bibliography

[1] MyEtherWallet (MEW). *What is Gas?* URL: https://kb.myetherwallet.com/en/transactions/what-is-gas (visited on 01/10/2020).

[2] ISO/IEC 2019. *IT Security and Privacy — A framework for identity management - Part 1: Terminology and concepts*. 2nd ed. 2019, p. 1.

[3] adria0. *Solidity RSA Sha256 Pkcs1 Verification*. 2019. URL: https://github.com/adria0/SolRsaVerify (visited on 01/05/2020).

[4] Afr. *Is there a limit for transaction size?* 2016. URL: https://ethereum.stackexchange.com/questions/1106/is-there-a-limit-for-transaction-size (visited on 01/12/2020).

[5] Muhammad Salek Ali, Koustabh Dolui, and Fabio Antonelli. "IoT Data Privacy via Blockchains and IPFS". In: (2017).

[6] Jamila Alsayed Kassem et al. "DNS-IdM: A blockchain identity management system to secure personal data sharing in a network". In: *Applied Sciences* 9.15 (2019), p. 2953.

[7] BitDegree. *MetaMask Wallet Review*. 2020. URL: https://www.bitdegree.org/tutorials/metamask-wallet-review (visited on 01/06/2020).

[8] Eric Borgsten and Oskar Jiang. "Authentication using Smart Contracts in a Blockchain". MA thesis. 2018.

[9] Vitalik Buterin et al. "Ethereum white paper.(2013)". In: *URL https://github.com/ethereum/wiki/wiki/White-Paper* (2013).

[10] Kostas Chalkias. *Demonstrate how Zero-Knowledge Proofs work without using maths*. 2017. URL: https://www.linkedin.com/pulse/demonstrate-how-zero-knowledge-proofs-work-without-using-chalkias (visited on 12/30/2019).

[11] Ali Darwish, Ahmed El Zarka, and Fadi Aloul. "Towards understanding phishing victims' profile". In: *2012 International Conference on Computer Systems and Industrial Informatics*. IEEE. 2012, pp. 1–5.

[12] D Eastlake III and T Hansen. "RFC 4634-US Secure Hash Algorithms (SHA and HMAC-SHA)". In: *Motorola Labs and AT &T Labs* (2006).

[13] Ethereum. *Solidity*. URL: `https://solidity.readthedocs.io/en/v0.5.0/` (visited on 01/01/2020).

[14] Ethereum. *Solidity in Depth - Types*. URL: `https://solidity.readthedocs.io/en/v0.5.0/types.html` (visited on 01/01/2020).

[15] Forest Fang. *Ethereum Solidity: Memory vs Storage and When to Use Them*. 2018. URL: `https://medium.com/coinmonks/ethereum-solidity-memory-vs-storage-which-to-use-in-local-functions-72b593c3703a` (visited on 01/06/2020).

[16] Donn Felker. *Self destructing smart contracts in Ethereum*. 2018. URL: `https://articles.caster.io/blockchain/self-destructing-smart-contracts-in-ethereum` (visited on 01/01/2020).

[17] Ganache. URL: `https://www.trufflesuite.com/ganache` (visited on 01/06/2020).

[18] Manish Gupta. "Pharming attack designs". In: *Encyclopedia of Information Ethics and Security*. IGI Global, 2007, pp. 520–526.

[19] ICAEW. *History of blockchain*. URL: `https://www.icaew.com/technical/technology/blockchain/blockchain-articles/what-is-blockchain/history` (visited on 12/30/2019).

[20] Tommy Koens, Coen Ramaekers, and Cees Van Wijk. *Efficient Zero-Knowledge Range Proofs in Ethereum*. Tech. rep. Technical Report, 2018.

[21] Yi Liu et al. "An efficient method to enhance Bitcoin wallet security". In: *2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. IEEE. 2017, pp. 26–29.

[22] Christian Lundkvist. *Introduction to zk-SNARKs with Examples*. 2017. URL: https://media.consensys.net/introduction-to-zksnarks-with-examples-3283b554fc3b (visited on 12/30/2019).

[23] Avijit Mallik et al. "Man-in-the-middle-attack: Understanding in simple words". In: *International Journal of Data and Network Science* 3.2 (2019), pp. 77–92.

[24] Rowan Marley. *How Blockchain Can Fight DDoS Attacks*. 2017. URL: https://vocal.media/theChain/how-blockchain-can-fight-ddos-attacks (visited on 01/10/2020).

[25] Lukas Marx. *Storing Data on the Blockchain: The Developers Guide*. 2018. URL: https://malcoded.com/posts/storing-data-blockchain/#using-transactions-to-store-data-on-the-blockchain (visited on 01/06/2020).

[26] MetaMask. URL: https://metamask.io (visited on 01/06/2020).

[27] Microsoft. *Introduction to ASP.NET Core*. 2019. URL: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1 (visited on 01/05/2020).

[28] Maryanne Murray. *Blockchain explained*. 2018. URL: http://graphics.reuters.com/TECHNOLOGY-BLOCKCHAIN/010070P11GN/index.html (visited on 12/30/2019).

[29] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. Manubot, 2019.

[30] Okta. *What Is Identity Management and Access Control?* URL: https://www.okta.com/identity-101/what-is-identity-management-and-access-control (visited on 12/30/2019).

[31] Sovrin. URL: https://sovrin.org (visited on 01/10/2020).

[32] TANYA. *What is gas?* 2019. URL: https://support.blockchain.com/hc/en-us/articles/360027772571-What-is-gas- (visited on 01/10/2020).

[33] Hong Kong Polytechnic University. *Identity and Access Management (IAM)*. URL: https://www.polyu.edu.hk/ags/Newsletter/news0911/IAM_details.html (visited on 12/30/2019).

[34] uPort. URL: https://uportlandia.uport.me (visited on 01/10/2020).

[35] Bart Van Delft and Martijn Oostdijk. "A security analysis of OpenID". In: *IFIP Working Conference on Policies and Research in Identity Management*. Springer. 2010, pp. 73–84.

[36] Fabian Vogelsteller et al. *web3.js Documentation*. 2019. URL: https://web3js.readthedocs.io/en/v1.2.4 (visited on 01/08/2020).

[37] Wikipedia. *Certificate authority*. 2019. URL: https://en.wikipedia.org/wiki/Certificate_authority (visited on 01/06/2020).

[38] Wikipedia. *Denial-of-service attack*. 2019. URL: https://en.wikipedia.org/wiki/Denial-of-service_attack (visited on 01/10/2020).

[39] Wikipedia. *Digital identity*. 2019. URL: https://en.wikipedia.org/wiki/Digital_identity (visited on 12/30/2019).

[40] Wikipedia. *Digital signature*. 2019. URL: https://en.wikipedia.org/wiki/Digital_signature (visited on 12/30/2019).

[41] Wikipedia. *Hash function*. 2019. URL: https://en.wikipedia.org/wiki/Hash_function (visited on 12/30/2019).

[42] Wikipedia. *InterPlanetary File System*. 2019. URL: https://en.wikipedia.org/wiki/InterPlanetary_File_System (visited on 12/30/2019).

[43] Wikipedia. *OpenID*. 2019. URL: https://en.wikipedia.org/wiki/OpenID (visited on 01/12/2020).

[44] Wikipedia. *OpenSSL*. 2019. URL: https://en.wikipedia.org/wiki/OpenSSL (visited on 12/30/2019).

[45] Wikipedia. *PKCS 1*. 2019. URL: https://en.wikipedia.org/wiki/PKCS_1 (visited on 01/05/2020).

[46] Wikipedia. *Salt (cryptography)*. 2019. URL: https://en.wikipedia.org/wiki/Salt_(cryptography) (visited on 01/10/2020).

[47] ZoKrates. *Introduction*. URL: https://zokrates.github.io (visited on 01/03/2020).