electronics

MDPI

Article

VaultPoint: A Blockchain-Based SSI Model that Complies with OAuth 2.0

Seongho Hong and Heeyoul Kim *

Department of Computer Science, Kyonggi University, Suwon 16227, Korea; elHongsh@gmail.com
* Correspondence: heeyoul.kim@kgu.ac.kr

Received: 10 July 2020; Accepted: 29 July 2020; Published: 31 July 2020

Abstract: An identity management including authentication and authorization in a network environment is a critical security factor. Various models for identity management have been developed continually, from the silo model to the federated model and to the recently introduced self-sovereign identity (SSI) model. In particular, SSI makes users manage their own information by themselves independently of any organizations. SSI utilizes the newly emerged blockchain technology and many studies of it are in progress. However, SSI has not had wide public use because of its low compatibility and inconvenience. This is because it involves an unfamiliar user experience and an immature process. To solve this problem, this paper proposes a new blockchain-based SSI model that complies with the popular and mature standard of OAuth 2.0. Using blockchain, the proposed model secures users' data sovereignty where users can use and control their own information in a decentralized manner, instead of depending on a specific monopolistic service-providers. Users and clients who are familiar with the existing OAuth can easily accept the proposed model and apply it, which makes both usability and scalability of the model excellent. This paper confirmed the feasibility of the proposed model by implementing it and a security analysis was performed. The proposed model is expected to contribute to the expansion of both blockchain technology and SSI.

Keywords: self-sovereign identity; blockchain; OAuth; authentication; authorization

1. Introduction

Identity management models to authenticate and authorize users in the Internet environment have been continually developed by addressing the problems of the existing models. In the early silo model, individual service providers possessed user information and directly authenticated the users. The silo model was limited, however, because only the service provider who possessed the user data could perform authentication. This led to the problem of password fatigue among users as Internet services became increasingly diverse.

To solve the problem of the silo model, the federated model was developed to perform authentication by delegating authentication to a certain service. The federated model was implemented in various forms [1,2]. One of them was the single sign-on [3], where the delegated authentication server processes all the authentication in a single network based on the SAML (Security Assertion Markup Language) protocol. Another approach was OAuth, where multiple third-party services delegate the authentication and authorization to a certain service, such as Google and Facebook, based on HTTP. The federated model helped to reduce password fatigue, but the authentication service came to possess huge amounts of user data. This created management and security issues [4] because the service may violate privacy of users by abusing user data it possesses [5]. Another problem is that those third-party services cannot work properly if there is a temporary failure or a permanent suspension of the authentication service.

The user-centric model [6] was created as a way to give data sovereignty to users and solve the problems of previous models. There was a representative service named OpenID [7] whose wide dissemination failed because of the unfamiliar concept of its authentication process [8]. Afterwards, several authentication services similar to OpenID were created. However, most of those services were not very different from the federated model and they had weak public appeal. With their vulnerability to phishing attacks, the range of use was largely restricted [9].

The self-sovereign identity (SSI) model, which was made possible by the emergence of blockchain technology, solved the existing shortcomings while achieving the same goal as the user-centric model [10]. The problem of data reliability of the existing OpenID was solved by the transparent and consistent characteristics of blockchain [11,12]. Uport [13] and Sovrin [14] are two representative blockchain-based SSI models. Efforts are being made to standardize the SSI model, such as discussions of the decentralized identifier (DID) [15] in W3C (World Wide Web Consortium). However, many problems must be solved to enable dissemination of the SSI model. Each SSI model has its own process for authentication and authorization. This means that users must learn a new authentication and authorization process for each SSI model. Furthermore, there is the problem that service developers must implement this process for each SSI model separately to link their service to the SSI models. The SSI models have made efforts to solve this problem by providing a tutorial page to help users learn the new process or by providing a library for easy development. However, this does not solve the fundamental problems described above.

This paper proposes a novel blockchain-based SSI model that solves those problems. The proposed model follows the concept of the SSI model and complies with the OAuth 2.0 framework at the same time. OAuth 2.0 is a mature authorization standard [16,17] with wide public use. By complying with OAuth, the proposed model could not only make development easy but also reduce users' burden of learning new authentication and authorization process because they are already familiar with OAuth. In the proposed model, user-centric authentication and authorization are made possible with a design that makes each user play the role of the authorization server in OAuth using the user's own device. By using blockchain, the proposed model has increased availability as users can stably manage their information, and it provides a decentralized authentication and authorization process that is not restricted to a certain service provider, such as Google.

The proposed model has the following contributions. First, it is the first SSI model that complies with OAuth 2.0 standard, which results in high reliability and interoperability. Second, it provides novel user-centric authentication and authorization which are controlled under a user's own device with the help of blockchain ledger. Third, from the viewpoint of service developers, the proposed model can be easily applied to their service because it follows the flow of OAuth 2.0. Fourth, it enables a user to manage personal information in a both secure and high accessible way by storing the information in the blockchain after encryption.

The rest of this paper presents the following. Section 2 shows how OAuth 2.0 works and it examines existing studies related to SSI. Section 3 describes the structure and processes of the proposed model. Section 4 displays the results of implementing the proposed model. Section 5 provides the results of a security analysis and Section 6 contains conclusions.

2. Related Work

2.1. The OAuth 2.0 Framework

OAuth is an authorization framework in which a third-party application is delegated limited right to access the user information that is stored in another web service. OAuth 2.0 provides more development convenience and has a simpler authentication process than OAuth 1.0 and OAuth 1.1 [18]. It does this by removing a complicated encrypting process and using the HTTPS protocol. Figure 1 describes the entities that constitute OAuth and their roles.

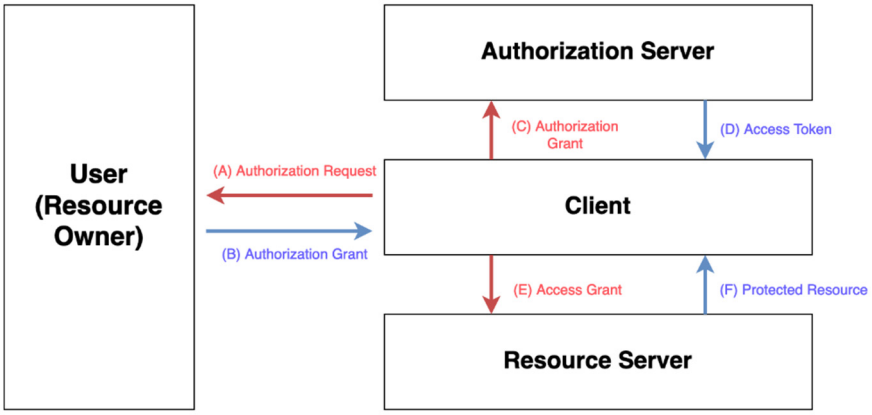

Figure 1. The abstract flow of OAuth 2.0.

- User (resource owner)

The user is the owner of the resource requested by a client, which is described next. A user can approve or deny access to the resource. Although a device or file can become a resource owner in some case, the user is the resource owner in most cases. The proposed model follows this convention.

- Client

A client is an entity that provides the Internet service that the resource owner wants to use. The client wants to obtain an access token from the authorization server, which means the right to acquire the user information.

- Authorization Server (AS)

The authorization server issues an access token to the client after successfully verifying authentication and authorization.

- Resource Server (RS)

The resource server hosts the protected resources such as a user's personal data. The client can request user information to the RS by using the access token.

OAuth 2.0 has four grant types (authorization code, implicit, resource owner and password credential and client credential). Although each type has its own scenario, the authorization code grant type has become the de facto standard for user authentication. Most clients that use OAuth 2.0 work on a server interacting with the user's web browser and this grant type was created by focusing on the server-based clients. The typical flow of the authorization code grant type is described below and our model also uses this grant type.

1. As shown in Figure 1, when the user accesses to the client, the client sends an authorization request to the user. The request includes the client_id, response_type, state and redirect_URI together as parameters. These parameters are:

    - client_ID (mandatory): an identifier of the client that provides the service.
    - response_type (mandatory): this value must be set to "code" for the authorization code.
    - state (recommended): used as a countermeasure against CSRF (Cross Site Request Forgery) attacks.
    - redirect_URI (optional): the URI (Uniform Resource Identifier) to be redirected for response to the request.

2. The user grants or denies the authorization request of the client with the help of AS, as shown in Figure 2. In general, AS's authentication of the user is included in this process. After the user's granting, AS redirects the user back to the client with the following values.

   - auth code (mandatory): a one-time code to obtain an access token.
   - state (recommended): the same value received from the client in step 1.

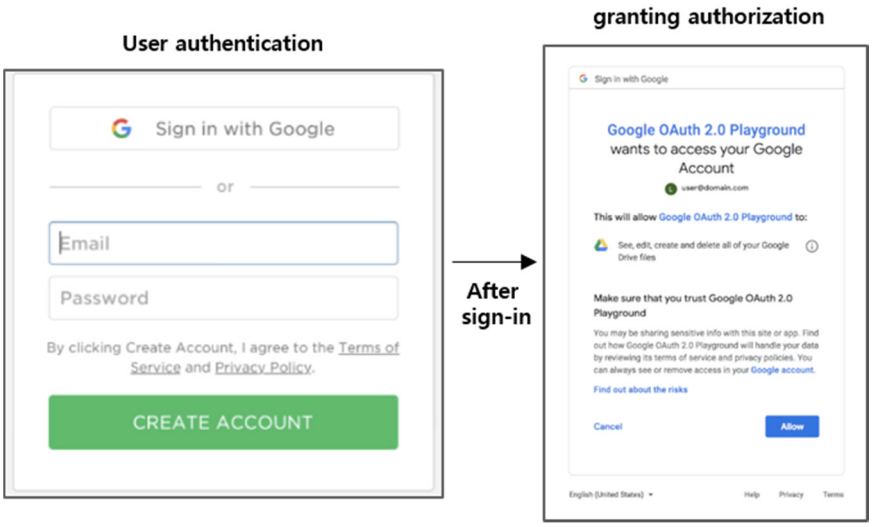

Figure 2. An example screenshot showing that a user signs in with Google ID and then grants authorization request from the client.

3. The client delivers the auth code to AS, together with client ID, client secret, and redirect URI. AS authenticates the client with the client ID and client secret, and then validates the auth code.

   - grant_type (mandatory): "authorization_code" is set for the authorization code grant type.
   - code (mandatory): the auth code received in step 2.
   - redirect_URI (mandatory): the path that shows where to return the response when the user requests.
   - client_ID (mandatory): a unique identifier of the client.
   - client_secret (mandatory): a credential used for authentication of the client.

4. AS issues and delivers an access token, which allow the client to access the user information.

   - access_token (mandatory): a credential used to access protected resources.
   - token_type (mandatory): there are "bearer" and "mac" types, and at present, the "bearer" type is generally used.
   - scope: informs the client of the access token issued.

5. After the client receives a valid access token, it can present the token to the resource server (RS) and obtain the protected resource which it wants to access.

2.2. Self Sovereign Identity

SSI is an identity management model where an individual user owns and controls his identity and personal information [19]. SSI should satisfy several requirements including decentralization, portability, simplicity and usability [6]. However, it is difficult for each user to establish such environment satisfying the requirements above with his own hands [20]. Moreover, the decentralized nature of SSI makes it hard to guarantee reliability and integrity [21].

Applying distributed ledger technology (DLT) and the smart contract technology of blockchain to SSI can solve the problems discussed earlier. DLT is a storage technology that shares identical information among participants in a P2P network for the purpose of guaranteeing integrity and preventing forgery attacks. Blockchain implements DLT by generating blocks containing the hash value of a previous block via the consensus process. DLT prevents forgery because an attacker who attempts to forge the data stored in some block needs to forge the hash values of the block and all the subsequent blocks [22]. A smart contract is a kind of program that operates automatically on a blockchain platform [23]. In Ethereum, a smart contract is developed with Solidity and deployed in the blockchain by a specific transaction whose destination address is 0. The execution result of a smart contract is also recorded in blockchain after being validated by nodes using identical parameters and states. Because the logic and states in the smart contract is transparent to all participants, the reliability and integrity problem which was a significant issue of SSI can be solved.

Recently several blockchain-based SSI models are being developed. Among them, Sovrin [14] is a project that began with the purpose of solving the problem that a considerable amount of identity is created in a duplicated and repeated manner in the on-line environment. Sovrin has an independent blockchain network and it produces fast consensus by using a modified PBFT algorithm names Plenum. However, most of the operational authority of the blockchain is granted to Guardian and Admin. Sovrin stores data in public claims or private claims according to the importance of the data and it has a characteristic that claim and identifier are not expressed in a direct association. Because a private claim is stored in off-chain storage and it is not recorded in blockchain, there occurs an accessibility problem of using a specific client program. Uport [13] is similar to Sovrin and it was developed based on Ethereum. By distributing the ID restoration authority to a socially reliable party named the Trustee, ID owners can restore their ID when they lose the device that has authority to control their DID. It can interact with users in a variety of forms, such as QR codes, emails and push notifications, and it complies with the DID format suggested by W3C. Uport supports various libraries so developers can incorporate it into a variety of environments. However, third parties depend heavily on these libraries to follow the distinctive authentication process of Uport.

3. VaultPoint

Although SSI models that use blockchain have been developed and proposed, most have not been broadly adopted. The fundamental problem is that each model adopts a unique authentication and authorization process, which creates technical barriers when existing systems try to incorporate the SSI model. Users face the unfamiliarity and inconvenience of using the new process. Furthermore, the security of the models' unique authentication and authorization processes have not been sufficiently analyzed.

To solve these problems, this paper proposes a new SSI model named VaultPoint. The proposed model complies with the broadly used OAuth and it provides users with familiar experience by designing novel authentication and authorization processes based on OAuth. By using blockchain, the proposed model provides decentralization and integrity of user and client information and it guarantees the reliability of the authentication and authorization processes. The proposed model not only solves the information centralization and the privacy issue of the existing federated ID management models governed by major companies, but it also provides users with secure accessibility to and sovereignty over their own information. Figure 3 shows the system architecture of VaultPoint. The role of each system component is described below.

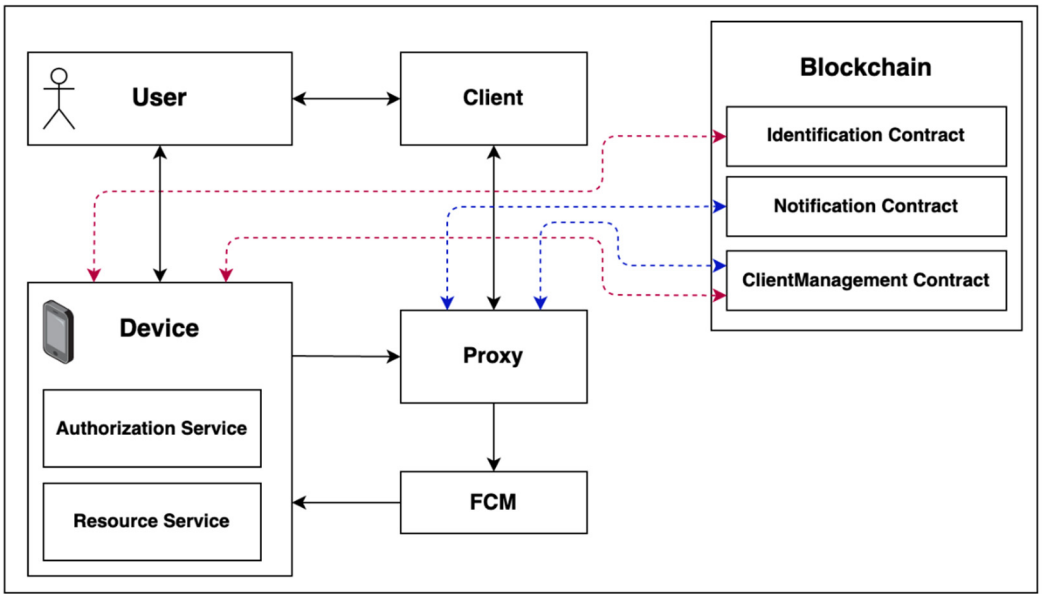

Figure 3. System Architecture of VaultPoint.

- User

Someone who uses a service provided by the client. The user accesses the client through web browser and he controls the authorization process with his own device.

- Client

An entity that provides a service to the user. The client is willing to be granted the right to access the user information by using VaultPoint.

- User device

A device like a mobile phone where an application that manages the user's identity is installed. The user device performs the roles of both authorization server and resource server in OAuth. Growing out of the existing centralized identity management models, it helps users manage their own identity and personal data. The device validates authorization requests and it delivers user information which is encrypted and stored in blockchain to the client.

- Proxy

A component that is responsible for delivering the requests to the proper user device. The response of the device is transferred to the proxy, which then delivers it to the client. Each client shares a client secret with the proxy, so the proxy performs the role of validating the client.

- FCM

Firebase Cloud Messaging which is a messaging system for mobile devices. The proxy utilizes FCM as a means of delivering the requests to the devices. A device token is needed for the correct delivery of a push message [22] and the corresponding token can be acquired through the notification contract.

- Blockchain

A decentralized system that records both client data and user data, manages the client's authority information and scope that the user authorizes and stores information required to connect to the user device. VaultPoint uses the Ethereum platform and the following three types of smart contracts are

run in blockchain. An identification contract is created by each user and it manages the user's own information. A notification contract manages the device tokens required for the delivery of push message to the device. The client management contract stores the name of the client service, client ID and scope in a succinct form so that users can refer to it when granting authorization to access data.

3.1. Smart Contract

In VaultPoint, three types of smart contracts are deployed and executed in blockchain. The notification contract and client management contract are deployed in the early phase of system set-up and each operates in one instance. The identification contract operates in different instances by each user and users generate their instance following the user registration process in Section 3.2.1.

3.1.1. Identification Contract

An identification contract stores the personal information of the user and it provides appropriate information to granted clients. As shown in Figure 4, a user's personal information is expressed in one default claim and multiple claims where a claim is a statement describing a specific attribute of the subject. The default claim comprises the user's basic information, such as the email address, nickname and gender, and it is disclosed to the public without being encrypted. Other than the email address and nickname, users can choose whether to provide the items in the default claim. Besides the default claim, users can create various types of claims, depending on the purpose of the claim. Each claim is stored in an encrypted form and those claims are managed as key-value pairs. Users can create, edit and delete the default claim and other claims using their identification contract, which is supported by update, upsert and delete functions. This contract also has an access control mechanism, which ensures that only the proper owner executes the functions.

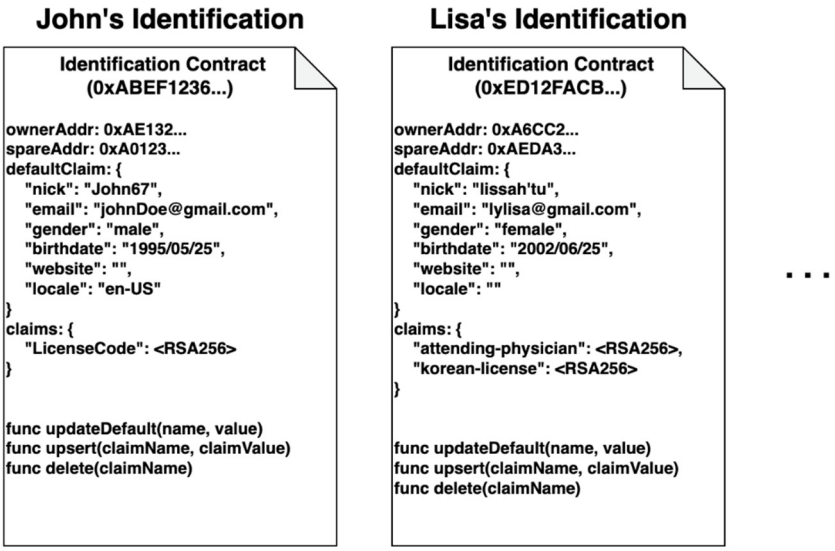

Figure 4. An example of identification contracts.

3.1.2. Notification Contract

A notification contract is used when the proxy delivers the client's authorization request to the user device. Proxy can acquire appropriate device token from this contract using the user's email address for the FCM push notification.

As shown in Figure 5, The token structure used in this contract is composed of device type, device token and Ethereum address and it binds the device token for push notification. with the Ethereum address of the device user. Each user's token is newly created by the user device following

the user registration process in Section 3.2.1. The device gets the device token from the FCM server and the Ethereum address to create or modify his token.

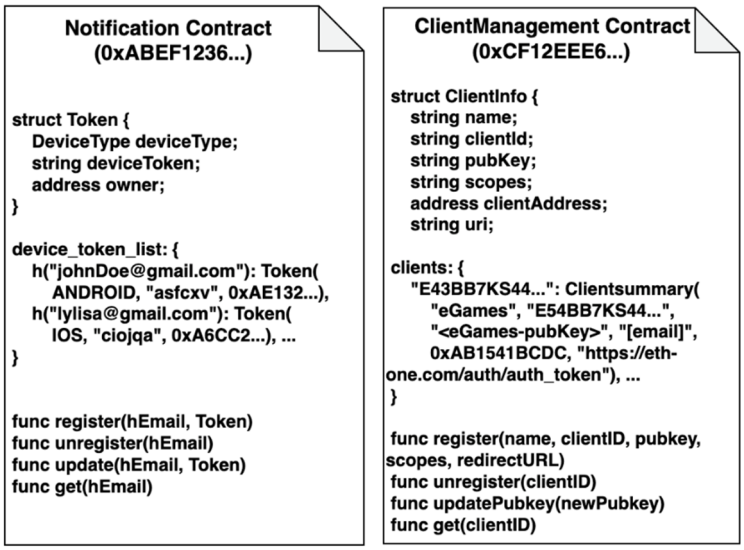

Figure 5. An example of notification contract and client-management contract.

These tokens are stored as key-value pairs in the notification contract and the hash value of the user's email address plays the role of key. A notification contract provides a register function, an update function and an unregister function. Because the user's email address is a unique identifier and the contract verifies whether the requester of the function is a legitimate owner of the target token or not, it prevents both duplicate registration and malicious modification.

3.1.3. Client Management Contract

A client management contract is used to describe the client who requests the use of user data and to store the resource list requested from each client. A client who wants to be authorized registers his ClientInfo. The ClientInfo includes the client's name, a public key that will be used for secure transmission of user data, a list of resource authority requested to users, his Ethereum address and URI for delivery of auth code, as shown in Figure 5. Similarly, a client can create or modify his own ClientInfo only because the ownership of the ClientInfo is validated with corresponding Ethereum address.

3.2. Registration

Users must perform the user registration in Section 3.2.1 to use VaultPoint. Similarly, clients must perform the client process in Section 3.2.2 to provide services. Here, it is assumed that both the users and the clients already possess normal Ethereum accounts.

3.2.1. User Registration

By performing this user registration process, users create and possess self-sovereign identity which uses their email address as an identifier. Although standardizing the Decentralized Identifiers (DIDs) is being discussed in W3C, it has not been publicly disseminated. An identifier that uses an email address has the advantage of being simple and easy to understand, compared to a decentralized identifier. Users install the VaultPoint application in their device and follow the registration procedure shown in Figure 6.

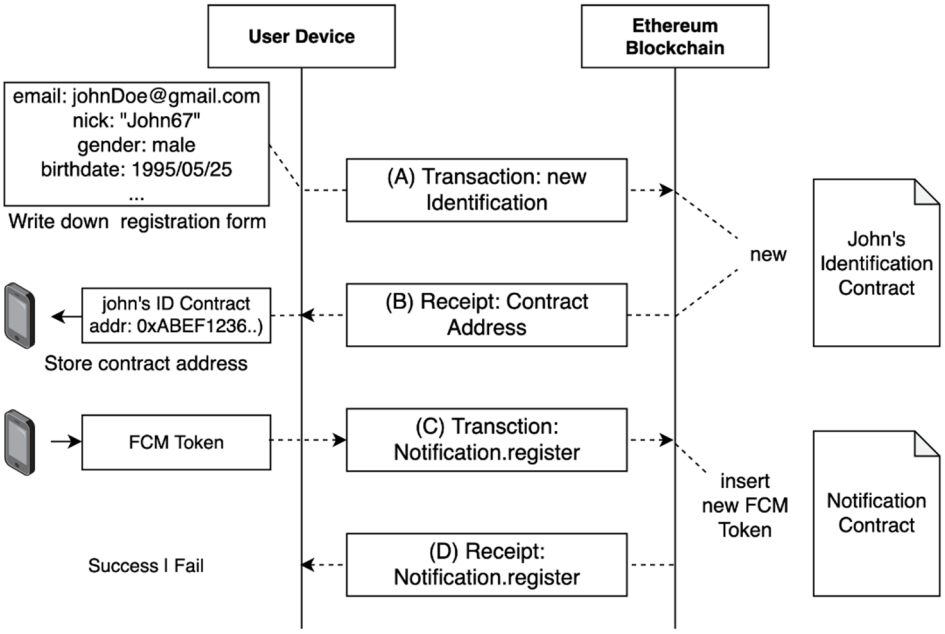

Figure 6. The flow of user registration process.

(A) The user writes the personal information to be included in the default claim in his device. The user device makes a transaction that creates the creation of a new identification contract for the user using the user's Ethereum account. The transaction including the default claim above is submitted to Ethereum blockchain.

(B) Blockchain executes the transaction in (A) and the user's identification contract is created. A receipt for this transaction is delivered to the user device and the identification contract address in the receipt is stored in the device.

(C) The device submits a transaction that registers his device token for FCM in the notification contract.

(D) The notification contract in blockchain stores the user's token and delivers a receipt, to the user device. Through these processes, the user successfully creates his digital identity, which he possesses and controls.

3.2.2. Client Registration

Clients must first register themselves in VaultPoint and be issued a client ID and client secret, In the existing OAuth-based services, AS generally provided a client registration interface. For VaultPoint, the proxy component plays this role. The flow for client registration is shown in Figure 7.

(A) The client creates a pair of public and private keys (Pub_c, Prv_c) and requests his registration to the proxy. This key pair is used for secure transfer of user data later.

(B) The proxy creates a new client ID and provides a web form that contains this ID and the address of the client management contract to the client. The client writes the name of the service that he provides, a list of authorities to be granted from the user and his public key Pub_c in this form.

(C) The transaction to store the client information in the form is created and submitted to the blockchain. Here, the transaction is created using the client's Ethereum account.

(D) The blockchain executes this transaction to register the client information in the client management contract and then transaction receipt is transferred to the client. Client reviews the receipt to verify that he was correctly registered.

(E) The client delivers his Ethereum account address to the proxy together with the receipt.

(F) The proxy verifies that the client's information was correctly registered in the client management contract by using the received Ethereum address and client ID. Then, it creates a new client secret and delivers it to the client.

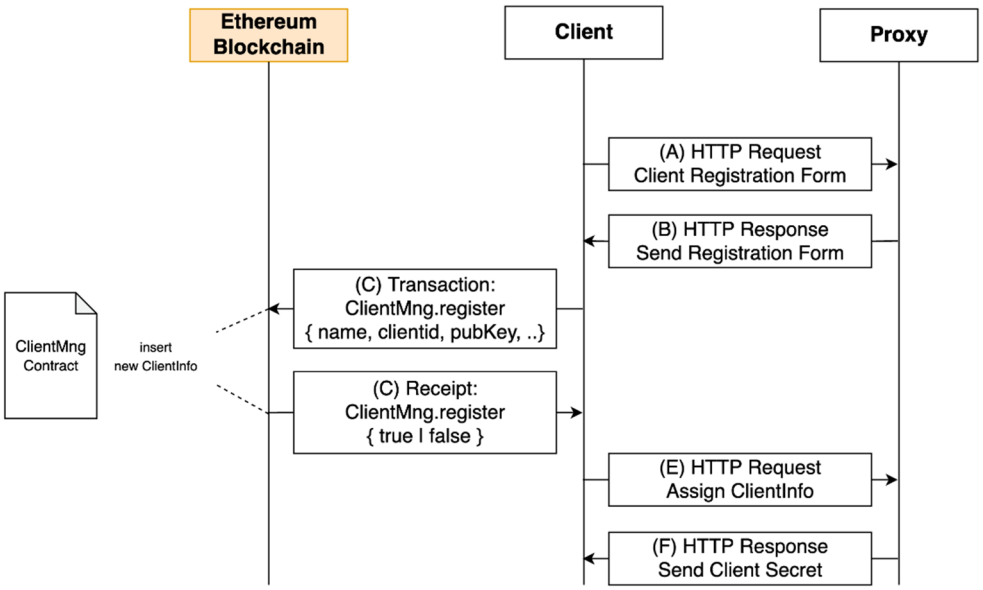

Figure 7. The flow of client registration process.

3.3. Authentication and Authorization

In VaultPoint, authorization is the process by which a client is granted authority to access user information. VaultPoint follows the authorization code grant type of OAuth, which comprises the following three phases. First, a client asks a user for authorization and receives an auth code (Section 3.3.1). Second, user authentication in Section 3.3.2 is performed if it is required during the first phase. Third, the client is issued an access token that gives him access to the user information using the auth code (Section 3.3.3).

3.3.1. Client Authorization

A client must be authorized by the user to get the desired user information. When approval is given an auth code is provided to the client. In the existing OAuth-based services, the authorization server to which the user belongs performs the client authorization process by interacting with the user. In VaultPoint, this process is performed in the user device with help from the proxy. The flow for client authorization is shown in Figure 8.

(A) When the user approaches the client's service, the client creates an authorization request for access to the user's information. This request is redirected to the proxy (instead of the existing AS) through the user. If the user has not been authenticated, the process in Section 3.3.2 is performed, followed by the next process.

(B) The proxy acquires the device token for the user device from the notification contract of blockchain using the user's email address. It then transfers an FCM-based push notification to the user device using this token, which includes the client's request.

(C) The device acquires client information from the client management contract of blockchain by using the received client ID. Based on this information, the device checks the name and identity of the requesting client and learns the scope of information that client wants. The user can approve or deny this authorization request by looking at the information displayed on the device. When the result is delivered to proxy, the proxy delivers it to the client. If the user has given approval, an auth code is created and delivered together to the client.

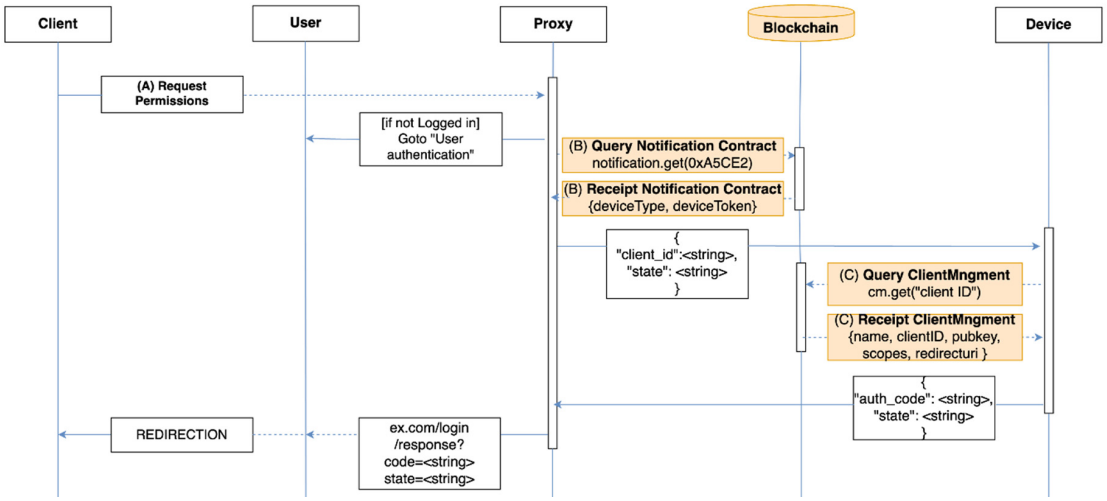

Figure 8. The flow of client authorization process.

3.3.2. User Authentication

For users to approve an authorization request from the client, they first must be verified that they are the correct subject of the client's request. With services that support OAuth, AS verifies the user with authentication tools such as a password. With VaultPoint, however, the user device plays the role of AS so it is not necessary to use a complicated authentication tool for user authentication. The user only needs to verify that his authentication request has successfully arrived at his device. In addition, VaultPoint creates a random secret code for each request to prevent impersonation attacks where an attacker deceives users by sending another authentication request simultaneously. The flow for user authentication is shown in Figure 9.

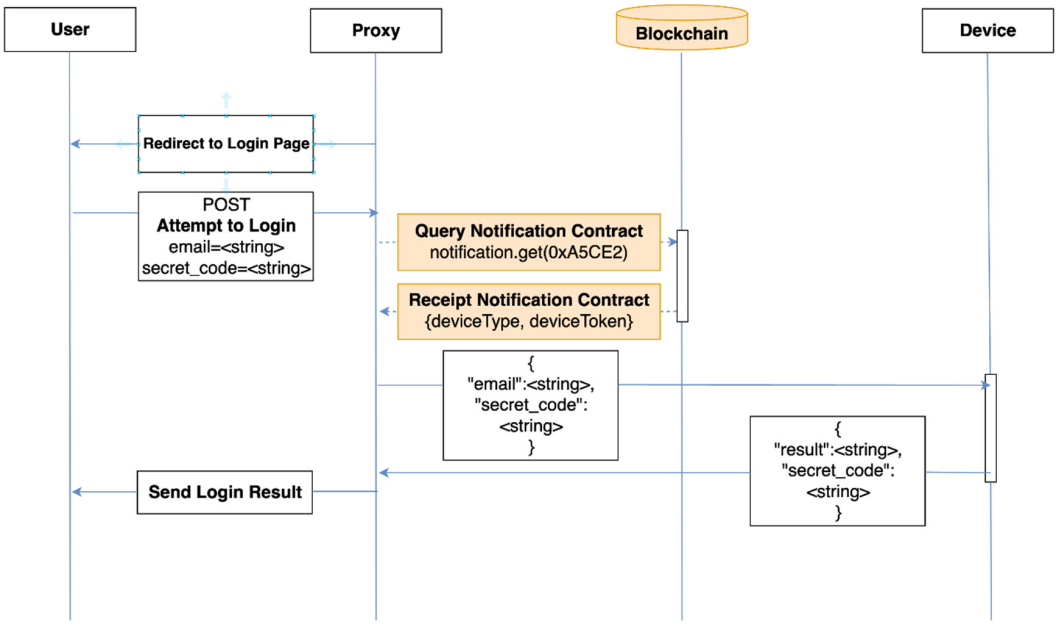

Figure 9. The flow of user authentication process.

(A) When a user requests authentication, the proxy creates a new secret code and provides it to the user. The user delivers his ID (email address) to the proxy.

(B) The proxy acquires the device token of the user from the notification contract of blockchain. The proxy notifies the user device that an authentication request arrived using this token.

(C) The user checks whether the secret code that arrived along with the authentication request to his device is identical to the value received in (A). If the values are identical, user authentication is completed and the result is delivered to the proxy.

3.3.3. Issuance of an Access Token

The auth code obtained in Section 3.3.1 means that authorization request was approved by the user. For the client to acquire user information, the auth code should be transferred to AS and the corresponding access token should be issued first. With VaultPoint, the user device confirms the auth code and issues an access token through the proxy. This process is automatically performed without additional intervention from the user. The flow requesting and issuing the access token is shown in Figure 10.

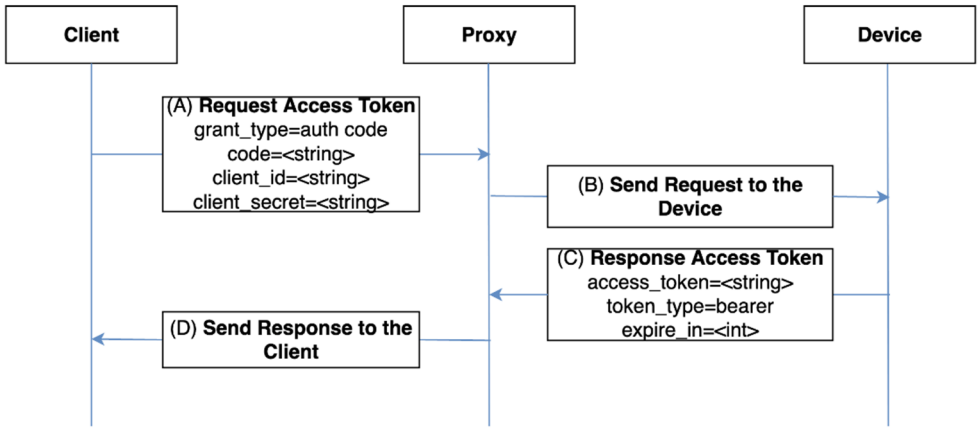

Figure 10. The flow of access token request and issuance.

(A) The client delivers auth code, client ID and client secret to the proxy and requests access token. Client secret is a value shared between the client and the proxy by the client registration process and it is used to verify the client.

(B) The proxy finds the user device that issued the auth code in Section 3.3.1 and delivers the request to it.

(C) The device verifies whether the auth code is what it created previously. If correct, the device newly issues an access token and delivers it to the proxy.

(D) The proxy delivers the received access token to the client.

3.4. Access to User Information

In VaultPoint, the user device performs the role of resource server and the actual user information is stored in the blockchain. This approach has the advantage that user information is not lost when the device is missing or replaced. User information is managed through the user's identification contract. Because the blockchain data is transparent to all participants, all claims except the default claim are encrypted by the users' private key to prevent from being exposed. The client who acquired a valid access token in Section 3.3.3 can access the user information through the following process. The flow for user information acquisition is shown in Figure 11.

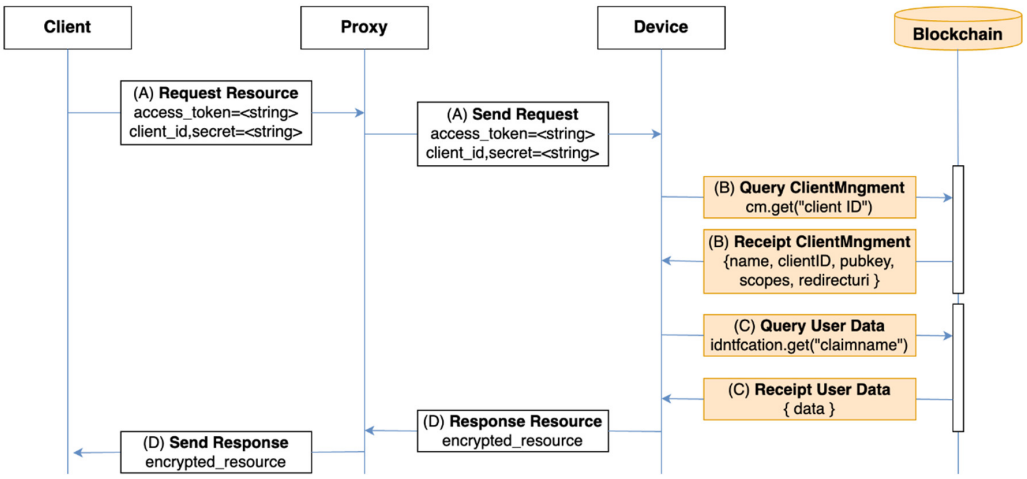

Figure 11. The flow of acquisition of user information.

(A) The client request user information to the proxy and delivers access token, client ID and client secret together. The proxy delivers this request to the user's device using FCM, similarly to Section 3.3.

(B) The user can access to client information using the client ID given from Proxy. Result of query show what sort of information client want and need to get.

(C) The user device verifies access token and checks its validity. Then it collects the user information that the client requested from its identification contract. The encrypted user information is decrypted using the user's private key. This process can be skipped if the device has up-to-date user information.

(D) The device gets the client's public key Pub_c from the client management contract and transfers the user information to the proxy after encrypting it with the Pub_c. The proxy delivers this information to the client. Finally, the client successfully gets the requested information by decrypting it using his private key Prv_c.

4. Implementation

The proposed VaultPoint was implemented and its soundness was confirmed through an experiment. Table 1 describes the implementation environment of each component of VaultPoint. The implemented smart contracts were deployed to Kovan Ethereum testnet [24] and the addresses of contracts are shown in Table 2. The device application and proxy were connected to the blockchain by using Ethereum nodes provided by Infura [25] as an entry point. A client providing a mock web service connected to VaultPoint was also implemented to show that applying VaultPoint is simple and easy.

Table 1. Implemented components and their environments.

Component	Environment
Proxy	Django on Ubuntu 18.04
Client	Gorilla on Ubuntu 18.04
Device App	Xamarin on Android 8.0 Oreo (API 26), tested on Galaxy A7 device
Smart contract	Solc 4.26 on Kovan Test Network

Table 2. Addresses of smart contracts deployed to Kovan testnet by VaultPoint.

Contract Name	Address
Notification	0x913464b9cD148874840EB0906fDa04e274F01DbB
Identification	0x4c70902a3Ef0279eBcE14967a8b66aEf22e87dd5
ClientManagement	0x0025182d23AAA37c2D5a642415F7Dc87022B82Ff

The following figures reflect the operations of VaultPoint in the experiment. Figure 12 shows an example of a user's registration using his mobile application. The user is given a form for writing his information as shown in Figure 12B and he can write additional information once the registration is complete. The user can scan the QR code of the private key and address of his Ethereum account, as shown in Figure 12A. When the user touches the button at the bottom of Figure 12B, the user registration process in Section 3.2.1 is executed and the user's identification contract is created. Figure 13 shows the receipt of the user's contract creation transaction executed in Kovan testnet.

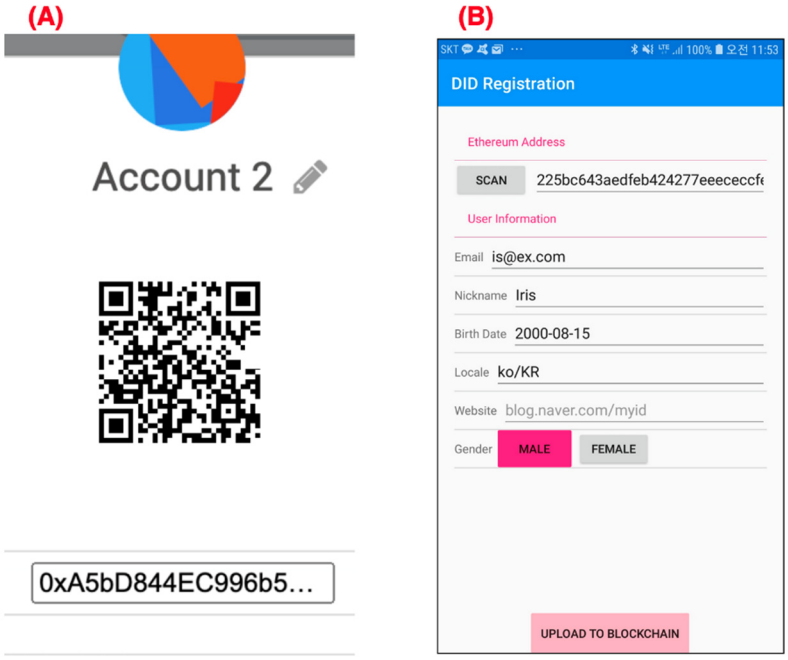

Figure 12. Example screenshots of user registration: (**A**) scanning Ethereum key and (**B**) writing user information.

Figure 13. An example receipt of identification contract creation transaction in user registration.

Figure 14 shows an example where a client who provides a mock service named Eth-one registers it following the process in Section 3.2.2. As shown in Figure 14A, the client requests registration to the proxy and receives a registration form to fill in the necessary information. Afterward, the client makes and submits a transaction that requests for the registration in the client management contract using its Ethereum account. This process is performed with the help of Metamask [26], which is a popular Ethereum wallet, as shown in Figure 14B and the client waits for a while until this pending transaction is confirmed and recorded in the block. Figure 15 shows the receipt of this client registration transaction. Once the receipt of the transaction is delivered to the proxy, the proxy verifies its validity and creates a client secret. The client receives this client secret as shown in Figure 14C.

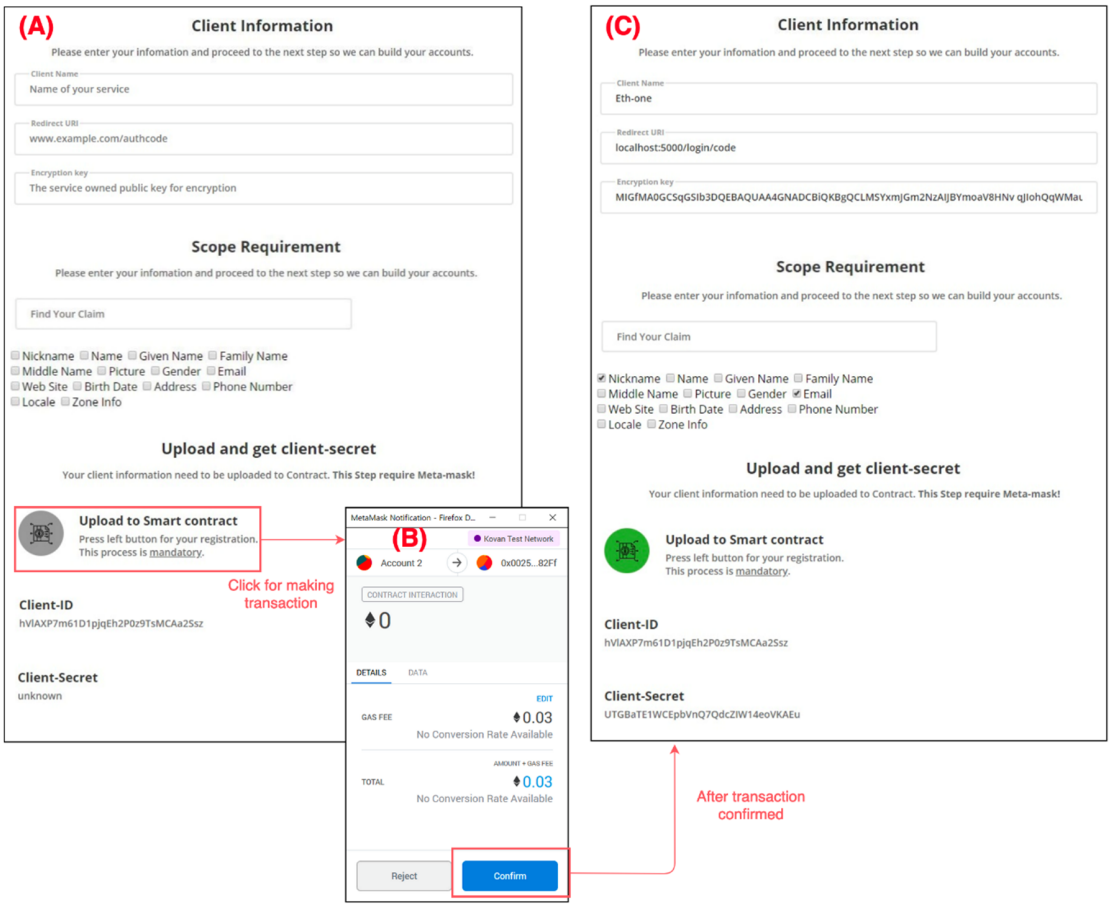

Figure 14. Example screenshots of client registration: (**A**) initial registration form, (**B**) register transaction submission and (**C**) acquisition of client secret after registration.

Figure 16 shows an example of the screen views that appear when a user confirms and approves the client's authorization request and an example of user authentication that is performed during the former process. Both Figure 16A and B are displayed on the user's web browser, whereas both Figure 16C and D are displayed on the user's device. As shown in Figure 16A, when the user approaches the client service through the browser and touches the "Sign in with VaultPoint" button, the client requests authorization to get the user information and the user is redirected to the proxy. If user authentication is required, the proxy provides a web form having a random secret code (Figure 16B) to the user. When the user inserts his email address, the proxy delivers the authentication request to the user device using blockchain and FCM, and the device displays this request as in Figure 16C. The user verifies whether the secret code "LTZP-QRVD" displayed on his browser in Figure 16B is equal to the secret code displayed on his device in Figure 16C and selects accept. Afterward, the authorization process in Section 3.3.1 continues. The information about the client who requested authorization and

requested items are displayed in the user device, as shown in Figure 16D. When the user selects the "Accept" button, a new auth code is created and delivered to the client. Afterward, the process in Section 3.3.3 where the client is issued an access token continues without any interaction with the user.

Figure 15. An example receipt of client registration transaction executed in Kovan testnet.

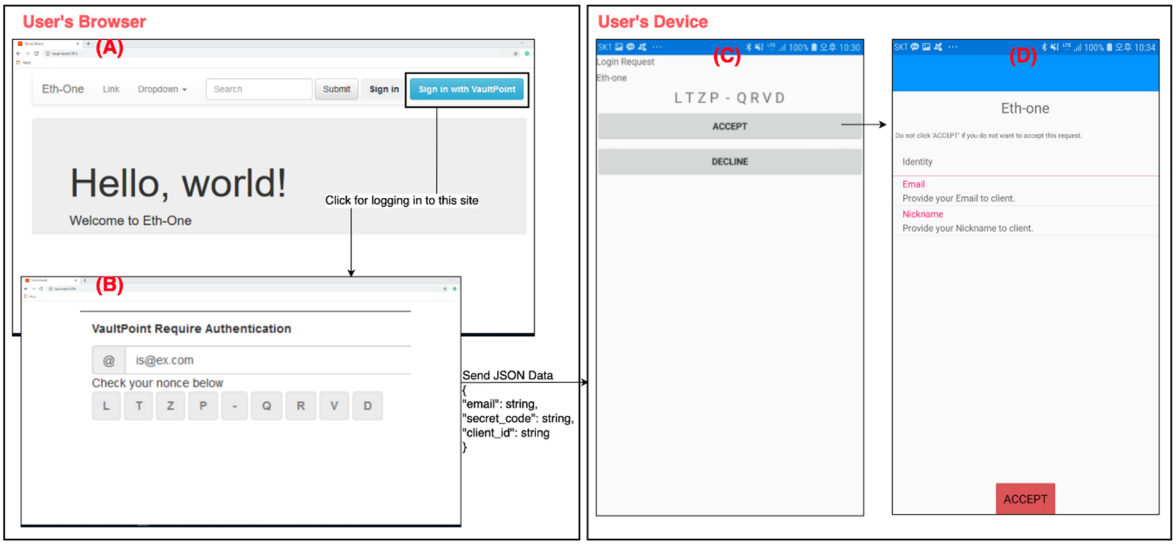

Figure 16. Example screenshots of authentication and authorization process: (**A**) user's access to client service, (**B**) user authentication request from the proxy, (**C**) secret code displayed on the user's device and (**D**) authorization request of client.

The client who successfully acquired the access token gets the desired user information by preforming the process in Section 3.4. As shown in upper right corner of Figure 17, the client acquired the user's nickname and utilized it.

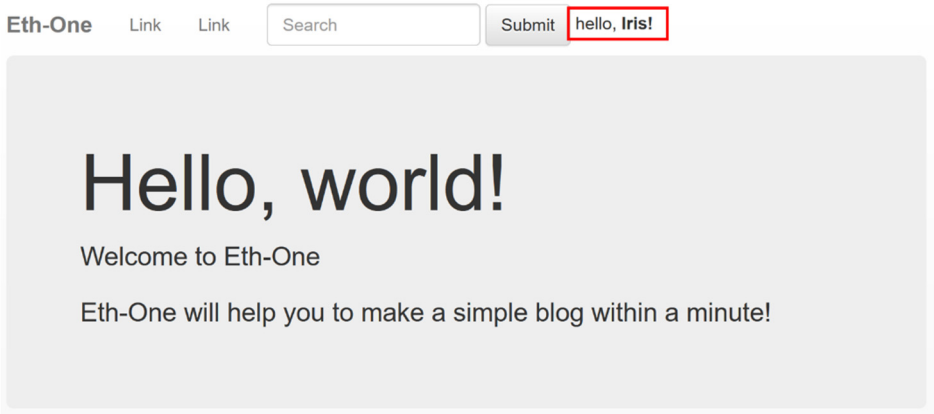

Figure 17. An example screenshot showing that user information has been acquired by client.

5. Security Analysis

This section provides the security analysis of the proposed VaultPoint. OAuth 2.0 with which VaultPoint complies is being used broadly and existing security analyses have confirmed its security. Thus, the security analysis on VaultPoint was conducted under the assumption that OAuth 2.0 is safe. The security of VaultPoint against two major threats, which are identity theft and leakage of personal information, is explained below.

5.1. Prevention of Identity Theft

A malicious attacker's theft of a user's identity is a very serious threat. Here we consider three attack scenarios where a hacker attempts identity theft, along with the countermeasures taken by VaultPoint.

In the first attack scenario, an attacker approaches the client disguising himself as the target user. If the attacker succeeds in replacing the FCM token of the target user in the notification contract with his own token, the authentication request in Section 3.3.2 is sent to the attacker's device instead of the target user's device, which results in success of the attack. However, it is very hard to forge the FCM token because blockchain guarantees the integrity of data recorded in the smart contracts. The only way to replace the token is to use the update function of the contract and before update the contract verifies whether the requester is the owner of the Ethereum address bound to the token. Therefore, the attacker cannot replace the token of the target user without having the user's Ethereum account and corresponding private key.

The second attack scenario occurs when the target user requests authentication. The attacker pretends to be the target user and requests authentication simultaneously. In that case, two authentication request messages are delivered to the target user's device. If the attacker is lucky, the attacker's request arrives first and the user may approve it because he thinks it is the legitimate request that he created. To prevent this kind of attack, VaultPoint generates and uses a random secret code for each authentication request. Normal users can verify their own authentication request by checking whether if the secret code provided on the browser at the time of requesting is equal to the secret code displayed on their device.

In the third attack scenario, an attacker is authenticated as himself but then provides target user's email address and fake information to the client. In that case, the attacker can access the client as if he is the target user and perform malicious activities causing damage to the target. To prevent this possibility, VaultPoint supports the client's verification by sending the email address of the user from whom the client requests authorization together when the proxy delivers the user information to the client.

5.2. Preventing the Leakage of Personal Information

Users' personal information is stored in their identification contact in blockchain. This has the advantage that the information is not lost in case of an accident like as missing device. Because of the transparency of blockchain, however, there is a possibility that an attacker accesses to the user's identification contract directly without authorization. Although the address of this contact is not disclosed to the public, the attacker can discover the contract address of the target user by visiting, the addresses that he learned from examining all the transaction. In VaultPoint, all claims inside the contract (except for the default claim) are encrypted to prevent this kind of threat. Moreover, VaultPoint performs access control for the functions provided by this contract using the function modifier in Solidity. It checks whether the sender of the transaction that seeks to access the contract is the owner of the contract. If not, the transaction is reverted.

VaultPoint has a decentralized form where each user device performs the role of the authorization server, instead of depending on a centralized authorization server. The proxy is the medium that connects a client and the user device. The proxy is a single point of failure and it becomes an attractive target for attackers. However, this type of threat is identical to the threats against the existing authorization servers that support OAuth. In addition, the proxy was implemented based on the cloud computing to be fault tolerant and highly available. Furthermore, the damage from attacks on the proxy is significantly less than the one on the existing authorization servers because the personal and critical information of the users is not stored in the proxy.

As the proxy performs the role of receiving user information from the user device and transferring it to the client, a malicious proxy might collect and misuse multiple users' information. To minimize this risk, VaultPoint encrypts user information in the user device with the client's public key before the transfer as shown in Figure 18, so the proxy cannot acquire user information by decrypting it in the middle of the transmission. The integrity of the client's public key is guaranteed, as it is managed by the client management contract in blockchain. Hence, neither the proxy nor outside attackers can change or forge the public key.

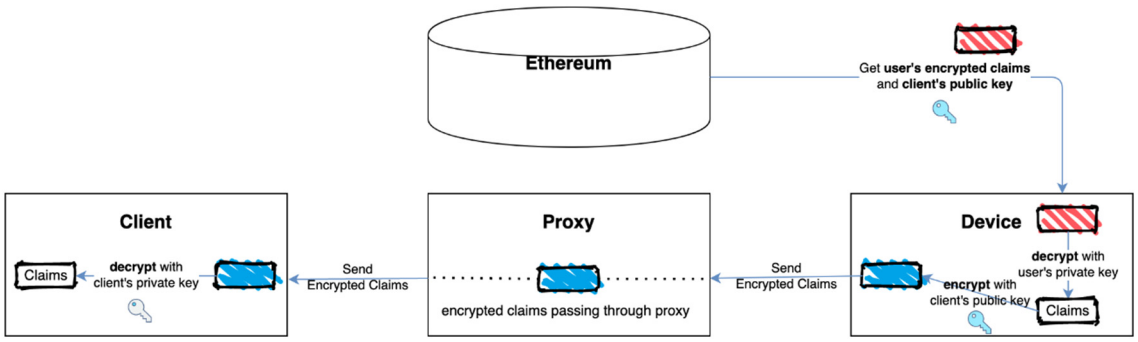

Figure 18. Secure delivery of user's claim from user's device to client.

An attacker can attempt to acquire user information by eavesdropping messages transferred among the components of VaultPoint through the network. VaultPoint prevents this threat as follows. First, similarly to OAuth 2.0, eavesdropping is prevented by using the HTTPS protocol for communication among the user, client and proxy. The HTTPS protocol is also used when the proxy requests a push message to the FCM server and XMPP over TLS protocol is used when the FCM server delivers messages to the user device, which also prevents malicious attacker's eavesdropping. The Infura service, which was used to connect to the blockchain, also supports HTTPS and the transaction itself prevents falsification through the digital signature. Moreover, the user information is transferred in an encrypted form as described above. Therefore, the combination of these means effectively prevents data leakage.

6. Conclusions

This paper proposed a new blockchain-based SSI model that complies with OAuth 2.0. With help from blockchain, users have increased accessibility to their information without being restricted to a certain service. Users can perform authentication and authorization using their own device. The proposed model provides a user experience that is similar to the existing OAuth procedure and it can be easily applied to those clients who have been using OAuth. This makes the scalability of the proposed model excellent. This study showed the feasibility of the proposed model through an implementation and the security analysis shows the proposed model is secure against identity theft and information leakage.

The proposed model provides a solution to the problem that user information is managed in a monopolistic manner by several major IT companies. It helps users secure data sovereignty, which refers to the right to use and control one's own information. A future study will investigate the possibility of a new type of user-centric web, based on this proposed model, where user can directly manage the history of their activity and the contents that they created and shared with other users.

Author Contributions: Conceptualization, H.K.; Methodology, S.H.; software, S.H.; validation, S.H. and H.K.; writing—original draft, S.H. and H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Ministry of Science ICT Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education(2018R1C1B6002903).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nongbri, I.; Hadem, P. A Survey on Single Sign On. *IJCRT* **2018**, *6*, 595–602.
2. Radha, V.; Reddy, D. A Survey on Single Sign-On Techniques. *Procedia Technol.* **2012**, *4*, 134–139. [CrossRef]
3. Jiang, J.; Duan, H. A Federated Identity Management System with Centralized Trust and Unified Single Sign-On. In Proceedings of the 6th International ICST Conference, Harbin, China, 17–19 August 2011; pp. 785–788.
4. Khattak, Z.; Sulaiman, S. A Study on Threat Model for Federated Identities in Federated Identity Management System. In Proceedings of the International Symposium on Information Technology, Kuala Lumpur, Malaysia, 15–17 June 2010.
5. Isaak, J.; Pouwelse, J. User Data Privacy: Facebook, Cambridge Analytica and Privacy Protection. *IEEE Comput.* **2018**, *8*, 56–59. [CrossRef]
6. Ferdous, S. In Search of Self-Sovereign Identity Leveraging Blockchain Technology. *IEEE Access* **2019**, *7*, 103059–103079. [CrossRef]
7. David, R.; Drummond, R. OpenID 2.0: A Platform for User-Centric Identity Management. In Proceedings of the 2nd Workshop on Digital Identity Management, Alexandria, VA, USA, 3 November 2006; pp. 11–16.
8. San-Tsai, S.; Eric, P. What Makes Users Refuse Web Single Sign-On? An Empirical Investigation of OpenID. In Proceedings of the 7th Symposium on Usable Privacy and Security, Pittsburgh, PA, USA, 20–22 July 2011; pp. 1–20.
9. Delft, B.; Oostdijk, M. A Security Analysis of OpenID. In Proceedings of the Second IFIP WG 11.6 Working Conference on Policies and Reseach Management (IDMAN), slo, Norway, 18–19 November 2010; pp. 73–84.
10. Paul, D.; Fabien, P. A First Look at Identity Management Schemes on the Blockchain. *IEEE Secur. Priv.* **2019**, *16*, 20–29.
11. Goodell, G.; Aste, T. A Decentralized Digital Identity Architecture. *Front. Blockchain* **2019**. [CrossRef]
12. Van, B.; Rico, H. Self-Sovereign Identity Solutions: The Necessity of Blockchain Technology. Available online: https://arxiv.org/pdf/1904.12816.pdf (accessed on 30 July 2020).
13. uPort: A Platform for Self-Sovereign Identity. Available online: https://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf (accessed on 1 July 2020).
14. Sovrin-Protocol-and-Token-White-Paper. Available online: https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf (accessed on 1 July 2020).
15. Decentralized Identifiers (DIDs). Available online: https://www.w3.org/TR/did-core/ (accessed on 1 July 2020).

16. Fett, D.; Kusters, R. A Comprehensive Formal Security Analysis of OAuth 2.0. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019.
17. OAuth 2.0 Authorization Framework. Available online: https://tools.ietf.org/html/rfc6749 (accessed on 1 July 2020).
18. Differences between OAuth 1 and 2. Available online: https://www.oauth.com/oauth2-servers/differences-between-oauth-1-2/ (accessed on 1 July 2020).
19. Der, U.; Jahnichen, S. Self-Sovereign Identity—Opportunities and Challenges for the Digital Revolution. Available online: https://arxiv.org/abs/1712.01767 (accessed on 10 July 2020).
20. Mainka, C.; Mladenov, V. Do Not Trust Me: Using Malicious IdPs for Analyzing and Attacking Single Sign-on. In Proceedings of the IEEE European Symposium on Security and Privacy, Saarbrücken, Germany, 21–24 March 2016.
21. Zheng, Z. An Overview of Blockchain Technology: Architecture, Consensus, and Trends. In Proceedings of the 6th IEEE International Congress on Big Data, Honolulu, HA, USA, 25–30 June 2017; pp. 558–560.
22. Warren, I.; Meads, A. Push Notification Mechanisms for Pervasive Smartphone Applications. *IEEE Pervasive Comput.* **2014**, *13*, 61–71.
23. Chinchilla, C. A Next-Generation Smart Contract and Decentralized Application Platform, Ethereum Whitepaper. Available online: https://ethereum.org/en/whitepaper/ (accessed on 27 July 2020).
24. Kovan. Proposal: Kovan Testnet. Available online: https://kovan-testnet.github.io/website/proposal/ (accessed on 10 July 2020).
25. Infura. Ethereum API|IPFS API Gateway|ETH Node as Service|Infura. Available online: https://infura.io (accessed on 10 July 2020).
26. Marks, E.; Akers, A. Metamask/Metamask-Docs: Metamask Project Documentation. Available online: https://github.com/MetaMask/metamask-docs (accessed on 10 July 2020).