PRIOIDC: A CLIENT-ACCESS-HIDDEN EXTENSION FOR OPENID-CONNECT

Name of author

Address - Line 1
Address - Line 2
Address - Line 3

I. INTRODUCTION

To maintain each user's profile and provide individual services, each service provider needs to identify each user, which requires the users to be authenticated at multiple online services repeatedly. Single Sign-On (SSO) systems enable users to access multiple services (called relying parties, RP) with the single authentication performed at the Identity Provider (IdP). With SSO system deployed, a user only needs to maintain the credential of the IdP, who offers user's attributes (i.e., identity proof) for each RP to accomplish the user's identification. SSO system also brings the convenience to RPs, as the risks in the users' authentication are shifted to the IdP, for example, RPs don't need to consider to leakage of users' credentials. Therefore, SSO systems are widely deployed and integrated. The survey on the top 100 websites demonstrates that 24 websites (e.g., Google, Facebook and Twitter) serve as the IdP while 63 websites integrate the SSO service.

One basic requirement of SSO systems is the security, which includes two aspects: 1) the attacker should not be able to access the honest RP with the honest users' identity; 2) the identity injection will never succeed, that is, the attacker should not be able to make the honest user access the RP with an incorrect identity. Plenty of works are proposed for the security of SSO systems. Firstly, various standards, e.g., OAuth 2.0 [1], SAML [2] and OpenID Connect (OIDC) [3], are proposed to formalize the handling at each entity (i.e., the user, RP and IdP) and the information exchanges between the entities. Secondly, the standards, SAML, OAuth and OIDC, are formally analyzed, for example, a general Dolev-Yao style web model is proposed for the web infrastructure [4] and adopted to analyze the security of SAML, OAuth and OIDC []. Moreover, the typical implementations of SSO systems, e.g. Google, Facebook, Twitter and the corresponding RPs, are systematically analyzed [5] [6] [7] [8], which makes the security of SSO systems improved significantly.

The other important requirement of SSO systems is the privacy. As suggested in NIST SP800-63C [9], in SSO systems, 1) the user should be able to control the range of the attributes exposed to the RP, 2) multiple RPs should fail to link the user through collusion, 3) IdP should fail to obtain the trace of RPs accessed by a user. The first two properties are satisfied in the popular SSO systems. For example, in OAuth and OIDC, IdP exhibits the attributes requested by the RP and sends the attributes to the RP only when the user has provided a clear consent, which may also minimize the exposed attributes as the user may disagree to provide partial attributes. To prevent a possible correlation among users from multiple RPs, a Pairwise Pseudonymous Identifier (PPID) is suggested to be generated by the IdP for the user in each RP, which requires that the user's identifier in one RP should never be the same with or derivable from the ones of other RPs.

However, in widely deployed SSO systems, IdP knows which RP the user logs in, which reflects the service that user accesses and may be analyzed for various purposes, e.g., profiling and targeted advertising. In addition to the potential commercial purpose, exposing the identifier of accessed RP to the IdP, is required for security consideration in existing SSO systems [10]. Firstly, the identity proof should only be sent to the correct RP, which prevents the adversary from performing the impersonation attack with the leaked identity proof. Secondly, the identity proof should be bound with a specific RP and user, which ensures the identity proof is only valid in the certain RP, and avoids the misuse of identity proof, for example, the adversary fails to use the identity proof for a corrupted RP to access another RP on behalf of the victim user.

Two SSO systems (BrowserID [11] and SPRESSO [12]) are proposed to hide the user's accessed RPs from IdP, while ensuring the security of SSO systems simultaneously. In BrowserID, the user is responsible for sending the identity proof correctly and binding the proof with the correct RP using a newly generated private key, while the corresponding public key is bound with the email address by the IdP who fails to obtain the information of accessed RP. In SPRESSO, the identity proof is bound with an encryption of the RP's domain name by the IdP who knows the user's identity but not the plaintext of the RP's information, and sent to the exact RP by a newly introduced trusted entity (called FWD) who obtains the RP's domain name but not the identity of the user.

BrowserID and SPRESSO are both redesigns of SSO systems, and therefore incompatible with existing widely deployed SSO systems (e.g., OAuth, OIDC and SAML). Moreover, the new SSO systems require a complicated, formal and thorough security analysis of both the designs and various implementations. As shown in [], vulnerabilities have been found in the implementation of BrowserID. More importantly, in order to provide the same identity to the RP in the multiple logins of a user, both BrowserID and SPRESSO use the email address as the identity, which makes the user linkage (from multiple RPs) possible.

In this paper, we propose an extension (named PriOIDC) of existing widely adopted SSO system (i.e., OIDC), which preserves the systematically and thoroughly analyzed security, and achieves the fully privacy. That is, (1) the security design in OIDC is inherited to prevent the impersonation attack and identity injection attack, (2) the privacy enhance mechanisms (e.g., the clear consent from the user and the PPID) are retained, (3) a new mechanism is introduced to hide the user's accessed RP from IdP. Unlike designing and deploying a newly designed SSO systems, we only need to analyze the compliance of the new function (i.e., hiding the user's accessed RP) and the influence to the security introduced by the new mechanism. And, the deployment of PriOIDC only requires: (1) IdP provides a set of public parameters and generates the PPID with a newly provided algorithm, (2) RP integrates the SSO service with a new software development kit (SDK) whose interface remains unchanged, (3) the user installs an extension to access RP with full privacy anywhere as no storage is required in the user side.

To hide the user's accessed RP from IdP, PriOIDC avoids the potential leakage in the identity proof (i.e., id token in OIDC) and the corresponding message transmission. Moreover, PriOIDC enables only the RP (and the user) to derive the user's unique identifier in this RP from different PPIDs, which avoids the user linkage and allows the RP to provide the individual service with the user's identifier. PriOIDC achieves these as follows:

- A new algorithm is proposed to negotiate the RP's identifier between the user and RP for each login. Therefore, the RP's identifier in multiple id tokens are different, and IdP fails to infer RP's information in the generation of one or multiple id tokens. Moreover, neither RP nor the user may control the generated identifier, which avoids the misuse of the id token. The detailed analysis is provided in Section *.
- A browser extension is introduced to transmit the messages (i.e., request and response) related with the id token. IdP fails to infer the RP's information through the network traffic, and the extension ensures only the correct RP receives the id token.
- A new generation algorithm of PPID is provided, which makes the PPIDs for one user in one RP indistinguishable from others (e.g., different users in different RPs), while only the RP (and the user) has the trapdoor to derive the unique identifier from different PPIDs for one user in one RP.

The main contributions of PriOIDC are as follows:

- We propose a pratical extension for OIDC, which inherits the systematically and thoroughly analyzed security and privacy mechanisms of OIDC, and achieves the full privacy for users by hiding the accessed RPs from IdP.
- We developed the prototype of PriOIDC. The evaluation demonstrates the effectiveness and efficiency of PriOIDC. We also provide a systematical analysis of PriOIDC to prove that PriOIDC introduces no degradation in the security of OIDC.

The rest of this paper is organized as follows. We introduce the background and the threat model in Sections II and III. Section IV describes the design and details of PriOIDC. A systematical analysis is presented in Section V. We provide the implementation specifics and evaluation in Section VI, then introduce the related works in Section VII, and finally draw the conclusion.

II. BACKGROUND

OpenID Connect 1.0 protocol is designed as extension protocol of OAuth 2.0 protocol. OAuth 2.0 is specifically designed for user authorization. It allows third party to access user's personal protected resources from resource holder. In OAuth 2.0 system everyone carrying user's access token is able to achieve user's protected resources from resource holder. Access token is not bound with any RP so that it is not appropriate for authentication. OpenID Connect offers an additional id token for user identifying so that it can be used in both authentication and authorization.

II-A. OpenID Connect Flows

OpenID Connect enables RP to verify the identity of a user based on the authentication performed by IdP. As OpenID Connect can be used in both authentication and authorization, it provides three kinds of credentials for the authentication response, containing *code, token, id_token*. *Code* and *token* is defined in OAuth 2.0 and *id_token* is offered only by OpenID Connect. The credential chosen is decided by the response_type value in authentication request. According to the different choices of response_type value, the use of OpenID Connect protocol can be classified as three flows: Authorization Code Flow, Implicit Flow and Hybrid Flow. The relation of response_type and flow type is showed in table I

II-A1. Implicit Flow

OpenID Connect implicit flow is shown in Figure 1. All dashed lines in the figure represent the redirection by browser and solid lines represent direct network calls. Parameters on lines are important data transmitted during this call.

Table I: OpenID Connect response_type Values

Response_type Value	Flow
code	Authorization Code Flow
id_token	Implicit Flow
id_token token	Implicit Flow
code id_token	Hybrid Flow
code token	Hybrid Flow
code id_token token	Hybrid Flow

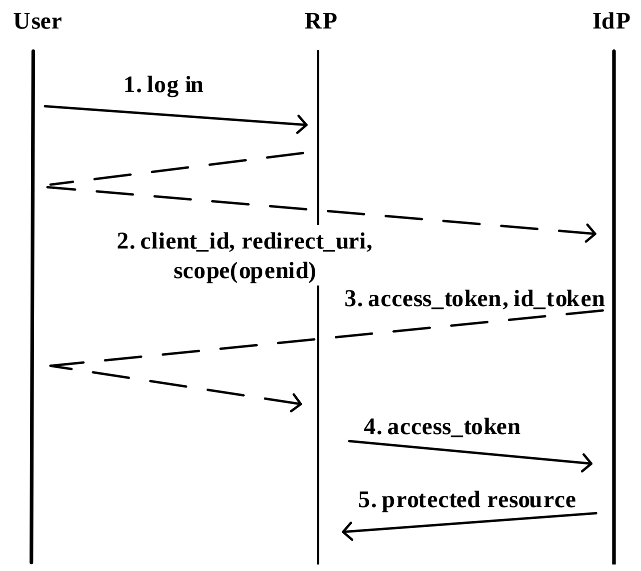
User
RP
IdP
1. log in
2. client_id, redirect_uri, scope(openid)
3. access_token, id_token
4. access_token
5. protected resource

Fig. 1: OpenID Connect 1.0 Implicit Flow

The OpenID Connect implicit flow is described as following steps:

- Step 1: User tries to log in RP.
- Step 2: RP constructs token request and redirects user to IdP. Client_id represents RP's identity, redirect_uri represents the RP's address waiting for token and scope represents the permissions RP required from IdP. In OpenID Connect protocol, scope must contain *openid*.
- Step 3: If IdP has authenticated user it is going to redirect user's authentication response. As response_type requires *token, id_token*, IdP's response contains access_token and id_token. RP can identify a user by id_token.
- Step 4, 5: RP is able to obtain user's protected resource from IdP by access_token.

II-A2. Other Flows

Authorization code flow is similar to implicit flow. IdP firstly sends RP the authorization code instead of tokens. Then RP need use the code and a secret shared by RP and IdP to exchange for tokens with IdP. Hybrid flow is the combination of implicit flow and authorization code flow. RP is able to obtain code and token from IdP at the same time.

II-B. Security Consideration

An RP must register a unique ID at IdP. To protect users' privacy, IdP should receive user's consent for specific RP before sending the PII to this RP. So IdP must get a valid ID from RP to represent RP's identity. And it has been discussed that the id token must be bound to the specific RP to avoid the reuse of token. It also requires that RP should provide the ID to IdP.

The redirect_uri registered at IdP can avoid a malicious opponent to get a user's id token. IdP compares the redirect_uri in the authentication request and uploaded during registration. Only when the redirect_uri in the authentication has been uploaded during registration IdP is going to send the id token to requester. It guarantees that only the owner of the registered redirect_uri is able to receive the id token issued for its registrant.

II-C. Dynamic Registration

Dynamic registration [13] is a function IdP provides RP to re-register its information at IdP. For dynamic registration, IdP issues each RP a registration token when the first registration of RP is finished. RP is able to register a new ID and redirect uri at IdP using the registration token.

To register a new RP at the IdP, firstly RP sends an HTTP POST message to the IdP with the parameters containing redirect uri, response type and other metadata parameters. This message is sent with the registration token. Upon successful registration, IdP generates a unique client id and returns it back with other registered metadata parameters.

III. SYSTEM OVERVIEW

PriOIDC allows user to conduct single sign on with out leaking login information to IdP. And even multi RPs' collusion can not trace the user. In this section, we are going to make an overview of our user privacy respecting protocol based on OpenID Connect 1.0.

III-A. Anonymity in OpenID Connect System

In OpenID Connect systems IdP gets RP's identity by client_id or redirect_uri in request (Figure 1 step 2) and gets user's identity when authenticating the user (Figure 1 step 3). As IdP has to provide RP a user's authenticator bound with user's identity, it's not possible to keep user anonymous in IdP without modifying the structure of current SSO system. So it is only feasible to protect user's privacy by keeping RP anonymous in IdP. But it introduces new challenges.

III-A1. Challenges

The simplest way to make RP anonymous in IdP is using random client_id and redirect_uri in each authentication. But the simple method will introduce some problems in two fields.

Using random client_id and redirect_uri results in the failure of authentication. In OpneID Connect system, IdP only accepts a request when the client_id and redirect_uri have been registered at IdP. So IdP will drop the request with random client_id and redirect_uri, in another word unregistered parameters, as the invalid request. Additionally to protect user's privacy from RPs' collusion attack, it's required that IdP should provide different user_ids for different RPs [3]. It means that user_id is bound to client_id, so random client_id means the user_id is random too. As RP wants to provide a user personalize service it must identify a user with a constant identity. So randomness of client_id is not appropriate for widely used SSO systems.

In the other field, anonymous RP with random client_id and redirect_uri causes secure problems. To avoid the misuse of id_token among different RPs, RP checks the validation of id_token. An client_id represents a specific RP's identity, a id_token with this client_id is only valid in this specific RP. But when using a random client_id, different RPs may share the same client_id. When a user log in a malicious RP, this RP possibly logs in other RPs with the user's id_token if they have the same client_id. Additionally redirect_uri is the address where RP waits for the id_token. Before issuing a id_token, IdP will check the validation of redirect_uri to avoid attacker getting the id_token. If the redirect_uri is random, IdP can no more protect user from sending id_token to an attacker.

III-A2. Solutions against the problems

With dynamic registration, a RP can register new random client_id and redirect_uri before sending a request to IdP for id_token. And to avoid IdP finding out RP's identity through dynamic registration, the requirement of registration token is omitted. IdP will delete the expired registration to reduce storage stress.

To identify a user in different logins, RP must have the ability to transform the user_id provided by IdP into a constant user identity for each user. Most of current SSO system generate user_id as a random character string. So a new user-id-generating algorithm has to be created for user authentication. As user_id is required to be bound to random client_id to protect from RPs' collusion, client_id should be the primary input parameter to user-id-generating algorithm. To make user_id able to be transformed into a constant user identity, it is a feasible way that generating client_id through a client-id-generating algorithm. The user-id-generating algorithm and client-id-generating algorithm will be described detailedly in Section IV.

Misuse of id_token only happens when different RPs use the same client_id. Although IdP will keep the registered client_id unique, an attacker is possible to be the executor of registration (RP or user) and tamper with the failed registration result. So victim will regard the repetitive client_id as a validate one. To prevent misuse of id_token, client-id-generating algorithm should require two random parameters respectively generated by RP and user. So even if an attacker possesses a user's id_token (or negotiates a client_id with RP), he is unable to negotiate the same client_id with a RP

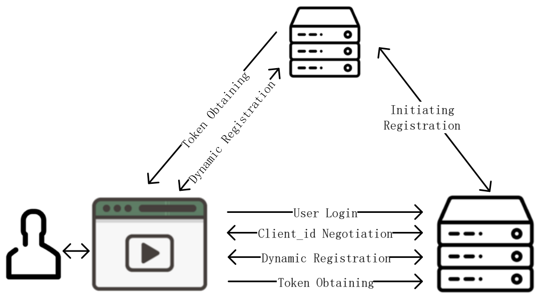

Fig. 2: Overview of System

(or get the id_token with same client_id from user).

As redirect_uri is random, IdP is going to send id_token to the invalidate address. User agent must intercept the id_token redirection from IdP and send id_token to RP. In PRISSO system IdP issues RP certification for each RP. A RP certification contains RP's identity and its address for token acceptance. User gets the real acceptance address of RP from certification and makes sure that the id_token is going to be sent to the RP. RP certification is also useful in defending phishing attack.

III-B. Overview of proposed scheme

The procedure of PRIOIDC can be divided into two parts: Initiating registration and Login procedure. Login procedure contains user login, client_id negotiation, dynamic registration and token obtaining. The overview is shown in Figure 2

— Initiating Registration: RPs and users register at IdP. IdP generates unique basic_client_id for each RP and unique user id for each user.
— User Login: User starts log in an RP. If user is labeled as logged in, RP is going to offer service to user. Otherwise RP requires user to start SSO procedure.
— Client_id Negotiation: For each SSO procedure, RP is going to start client_id negotiation with user. Client_id is a random number generated by client-id-generating algorithm unrelated with any RP. A client_id represents login from a user to an RP.
— Dynamic Registration: To make the client_id generated by negotiation between user and RP, user is going to register client_id at IdP by the dynamic registration API provided by IdP. IdP is going to check whether client_id is unique and ask RP to restart client_id negotiation for another client_id.
— Token Obtaining: After dynamic registration success, RP is going to redirect token request to IdP. IdP firstly authenticates user and then generates id token for RP. Id token contains RP's client_id and user id. Client_id is provided by RP and user id is generated through sser-id-generating algorithm by IdP. RP is able to get the constant user identity from user id.

III-C. Roles in PriOIDC

To achieve the goals outlined in Section I, the requirements and restrictions of abilities owned by each roles in single-sign-on system is defined as following:

- **User** is able to generate RP's client_id with RP by key data exchanging. User is able to register client_id in IdP. User is able to modify redirect_uri in token request and redirect token response to the correct RP's redirect_uri. User need not to store any data in its computer so that user is able to conduct single sign on in any computer.
- **RP** is able to generate client_id with user by key data exchanging. RP is able to check the client_id registration result. RP is able to get a constant user identity from id_token generated by IdP. RP is unable to find out its user's identity in other RPs even through RPs' collusion.
- **IdP** is able to generate user's id for specific login by client_id and user's unique id in IdP. IdP is unable to find out RP's identity by client_id. IdP is unable to find out the relevance between an RP's different client_ids.
- **User agent** is the software used by the user, such as browser and the application on the mobile device.

More detailedly, the enhanced protocol is going to provide new client-id-generating and user-id-generating algorithm: Client_id is random in each logins to make RP anonymous in IdP. There is a one-to-one correspondence between client_id and user_id provided by IdP so that user id is random. So RP is able to transform user_id IdP into constant user identity and RPs are unable to find the relationship between one user's user_ids for different RPs.

III-D. Threat Model

Considered different attack scenarios, malicious opponent can be divided into following situations: malicious IdP, malicious RP, malicious user and external attacker. In different situations malicious owns different abilities.

Curious IdP As IdP service is usually provided by a leading internet company. In consideration of in consideration of, Idp is considered secure but curious. Phishing attack on IdP is not considered. It means IdP would not try to do the impersonation attack or abduction attack. But an IdP is probably interested in user's login trace in RPs so that it is able to deduce user's interests and behavioral traits. During SSO login, as IdP need authenticate the user, so IdP has the ability to collect the user's information. And IdP is able to get client_id and redirect_uri from token request. IdP is also able to store each user's login history and analyze each client_id and redirect_uri to find out the relevance among each login.

Malicious RP There are two kinds of malicious RPs. The first is the legal RP owned by malicious opponent and the other is phishing site. Because everyone is able to register as an RP at IdP, it is considered that RP can be fully controlled by malicious. A malicious RP is going to conduct impersonation attack and privacy undermining attack on user. As some attack methods require attacker act as both RP and user, to avoid the repetitive description malicious RP and malicious user is defined: If attacker acts as an RP in attack, attacker is considered as malicious RP. If attacker only acts as a user, it's malicious user. A malicious RP's goals include: 1) Getting id_token from user which is validate in other RPs. 2) Deducing user's login trace by colluding with other RPs. Malicious RP is able to make fake basic_rp_id and conduct client_id negotiation with user. Malicious RP is also able to construct the id_token request to IdP and receive id_token from IdP. In phishing attack, it is considered that user trusts attacker completely.

Malicious User A malicious user is only going to conduct impersonation attack. In the attack malicious opponent acts as both user and RP, user is able to conduct client_id negotiation, construct dynamic registration request. User is also able to temper all the data transformed through itself. It is considered that the user agent is trustful, but there are external attacker trying to exploit the flaw of user agent.

IV. DESIGN OF PROTOCOL

IV-A. Client-id-generating and User-id-generating algorithm

Client-id-generating and User-id-generating algorithm are created based on Discrete Logarithm problem [14]. IdP carefully chooses a big prime p [15] for system. When a RP initialize registration at IdP, IdP will provide RP a unique primitive element module p. It's used in RP as the basic_rp_id and RP will generate another primitive g from basic_rp_id for further client_id negotiation. As p is a prime and a is a primitive element module p, if α is a relatively prime of *p-1*, $a^{\alpha} modp$ is another prime element module p.

For each login process, the user and RP negotiate the temporary $client_{id}$ for the RP registration at the IdP. While starting a login procedure, there is **Diffie-Hellman key Exchange** [16] between RP and user. Firstly RP sends $pk\_rp = g^{x} modp$ to user, and x is a random number. After receiving the pk_rp, user continue generating the random number y until $r = pk\_rp^{y} modp$ is a relative prime of *p-1*. Then user sends $pk\_user = g^{y} modp$ to RP so that both user and RP can get $r = g^{xy} modp$. So the client_id is generated as:

$$client\_id = basic\_rp\_id^{r} modp$$

such that client_id is another primitive element module p.

To identify users, IdP keeps a unique id for each user. After receiving a client_id, IdP will generate the one-to-one correspondence user_id

$$user\_id = client\_id^{id} modp$$

so

$$user\_id = basic\_rp\_id^{r \cdot id} modp$$

As r is a relative prime of *p-1*, according to **Extended Euclidean** algorithm RP can get r^{-1} and let $1 = r \cdot r^{-1} mod(p-1)$. While receiving user_id from IdP, RP can get a user identity

$$user\_rp\_id = user\_id^{r^{-1}} modp$$

so

$$user\_rp\_id = basic\_rp\_id^{id} modp$$

For one user in a RP, user_rp_id is constant. But user_rp_ids are disparate in RPs because basic_rp_ids are different in each RP.

IV-B. Login flow

User firstly logs in an RP. If RP find that user is unauthenticated, RP is going to negotiate a new client_id with user. Then user starts dynamic registration and forward the registration result from IdP to RP. If registration succeeds RP will construct a token request and redirect user to IdP. IdP authenticates user and generates an id_token of user for RP. Id_token is sent to RP and RP gets user_rp_id from id_token. RP is going to identify the user through user_rp_id.

IV-B1. Initiating Registration
If an RP wants to join the SSO system, it must do the initialization registration at IdP. As well as traditional SSO system, IdP is going to inspect the real identity of RP and store RP's information on IdP's server. During registration procedure RP sends its URL for id_token acceptance to IdP. IdP generates a unique primitive element module *p* for RP as basic_rp_id. Then IdP puts basic_rp_id, URL and the prime *p* together and encodes them to Json Web Token. This token is called rp_certificate. A typical rp_token carries the following information: [language=[ANSI]C,basicstyle=] `"alg": "RSA", "type": "certificate" . "iss": IdP URL, "sub": basic`$_r p_i d$,"$name$" : $RPname$,"$redirect_u ri$" : URL

Same as RP, user need to register at IdP. IdP is going to generate unique user id for each user during registration.

IV-B2. Client_id Negotiation
An attacker is able to be the man in the middle between RP and user in client_id negotiation using phishing attack. When a user logs in attacker's website, attacker logs in another RP as a user. In client_id negotiation, attacker just transmits user and RP's requests and responses to each other. As a result, attacker shares the same client_id with user and RP and gets a id_token valid in RP from user. So besides of generating client_id, RP has to send its rp_certificate to user in this phase. It protects user from sending id_token to malicious opponent. As rp_certificate contains RP's name, it allows user can identify the real RP's identity when doing login.

IV-B3. Dynamic Registration
User generates IdP's registration URL by *iss* from rp_certificate. The client_id negotiation is described in client-id-generating algorithm. Dynamic registration starts after client_id negotiation. User generates a random redirect_uri and sends it to IdP as well as client_id. IdP checks the uniqueness of client_id and sends the result success or fail back. If registration fails, user is going to restart client_id negotiation. Otherwise user will forward the registration response to RP.

IV-B4. Obtaining Token
RP firstly redirects user to IdP with its token request. User generates IdP's authenticate URL by *iss* from rp_certificate and compared it with RP's redirect location. If they point the same address, user is going to continue the login. To keep the advanced protocol same as OpenID Connect 1.0, after authenticating a user IdP is going to redirect the user to the redirect_uri of RP with id_token as parameter. As redirect_uri is random, user stops the redirection. User then sends id_token to the URL received from client_token negotiation to defend man-in-the-middle attack. RP gets user_id from id_token and gets user_rp_id computed from user_id. If it's the first time user logs in RP, RP is going to finish the registration. Otherwise RP searches user profile through user_rp_id.

V. SECURITY ANALYSIS

In SSO system, malicious opponent's attacks can be concluded into 3 goals:

1) Privacy undermining attack: Malicious opponent tries to get user's login trace on different RPs.
2) Impersonation attack: Attacker tries to log in RP as a victim's identity. In this way, attacker can get the full control of victim's account in RP.
3) Abduction attack: Attacker also tries to lead user to upload users personal information to it. To achieve this goal, there are two ways. The first is letting a victim log in an RP as attacker's identity. In this way, if the RP is online storage system, victim may upload its privacy data to attacker's account. The other way is phishing attack. A malicious RP disguises it as another RP and abducts user to upload some information.

V-A. Privacy undermining attack

PRIOIDC tries to protect user's privacy by keeping RP anonymous to IdP. IdP is able to get client_id and redirect_uri. As redirect_uri is generated by user, it will show nothing about RP. IdP can only undermine user's privacy by get RP's identity from client_id. It's described in Client-id-generating algorithm: $client\_id = basic\_rp\_id^r modp$. p is a large prime and basic_rp_id is a primitive element module p. And r is the random number generated by user and RP. IdP can only find out RP's real identity by finding out r^{-1} and let $1 = r \cdot r^{-1} mod(p-1)$, so that

$$basic\_rp\_id = client\_id^{r^{-1}} modp$$

But r is secret shared by user and RP, and according to **Discrete Logarithm** problem calculating r from client_id is difficult. So basic_rp_id is invisible to IdP. In other way if IdP gets a user's repeatedly login, it is going to find out whether they are about the same RP. If there are two client_ids from the same RP marked as $client\_id_1 = basic\_rp\_id^{r_1} modp$ and $client\_id_2 = basic\_rp\_id^{r_2} modp$. Client_id$_1$ and client_id$_2$ meet the following formula

$$client\_id_1 = client\_id_2^{r_2/r_1} modp$$

So that only when knowing r_1 and r_2 IdP can find out the relevance between Client_id$_1$ and client_id$_2$. But r_1 and r_2 are invisible to IdP. So IdP is never able to undermine user's privacy.

RPs try to find out user's login trace in three ways: 1) Getting the user's unique id in IdP. 2) Finding the relevance among user_rp_ids. 3) Deducing user's login trace from IP address. As user's id is used in generating user_id in id_token, RP is able to obtain $user\_id = client\_id^{id} modp$. Client_id is primitive element module p. Although client_id, user_id and p are known by RP, according to **Discrete Logarithm** problem calculating id from user_id is difficult. For different RPs, they are able to get user's user_rp_id. User_rp_ids from different RPs can be marked as $user\_rp\_id_1 = basic\_rp\_id_1^{id} modp$ and $user\_rp\_id_2 = basic\_rp\_id_2^{id} modp$. As basic_rp_id$_1$ and basic_rp_id$_2$ are primitive element module p, there is $0 < \alpha < p$ and $basic\_rp\_id_1 = basic\_rp\_id_2^{\alpha} modp$.So user_rp_id$_1$ and user_rp_id$_2$ meet the following formula

$$user\_rp\_id_1 = user\_rp\_id_2^{\alpha} modp$$

So RP is able to deduce the relevance between user_rp_id$_1$ and user_rp_id$_2$ only when knowing α. As basic_rp_id is generated by IdP and calculating α from basic_rp_ids, RP is never able to find the relevance. If an RP does not use the basic_rp_id from IdP, user is able to find it dishonest through rp_certificate and stop the login. Most of current users use dynamic IPs so that it is impossible to get user's login trace from user's IP.

V-B. Impersonation attack

RP conducts impersonation attack by getting user's id_token which is valid in other RPs. OpenID Connect protocol protect id_token from malicious RP by keep RP owns unique client_id and check RP's redirect_uri during login. Unique client_id makes one RP's id_token invalid in other RPs. And IdP only redirects id_token to it's relevant RP's redirect_uri registered in IdP so that attacker is never able get RP's id_token. There are three conditions for a malicious to try getting a validate id_token. 1) Malicious RP has already finished client_id negotiation with an RP as a user. As client_id is generated by both RP and user, malicious RP is unable to get the id_token with the same client_id. 2)Malicious RP has got a user's id_token, same as condition 1 malicious RP is unable to negotiate the same client_id with another RP. 3) Malicious RP acts as the man in the middle between RP and user. As RP sends its URL in rp_certificate user only sends its id_token to this URL so that attacker can never achieve id_token. As a summary, malicious is unable to conduct impersonation attack.

Malicious user is only able to conduct impersonation attack by tempering id_token. If attacker has already get victim's user_rp_id, attacker is able to calculate $user\_id = user\_rp\_id^{r} modp$. r is shared by RP and attacker. However id_token is protected by the signature generated by IdP so that it is impossible for attacker to log in RP as victim.

V-C. Abduction attack

To lead user to login an RP as attacker, attacker needs to make sure that user receive a malicious token from IdP. As https is used to protect parameters transforming between user and IdP, it's impossible to temper user's token during transmission. The other way to conduct the attack is phishing attack on IdP. In traditional SSO protocol such as OAuth 2.0 and OpenID Connect, it is possible for malicious to conduct phishing attack on IdP. As it is shown in 1 step 2, the request from user to IdP is built by RP. If an malicious RP set the IdP'url as its phishing site, an unwary user may input its id and password on the phishing website so that attacker is able to get the full control of user's account. In PriOIDC as RP_Cert contains IdP's url, user agent is going to compare the IdP's url in request and RP_Cert. If they are not matched, the request is deemed invalid.

Phishing attack on RP in SSO system is quite different from it in normal website. In SSO system even an unwary user has visited a phishing RP's website, IdP is going to ask user to make sure RP's identity in 1 step 2. The identity is bound with RP's client id and client id is bound with its redirect uri. If malicious RP constructs the request in 1 step 2 to IdP with its personal client id, user is able to find out the true identity of RP and protect itself from phishing attack. In traditional SSO system if malicious uses a client id of another RP, IdP is going to redirect user to the corresponding redirect uri. In PriOIDC user agent is going to compare redirect uri from RP with the redirect uri in RP_Cert.If uris are not matched, the request is regarded invalid. A phishing RP can never achieve another RP's token and never lead user to log in its website.

V-D. Discussion

An external attacker is also taken into account in SSO system. External attacker is able to capture and temper all the network flow through user, RP and IdP. External attacker's targets include impersonation attack, abduction attack and privacy undermining attack. If an attacker keeps its eye on a specific user, it is able to find that the user's login on different RPs. So it is easy for an external attacker to draw a user's login trace. Privacy protection is not effective for external attacker. To protect user from privacy leaking a proxy is probably a appropriate scheme. Proxy is able to mix multi-user's request and keep user's login trace invisible to attacker. User's dynamic IP makes proxy impossible to get user's login trace from user's IP External attacker is going to steal user's id_token from network flow to make the attack and it is also going to make the attack by temper user's id_token into attacker's id_token when id_token is transformed on the network. As all the network flows are protected by https, external attacker is unable to conduct the attacks.

VI. EVALUATION

The prototype system runs on Thinkcentre M8600t with an Intel Core i7-6700 CPU, 500GB SSD and 8GB of RAM running Windows 10.

VI-A. Implementation

Implementation of system contains modification of IdP as well as RP and creation of user agent. User agent runs on chrome 71.0.3578.98 as its extension.

System's parameters are carefully chosen in specification about **Difie-Hellman** algorithm. p is one of primes provided by the specification and a is its generator. All the primitive elements module p is generated by a.

Compared with formal openid connect system, the work we do is shown as following:

- Modifying RP registration so that IdP is able to offer RP_cert to RP.
- Providing RP's client_id negotiation interface.
- Providing RP's dynamic registration acceptance interface.
- Implementing user-id-generating algorithm at IdP.
- Implementing the function of getting user_rp_id from user_id at RP.
- Realizing function of client_id negotiation, dynamic registration, id_token transmitting and so on at user agent.

VI-B. Storage

As the prime p is 2048-bit-length, storage of client_id, user_id and user_rp_id are no larger than 512 Bytes as hexadecimal. We consider they are all 512 Bytes in evaluation.

For IdP and RP's user Personally Identifiable Information (PII) storage, it changes from a short user id into a 512 Bytes id. It is assumed that an IdP owns 100 million users and an RP owns 10 million users. If a user's PII costs 500 Bytes extra storage so that IdP need to offer 50 billion Bytes (less than 50 GB) storage and RP need to offer 5 billion Bytes (less than 5 GB) storage. The extra cost of storage can be omitted.

For IdP's dynamic registration storage, the data contains RP's client_id and redirect_uri. We consider that each dynamic registration data cost no more than 550 Bytes storage. And for each client_id IdP can set the lifetime of validity. It is assumed that for each client_id its lifetime is 2 minutes and during 2 minutes there are 1 million requests for dynamic registration. So IdP need to offer 550 million Bytes (about 500 MB) storage for dynamic registration. The extra cost of storage can be omitted.

For user's login log stored in RP and IdP, RP and IdP are able to transform PII into a shorter hash characters. So it almost cost no more extra storage.

Table II: Benchmark Result

phase	time (ms)
Client_id Negotiation (RP)	49
Client_id Negotiation (user)	2967
Dynamic registration (IdP)	16
Dynamic registration (user)	1001
Obtaining Token (IdP)	369
Obtaining Token (RP)	19
Network Cost	12
Total Time	4433

VI-C. Timings

Table II shows the result of the time cost in PRISSO's each phases. We log in the prototype 100 times and figure out the average time cost. It can be found that the most of time consumed in client_id negotiation phase, dynamic registration conducted by user and IdP providing id_token. They cost 4337ms in average which is more than 90% of total time. In client_id negotiation to confirm $r = pk\_rp^y mod p$ is a relative prime of $p-1$ user has to continue generating y until r is validate which costs most of time. In dynamic registration user need check validation of basic_rp_id and IdP's URL by rp_certificate, calculate client_id by basic_rp_id, r and check the result of registration and forward it to RP. In SSO system if user firstly log in an RP it is necessary for user to confirm permission of login in the specific RP. It is showed as user has to press the confirm button in IdP's website. In PRISSO client_id is random so that every login for a user is first login. So every login requires user to press a button redundantly. Even the press action is conducted by chrome extension, it costs some time.

We also do login in traditional OpenID-Connect system 100 times and get a total time cost 44ms in average. Compared with traditional system, PRISSO's time cost is about 100 times.

VI-D. Optimizing

As the most time cost is in client_id negotiation and dynamic registration and these two phases are transparent to user. To reduce time cost we move client_id negotiation and dynamic registration to website initiation. When user visit RP's login page user agent conducts client_id negotiation and dynamic registration during page loading. So for a user its login procedure starts at obtaining token and network time cost is halved. The total time cost is about 406ms and the system possesses practicability.

VII. RELATED WORKS

In 2014, Chen et al. [10] concludes the problems developers may face to in using sso protocol. It describes the requirements for authentication and authorization and different between them. They illustrate what kind of protocol is appropriate to authentication. And in this work the importance of secure base for token transmission is also pointed.

In 2016, Daniel et al. [6] conduct comprehensive formal security Analysis of OAuth 2.0. In this work, they illustrate attacks on OAuth 2.0 and OpenID Connect. Besides they also presents the snalysis of OAuth 2.0 about authorization and authentication properties and so on.

Besides of OAuth 2.0 and OpenID Connect 1.0, Juraj et al. [17] find XSW vulnerabilities which allows attackers insert malicious elements in 11 SAML frameworks. It allows adversaries to compromise the integrity of SAML and causes different types of attack in each frameworks.

Other security analysis [5] [7] [8] [18] [19] on SSO system concludes the rules SSO protocol must obey with different manners.

In 2010, Han et al. [20] proposed a dynamic SSO system with digital signature to guarantee unforgeability. To protect user's privacy, it uses broadcast encryption to make sure only the designated service providers is able to check the validity of user's credential. User uses zero-knowledge proofs to show it is the owner of the valid credential. But in this system verifier is unable to find out the relevance of same user's different requests so that it cannot provide customization service to a user. So this system is not appropriate for current web applications.

In 2013, Wang et al. proposed anonymous single sign-on schemes transformed from group signatures. In an ASSO scheme, a user gets credential from a trusted third party (same as IdP) once. Then user is able to authenticate itself to different service providers (same as RP) by generating a user proof via using the same credential. SPs can confirm the validity of each user but should not be able to trace the users identity.

In 2018, Han et al. [21] proposed a novel SSO system which uses zero knowledge to keep user anonymous in the system. A user is able to obtain a ticket for a verifier (RP) from a ticket issuer (IdP) anonymously without informing ticket issuer anything about its identity. Ticket issuer is unable to find out whether two ticket is required by same user or not. The ticket is only validate in the designated verifier. Verifier cannot collude with other verifiers to link a user's service requests. Same as the last work, system verifier is unable to find out the relevance of same user's different requests so that it cannot provide customization service to a user. So this system is not appropriate for current web applications.

BrowserID [22] [23] is a user privacy respecting SSO system proposed by Molliza. BrowserID allows user to generates asymmetric key pair and upload its public to IdP. IdP put user's email and public key together and generates its signature as user certificate (UC). User signs origin of the RP with its private key as identity assertion (IA). A pair containing a UC and a matching IA is called a certificate assertion pair (CAP) and RP authenticates a user by its CAP. But UC contains user's email so that RPs are able to link a user's logins in different RPs.

SPRESSO [12] allows RP to encrypt its identity and a random number with symmetric algorithm as a tag to present itself in each login. And token containing user's email and tag signed by IdP is also encrypted by a symmetric key provided by RP. During parameters transmission a third party credible website is required to forward important data. As token contains user's email, RPs are able to link a user's logins in different RPs.

All the SSO system protocols above are quite different from current popular SSO protocol. So it is difficult for IdPs and RPs to remould their system into new protocols.

VIII. CONCLUSION

IX. REFERENCES

[1] Dick Hardt, "The oauth 2.0 authorization framework," *RFC*, vol. 6749, pp. 1–76, 2012.

[2] Scott Cantor, Internet2 John Kemp, and Nokia Rob Philpott, "Assertions and protocols for the oasis security assertion markup language," .

[3] J. Bradley N. Sakimura, NRI, "Openid connect core 1.0 incorporating errata set 1," https://openid.net/specs/openid-connect-core-1_0.html#CodeFlowSteps.

[4] Daniel Fett, Ralf Küsters, and Guido Schmitz, "An expressive model for the web infrastructure: Definition and application to the browser ID SSO system," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, 2014, pp. 673–688.

[5] Rui Wang, Shuo Chen, and XiaoFeng Wang, "Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services," in *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, 2012, pp. 365–379.

[6] Daniel Fett, Ralf Küsters, and Guido Schmitz, "A comprehensive formal security analysis of oauth 2.0," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, 2016, pp. 1204–1215.

[7] Yuchen Zhou and David Evans, "Ssoscan: Automated testing of web applications for single sign-on vulnerabilities," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, 2014, pp. 495–510.

[8] Hui Wang, Yuanyuan Zhang, Juanru Li, and Dawu Gu, "The achilles heel of oauth: a multi-platform study of oauth-based authentication," in *Proceedings of the 32nd*

Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016, 2016, pp. 167–176.

[9] Paul A Grassi, M Garcia, and J Fenton, "Draft nist special publication 800-63c federation and assertions," *National Institute of Standards and Technology, Los Altos, CA*, 2017.

[10] Eric Y. Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague, "Oauth demystified for mobile application developers," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, 2014, pp. 892–903.

[11] Mozilla Developer Network (MDN), "Persona," https://developer.mozilla.org/en-US/docs/Archive/Mozilla/Persona.

[12] Daniel Fett, Ralf Küsters, and Guido Schmitz, "SPRESSO: A secure, privacy-respecting single sign-on system for the web," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, 2015, pp. 1358–1369.

[13] J. Bradley N. Sakimura, NRI, "Openid connect dynamic client registration 1.0 incorporating errata set 1," https://openid.net/specs/openid-connect-registration-1_0.html.

[14] Peter Shiu, "Cryptography: Theory and practice (3rd edn), by douglas r. stinson. pp. 593. 2006. (hbk) 39.99. isbn 1 58488 508 4 (chapman and hall / crc).," *The Mathematical Gazette*, vol. 91, no. 520, pp. 189, 2007.

[15] Tero Kivinen and Mika Kojo, "More modular exponential (MODP) diffie-hellman groups for internet key exchange (IKE)," *RFC*, vol. 3526, pp. 1–10, 2003.

[16] Whitfield Diffie and Martin E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[17] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen, "On breaking SAML: be whoever you want to be," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, 2012, pp. 397–412.

[18] Ronghai Yang, Guanchen Li, Wing Cheong Lau, Kehuan Zhang, and Pili Hu, "Model-based security testing: An empirical study on oauth 2.0 implementations," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*, 2016, pp. 651–662.

[19] Hui Wang, Yuanyuan Zhang, Juanru Li, Hui Liu, Wenbo Yang, Bodong Li, and Dawu Gu, "Vulnerability assessment of oauth implementations in android applications," in *Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA, December 7-11, 2015*, 2015, pp. 61–70.

[20] Jinguang Han, Yi Mu, Willy Susilo, and Jun Yan, "A generic construction of dynamic single sign-on with strong security," in *Security and Privacy in Communication Networks - 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings*, 2010, pp. 181–198.

[21] Jinguang Han, Liqun Chen, Steve Schneider, Helen Treharne, and Stephan Wesemeyer, "Anonymous single-sign-on for n designated services with traceability," in *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, 2018, pp. 470–490.

[22] Daniel Fett, Ralf Küsters, and Guido Schmitz, "Analyzing the browserid SSO system with primary identity providers using an expressive model of the web," in *20th European Symposium on Research in Computer Security (ESORICS)*, 2015, pp. 43–65.

[23] Daniel Fett, Ralf Küsters, and Guido Schmitz, "An expressive model for the web infrastructure: Definition and application to the browserid SSO system," *CoRR*, vol. abs/1403.1866, 2014.