PriOIDC: A Client-Access-Hidden Extension for OpenID-Connect

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 the leakage of users' credentials. Therefore, SSO systems are widely deployed and integrated. The survey on the top 100 websites from SimilarWeb [**?**] demonstrates that 25 websites excluded those which are not available for browser accessing do not integrate the SSO service.

However, SSO systems introduces new privacy issues in authentication which exposes the users' login traces. An adversary has the ability to track users' login trace in multiple ways. It is described more detailedly:

- The adversary is able to act the IdP who always knows which RP a user has accessed;
- Multiple RPs controlled by the adversary may link the same user who logs in each RPs.

The leaking of users' login trace can be easily avoided in traditional authentication systems by using different usernames for multiple services.

Currently the SSO systems widely accepted by users, such as Google Identity and Facebook Login are provided by the large enterprise who has already held the significant quantity of users' data. For example, Facebook collects user data, such as where you live, your age, gender, level of education, employment details, language and so on, used for commercial purpose or something possibly even worse, such as political purpose. It is reported in 2016 the company Cambridge Analytica utilized the 50 million people's profiles leaked by Facebook to build the portrait of voters, in order to target them with personalised political advertisements [**?**]. Google and Facebook seem to become the real Mr. Know It All, as they know who you are, where you live, what you interest in and so on, as long as you have to use the service provided by them. However, what is even worse is that they apparently are interested in what you have done in other applications without of their control. For example, Google would like to offer $20 gift card for those people who are willing to accept the Screenwise Meter [**?**] (mobile app and web extension) which allows Google to see what you are doing in other applications. However, with Google Identity service (Google's SSO system), it has the ability to surveil what applications people accessed even without any additional payment, which is to be used to draw their portraits. Similarly, the controller of multiple services has the ability to link the users using these services which respectively hold partial users' sensitive profiles to build the users' portraits.

Several schemes are proposed to protect users' privacy. To avoid users from being linked by multiple RPs, it requires that the user's identifier in one RP should never be the same with or derivable from the ones of other RPs to prevent a possible correlation among users from multiple RPs. For example, in OIDC and SAML a Pairwise Pseudonymous Identifier (PPID) or named Pairwise Subject Identifier is suggested to be generated by the IdP for the user in each RP [**?**], [3]. It is not defined by neither OIDC nor SAML how to generate the distinct user identifier. For example, in MITREid Connect, the open OIDC implementation, PPID is generated by the `Java.Util.UUID` provided since JAVA 1.5 and specifically bound with corresponding RP identifier.

Additionally, to avoid users from being tracked by IdP, two SSO systems (BrowserID [11] and SPRESSO [12]) are proposed to hide the user's accessed RPs from IdP who fails to obtain the information of accessed RP. In BrowserID, the identity proof is signed with the private key generated by user while the corresponding public key is bound with users' email by IdP who need not obtain the information of accessed RP. In SPRESSO, RP uses the encrypted RP domain and a nonce as the identifier, so that the real identity of RP is never exposed to IdP.

However, there is no existing SSO system protecting users' privacy comprehensively. Currently proposed designs for privacy protection are only available in dealing with one privacy leaking problem, none of these systems are able to deal with the both privacy issues at the same time. Therefore, it means that the adversary always have at least one way to track users.

Widely deployed SSO systems (e.g. OIDC) are unable to hide RPs' identity from IdP for security considerations. Firstly, the identity proof should only be sent to the correct RP, which prevents the adversary from performing the imper-

sonation 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. However, although BrowserID and SPRESSO achieve the goal of hiding RP from IdP, distinct user identifiers are not available in these systems. As distinct user identifier has to be bound with specific RP, to decide which user identifier is to be used for specific authentication, IdP has to know which RP the request is from. Therefore, 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 the first scheme which deals with all the privacy issues introduced by SSO comprehensively. Recluse enables the RP to hide its identity from IdP for users authentication, as well as IdP is able to provide distinct user identifiers for each RP. To achieve the above goals, we proposed the scheme for RP and user identifier generating, which allows that, 1) RP has the ability to offer the changing RP identifiers to IdP in each authentication, from which the real RP identity is not possible to be derived without the trapdoor; 2) IdP is able to generate unique user identifier (`user_idp_id`) bound with specific RP identifier, from which the user identifier in RP(`user_rp_id`)is to be derived with the trapdoor. However, multiple RPs are unable to link the user by `user_rp_id` or `user_idp_id`.

Moreover, Recluse is implemented based on OIDC with the support of Dynamic Registration [15]. For OIDC system the Reluse only requires: (1) a new set of public parameters and web interfaces are provided additionally; (2) the new RP identifier and PPID generating algorithm is supported. Compared with BrowserID and SPRESSO, Recluse does not only deal with the privacy issues comprehensively but also be compatible with traditional OIDC system, which is not achieved by neither BrowserID and SPRESSO. BrowserID requires that the user's identity proof should be generated by user's browser, as well as SPRESSO has to introduce new trustful party into the system.

To deal with the security considerations introduced by hiding RP in OIDC: 1) the identity proof should only be sent to the correct RP; 2) identity proof should not be misused. The following requirements should be fulfilled by Recluse:

- 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 authentications are different, and IdP fails to infer RP's information or link it in different authentications. Moreover, neither RP nor the user may control the generated identifier, which avoids the misuse of the identity proof. The detailed analysis is provided in Section V.
- A browser extension is introduced to transmit the messages (i.e., request and response) related with the authentication, which 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.

We build the prototype system by running the Recluse IdP on the modified MITREid Connect, RP on the SpringMVC frame work and extension on chrome browser. Finally we prove the availability of the Recluse and evaluate the delay introduced by Recluse.

The main contributions of Recluse are as follows:

- We propose a new scheme which deals with all the privacy issues introduced by SSO comprehensively. It has the ability to prevent IdP from tracking users' login trace, as well as multiple RPs are unable to link the users either.
- We developed the prototype of Recluse. The evaluation demonstrates the effectiveness and efficiency of Recluse. We also provide a systematic analysis of Recluse to prove that Recluse introduces no degradation in the security of Recluse.

The rest of this paper is organized as follows. We introduce the background and the threat model in Sections II and **??**. Section IV describes the design and details of Recluse. 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 draw the conclusion finally.

II. Background

Recluse is an extension of OIDC to prevent the IdP from inferring the user's accessed RP, with the security of SSO systems under consideration. This section provides the necessary background information about OIDC and adopts OIDC as the example to present the security consideration of SSO systems.

A. OpenID Connect

OpenID Connect (current version 1.0) is an extension of OAuth (current version 2.0). The OIDC or OAuth systems contains following entities:

- **User** is the entity to be authenticated in this system who holds the credentials for the IdP. User takes part in the the system through the user agent.
- **User agent** is the software used by the user, such as browser and the application on the mobile device. User agent is required to transmit the authentication request and identity proof between IdP and RP correctly.
- **IdP** is the entity who authenticates the user and provide the identity proof. IdP authenticates the user, verifies the

authentication request from RP, generates user's PPID and issues the identity proof signed with its private key. Besides, IdP provides the notification to user about the range of exposed attributes to RP and guarantees that the identity proof should only be sent to the corresponding RP.

- **RP** is the entity who provides the service and need to identify the user. RP builds the authentication request to IdP with its identifier and endpoint for identity proof. RP identifies a user through the PPID in identity proof.

OAuth is originally designed for authorizing the RP to obtain the user's personal protected resources stored at the resource holder. That is, the RP obtains an access token generated by the resource holder after a clear consent from the user, and uses the access token to obtain the specified resources of the user from the resource holder. However, plenty of RPs adopt OAuth 2.0 in the user authentication, which is not formally defined in the specifications [1], [14], and makes impersonation attack possible [8], [10]. For example, the access token isn't required tp be bound with the RP, the adversary may act as a RP to obtain the access token and use it to impersonate as the victim user in another RP.

OIDC is designed to extend OAuth for user authentication by binding the identity proof for authentication with the information of RP. OIDC provides three protocol flows: authorization code flow, implicit flow and hybrid flow (i.e., a mix-up of the previous two flows). In the authorization code flow, the identity proof is the authorization code sent by the IdP, which is bound with the RP, as only the target RP is able to obtain the user's attributes with this authorization code and the corresponding secret.

The implicit flow of OIDC achieves the binding between the identity proof and the RP, by introducing a new token (i.e., id token). In details, id token includes the user's PPID (i.e., `sub`), the RP's identifier (i.e., `aud`), the valid period and the other requested attributes. The IdP completes the construction of the id token by generating the signature of these elements with its private key, and sends it to the correct RP through the redirect URL registered previously. The RP validates the id token, by verifying the signature with the IdP's public key, checking the correctness of the valid period and the consistency of `aud` with the identifier stored locally. Figure 1 provides the details in the implicit flow of OIDC, where the dashed lines represent the message transmission in the browser while the solid lines denote the network traffic. The detailed processes are as follows:

- Step 1: User attempts to login at one RP.
- Step 2: The RP redirects the user to the corresponding IdP with a newly constructed request of id token. The request contains RP's identifier (i.e., `client_id`), the endpoint (i.e., `redirect_uri`) to receive the id token, and the set of requested attributes (i.e., `scope`). Here, the `openid` should be included in `scope` to request the id token.

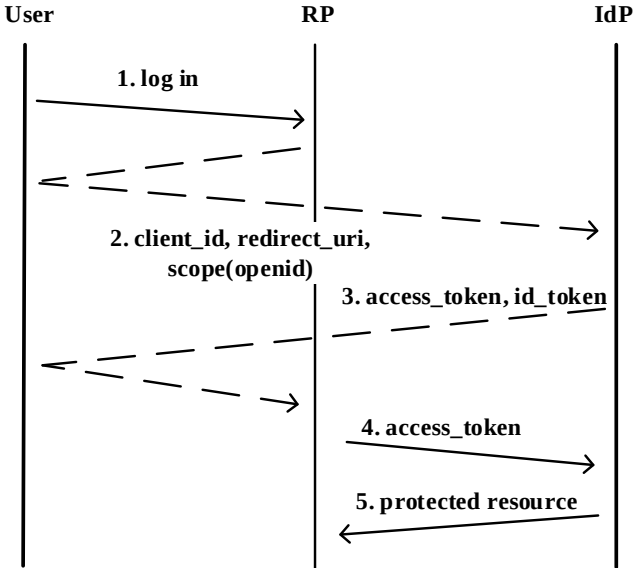

Fig. 1: The implicit protocol flow of OIDC.

- Step 3: The IdP generates the id token and the access token for the user who has been authenticated already, and constructs the response with endpoint (i.e., `redirect_uri`) in request if it is the same with the registered one for the RP. If the user hasn't been authenticated, an extra authentication process is performed.
- Step 4, 5: The RP verifies the id token, identifies the user with `sub` in the id token, and requests the other attributes from IdP with the access token.

Dynamic Registration. The id token (also, the authorization code) is bound with the RP's identifier. OIDC provides the dynamic registration [15] mechanism to register the RP dynamically. After the first successful registration, RP obtains a registration token from the IdP, and is able to update its information (e.g., the `redirect URI` and the response type) by a dynamic registration process with the registration token. One successful dynamic registration process will make the IdP assign a new unique client id for this RP.

B. Security Consideration

Widely deployed SSO systems, such OIDC, are designed with the following security considerations, and various implementations of IdP and RP are also analyzed with the same security principles under the assumption that IdP is trusted. Here, we list the security considerations:

- **Content Checking:** The contents in the identity proof are generated under a clear consent of the user. The contents include the RP's information and the range of exposed attributes.
- **Confidentiality:** The confidentiality of the identity proof is ensured, that is, only the target RP obtains the identity proof which will never be leaked by the honest RP. The

HTTPS connection is used to protect the identity proof between the IdP and the user, while the trusted user agent (e.g., the browser) ensures the identity proof only sent to the correct URL (of RP) which is confirmed by the user and the IdP.
- **Integrity:** No one except the IdP is able to construct a valid identity proof Any modification in the identity proof makes the identity proof invalid.
- **Binding:** The identity proof is only valid for the target RP, as it is bound with only the target RP, and the honest RP has the ability to verify the consistency.

III. Challenges and Solutions

As SSO systems introduce the novel way for the RPs to identify the user, the authentication security and users' privacy should be considered.

A. Threat Model

In SSO systems, an adversary tries to break the authentication security in following ways:

- Impersonation Attack: Adversaries log in to the honest RP as the honest user.
- Identity Injection: Honest user logs in to the honest RP under adversaries' identity.

Besides, the adversary also has the interests in users' login traces, the private issues introduced by SSO systems. To undermine a user's privacy, the adversary tries to achieve the following goals:

- Adversary finds out which RP a user has accessed by acting as the honest IdP.
- Adversary links the same user in multiple RPs by controlling these RPs.

In SSO systems, IdP has the max authority in this system. Therefore, IdP should be considered honest but curious. Otherwise, an malicious IdP has the ability to log in to any RP as any honest user (impersonation attack) and enforce any honest user to log in honest RP under an adversary's identity (identity injection). Moreover, a user's login trace is never able to be hidden from collusion between IdP and RP. It is considered that any RP could be corrupted and any user may be the adversary. User agent is considered completely honest but under control of the user. Therefore, the user agent is seemed as a part of user. The ability of each entity acted by adversary are shown as follows:

- **Curious IdP** acts as an completely honest IdP.
- **Malicious RP** has the ability to build any response, as well as the authentication request, for user's requestion.
- **Malicious User** is able to intercept and tamper all the data transmitted through itself.
- **Network Attacker** has the ability to listen all the IP address on the Internet but unable to tamper any the network flows as they are protected by various ways, such as TLS.

However, Identity Injection only occurs when 1) IdP is dishonest; 2) the transmission between RP and IdP is corrupted by either corrupted user agent or unprotected network flows. Therefore, Identity Injection is not considered.

B. Challenges

To protect users from the privacy issues introduced by SSO system, the scheme should simultaneously achieve the following goals:

- Hiding RP's identity from IdP.
- Providing distinct user identifier for each RP.

However, it will introduce prominent challenges.

As it has been described in Section I, it is required to expose the identifier of users' accessed RP for security consideration. Hiding RP's identity from IdP breaks the security considerations listed in Section **??**.

- **Break Binding:**
- **Break Confidentiality:**
- **Break Content Checking:**

Additionally, it introduces another challenge to provide distinct user identifier while hiding RP's identity

- **RP is unable to identify the user:**

In OIDC, the authentication request from RP to IdP exposes RPs' identity by the parameters, `client_id` (RP's identifier) and `redirect_uri` (RP's endpoint for token receiving). To hide accessed RP from IdP requires that `client_id` and `redirect_uri` should not represent an RP any more. As these parameters have played important roles in authentication, it introduces prominent challenges.

Challenges in Authentication Procedure. The `client_id` and `redirect_uri` are required in both authentication request and users' identifier proof generation, so that the omission of these parameters introduces challenges in authentication procedure. The challenges are as follows:

- OIDC is designed for the centralized systems. Therefore, prior coordination is required between RP and IdP so that RP registers its individual attributes (i.e., `redirect_uris`) and gets client attributes (i.e., `client_id`) issued by IdP. While the authentication request is transmitted from RP, IdP verifies the validation of `client_id` and `redirect_uri` because it only provide service to those RPs already registered. Therefore, if an RP builds the authentication request without `client_id` and `redirect_uri`, IdP considers it invalid.
- The identity proof issued (called `id_token`) by IdP provides the PPID (called `sub`) of the user for RP. It means that the `sub` is solely bound with an RP to avoid linking the user through RPs' collusion. However, once the IdP doesn't know the RP accessed by user, it is impossible for IdP to provide the corresponding PPID for RPs. Therefore, RP has no ability to identify a user any more.

Challenges in Security. In addition to playing the important roles in authentication procedure, the client_id is required to avoid the misuse of id_token and the redirect_uri guarantees the id_token should be transmitted to corresponding RP. The of client_id and redirect_uri introduces the following challenges:

- While RP registers with IdP, IdP gets the redirect_uris which lists all the endpoints of RP waiting for id_token. Therefore, once IdP receives an authentication request from the RP, it compares the redirect_uri in request with the registered endpoints. If the redirect_uri has not been registered before, IdP considers this request malicious. This means if the user is honest and transmissions among user, IdP and RP are well protected (e.g., using TLS), the attacker has no possibility to get the id_token. However, without the endpoint provided by RP, IdP is unable to guarantee the id_token is properly transmitted to the corresponding RP.
- The id_token includes the RP's identifier (called aud), user's PPID and IdP's identifier (called iss), representing the id_token is issued by iss for aud to confirm that the owner's (of this id_token) identity is sub. Therefore, RP compares the aud with its own identifier to make sure whether this id_token is issued for itself. However, once IdP doesn't know the correct identifier of RP, the id_token issued by IdP doesn't include the correct aud any more. It results in the misuse of id_token.

Solutions. To fulfill the requirement of client_id and redirect_uri when building the authentication request without exposing RP's identify, the simplest way is using random client_id and redirect_uri. However, using the completely random parameters still faces the above challenges, so that several methods should be proposed to deal with these challenges. The methods are as follows:

- The RP's identifier should be generated beyond any entities' control to avoid the misuse of user's identity proof, which only happens when different RPs use the same client_id. Attackers tries to obtain an honest user's id_token in an honest RP through various methods, such as leading the honest RP use the same client_id with a corrupted one, which will be described detailedly in Section V. Therefore, the appropriate method to generate the id_token is negotiation between the user and RP.
- To identify a user in different logins, RP should have the ability to transform the user's PPID into a constant user identifier for each user. Some OIDC systems only use a random character string as the PPID (e.g., MITREid Connect). Therefore, the new user PPID generating algorithm should be created for user authentication while only the corresponding RP has the trapdoor to derive the unique identifier from PPID. To protect users from linkage by RPs' collusion, both the PPID and the unique identifier should be bound with the (not completely) random RP's identifier. The PPID generating algorithm and RP's identifier generating algorithm will be described detailedly in Section IV.
- With dynamic registration, an RP has the ability to re-register the new client_id and redirect_uri with IdP. Therefore, it is needed that before the authentication request is transmitted to IdP, RP should re-register the newly generated client_id and completely random redirect_uri with IdP. The registration should be conducted by the user to avoid direct interactive between RP and IdP. However, the specification [15] of OIDC dynamic registration requires the registration request should carry a bearer token as well as the new client_id is generated by IdP. To avoid IdP finding out RP's identity through dynamic registration, the requirement of registration token should be omitted. It is also needed to enable RP to assign the specific client_id. It is observed that although client_id is defined to be generated by IdP, some OIDC systems (e.g., MITREid Connect) enable the client_id be the input attribute.
- Using the random redirect_uri makes the responsibility to check whether the redirect_uri has been registered is migrated to user's agent (e.g., browser) as it is impossible for IdP to guarantee the id_token is properly transmitted to the corresponding RP any more. However, to avoid the query for registered endpoints list from user agent exposing the RP's identity, IdP should issue the RP Certification for each RP which includes the RP's basic attributes and the registered endpoints list with a signature. User agent should have the ability to take the responsibility to check whether the redirect_uri has been registered through the RP Certification.

IV. Design of PRIOIDC

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.

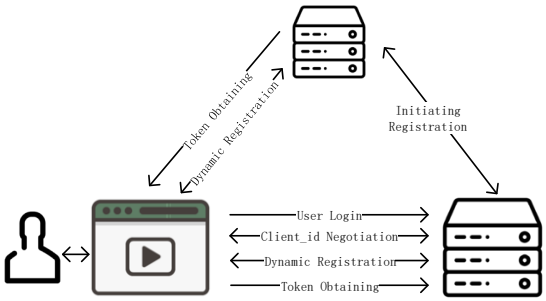

Fig. 2: Overview of System

— 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.

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

Client-id-generating and User-id-generating algorithm are created based on Discrete Logarithm problem [16]. IdP carefully chooses a big prime *p* [17] 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*, $a^{\alpha} mod p$ 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** [18] between RP and user. Firstly RP sends $pk\_rp = g^x mod p$ to user, and *x* is a random number. After receiving the pk_rp, user continue generating the random number *y* until $r = pk\_rp^y mod p$ is a relative prime of *p-1*. Then user sends $pk\_user = g^y mod p$ to RP so that both user and RP can get $r = g^{xy} mod p$. So the client_id is generated as:

$$client\_id = basic\_rp\_id^r mod p$$

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

$$user\_id = client\_id^{id} mod p$$

so

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

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

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

so

$$user\_rp\_id = basic\_rp\_id^{id} mod p$$

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.

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.

1) 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:

```
{
    "alg": "RSA",
    "type": "certificate"
}.
{
    "iss": IdP URL,
    "sub": basic_rp_id,
    "name": RP name,
    "redirect_uri": URL
}
```

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

2) 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.

3) 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.

4) 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.

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.

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 $user\_id = user\_rp\_id^r mod p$. 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.

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.

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.

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.

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.

TABLE I: 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

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.

C. Timings

Table I 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 $r = pk\_rp^y modp$ 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.

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. [4] 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. [19] 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 [6] [7] [8] [20] [21] on SSO system concludes the rules SSO protocol must obey with different manners.

In 2010, Han et al. [22] 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 user's identity.

Anonymous SSO schemes prevents the IdP from obtaining the user's identity for RPs who do not require the user's identity nor PII, and just need to check whether the user is authorized or not. These anonymous schemes, such as the anonymous scheme proposed by Han et al. [23], allow user to obtain a token from IdP by proving that he/she is someone who has registered in the Central Authority based on Zero-Knowledge Proof. RP is only able to check the validation of the token but unable to identify the user. In 2018, Han et al. [23] 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 [13] [24] 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

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, "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.

[5] Daniel Fett, Ralf Küsters, and Guido Schmitz, "The web SSO standard openid connect: In-depth formal security analysis and security guidelines," in *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, 2017, pp. 189–202.

[6] 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.

[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] 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.

[14] Michael B. Jones and Dick Hardt, "The oauth 2.0 authorization framework: Bearer token usage," *RFC*, vol. 6750, pp. 1–18, 2012.

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.