Recluse: A Privacy-Respecting Single Sign-On System Avoiding User Linkage and Tracing

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 [1] demonstrates that only 25 websites (excluding the ones not for browser accessing) do not integrate the SSO service.

In addition to the convenience for both the users and RPs, current SSO systems introduce a new privacy leakage risk for the users. Instead of maintaining the user's information (including identifier) independently in those systems not integrating the SSO service, the IdP maintains the user's attributes and identity proof in SSO systems, which allows the IdP or the colluded RPs to infer the access trace of a specified user [2]. In details, the privacy leakage risks include:

- Identity linkage [3]–[5], the colluded RPs may link a user if the user's identifiers (generated by the IdP) in these RPs are the same or derivable from the others, and use the attributes maintained in each RP to profile a user.
- Access tracing [5]–[7], the IdP knows which RP a specified user has accessed, as the construction and transmission of the identity proof provide the IdP the identifiers and URLs of the user's accessed RPs.

The privacy leakage allows the widely deployed SSO systems (e.g., Google Identity and Facebook Login) to perform the long-term profile of the users based on the information of accessed RPs. Firstly, Google and Facebook collect the accessed RPs' identifiers without any limit, as the RP's information is necessary in the construction of identity proof; Secondly, it's feasible for Google and Facebook to model the user's behaviours and monitor the specified users, unlike the user session re-identification in DNS queries where the similarities is needed to classify the user's queries [8], IdP (e.g., Google and Facebook) may classify the multiple access traces based on the maintained user's unique identity. Thirdly, Google and Facebook are curious about users' behaviours in each RP (may be the competitors), for example, Screenwise Meter [9] and Onavo [10] are provided to collect all the traffic of the victim. Moreover, the users' attributes are analysed for various purposes, for example, 50 million people's profiles were leaked by Facebook and utilized by Cambridge Analytica to build the portrait of voters for personalised political advertisements [11].

Various schemes [3], [4], [6], [7] are proposed to protect the user's privacy in SSO systems. However, they either achieve the identity unlinkage [3], [4], or prevent the IdP from tracing the users [6], [7].

To prevent the colluded RPs from performing the identity linkage, the straightforward solution is that the user's identifier in one RP should never be the same with or derivable from the ones of other RPs, which is already specified in the widely adopted SSO standards (e.g., OIDC and SAML), called a Pairwise Pseudonymous Identifier (PPID) in OIDC [4] and Pairwise Subject Identifier in SAML [3]. This requirement is also widely satisfied in various SSO implementations. For example, in MITREid Connect (an open-source OIDC implementation), PPID is a random sequence generated by the `Java.Util.UUID` provided by Java and the binding between the PPID and the RP is only maintained by the IdP, which ensures PPID for different RP is independent from each other and no one except the IdP and the corresponding RP may infer the RP from PPID.

To prevent IdP from tracing the RPs accessed by the user, two SSO systems (BrowserID [6] and SPRESSO [7]) are proposed to hide the user's accessed RPs from IdP in the construction and transmission of identity proof. In BrowserID, the identity proof is signed with the private key generated by user, and transmitted to the RP through the user directly, while the corresponding public key is bound with users' email by IdP who needs 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, while the identity proof is transmitted to the RP through a trusted entity (named FWD) who doesn't know the user's identity.

However, none of existing SSO systems hide the user's trace from both the IdP and colluded RPs, the curious IdP obtains the users' traces in OIDC and SAML [3], [4],

while the colluded RPs link the user with the same email address in BrowserID [6] and SPRESSO [7]. The fundamental challenges to hide the user's trace from both the IdP and colluded RPs are as follows:

- The identity proof should be bound with one RP and transmitted only to this RP, avoiding the misuse of identity proof by the malicious RPs. In SAML and OIDC, the binding and transmit check are performed by the IdP, which makes it fail to prevent access tracing.
- The identity proof should be bound with one user, avoiding the misuse of identity proof by the malicious users. Moreover, the identities for one user in one RP should be the same or derivable from the previous ones, allowing the RP to provide the continuous service. In BrowserID [6] and SPRESSO [7], the email address is adopted as the identities, which is the same among different RPs and allows the RPs to infer the user's trace through the identity linkage.

Moreover, BrowserID and SPRESSO are both redesigns of SSO systems, and therefore incompatible with existing widely deployed SSO systems (e.g., OAuth, OIDC and SAML). 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.

In this paper, we propose a privacy-REspecting Single Sign-On System aChieving UnLinkable USErs' Traces from both the IdP and RPs, named Recluse. To achieve this, we rely on the user to achieve the trusted transmit and check of identity proof as in BrowserID [6] and SPRESSO [7], and propose two algorithms to achieve:

- RP's identifier generation , which makes the RP's identifier in multiple authentications 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.
- PPID generation, 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.

Moreover, Recluse is implemented based on OIDC with the support of Dynamic Registration [12]. 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.

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 have analyzed the privacy issues in SSO systems systematacially, and propose a scheme which hides the user's trace from both the IdP and RPs, for the first time.
- 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 **??** 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 VII, then introduce the related works in Section IX, 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, which 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 [13], [14], and makes impersonation attack possible [15], [16]. 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.
- 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 [12] 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.

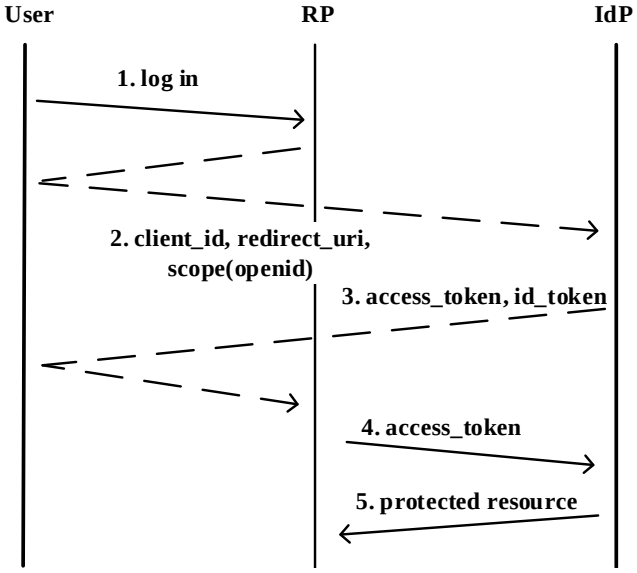

Fig. 1: The implicit protocol flow of OIDC.

B. Security Consideration

It has been described in [7] that 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.

It is proved by the analysis of current widely deployed SSO systems [15]–[21] as the attacks on the authentication illustrated in these works should be classified into these two categories.

The impersonation attack allows the adversary obtain the full control of user's account in the RP so that adversary has the ability to do anything it wants in the identity of the user. The identity injection leads the honest user log in an honest RP in the identity of the adversary which might leak the privacy of the user, for example, the user might log in the file hosting service such as OneDrive or iCloud as the adversary unconsciously and uploads the personal files to the adversary's account. To deal with the threat of the adversary, widely deployed SSO systems, such OIDC, are designed with the following security considerations, and various implementations of IdP and RP are also analysed 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. For example, it has been described in [15] that the content checking provides a way for user to check the identity of the RP which is able to avoid the leaking of user's authentication information. It can be utilized by the adversary while the Binding (described later) is also broken to obtain the user's identity proof which makes the impersonation attack possible.

- **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. For example, it has been illustrated in [15], [17] the SSO systems must transmitted the identity proof to the corresponding RP and illustrated in [16] that teh transmission must be protected by various ways such as TLS. Otherwise, it also results the leakage of user's identity proof.
- **Integrity:** No one except the IdP is able to construct a valid identity proof. And any modification in the identity proof makes the identity proof invalid. The analysis in [16] claims that the identity proof in SSO systems must be the private and unforgeable information provided by the IdP to avoid the impersonate attack. It is also required that the identity proof should not be tampered during transmission to avoid the identity injection.
- **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. It has been described in Section **??** that the binding of RP and identity proof avoid the adversary to use the identity proof of an honest user received by the corrupted RP to log in other honest RPs (impersonation attack).

*However, the privacy considered in these works is the protected resources held by the IdP which is not necessary in authentication. Besides, the adversary also has the interests in users' login traces, the private issue introduced by SSO systems described in Section **??**. 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 controlled by adversary.

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, 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 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. Moreover, as network flows are protected by various ways, such as TLS, the network attacker is not considered. 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.

To explicitly illustrate how an adversary works in the SSO system, the `authentication flow` is created to defined the authentication of specific IdP, RP and user. For example, now there are IdP, $User_A$, $User_B$, RP_A and RP_B, who are able to form 4 `authentication flows`, (IdP, $User_A$, RP_A), (IdP, $User_A$, RP_B), (IdP, $User_B$, RP_A) and (IdP, $User_B$, RP_B). An adversary has the ability to act one or more entities in single or multiple `authentication flows`. That is, an adversary is able to act as, i) the single entity in one `authentication flow`, such as the curious IdP; ii) the same entity in multiple `authentication flows`, such as acting as different RPs for the same honest user ; iii) the different entities in multiple `authentication flows`, such as acting as the RP for the honest user and the user for the honest RP at the same time. However, it is considered an adversary should not act as both the IdP and RP in single `authentication flow`.

B. Challenges

To protect users from the privacy issues introduced by SSO systems, 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 II.

- **Breaking Binding:** To hide RP's identity, IdP is unable to know which RP the identity proof is issued for. Therefore, the identity proof is no longer bound with the specific RP, which results in the misuse of identity proof. An adversary has the ability to achieve an honest user's identity proof by various ways, for example, once the user logs in the corrupted RP controlled by the adversary

with his/her identity proof, the adversary is able to access other honest RPs with the honest user's identity by using this identity proof (Impersonation Attack).

- **Breaking Confidentiality:** To hide RP's identity, IdP is unable to know the correct endpoint provided by the RP to receive the identity proof. For example, in OIDC, IdP holds the list of all the endpoints of RP waiting for `id_token`, so IdP is able to guarantee that the identity proof is only to be sent to the endpoint in this list. Without the endpoint representing RP's identity, an RP controlled by the adversary has the ability to build the authentication request by setting another honest RP's identifier (if RP's identifier is used in a way without exposing RP's identity) and the adversary's endpoint. IdP is to send the identity proof issued for the honest RP to the adversary. Therefore, the adversary has the ability to achieve the identity proof valid in honest RPs, which results in Impersonation Attack.
- **Ignoring Content Checking:** To hide RP's identity, IdP is unable to know the RP's unique real name, which represents the RP. Therefore, the notification of target RP's identity to user is no longer provided by IdP. An adversary has the ability to utilize this vulnerability as well as breaking confidentiality to achieve honest user's valid identity proof for other honest RPs (Impersonation Attack). Additionally, as SSO systems require user's clear consent for certain login to specific RP, the phishing attack can be avoided in some situations by RP's name checking. The ignoring of content checking breaks the protection from phishing attack.

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

- **RP is unable to identify the user:** To hide RP's identity, the single RP's multiple authentication requests should be considered from different RPs by IdP. However, the user identifier provided by IdP is solely bound with an RP to avoid linking the user through RPs' collusion. It means that the single user's multiple identifiers for one RP will never be constant. Therefore, RP is unable to identify the user no longer.

C. Solutions

To deal with the challenges introduced by hiding RP's identity from IdP, the following methods are proposed:

- **Providing the RP identifier which is only valid in corresponding RP without exposing RP's real identity.** The RP identifier should be generated in a way so that for each authentication the identifiers are different. Moreover, the generation should be beyond any entities' control to avoid the misuse of user's identity proof, which happens when different RPs use the same identifier. Therefore, we propose the scheme that the identifier should be generated under the negotiation between the user and RP. However, in this way, the malicious RP and user have the ability to build any negotiation requests and responses they need. Adversaries try to the honest RP use the same identifier with a corrupted one to obtain an honest user's identity proof valid in other honest RPs, which will be analysed detailedly in Section V. Moreover, the curious IdP tries to derive the real identity of RP from the RP identifier. It is also to be analysed in Section V.
- **Providing the distinct user identifier which make RP able to identify the user.** It is required that the user identifier provided by IdP (named `user_id`) should be different in each authentication, but RP is able to derive the specific user identifier for each R-P (named `user_rp_id`) which is constant for each RP with the `user_id`. However, the current ways to generate user identifier, such as using random character string as user identifier and binding it with specific RP identifier in database, is not appropriate. Therefore, we proposed a novel `user_id` generating algorithm associated with RP identifier generating which allows only the corresponding RP has the trapdoor to derive the `user_rp_id` from `user_id`. In this way, malicious RPs try to link the user by the `user_id`, which is also to be analysed in Section V.
- **Binding the RP identifier with RP's attributes.** It is required that the identity proof issued for specific RP identifier should be sent to the corresponding endpoint. However, RP identifier is generated temporarily by user and RP unrelated with any RP, so that IdP is unable to decide which endpoint the identity proof should be sent to. Therefore, the enhanced user agent is required to guarantee the identity proof's transmission without introducing new trustful entity into SSO system. It is required that the RP identifier generation should be based on the basic identifier element (named `basic_rp_id`) issued by RP, as well as the `basic_rp_id` should be bound with specific endpoint. Moreover, if IdP publishes the relationship of all the RPs on its website, unless user agent caches all the relationship, the access for specific RP's relationship is to expose the user's accessed RP. Therefore, IdP should offer the certification signed with its private key for each RP, which contains the RP's `basic_rp_id` and endpoint list. User agent should have the ability to verify this certification. Similarly, the responsibility of notifying user with RP's identity should be shifted to user agent. Same as binding the RP identifier with correct endpoint, RP's certification should contains RP's name and user agent should show it to user clearly while authenticating.

IV. Design of Recluse

The overview of login flow is shown in Figure 2, which contains RP identifier negotiation, dynamic registration and token obtaining. The of each phase in login flow is shown as

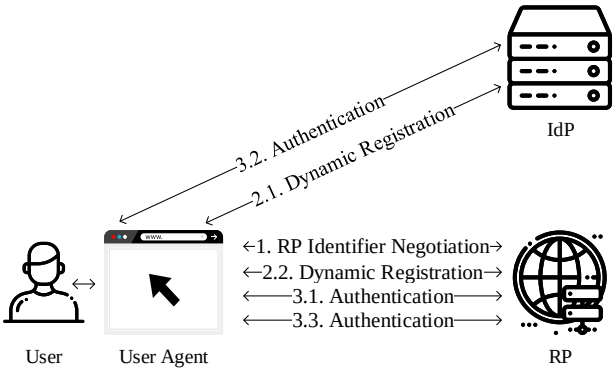

Fig. 2: Overview of System

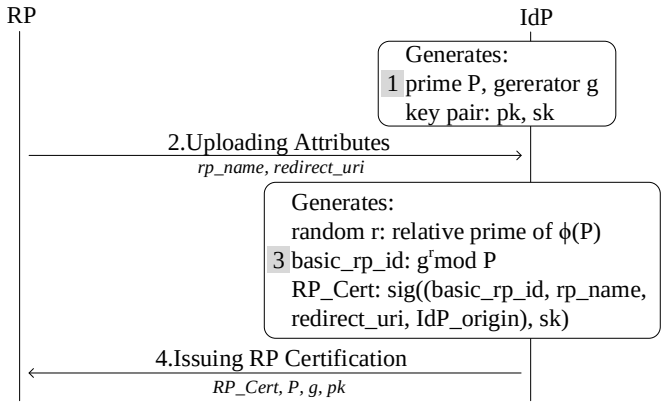

Fig. 3: Prior Registration

follows:

1. RP Identifier Negotiation: The negotiation is required to generate the RP identifier. For each SSO procedure, user is going to start negotiation with user. RP identifier is a random number which does not represent any RP, generated by rp-id-generating algorithm. However, the identifier is bound with specific authentication which is able to be confirmed by user and RP. The details of identifier generation is shown in Section **??**.
2. Dynamic Registration: Dynamic registration is required to in Recluse to make sure the identifier generated by negotiation is valid in IdP. In OIDC system, RP should register its attributes (e.g., the endpoint for identity proof) with IdP and obtain the RP identifier generated by IdP so that IdP is able to authenticate the user for specific RP. Therefore, to make sure the newly generated RP identifier in RP Identifier Negotiation is valid in IdP, the dynamic registration is required before the authentication request is transmitted to IdP. IdP is to check whether the identifier is unique and require RP to restart identifier negotiation if the identifier has be used by another RP.
3. Authentication: After dynamic registration, RP builds the authentication request and redirects it to IdP through user agent. After receiving the request, IdP firstly authenticates user and then issues identity proof for RP, which contains the user id generated through the user-id-generating algorithm. Then IdP redirects the identity to RP through user agent, and RP identify the user through identity proof. The methods of user identifier generation and RP identifying the user are shown in Section **??**.

However, in addition to the login flow in each authentication, the initial registration is required before the RP and user integrates the Recluse service. The initial registration allows the RP to achieve the necessary parameters from the IdP and user to sign on in the IdP. In Recluse system, the initial registration is conducted by each RP and user only once, however, the login flow is conducted in each authentication integrally each time.

A. Initial Registration

The initial registration between RP and IdP is shown as Figure 3. The registration process is as follows:

1. Uploading Attributes: Firstly, IdP generates its prime P, the primitive root g of P, used for RP and user identifier generation and the key pair pk, sk.
2. The RP uploads it attributes, such as its name, endpoint, identity proof (e.g., business license) and so on.
3. Issuing RP Certification: IdP verifies the identity of RP and generates the RP certification including $basic\_rp\_id$, rp_name, `redirect_uri` and `IdP_origin`. The RP certification is used for user agent to verify the basic attributes, such as $basic\_rp\_id$ and `redirect_uri`.
4. IdP returns the RP certification, P, g and pk to RP.

The initial registration required for IdP to verify the basic attributes of RP, such as name, endpoints for identity proof, so that Idp is able to provide the RP certification to RP which includes the unique identifier for each RP and its attributes. With the RP certification, user agent has the ability to verify the RP's endpoint for identity proof and notify user with RP's identity. Additionally, the parameters, prime P (used for user id generating) with its generator g, public key of IdP pk is provided in registration as well. Same as RP, user need to register with IdP and IdP generates unique user id for each user.

B. Rp-id-generating and User-id-generating algorithm

The rp-id-generating and user-id-generating algorithm are created based on Discrete Logarithm problem [22]. IdP carefully chooses a big prime P and its primitive root `g` as generator for system. When the RP registers with IdP, IdP provides a unique primitive root as the RP's root identifier (called $basic_rp_id$).

For each login process, the user and RP negotiate the temporary RP identifier bound with specific authentication.

[1] P is generated as $P = q \cdot 2 + 1$, while q is prime as well.

1) rp_id generating: While starting a login procedure, there is Diffie-Hellman key Exchange [23] between RP and user, through which the random r is generated. However, to make sure that there is r^{-1}, that $r \cdot r^{-1} = 1 mod \phi(P)$, r should be the relative prime of $\phi(P)$, so that if r is even r should be added by one. Although there is little possibility that r is the multiple of p or q, it is not considered in the illustration. However, the re-negotiation is required in the practical system if r is the multiple of p or q. The RP identifier is generated as:

$$rp\_id = basic\_rp\_id^r mod P \tag{1}$$

such that rp_id is another primitive element module p. And r^{-1} is generated through Extended Euclidean algorithm.

2) $user\_id$ generating: IdP labels each user at IdP with the unique identifier called $baisc\_user\_id$. To generate the specific user identifier for each rp_id, the algorithm is

$$user\_id = rp\_id^{basic\_user\_id} mod P \tag{2}$$

so

$$user\_id = basic\_rp\_id^{r \cdot basic\_user\_id} mod P \tag{3}$$

3) Trapdoor: While receiving $user\_id$ from IdP, RP is able to derive the constant user identifier from if

$$user\_rp\_id = user\_id^{r^{-1}} mod P \tag{4}$$

so

$$user\_rp\_id = basic\_rp\_id^{(1 mod \phi(P)) \cdot basic\_user\_id} mod P \tag{5}$$

so

$$user\_rp\_id = basic\_rp\_id^{basic\_user\_id} mod P \tag{6}$$

For single user in a RP, $user\_rp\_id$ is unchanged. However, $user_rp_id$s are distinct in each RP because $basic_rp_id$s are different in each RP.

C. Login Flow

The login flow is shown as Figure 4, in which the rp_id is generated in step 6, the $user\_id$ is generated in step 15 and the RP derives the $user\_rp\_id$ in step 18.

1) RP Identifier Negotiation: RP identifier negotiation starts form step 1 to step 7. The user accesses the service provided by RP in his/her browser. To log in this RP, user needs to click the login button offered by Recluse. Firstly, the user agent sends the `Start Negotiation` request to RP, so that RP generates the random sk_rp and $pk\_rp = g^{sk\_rp} mod P$ as the private key and public key for DH Key exchanging. Secondly, RP builds the `Negotiation Response` with newly generated pk_rp as well as the `RP_Cert` issued by IdP. User agent similarly generates random sk_user and pk_user, and $r = pk\_rp^{sk\_user} mod P$. However, to make sure that r is the relative prime of $\phi(P)$, it is required that r should be odd and the greatest common divisor of r and $\phi(P)$ is 1. Then user agent continues the `Negotiation` sending pk_user and r to RP. RP generates the local r in the same way as user agent and compares the local r and user agent generated r. If rs are equal, RP generates $rp\_id = basic\_rp\_id^r mod P$, as well as r^{-1} through Extend Euclidean algorithm, which meets $r \cdot r^{-1} = 1 mod \phi(P)$. Finally RP transmits the rp_id to user agent.

2) Dynamic Registration: Dynamic registration is from step 8 to step 13. While user agent receives teh rp_id from RP, it is required the rp_id from RP should be equal with it generated by user agent. Then user agent generates the `fake_uri` which contains the random string and keeps it for further identity proof transmission. User agent sends the `Dynamic Registration` request to IdP with newly generated rp_id and `fake_uri` and redirects the `Dynamic Registration Response` to RP.

3) Authentication: Authentication is from step 14 to step 19. After dynamic registration, RP builds the `Authentication Request` including rp_id as well as the `redirect_uri` representing the endpoint, and redirects it to IdP through user agent. User agent tampers the authentication request, compares rp_id with the local one, verifies the validation of the `redirect_uri` and replaces it with the fake one. Then user agent transmits the `Authentication Request` to IdP. After receiving the request, IdP firstly authenticates user and then generates $user\_id = rp\_id^{basic\_user\_id} mod P$. The identity proof signed with IdP's private key including the $user\_id$ is redirected to the `fake_uri` through user agent, who intercepts the transmission and transmit it to the endpoint `redirect_uri` in authentication request. Finally, RP derives the constant $user\_rp\_id$ from $user\_id$. If the $user\_rp\_id$ has already been registered, RP send `Authentication Finished` with the message `success` to user agent.

V. Security Analysis

In order to prove the privacy and security properties of Recluse system, we firstly demonstrate that for any adversary in the system, a user's access to an specific RP untraceable from it to another one. Besides, we illustrate the potential attacks discussed in the previous work about SSO security and the security issues introduced by Recluse.

A. Privacy

We now define the untraceability of SSO system. It is in the SSO system, for an adversary controlling curious IdP, it is impossible to inspect whether two `authentication flow`s are to same RP or not, however, for an adversary controlling malicious RPs, it is impossible to inspect whether two `authentication flow`s are from same user or not.

Assuming there are two `authentication flow`s from the same user to the same RP. In Recluse system, for the curious IdP, the only parameter related with RP is rp_id, as other parameters are related or generated by user, such as

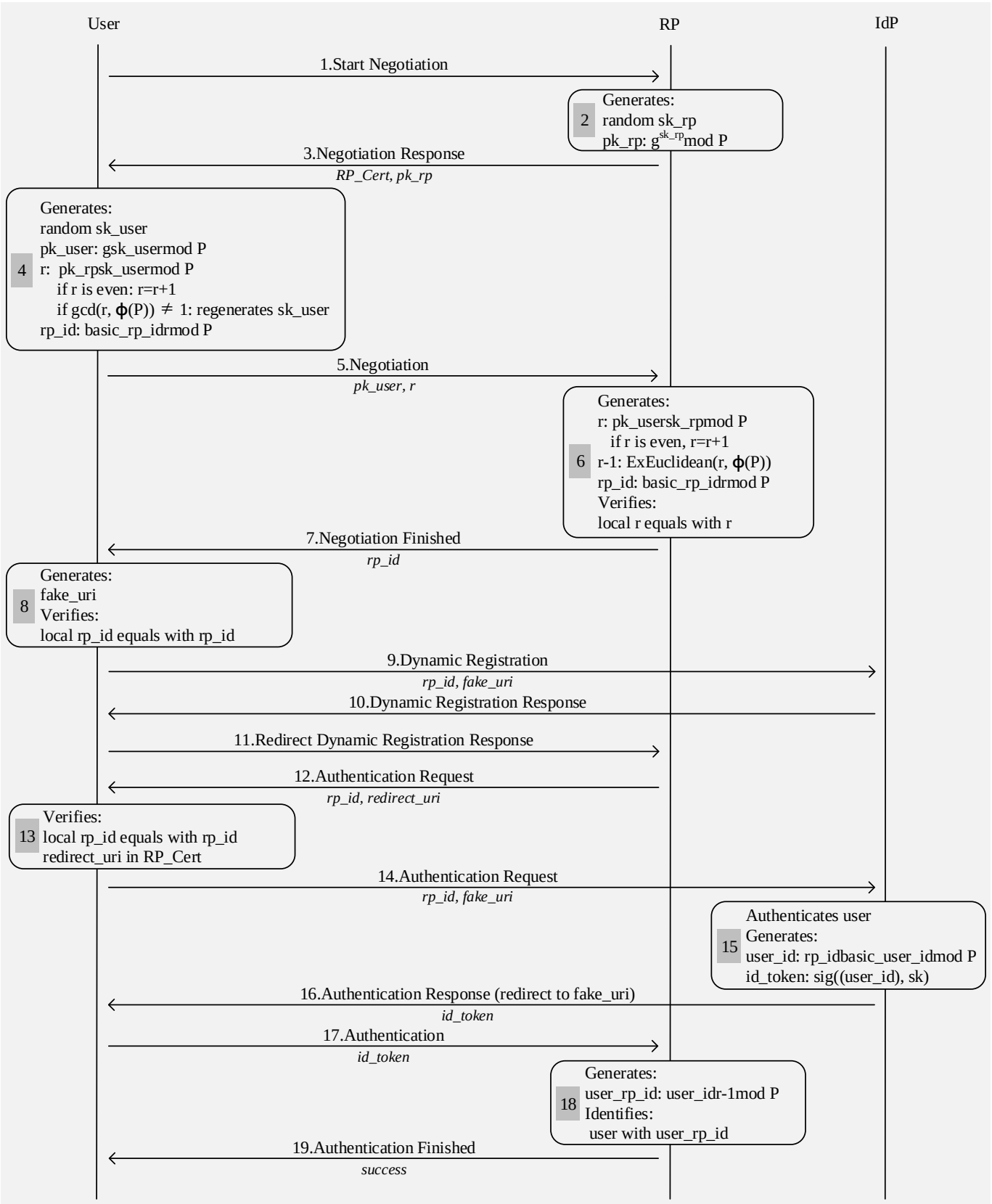

Fig. 4: Login Flow

$fake\_uri$. To break the untraceability, IdP tries to derive the $basic\_rp\_id$ of RP from rp_id or inspect whether the rp_ids are generated by the same $basic\_rp\_id$. However, as rp_id is generated by the formula (1), so

$$basic\_rp\_id = rp\_id^{r^{-1}} modP \tag{7}$$

However, r and r^{-1} is unknown to the IdP, so that IdP is unable to derive the $basic\_rp\_id$ from the rp_id. Moreover, even the IdP suspects that the rp_id is generated by the specific RP, it is impossible for IdP to verify it as the rp_id can be generated based on any primitive root of P. Besides, assuming that the rp_ids in different `authentication flow`s are rp_id_1 and rp_id_2 generated by r_1 and r_2. There is

$$rp\_id_1 = rp\_id_2^{r_1/r_2} modp \tag{8}$$

So only the entity who carries the r_1 and r_2 is able to verify whether rp_id_1 and rp_id_2 generated by the same RP.

Inspecting whether the users in two `authentication flow`s are the same user relies on the $baisc\_user\_id$ or the relation between $user_id$s or $user\_rp\_ids$. Assuming the same user log in different RPs where the $user_id$s are $user\_id_1$ and $user\_id_2$, $user_rp_id$s are $user\_rp\_id_1$ and $user\_rp\_id_2$, and $basic_rp_id$s are $basic\_rp\_id_1$ and $basic\_rp\_id_2$. We define that $\alpha = \log_{basic\_rp\_id_2} basic\_rp\_id_1$. There is

$$user\_rp\_id_1 = user\_rp\_id_2^{\alpha} \tag{9}$$

and

$$user\_id_1 = user\_id_2^{\alpha \cdot r_1/r_2} \tag{10}$$

The α is unknown to the malicious, so that the adversary is unable to inspect whether the two `authentication flow`s are from the same user. However, the malicious also tries to lead the same user in two `authentication flow`s using the same $user\_rp\_id$ or $user\_id$. According to formula (2) and (4), the user should use the same $basic\_rp\_id$ or rp_id in two `authentication flow`s. So the $user\_rp\_id$ is impossible to be same as $basic\_rp\_id$ is issued by IdP and verified by user agent. And rp_id is generated through the negotiation between user and RP, so that RP is unable to lead the user to use the same rp_id in different `authentication flow`s.

B. Security Consideration in OpenID Connect

As it has been defined in [4] Section 16, the security consideration of the OIDC design contains authentication and authorization. Now we list the security consideration of authentication and prove that Recluse achieves this goals.

1) **Server Masquerading.** The malicious server might masquerade as the RP or IdP using various ways through which user might leak its identity proof for RP or credentials on IdP. The Recluse mitigate this threat in following ways. 1) The server visited through user agent is authenticated by HTTPS; 2) The user agent verifies the RP certification which makes sure that the token is only sent to the corresponding RP instead of the masqueraded one; 3) The user agent only visit the IdP's endpoint listed in the RP certification which avoid the access to the masqueraded IdP.
2) **Token Manufacture/Modification.** The adversary might generate a bogus token or tamper the contents in the existing token, which enables the adversary log in any RP as any honest user. The receiver of token must have the ability to verify whether the token is issued by IdP without any modification or not. The token used in Recluse is signed by IdP with its private key, so that the token is unable to be constructed or modified.
3) **Access/ID Token Disclosure.** The adversary might try to obtain an honest user's token to the honest RP through various ways, which enables the adversary log in this RP as the honest user. It is required the tokenisi never exposed to anyone except the corresponding user and RP.
4) **Access/ID Token Redirect.** The malicious RP might use the token from an honest user to access other RPs as this user only if the token is also valid in other RPs. It is required the token issued for specific RP should bound with this RP which means the RP has the ability to verify whether a token is valid in itself. The token used in Recluse containing the rp_id and $user\_id$ is solely bound with specific RP and user, which is checked by the RP.
5) **Issuer Identifier.** The issuer identifier contained in the token should be completely same as it provided by the IdP, so that the verifier of token has the ability to obtain the corresponding public key. It is also implemented in Recluse.
6) **TLS Requirements.** To avoid network attacker the transmission in OIDC system should be protected by TLS. It is implemented in Recluse.
7) **Implicit Flow Threats.** In OIDC implicit flow, token is transmitted through user agent and the TLS protections are only between user agent and IdP, and between user agent and RP. It is required the token should not be leaked to the adversary by the user agent. The user agent of Recluse is deployed based on the Chrome browser as the trust base.

It is discovered that the security consideration of authentication in OIDC can be included into the security consideration in Section II as table **??**. however, the threats introduced by Recluse which breaks the security consideration and the methods to mitigate these threats has also been discussed in Section **??**.

C. Related Security Analysis

307 Redirect. It has been discussed in [17] that IdP might redirect the user to the RP immediately after the user inputs the credentials. For example, the HTTP response to the user's

TABLE I: Security Consideration

Security Consideration of SSO	Security Consideration of OIDC
Content Checking	Server Masquerading
Confidentiality	Server Masquerading, Access/ID Token Disclosure, TLS Requirements, Implicit Flow Threats
Integrity	Token Manufacture/Modification, Issuer Identifier
Binding	Access/ID Token Redirect

POST message with `username` and `password` might be the redirection to RP carrying user's identity proof. That is, as long as the 307 status code is used for this redirection, the user's credentials are also transmitted to the RP. However, in Recluse the redirections are intercepted by the user agent and rebuild the HTTP GET request to RP or IdP which is unable to leak the POST data of the user.

IdP Mix-Up. It has been illustrated in [17] that the adversary might intercept the the user's access to RP and modify the user's choice of IdP, assuming that RP integrates the SSO service provided by both the honest IdP (named `HIdP`) and malicious IdP (named `MIdP`). The user obtains the `access token` and `authorization code` from the `HIdP` and uploads them to the RP, however, the RP considers that the `access token` and `authorization code` are issued by the `MIdP`. Therefore, the RP tries to exchange for the protected resources using the `access token` and `authorization code` with `MIdP`. With this, the adversary carries the honest user's `access token` and `authorization code` which make the adversary able to log in this RP as the identity of the honest user. However, the threat is mitigated by the OIDC implicit flow, as the `ID token` issued by IdP contains the issuer identifier, so that RP is able to find out which IdP the token is from.

Cross-Site Request Forgery (CSRF). The CSRF attack on the RP makes the identity injection possible, through which the adversary might lead the honest user to upload the adversary's `ID token` to the RP. However, in Recluse the cross origin request should be repudiated by both RP and IdP excepted the request from the origin of the user agent. Therefore the CSRF attack on Recluse is impossible.

VI. Implementation

The Recluse prototype consists of the IdP modified from the traditional OIDC IdP, the user agent and the RP using the SDK of Recluse.

The implementation of Recluse IdP is based on the MITREid Connect, which is the open-source OpenID Connect implementation in Java on the Spring framework, one of the most popular MVC frame work. Until July 30, 2019, the project of MITREid Connect on github owns 233 uses and 994 stars. It has already been certified by the OpenID Foundation as well.

The implementation of user agent is based on the Chrome extension, the function provided by Google for developers to create the plug-in for Chrome browser. The main programming language of Chrome extension is JavaScript. The cryptographic computing of user agent is provided by the jsrsasign, which has 6878 uses and 1986 stars on github.

The implementation of RP SDK is also based on the Spring framework and the cryptographic computing is provided by the Java Platform, Standard Edition. An RP is able to use the service provided by the Recluse conveniently with this SDK. However, to make it convenient to evaluate the time cost of prototype system, we build the RP based on the Spring frame work instead of modifying the open-source RP implementation.

The modification of IdP introduces about 5 lines of code changing, including deleting 1 line about verifying the authority of dynamic registration, deleting 1 line about getting user identifier from database which is replaced by 3 lines about generating it through the user-id-generating algorithm. Additionally, to make the CORS (Cross-Origin Resource Sharing) available, we add 6 lines of configuration code. The implementation of user agent contains about 330 lines of code which imports 3 libraries and about 30 lines of configuration. The implementation of RP SDK is about 1100 lines. However, we easily build the simple RP with only 30 lines of code based on the RP SDK ignoring the auto generated code by Spring framework.

CORS. Same-origin policy enables the browser restrict the request from one origin to another, for example, the JavaScript code on the web page created by `http://www.A.com` defines the request to `http://www.B.com`, which carries the `Origin: http://www.A.com` in its http header. While the response of `http://www.B.com` is transmitted to browser, the browser is to check whether the response carries the `Access-Control-Allow-Origin: http://www.A.com` in the http header. If the `Access-Control-Allow-Origin` is missed, the response is intercepted by the browser. The request initiated by the Chrome extension belongs to the origin `chrome-extension://chrome-id`, while the chrome-id is provided by Google when the extension is uploaded to chrome web store. Therefore, it is required that the RP and IdP's web interfaces accessed by the user agent should add the `Access-Control-Allow-Origin: chrome-extension://chrome-id` in its http header to make CORS available.

VII. Evaluation

The consideration of usability about Recluse is time cost in each authentication. However, the Recluse also introduces the extra storage as IdP and RP has to remember the longer identifier of user and RP. But the storage cost is within the range of TBs, which is available to be ignored.

A. Settings

The prototype of Recluse has been implementation on the system consisting of 3 computers. The IdP is on the DELL

OptiPlex 9020 PC with an Intel Core i7-4770 CPU, 500GB SSD and 8GB of RAM running Window 10 pro. The RP is on the ThinkCentre M9350z-D109 PC with an Intel Core i7-4770s CPU, 128GB SSD and 8GB of RAM running Window 10 pro. The user agent is on the Acer VN7-591G-51SS Laptop with an Intel Core i5-4210H CPU, 128GB SSD and 8GB of RAM running Windows 10 pro. The system is linked through the D-Link DGS-1008D Unmanaged Gigabit Ethernet Switch.

Additionally, the version of Chrome is 75.0.3770.100, and the version of spring framework is 2.0.5.

B. Result

We run the authentication on Recluse prototype system 1000 times and get the average time of the whole authentication flow is 546 ms, in which the Negotiation costs 309 ms , the Dynamic Registration costs 129 ms, and the Authentication costs 107 ms.

The most time cost is introduced by the modular power and extended euclidean operation. It is evaluated that the time cost of each modular power operation in RP, user agent and IdP are about 30 ms, 100 ms and 30 ms. During the login flow contains modular power operation is conducted 4 times by RP, 3 times by user agent and 1 time by IdP.

C. Comparison

We also evaluate the time cost of traditional MITREid connect system in the same circumstance, which is about 130 ms. And the time cost of SPRESSO is about 400 ms. Therefore, the time cost of Recluse is acceptable.

VIII. Discussion

IX. Related Works

In 2014, Chen et al. [15] 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. [17] 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. [24] 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 [18] [19] [16] [21] [25] on SSO system concludes the rules SSO protocol must obey with different manners.

In 2010, Han et al. [26] 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.

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. [27], 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. [27] 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 [28] [29] 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 [7] 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.

X. CONCLUSION

REFERENCES

[1] "Top websites," https://pro.similarweb.com/#/industry/topsites/All/999/1m?webSource=Total, Accessed July 20, 2019.

[2] Paul A Garcia Michael E Fenton James L Grassi, "Digital identity guidelines," *NIST 800-63-3*, 2017.

[3] Thomas Hardjono and Scott Cantor, "Saml v2.0 subject identifier attributes profile version 1.0," *OASIS standard*, 2019.

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

[5] 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.

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

[7] 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.

[8] Hannes Federrath, Karl-Peter Fuchs, Dominik Herrmann, and Christopher Piosecny, "Privacy-preserving DNS: analysis of broadcast, range queries and mix-based protection methods," in *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security*, 2011, pp. 665–683.

[9] SYDNEY LI and JASON KELLEY, "Google screenwise: An unwise trade of all your privacy for cash," https://www.eff.org/deeplinks/2019/02/google-screenwise-unwise-trade-all-your-privacy-cash, Accessed July 20, 2019.

[10] BENNETT CYPHERS and JASON KELLEY, "What we should learn from facebook research," https://www.eff.org/deeplinks/2019/01/what-we-should-learn-facebook-research, Accessed July 20, 2019.

[11] Carole Cadwalladr and Emma Graham-Harrison, "Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach," https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election, Accessed July 20, 2019.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Torsten Lodderstedt, Mark Mcgloin, and Phil Hunt, "Oauth 2.0 threat model and security considerations," *RFC*, vol. 6819, pp. 1–71, 2013.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.