Recluse: A Privacy-Respecting Single Sign-On System Achieving Unlinkable Users' Traces

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, either achieving the identity unlinkage [3], [4], or preventing 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 PPIDs for different RPs are 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 encrypts its 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 trusted transmit 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 [12]–[14], vulnerabilities have been found in the implementation of BrowserID.

In this paper, we propose a privacy-REspecting Single Sign-On System aChieving un$L$inkable USErs' traces from both the IdP and RPs, named Recluse. To achieve this, we rely on the user to achieve the trusted transmit and correctness check of identity proof as in BrowserID [6], 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 VI.
- 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 may be implemented compatibly with OIDC based on the support of Dynamic Registration [15]. Reluse only requires the following modification on OIDC implementations: (1) an additional web service for providing a set of public parameters; (2) the support for generating the new RP identifier and PPID in user and RP. We implement Recluse prototype as follows: the IdP is based on the MITREid Connect, RP is a Java web service based on the SpringMVC framework and a chrome extension for the functions on the user side.

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 in Sections II, and the challenges with solutions briefly III. Section **??** describes the design and details of Recluse. A systematical analysis is presented in Section VI. We provide the implementation specifics and evaluation in Section VIII, then introduce the related works in Section X, and draw the conclusion finally.

II. Background

Recluse aims to preserve the users' privacy by hiding the users' traces from both IdP and RPs in SSO systems (e.g., the widely adopted OIDC SSO systems), with the security of SSO systems under consideration. This section adopts OIDC as the example to present the necessary background information and the security consideration of SSO sytems.

A. OpenID Connect

Typical SSO systems [3], [4], [7] contain the following entities:

- **IdP**. IdP maintains the user's attributes and credentials, performs the user authentication, generates the identity proof with its private key, binds it with the RP and sends the poof (or the reference) to the correct RP through the user. The generation of identity proof is processed differently in various SSO systems. For example, in OIDC, IdP generates a PPID for the user's first login at a RP, checks the consistency of the RP's information (the URL and identifier) between ones from the maintained database and the request, and requires the consent from the user about the exposed attributes.
- **User**. This entity completes the authentication at the IdP with securely maintained credentials, initiates a SSO process by requiring to login at a RP, relays the identity proof request from the RP to IdP, checks the scope of attributes exposed to RP, and transmits the identity proof (or the reference) from IdP to the RP correctly. Usually, these processes are handled by a user-controlled software (e.g., the browser), called user agent.
- **RP**. RP provides the individual services based on the identifier (i.e., account) from the identity proof. In details, RP constructs the identity proof request, sends the request to the IdP through the user, checks the correctness of the received identity proof, and parses the proof for necessary information. The details process varies in different systems, for example, RP needs to

register at the IdP for the identifier to be used in the construction of the identity proof request, and derives the user's account based on PPID.

OpenID Connect (current version 1.0), a typical SSO standard, defines the process at each entity and the protocol flows between entities. OIDC is an extension of OAuth (current version 2.0). 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 [16], [17], and makes impersonation attack possible [18], [19]. For example, the access token isn't required to 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 (distributed in the RP's registration).

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.

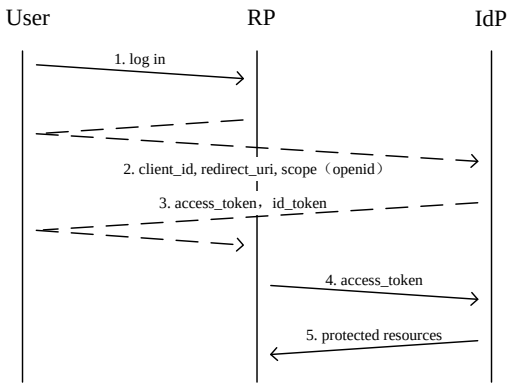

Fig. 1: The implicit protocol flow of OIDC.

Dynamic Registration. The id token (also, the authorization code) is bound with the RP's identifier (i.e., `client_id`). OIDC provides the dynamic registration [15] mechanism to register the RP for a new `client_id`, 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

As described in [7], the design of SSO systems is challenging, as the adversary may adopt various attacks [18]–[24] to achieve:

- Impersonation attack: Adversaries login to a RP as the victim user. Then, the adversary obtains the full control of user's account in the RP, and behaves arbitrarily under the victim's identity.
- Identity injection: Honest user logins to a RP with the adversaries' identity. Then, the privacy of the victim users may be leaked. For example, the victim user may upload the private data to the OneDrive or iCloud under the adversary's account.

To prevent impersonation attack and identity injection, SSO systems are designed and analysed with the following security considerations. In details, in addition to being bound with the correct user, which is the basic requirement of authentication, the identity proof should satisfy the following requirements:

- **Confidentiality.** The identity proof should only be obtained by the correct RP (and the user), no one else may get the identity proof [18]–[20]. Otherwise, the adversary may use the obtained identity proof to login

as the victim user on the specified RP. To avoid the unauthorized leakage of identity proof, (1) the user and IdP should perform additional checks in its generation and transmit to ensure that the identity proof is generated for the correct RP (i.e., the correct identifier) and sent to the exact RP (i.e., correct URL); (2) TLS is adopted to ensure the confidentiality during transmit; (3) a trusted user agent is deployed to ensure the identity proof is only sent to the URL specified by the IdP.

- **Integrity.** The valid identity proof should only be generated by the IdP, no one else may forge or modify a valid proof [19]. Otherwise, the adversary may replace the user's identifer in the proof for either impersonation attack and identity injection. To ensure the integrity of identity proof, (1) the signature is required for each identity proof using the unleaked private key of IdP; (2) the RP should check the signature correctly, and only accept the information protected by the signature, as the others may be tampered. (add reference such as the SAML attacks).
- **Binding.** The identity proof should be bound with only one RP and accepted only by this RP, no other honest RP may accept this identity proof. Otherwise, the adversary may impersonate as the victim user, for example, by pretending as a RP to collect the victim's identity proof and using it at any other honest RP. To ensure the correct binding, (1) IdP should include the RP's identifier in the proof and adopt the signature to avoid the modification, (2) RP should checks the consistency between its identifier with the one in the identity proof. (add attack reference)

In addition to the above security considerations, some other conventional checks are required in the RP to ensure the security. For example, the identity proof should be considered valid only in the specified period; the replayed identity proof needs to be rejected.

Beside the security, the privacy is also considered in SSO systems. For example, the user should be able to control the range of attributes exposed to the RP, which is achieved by one extra user's consent required by IdP.

III. Challenges and Solutions

In this section, we describe the main challenges in hiding the user's traces from both IdP and colluded RPs, and present the solutions of Recluse briefly.

A. Challenges

To prevent the IdP from inferring the user's trace, it is straightforward that IdP should never obtain any information identifying the user-accessed RP. The identifying information includes the RP's identifier and URL. However, it conflicts with the security requirements on identity proof described in Section II-B:

- **Leakage.** The confidentiality of identity proof is corrupted, and the leaked identity proof may result in the impersonation attacks [18]–[20]. The potential leakage is due to: (1) no reliable checks (from the IdP and user) during the generation of identity proof, as the IdP lacks the correct RP identifier to retrieve the exact information from the local storage, while the user fails to obtain the correct URL (or RP name) from IdP for the check. Therefore, the malicious RP may request the identity proof for another RP without being found by the IdP and user. (2) the lack of correct URL for the transmit, as without the correct RP identifier, IdP fails to extract the correct (locally stored) URL. The trusted user agent transmit the identity proof to the incorrect URL (provided by IdP) which is controlled by the adversary.
- **No Binding.** IdP fails to bind the identity proof with a specified RP, as it lacks the correct RP identifier for binding. On receiving the identity proof not bound to it, the RP either (1) rejects the proof and halts its service as no identity proof is bound to it, or (2) accepts the proof. The second case will make one identity proof be accepted by multiple RPs, which results in the misuse of identity proof for impersonation attacks and identity injection [18]–[20].

In addition to challenges from the security considerations, providing no RP's identifying information to the IdP also brings the challenge in preventing the identity linkage and RP's consecutive running, as:

No user's unique account for one RP, which varies for different RPs. Each RP provides the individual services for each user based on the unique account. In SSO systems, RP derives the account from the identifier (i.e., PPID in OIDC) in the identity proof. With the correct RP identifier, IdP ensures the PPID is unique for multiple logins at the same RP by one user. However, when IdP fails to obtain the exact RP identifier, IdP may provide (1) different PPIDs for user's multiple logins at the same RP, which makes RP fail to provide the consecutive and individual services; or (2) the same PPID (e.g., user's email address) for various RPs, which makes the identity linkage be possible for colluded RPs.

B. Solutions

Recluse aims to hide the users' traces from both the IdP and colluded RPs, without violating the security of SSO systems and interrupting RP's the consecutive and individual service. That is, Recluse ensures the confidentiality and binding of identity proof without leaking the RP's identifer to the IdP, and provides the unique and unlinkable PPID to the RP for the user's multiple logins.

User-centric confidentiality. In Recluse, instead of checking the information provided by the RP with the ones stored in the IdP, the user directly extracts from a RP certificate for the trusted necessary information in the generation and transmit of identity proof, which ensures the confidentiality. The RP

certificate contains the RP's correct identifying information (URL, name and RP identifier), and a signature from the IdP to ensure no modification nor forging on it. The user provides IdP a transformation of the correct RP identifier in the generation of identity proof; and sends the proof to the URL specified in the RP certificate. Therefore, the adversary fails to request the identity proof using others' identifiers, or obtain others' proof with it's URL.

Binding with a transformation of RP's identifier. The identity proof is bound with a transformation of the RP's identifier by the IdP, who cannot infer the original identifier from the transformation. RP only accepts the identity proof which is bound to a fresh transformation of its identifier. Moreover, the transformation of RP's identifier is generated corporately by the user and RP, preventing the adversary from constructing a same transformation for various RPs. Otherwise, the identity proof will be accepted by multiple RPs, which results in the misuse of the proof for the impersonate attack.

Deriving the unique account with the trapdoor. In Recluse, the user's account in a RP is a function of the RP's original identifier and the user's unique identifier. The calculation of the user's unique account is split into two steps, to prevent the IdP from obtaining the RP's identifier and avoid the RP to infer the user's unique identifier. In the first setp, IdP generates the PPID with the user's unique identifier and the transformation of the RP's identifier, which results in different PPIDs for multiple logins at a RP as various transformations are adopted. While, in the second step, the destination RP adopts the trapdoor of the RP's identifier transformation to derive the unique account from the PPIDs. Moreover, the accounts of a user in various RPs are different, as the original RP identifiers are different, which prevents the identity linkage.

IV. Assumption and Threat Model

Recluse contains only three entities, i.e., the user, IdP and RP; and doesn't introduce any other (trusted) entity.

Assumption. In Recluse, we assume the user agent deployed at the honest user is correct, and will transmit the messages to the correct destination without leakage. The TLS is correctly implemented at the user agent, IdP and RP, which ensures the confidentiality and integrity of the network traffic. We also assume the nonce is unpredictable by using a secure random number generator; and the adopted cryptographic algorithms, including the RSA and SHA-256 for the RP certificate, are secure and implemented correctly, that is, no one with the IdP's private key can produce a valid certificate, and the adversary fails to infer the private key. Moreover, the transformation of the RP's identifier and the user's account calculation are based on the Discrete Logarithm Problem, we assumes the adversary fails to infer r from $g^r modP$, where P is a large prime and g is the primitive root.

A. Threat Model

In Recluse, the adversary attempts to break the security and user's privacy under the following threat model.

Security. The adversary attempts to break the confidentiality, integrity or binding of the identity proof, for impersonating the victim user to access a RP, or making the user access a RP under an incorrect account. Same as traditional SSO systems [3], [4], [6], [7], we assume the IdP is honest, uncorrupted users and RPs behave correctly, while the user and RP controlled by the adversary may be malicious. The details are as follows:

Honest IdP. The IdP is well protected and the private keys for signing the RP certificate and identity proof are not leaked. In the initial registration of RP, IdP checks the correctness of RP's URL, assigns an unique original identifier, and generates the correct signature. For identity proof, IdP generates the proof only for the authenticated user, calculates the PPID based on the user's unique identifier and the user-provided transformation of RP identifer, binds the proof with the transformation, generates the signature correctly, and sends it only to the user.

Malicious User. The adversary may obtain the user's credential through various attacks, or register a valid account at the IdP and RP. The user controlled by the adversary may behave arbitrarily. For example, to login at a RP under a uncontrolled user's account, the adversary may send illegal login request to the RP, transmit a modified or forged identity proof request to the IdP, reply a corrupted or forged identity proof to the RP, choose a non-random nonce to participate in the generation of RP's transformation identifier.

Malicious RP. The adversary may work as RPs and behave arbitrarily, by controlling one or more compromised RPs, or registering as multiple valid RPs at the IdP. The malicious RP may attempt to make the identity proof bound with it be accepted by other RPs by using one or more chosen nonce for the RP's transformation identifiers, or receive an identity proof bound with other RP by sending another valid or invalid RP certificate instead of its own one, or providing an incorrect identity proof request (e.g. CSRF).

Collusion. The malicious users and RPs may collude to perform the impersonation attack and identity injection. For example, to login at the uncorrupted RP under the uncontrolled user's account, the adversary firstly attracts the uncontrolled user to access the malicious RP, then attempts to make the identity proof also valid for the uncorrupted RP, and finally works as a user to access this RP with the identity proof. To make the uncontrolled user login at the uncorrupted RP under an controlled account, the adversary acts as a user to obtain an identity proof for itself by accessing the uncorrupted RP, and works as a RP to make the uncontrolled user access the uncorrupted RP with this proof.

Privacy. The curious IdP may attempt to infer the user's access traces (i.e., which RPs are accessed by one user), by analyzing the RP's identifiers in or the receivers of the identity

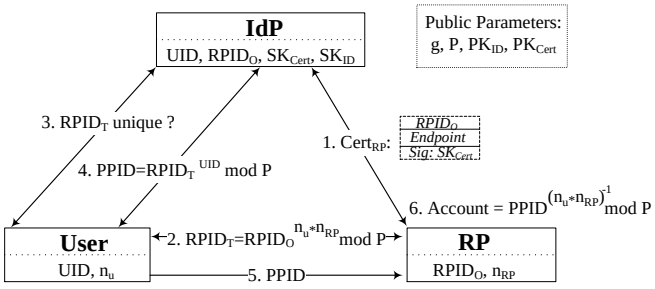

Fig. 2: Overview of Recluse.

proof generated for a user. The colluded (correct) RPs may link the accounts in each RP, using the same (or derivable) PPID in the identity proof. Same as SPRESSO [7], we assume that IdP will never collude with RPs. The user linkage based on other attributes are not considered in this work, which may be prevented by limiting the attributes exposed to each RPs.

V. Recluse

As analyzed in Section III-B, to ensure the security and privacy of SSO systems, Recluse has to achieve user-centric confidentiality of the identity proof, bind the proof with a transformation of RPs identifier, and allow only the exact RP to derive the user's unchange account with a trapdoor. We further divide these three solutions into 5 goals to be achieved in the design in Recluse:

- 1. Providing a self-verifying value for the user-centric check, to ensure the correct construction and transmission of identity proof (i.e., user-centric confidentiality);
- 2. Providing a RP identifier transforming algorithm, to ensure the security by binding the identity proof with this transformation, and preserve user's privacy by preventing the IdP from inferring the original RP identifier;
- 3. Checking the uniqueness of the transformation, to avoid one identity proof being accepted by two or more RPs, which will harm the security of SSO systems;
- 4. Unlinkable PPID, for one user, the PPIDs for different RPs should vary, to prevent identity linkage;
- 5. Unchange account, to allow only the destination RP having the trapdoor to derive the unchanged account for one user from varying PPIDs, which is necessary for providing the consecutive and individual services.

Recluse achieves these 5 goals using 5 phases as described in Figure 2, while the used notations are listed in Table I. The details are as follows, and the step refers to the one in Figure 2:

- RP initial registration: Applying for a RP certificate ($Cert_{RP}$), a self-verifying value, (Step 1, Goal 1);
- RP identifier transforming: Generating transformation (denoted as $RPID_T$) of RP's original identifier (denoted as $RPID_O$), (Step 2, Goal 2);
- Dynamic registration: Checking the global uniqueness of $RPID_T$, (Step 3, Goal 3);

TABLE I: The notations used in Recluse.

Notations	Definition
P	A large prime.
g	A primitive root modulo P.
$Cert_{RP}$	A RP certificate.
SK_{Cert}, PK_{Cert}	The private / public key for $Cert_{RP}$.
SK_{ID}, PK_{ID}	The private / public key for identity proof.
UID	User's unique identifier at IdP.
$PPID$	User's pseudonymous id in the identity proof.
$Account$	User's identifier at a RP.
$RPID_O$	RP's original identifier.
$RPID_T$	A transformation of $RPID_O$.
n_u	User-generated random nonce for $RPID_T$.
n_{RP}	RP-generated random nonce for $RPID_T$.
t	A trapdoor, $t = (n_u * n_{RP})^{-1} mod\ (P-1)$.

- PPID: Generating PPID at the IdP, and transferring PPID to the RP, (Step 4 and 5, Goal 4);
- Account: Calculating the user's account in the RP, (Step 6, Goal 5).

In Recluse, the IdP needs to be initialized during the IdP setup, for generating the public parameters for the users and RPs; the RP has to perform only one initial registration at the RP setup; and the user triggers the Step 2-6 in Figure 2 for each login at a RP. IdP setup is performed at the very beginning of Recluse, and will be invoked by the IdP to update the leaked SK_{Cert} or SK_{ID}, while g and P will never be modified. Recluse does not support multiple initial registrations for one RP, as the RP does not know the $UIID$ and fails to derive the user's new account from the old one under the Discrete Logarithm problem.

IdP setup. In the setup, the IdP generates two random asymmetric key pairs, (PK_{ID}, SK_{ID}) and (PK_{Cert}, SK_{Cert}), for calculating the signatures in the identity proof and $Cert_{RP}$, respectively; and provides PK_{ID} and PK_{Cert} as the public parameters for the verification of identity proof and $Cert_{RP}$. Moreover, IdP generates a strong prime P, calculates a primitive root (g) using ** algorithm[], and provides P and g as the public parameters. For P, we firstly randomly choose a large prime Q, and accept $2Q+1$ as P if $2Q+1$ is a prime. The strong prime P makes it easier to choose n_u and n_{RP} as descried in Section V-B. With g, we obtain all the integers less than P, by calculating $g^i mod\ P$ for $0 \leq i \leq (P-1)$.

A. Initial Registration

The RP invokes initial registration to apply a valid and self-verifying $Cert_{RP}$ from IdP (**Goal 1**) as provided in Figure 3, which contains three steps:

- 1. RP sends IdP a $Cert_{RP}$ request $Req_{Cert_{RP}}$, which contains the distinguished name $Name_{RP}$ (e.g., DNS name) and the endpoint to receive the identity proof.
- 2. IdP calculates $RPID_O = g^r mod\ P$ with a random chosen r which is coprime to $P-1$ and different from the ones for other RPs, generates the signature ($Sig_{SK_{Cert}}$) of $[RPID_O, Name_{RP}]$ using SK_{Cert}, and returns $[RPID_O, Name_{RP}, Sig_{SK_{Cert}}]$ as $Cert_{RP}$.

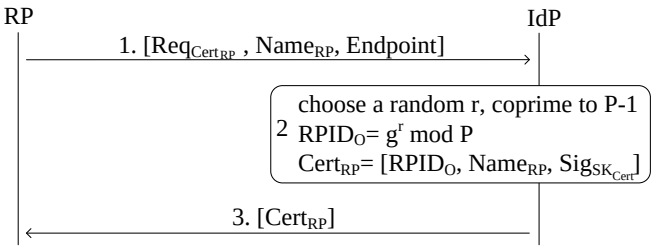

Fig. 3: RP initial registration.

- 3. IdP sends $Cert_{RP}$ to the RP who verifies $Cert_{RP}$ using PK_{Cert}.

Here, we further explain the two requirements in the generation of $RPID_O$ for **Goal 4**:

- r should be coprime to $P-1$. This makes $RPID_O$ be another primitive root, and satisfies the requirement of Discrete Logarithm problem to prevent the RP inferring UID from $Account$.
- r should be different with the ones for other RPs. Otherwise, the RPs who are assigned the same $RPID_O$, obtain the same PPID for a user, which makes identity linkage possible.

The honest IdP is assumed to generate the correct $RPID_O$. However, we may perform an external check on $Cert_{RP}$ and $RPID_O$, based on the idea of Certificate transparency. That is, IdP is required to provide the $Cert_{RP}$ to a log server, while a monitor checks the correctness of $Cert_{RP}$, i.e., no two valid $Cert_{RP}$ assigned to a same $RPID_O$ and $RPID_O$ is a primitive root modulo P based on the *** algorithm. Moreover, $Cert_{RP}$ is compatible with a X.509 certificate which is discussed in Section IX.

B. RP identifier transformation and Account calculation

In this section, we provide the calculation of $RPID_T$, $PPID$ and $Account$ separately, which is the foundation of each user's login process that described in Section V-C.

$RPID_T$. Similar to Diffie-Hellman key Exchange [25], the RP and user generate the $RPID_T$ cooperatively as follows:

- RP chooses a random odd number n_{RP}, and sends $Y_{RP} = {RPID_O}^{n_{RP}} mod\ P$ to the user.
- The user replies a random chosen odd number n_u to the RP, and calculates $RPID_T = {Y_{RP}}^{n_u} mod\ P$.
- RP also obtains $RPID_T$ with the received n_u.

Therefore, $RPID_T$ is denoted as Equation 1. The IdP fails to infer $RPID_O$ from $RPID_T$ (**Privacy in Goal 2**).

$$RPID_T = {Y_{RP}}^{n_u} = {RPID_O}^{n_u * n_{RP}} mod\ P \quad (1)$$

The generation ensures that $RPID_T$ cannot be determined by neither (malicious) user nor RP, which prevents the adversary from constructing a identity proof to be accepted by two or more RPs (**Security in Goal 2**). RP fails to control the $RPID_T$ generation, as it provides Y_{RP} before obtaining n_u and the modification of Y_{RP} will trigger the user to generate another different n_u. The Discrete Logarithm problem prevents the user from choosing a n_u for a specified $RPID_T$ on the received Y_{RP}.

Both n_{RP} and n_u are odd numbers, therefore $n_{RP} * n_u$ is an odd number and coprime to the even $P-1$, ensuring:

- $RPID_T$ is a primite root modulo P, which prevents the RP from inferring UID from $PPID$ (as analyzed for $RPID_O$, **Goal 4**).
- The inverse $(n_{RP} * n_u)^{-1}$ exists, that is, $(n_{RP} * n_u)^{-1} * (n_{RP} * n_u) = 1\ mod\ (P-1)$. The inverse serves as the trapdoor t for $Accout$, which makes:

$$(RPID_T)^t = RPID_O\ mod\ P \quad (2)$$

$PPID$. The IdP generates the $PPID$ based on the user's UID and the user-provided $RPID_T$, as denoted in Equation 3. $PPID$ varies for RPs due to the uniqueness of $RPID_T$, satisfying **Goal 4**.

$$PPID = {RPID_T}^{UID}\ mod\ P \quad (3)$$

$Account$. The RP calculates $PPID^t mod\ P$ as the user's account, where $PPID$ is received from the user and t is derived in the generation of $RPID_T$. Equation 4 demonstrates that $Account$ is unchanged to the RP during a user's multiple logins, satisfying **Goal 5**.

$$Account = ({RPID_T}^{UID})^t = {RPID_O}^{UID} mod\ P \quad (4)$$

C. User Login Process

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.

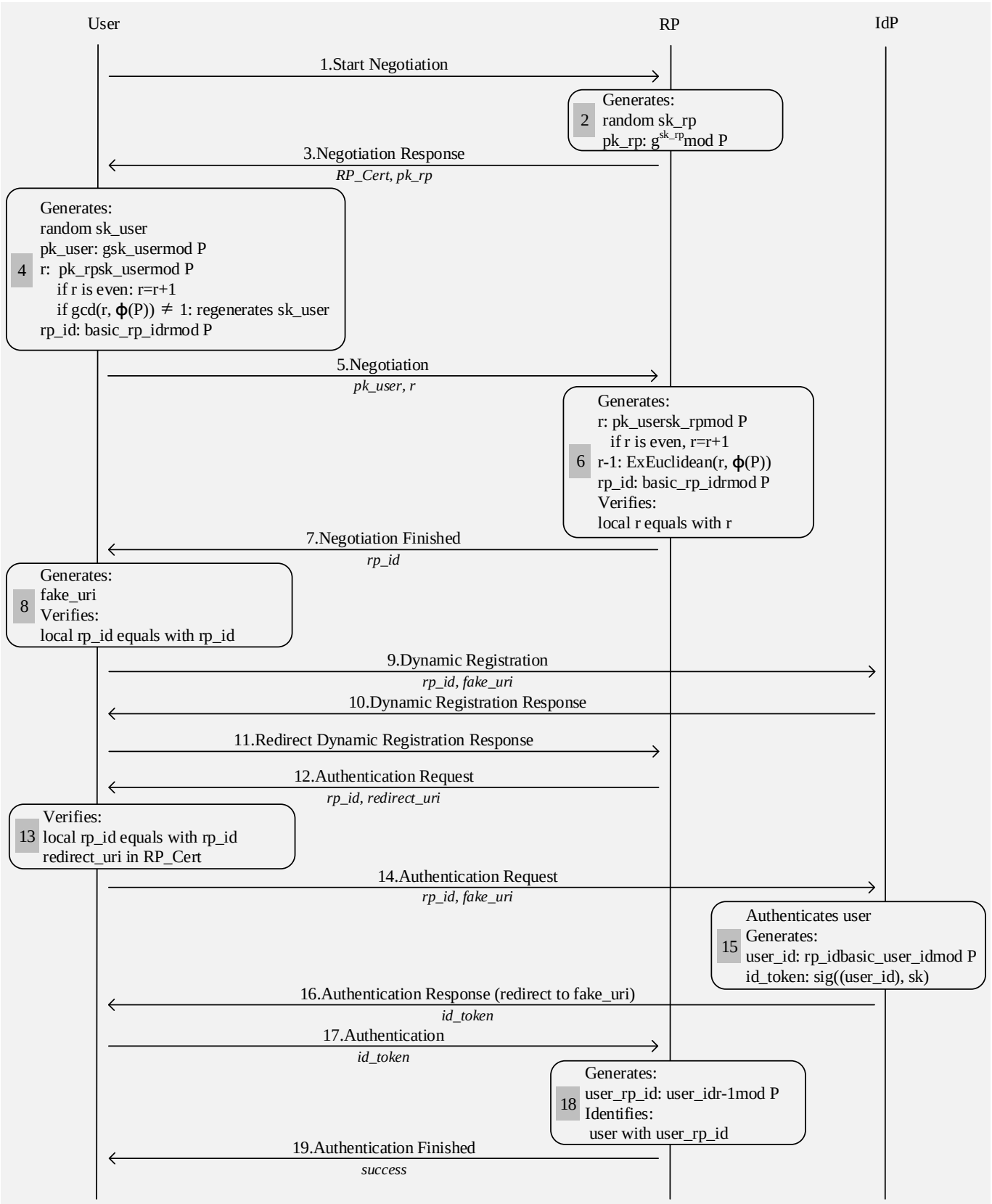

Fig. 4: Login Flow

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 $user\_id = rp\_id^{basic\_user\_id} modP$. 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.

D. Compatible with OIDC

VI. 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 undistinguishability of SSO system. It is in the SSO system, for an adversary controlling curious IdP, it is impossible to inspect whether two `authentication flows` are to same RP or not, however, for an adversary controlling malicious RPs, it is impossible to inspect whether two `authentication flows` are from same user or not.

Assuming there are two `authentication flows` 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 $fake\_uri$. To break the undistinguishability, 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 flows` 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 flows` 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 flows` are from the same user. However, the malicious also tries to lead the same user in two `authentication flows` 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 flows`. 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 flows`.

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 [20] 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 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.

TABLE II: 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

IdP Mix-Up. It has been illustrated in [20] 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.

VII. 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.

VIII. 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.

IX. Discussion

X. Related Works

In 2014, Chen et al. [18] 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. [20] 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. [26] 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 [21] [22] [19] [24] [27] on SSO system concludes the rules SSO protocol must obey with different manners.

In 2010, Han et al. [28] 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. [29], 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. [29] 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 [12] [30] 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.

XI. 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] 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.

[13] Daniel Fett, Ralf Küsters, and Guido Schmitz, "An expressive model for the web infrastructure: Definition and application to the browser id sso system," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 673–688.

[14] Connor Gilbert and Laza Upatising, "Formal analysis of browserid/-mozilla persona," 2013.

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.