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

***Abstract*—The Single Sign-On (SSO) System is widely deployed to bring the convenience to both the relying party (RP) and the users, by shifting the users' credentials to the identity provider (IdP) and reducing the user's maintained credentials. However, the privacy leakage is the obstacle to the users' adoption of SSO, as the curious IdP knows the user's accessed RPs while the RPs may link the user with the non-independent identifiers provided by the IdP. Existing solutions preserve the user's privacy either from the curious IdP or the colluded RPs, but fails to hide the user from the both entities. In this paper, we provide a SSO system, named Recluse, for the first time to hide the user's accessed RPs from the curious IdP and prevent the identity linkage by colluded RPs, based on the transformation of the RP identifiers and user's accounts on Discrete Logarithm Problem. Recluse doesn't introduce any other entities, and is compatible with OpenID-Connect, the widely deployed and analyzed SSO systems. Recluse is proved to provide the privacy without any degradation on the security of OpenID-Connect. The evaluation demonstrates that Recluse is efficient, about 200 ms for one user's login at a RP in our environment.**

***Index Terms*—Single Sign-On, security, privacy, trace, linkage**

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 in the view of any entities except the IdP and the user.

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 need 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 IdP who doesn't know the RP's identifying information, should bind the identity proof with the correct RP and transmit it to the exact RP, avoiding the misuse of identity proof by the malicious RPs.
- For one user, the IdP who doesn't know the RP's identifying information, should provide the PPIDs that are (1) un-linkable among RPs to avoid the identity linkage and (2) linkable in the destination RP among different logins for RP to provide the continuous service.

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 (same as in BrowserID [6]), and propose two algorithms to achieve:

- RP's identifier generation and transformation, 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 at the IdP for providing a set of public parameters; (2) the support for generating the new RP identifier (at the user and RP), PPID (at the IdP) and user's account (at RP). The prototype demonstrates that Recluse is incompatible with existing OIDC implementations.

The main contributions of Recluse are as follows:

- We have analyzed the privacy issues in SSO systems systematically, 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 IX and Section V describe the threat model and the design of Recluse. A systematical analysis is presented in Section VI. 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 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, in OIDC, 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 issuer, issuing time, the validity 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:

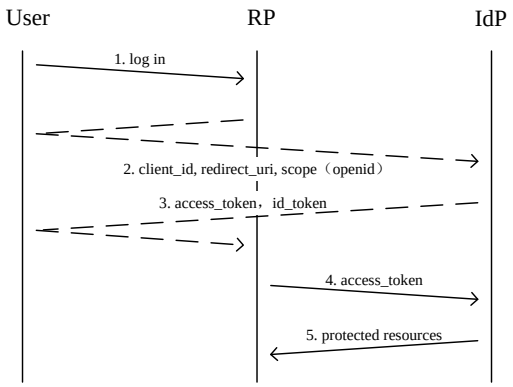

Fig. 1: The implicit protocol flow of OIDC.

- 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 (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 identifier in the proof for either impersonation attack and identity injection. There are two solutions to ensure the integrity of identity proof: (1) one is based on signature, IdP generates the signature for each identity proof with the un-leaked private key, while RP only accepts the information protected by the valid signature as the others may be tampered [21], [27]; (2) the other is based on a direct TLS communication between the IdP and RP, RP verifies the certificate of IdP and the TLS ensures the integrity of transmitted identity proof, which is adopted in authorization code flow in OIDC and OAuth 2.0 [18].
- **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. Currently, the binding is achieved in two ways: (1) one is based on the claim in the identity proof, IdP claims the destination RP's identifier in the identity proof and avoids the modification on it by generating a signature, while the RP checks the RP's identifier and only accepts the identity proof for it. (2) the other is based on the one-time nonce and RP authentication, IdP generates an one-time nonce, binds it with the user-requested RP's identifier, and provides the identity proof to the RP only when the RP's identifier is the same as the user-requested RP's identifier and the RP has completed the authentication at the IdP with the pre-assigned secret; therefore the RP will only receive the identity proof bound with it.

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.

C. Primitive Root

A number g ($0 < g < P$) is called a primitive root modular a prime P, if for $\forall y$ ($0 < y < P$), there is a (unique) number x ($0 \leq x < P - 1$) satisfying $y = g^x \pmod P$. Here, x is called the discrete logarithm of y modulo P. Given a large prime P and a number y, it is computationally infeasible to derive the discrete logarithm (here x) of y (detailed in [**?**]), which is called Discrete Logarithm Problem. The hardness of solving discrete logarithm has been a base of the security of several security primitives, including Diffie-Hellman key exchange and Digital Signature Algorithm.

To calculate the primitive root for a given large prime P, we first search the least primitive root g_m mod P, and then calculate the primitive root $g = g_m^t mod\ P$, where t is an integer coprime to $P - 1$. We checks whether a integrity μ is the primitive root modulo P where $P = 2Q + 1$ (Q is a prime), based on the lemma that an integer $1 < \mu < P - 1$ is a primitive root if and only if $\mu^2 \neq 1 \pmod P$ and $\mu^Q \neq 1 \pmod P$. The details are provided in [**?**], [**?**].

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 transmitting,

as without the correct RP identifier, IdP fails to extract the correct (locally stored) URL. The trusted user agent may 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 account satisfying (1) unique for one RP and (2) different from ones for other 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 the user's multiple logins at the same RP. 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 user identifier (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 un-linkable 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 transmitting 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 its 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 between correct entities. 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 without 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 $g^r mod P$, 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.

Collusion. The malicious users and RPs may collude to perform the impersonation attack and identity injection. For example, (1) 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 pretends as a user to access this RP with the received identity proof; (2) 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 redirect the uncontrolled user to the uncorrupted RP with this proof (e.g. CSRF).

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 content and timing of received messages, for example, inferring RP's identifiers in (or the receivers of) the identity proof. The colluded (malicious) RPs may link the accounts in each RP actively (providing incorrect messages) and passively (combining the received messages), 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 and the global network traffic analysis are not considered in this work, which may be prevented by limiting the attributes exposed to each RPs and proving the mixed traffic by accessing unwanted RPs.

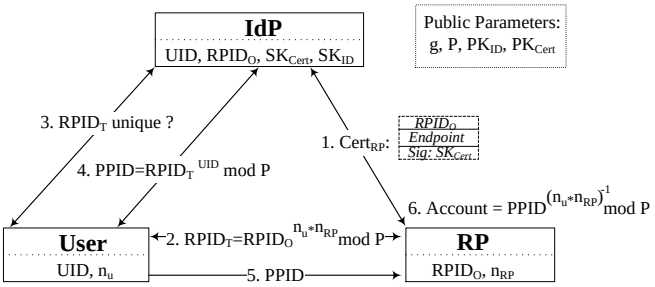

Fig. 2: Overview of Recluse.

V. Recluse

In this section, we firstly provide an overview of Recluse with the design goals and main process phases; then present initial registration of RP, following with the description of algorithms for calculating the RP identifier transformation, PPID and account. The detailed processing for each user's login is provided corresponding to the main process phases. Finally, we present how Recluse to be compatible with OIDC.

A. Overview

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 RP's 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 transformation, 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 from this transformation in the request;
- 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 (with 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);

TABLE I: The notations used in Recluse.

Notation	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 $RPID_T$.
Y_{RP}	Public value for n_{RP}, $(RPID_O)^{n_{RP}}\ modP$.
t	A trapdoor, $t = (n_u * n_{RP})^{-1} mod\ (P-1)$.

- 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);
- 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 UID nor the discrete logarithm of $RPID_O$ 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) as described in Section II-C, 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-C.

B. RP 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:

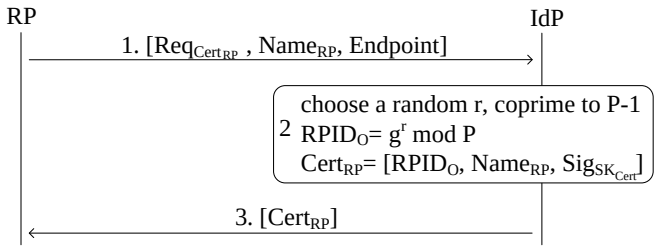

Fig. 3: RP initial registration.

- 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 $Cert_{RP}$.
- 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. The external check needs to be performed by a third party instead of RP who will be able to perform identity linkage with incorrect $RPID_O$. To perform the external check, 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 as descried in Section II-C. Moreover, $Cert_{RP}$ is compatible with a X.509 certificate which is discussed in Section VIII.

C. 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-D.

$RPID_T$. Similar to Diffie-Hellman key Exchange [26], 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 $PPID$ (**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**.

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

D. User Login Process

In this section, we present the detailed process for each user's login as shown in Figure 4. The process corresponds to the four phases defined in Section V-A.

For RP identifer transforming, the user and RP corporately process as follows. **(1)** The user sends a login request to trigger the negotiation of $RPID_T$. **(2)** RP chooses the random n_{RP}, and calculates Y_{RP} as described in Section V-C. **(3)** RP sends $Cert_{RP}$ with Y_{RP} to the user. **(4)** The user halts the login process if the provided $Cert_{RP}$ is invalid; otherwise, it extracts $PPID_O$ from $Cert_{RP}$, and calculates $RPID_T$ with a random chosen n_u as in Section V-C. **(5)** The user sends n_u and $RPID_T$ to the RP. **(6)** RP calculates $RPID_T$ using the received n_u with Y_{RP} as in Section V-C, and rejects the user's login request if the calculated $RPID_T$ is inconsistent with the received one. After that, RP derives the trapdoor t as in Section V-C, which will be used in calculating $Account$. **(7)** RP sends the calculated $RPID_T$ to the user, who will halt the login if the received $RPID_T$ is different from the cached one.

For dynamic registration, the user registers the RP at the IdP instead of RP as follows. **(8)** The user generates an one-time endpoint (used in Section V-E) if the received $RPID_T$ is accepted. **(9)** Then, the user registers the RP with the $RPID_T$ and one-time endpoint. **(10)** If $RPID_T$ is globally unique and is a primitive root module P, IdP sets the flag $RegRes$ as OK (otherwise $FAIL$), and constructs the reply in the form of [$RegRes$, $RegMes$, $Sig_{SK_{ID}}$] where $RegMes$ is the response to traditional dynamic registration containing $RPID_T$ with other attributes and $Sig_{SK_{ID}}$ is the signature of the other elements using the private key SK_{ID} (satisfying **Goal 3**). **(11)** The user forwards the registering result to the RP. The user obtains $RegRes$ directly as the connection between the user and IdP is secure, while the RP accepts the $RegRes$ only when $Sig_{SK_{ID}}$ is valid and $RegMes$ is issued for the $RPID_T$ within the expiration date. The user and RP will negotiate a new $RPID_T$ if $RegRes$ is $FAIL$.

To acquire the PPID, the user corporates with the RP and IdP as follows. **(12)** RP constructs an identity proof request with the correctly registered $RPID_T$ and the endpoint (the form of the request is detailed in Section V-E). **(13)** The user halts the login process if the received $RPID_T$ is different from the previous one. **(14)** The user replaces the endpoint with the registered one-time endpoint, and sends it with the identity proof request to the IdP. **(15)** IdP requires the user to provide the correct credentials if the user hasn't been unauthenticated; and rejects the request if the binding of $RPID_T$ and the one-time endpoint doesn't exist in the registered ones. Then, IdP generates the $PPID$ as in Section V-C, and constructs the identity proof with $RPID_T$, $PPID$, the valid period, issuing time and other attribute values, by attaching a signature of these elements using the private key SK_{ID}. **(16)** IdP sends the identity proof with the one-time endpoint to the user. **(17)** The user forwards the identity proof to the RP's endpoint corresponding to the one-time endpoint.

Finally, RP derives the user's unchanged $Account$ from $PPID$ as follows. **(18)** RP accepts the identity proof only when the signature is correctly verified with PK_{ID}, $RPID_T$ is the same as the negotiated one, the issuing time is less than current time, and the current time is in the validity period. If the identity proof is incorrect, RP returns login fail to the user who will trigger another login request. Otherwise, RP calculates the $Account$ as in Section V-C. **(19)**, After obtaining the user's unchanged $Account$, RP sends the login result to the user and begins to provide the individual service.

E. Compatible with OIDC

Recluse is compatible with the implicit protocol flow of OIDC (authorization code flow is discussed in Section VIII).

In Recluse, the formats of identity proof request and identity proof are the same as the ones in OIDC. In details, each

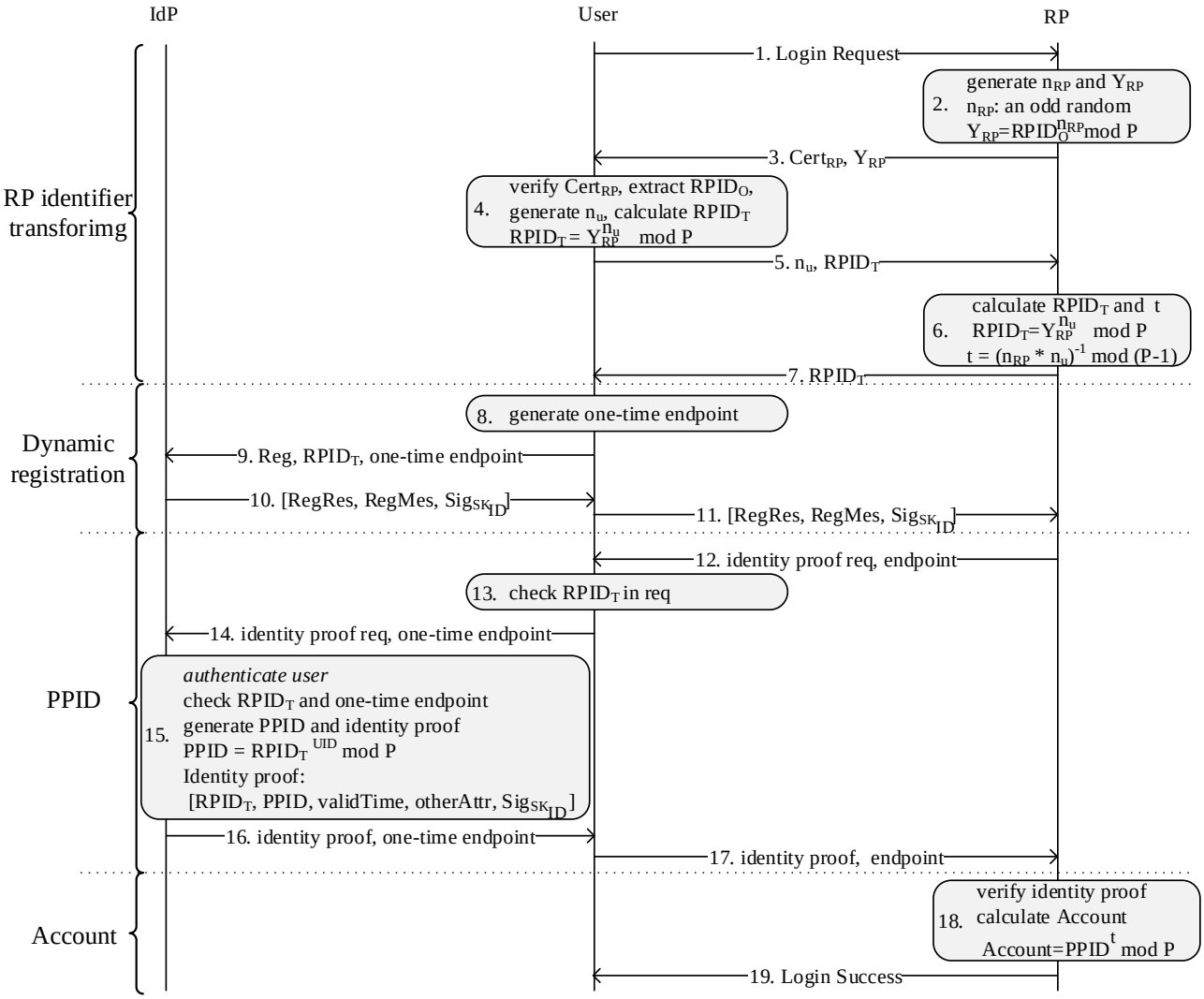

Fig. 4: Process for each user's login.

element of the identity proof request in OIDC is contained in Recluse as follows: the RP's identifier ($RPID_T$ in Recluse), the endpoint (one-time endpoint in the request from the user in Recluse) and the set of required attributes (also supported by Recluse although not listed in the description). The identity proof in Recluse is also exactly the same as the one in Recluse, which includes RP's identifer ($PPID_T$ in Recluse), the user's PPID, the issuer, validity period, issuing time, other requested attributes and a signature using the IdP's private key.

The same format of identity proof request and identity proof makes the verification same in OIDC and Recluse. The IdP, in both Recluse and OIDC, verifies the identity proof request, by checking whether the mapping of RP's identifier and endpoint exists in the registered ones. The RP, in both Recluse and OIDC, checks the correctness of identity proof, by checking that the signature is correct, the consistency of RP's identifer in the identity proof and the one it owns, the validity period, issuing time and the freshness (which is checked based on a nonce in OIDC, while $RPID_T$ serves as the nonce in Recluse).

The RP's extra processes needed in Recluse may be achieved using the existing interfaces defined in OIDC. Recluse requires that $RPID_T$ is globally unique, which can be achieved through the dynamic registration (described in Section II) provided by IdP in OIDC. In Recluse, the dynamic registration is invoked by the user instead of the RP to prevent the curious IdP infers the user's traces by analyzing the between the dynamic registration and identity proof request. Moreover, the endpoint in the dynamic registration request is replaced with an one-time endpoint, to avoid the RP's identifying information to be leaked to the IdP.

The modification required by Recluse at the RP may be achieved using existing interface provided by the implementations of OIDC. Based on the software development kit (SDK) in existing OIDC implementations for RP, the Steps 2,3, 6, 12 (in Figure 4) in RP may be integrated in the interface for constructing identity proof request; Step 18 in Figure 4 may be combined with the interface for verifying and parsing identity proof.

The processes at the user may be achieved through an extension at the user agent, which captures the identity proof (and request) for process without modifying existing message transmission at the IdP and RP (i.e., redirection mechanism), as described in Section VII. Only the RP initial registration and Step 15 in Figure 4 requires a modification at IdP.

VI. Security Analysis

In this section, we first prove the privacy of Recluse, i.e., avoiding the identity linkage of the colluded malicious RPs, and preventing the curious IdP from inferring the user's accessed RPs. Then, we prove that Recluse does not degrade the security of SSO systems by comparing it with OIDC, which has been formally analyzed in [**?**].

A. Privacy

Curious IdP. The curious IdP can only perform the analysis on the content and timing of received messages, however it fails to obtain the user's accessed RPs directly, nor infer classifies the accessed RPs for RP's information indirectly.

- The curious IdP fails to derive RP's identifying information (i.e., $RPID_O$ and correct endpoint) through a single login flow. IdP only receives $RPID_T$ and one-time endpoint, and fails to infer the discrete logarithm $RPID_O$ from $RPID_T$ without the un-leaked trapdoor t due to hardness of solving discrete logarithm, or the RP's endpoint from the independent one-time endpoint.
- It also fails to infer the correlation of RPs in two or more login flows from a single user or multiple users. The secure random number generator ensures the random for generating $RPID_T$ and the random string for one-time endpoint are independent in multiple login flows. Therefore, curious IdP fails to classify the RPs based on $RPID_T$ and one-time endpoint.
- It even fails to obtain the correlation of RPs through analyzing the timing of received messages. IdP fails to map user's accessed RP in the identity proof to the origin of dynamic registration based on timing, as both the dynamic registration and the identity proof request are sent by the user instead of the RP.

Malicious RPs. The malicious RPs may attempt to link the user passively by combining the PPIDs received by the colluded RPs, or actively by tampering with the provided elements (i.e., $Cert_{RP}$, Y_{RP} and $RPID_T$). However, these RPs still fail to obtain the user's unique identifier directly, or trigger the IdP to generate a same or derivable PPIDs.

- A single RP fails to infer the user's unique information (e.g., UID or other similar ones) passively. The $PPID$ is the only element received by RP that contains the user's unique information. However, RP fails to infer (1) UID (the discrete logarithm) from $PPID$, due to hardness of solving discrete logarithm; (2) or g^{UID} as the r in $RPID_O = g^r$ is only known by IdP and never leaked, which prevents the RP from calculating r^{-1} to transfer $Account = RPID_O^{UID}$ into g^{UID}.
- A single RP fails to actively tamper with the messages to make UID leaked. The modification of $Cert_{RP}$ will make the signature invalid and be found by the user. The malicious RP fails to control the calculation of $RPID_T$ by providing an incorrect Y_{RP} as another element n_u is controlled by the user. Also, the malicious RP fails to make an incorrect $RPID_T$ (e.g., 1) be used for $PPID$, as the honest IdP only accepts a primitive root as the $RPID_T$ in the dynamic registration. The RP also fails to change the accepted $RPID_T$ in Step 11 in Figure 4, as the user checks it with the cached one.
- Two or more RPs fail to classify the user passively. The analysis can only be performed based on $Account$ and $PPID$. The $Account$ is independent among RPs, as the $RPID_O$ chosen by honest IdP is random and unique. The $PPID$s are also independent due to the unrelated $RPID_T$.
- Two or more RPs fail to classify the user based on $PPID$s actively, by attempting to make $RPID_T$ correlative by manipulating n_{RP}. However, the random n_u chosen by the honest user will ensure the independence of $RPID_T$ to protect its own privacy.

Colluded RPs even fail to correlate the users based on the timing of users' requests, when the provided services are unrelated. For the related services, (e.g., the online payment accessed right after an order generated on the online shopping), the user may break this linking by adding an unpredicted time delay between the two accesses. The anonymous network may be adopted to prevent colluded RPs to classify the users based on IP addresses.

B. Security

Recluse protects the user's privacy without breaking the security. That is, Recluse still prevents the malicious RPs and users from breaking the integrity, confidentiality and binding of identity proof.

In Recluse, all mechanisms for integrity are inherited from OIDC. The IdP uses the un-leaked private key SK_{ID} to prevent the forging and modification of identity proof. The honest RP (i.e., the target of the adversary) checks the signature using the public key PK_{ID}, and only accepts the elements protected by the signature.

For the confidentiality of identity proof, Recluse inherits the same idea from OIDC, i.e., TLS, a trusted user agent and the checks. TLS avoids the leakage and modification during

the transmit. The trusted agent ensures the identity proof to be sent to the correct RP based on the endpoint specified in the $Cert_{RP}$. The $Cert_{RP}$ is protected by the signature with the un-leaked private key SK_{Cert}, ensuring it will never be tampered with by the the adversary. For Recluse, the checks at the IdP is exactly the same as OIDC, that is, checking the RP identifier and endpoint in the identity proof with the registered ones, preventing the adversary from triggering the IdP to generate an incorrect proof or transmit to the incorrect RP. However, the user in Recluse performs a two-step check instead of the direct check based on the RPID in OIDC. Firstly, the user checks the correctness of $Cert_{RP}$ and extracts $RPID_O$ and the endpoint. In the second step, the user checks that the RPID in identity proof is a fresh $RPID_T$ negotiated based on the $RPID_O$ and the endpoint is the one-time one corresponding to the one in $Cert_{RP}$. This two-step check also ensures the identity proof for the correct RP ($RPID_O$) is sent to correct endpoint (one specified in $Cert_{RP}$).

The mechanisms for binding are also inherited from OIDC. The IdP binds the identity proof with $RPID_T$ and $PPID$. The correct RP checks the binding by comparing the $RPID_T$ with the cached one, and provides the service to the $Account$ based on $PPID$.

Recluse binds the identity proof with $RPID_T$, instead of a random string unique for each RP assigned by IdP in OIDC. However, the adversary (malicious users and RPs) still fails to make one identity proof (or its transformation) accepted by the other correct RP. As the correct RP only accepts the valid identity proof for its fresh negotiated $RPID_T$, we only need to ensure one $RPID_T$ (or its transformation) never be accepted by the other correct RP.

- $RPID_T$ is unique in one IdP. The honest IdP checks the uniqueness of $RPID_T$ in its scope during the dynamic registration, to avoid one $RPID_T$ (in its generated identity proof) corresponding to two or more RPs.
- The mapping of $RPID_T$ and `issuer` globally unique. The identity proof contains the identifier of IdP (i.e., `issuer`), which is checked by the correct RPs. Therefore, the same $RPID_T$ in different IdPs will be distinguished.
- The $RPID_T$ in the identity proof is protected by the signature generated with SK_{ID}. The adversary fails to replace it with a transformation without invaliding the signature.
- The correct RP or user prevents the adversary from manipulating the $RPID_T$. For no extra benefits, the adversary can only know or control one entity in the login flow. The other correct one provides a random nonce (n_u or n_{RP}) for $RPID_T$. The nonce is independent from the ones previously generated by itself and the ones generated by others, which prevents the adversary controlling the $RPID_T$. , the correct

Recluse binds the identity proof with PPID in the form of $RPID_T^{UID}$, instead of a random string generated by the IdP. However, the adversary still fails to login at the correct RP using a same $Account$ as the uncontrolled user. Firstly, the adversary fails to modify the $PPID$ directly in the identity protected by SK_{ID}. Secondly, the malicious users and RPs fail to trigger the IdP generate a wanted $PPID$, as they cannot (1) obtain the uncontrolled user's $PPID$ at the correct RP; (2) infer the UID of any user from all the received information (e.g., $PPID$) and the calculated ones (e.g., $Account$); and (3) control the $RPID_T$ with the participation of a correct user or RP.

The design of Recluse makes it immune to some existing know attacks (e.g., CSRF, 307 Redirect, IdP Mix-Up [20] and Man-in-middle attack) on the implementations. The Cross-Site Request Forgery (CSRF) attack is usually exploited by the adversary to perform the identity injection. However, in Recluse, the correct user logs $RPID_T$ and one-time endpoint in the session, and perform the checks before sending the identity proof to the RP's endpoint, which prevents the CSRF attack. The 307 Redirect attacks [20] is due to the implementation error at the IdP, i.e. returning the incorrect status code (i.e., 307), which makes the IdP leak the user's credential to the RPs during the redirection. In Recluse, the redirection is intercepted by the trusted user agent which removes these sensitive information. In the IdP Mix-up attack, the adversary works as the IdP to collect the makes `access token` and `authorization code` from the victim RP. Same as OIDC, Recluse includes the `issuer` in the identity proof (protected by the SK_{ID}), avoiding the victim RP to send the sensitive information to the IdP. The user established the TLS connection with RP and IdP, avoids the Man-in-middle attack.

VII. Implementation And Evaluation

We have implemented the prototype of Recluse, and compared its performance with the original OIDC implementation and SPRESSO.

A. Implementation

We adopt SHA-256 to generate the digest, and RSA-2048 for the signature in the $Cert_{RP}$, identity proof and the dynamic registration response. We choose a random 2048-bit strong prime as P, and the smallest primitive root (3 in the prototype) of P as g. The n_u, n_{RP} and UID are 256-bit odd numbers, which provides no less security strength than RSA-2048 [**?**].

The IdP is implemented based on MITREid Connect [**?**]. an open-source OIDC Java implementation certificated by the OpenID Foundation [**?**]. Alghough, OIDC standard specifies that RP's identifier should be generated by IdP in the dynamic registration, MITREid Connect allows the user to provide a candidate RP identifier to the IdP who checks the uniqueness, which simplifies the implementation of Recluse. In Recluse, we add 3 lines Java code for generation of $PPID$, remove 1 line for checking the registration token in dynamic registration, while the calculation of $RPID_O$, $Cert_{RP}$, $PPID$,

and the RSA signature is implemented using the Java built-in cryptographic libraries (e.g., BigInteger)

The user-side processing is implemented as a Chrome extension with about 330 lines JavaScript code and 30 lines Chrome extension configuration files (specifying the required permissions). The cryptographic calculation in $Cert_{RP}$ verification, $RPID_T$ negotiation, dynamic registration, is based on an efficient JavaScript cryptographic library jsrsasign [**?**]. The Chrome extension clears the `referer` in the HTTP header, to avoid the RPs' URL leaked to the IdP.

We provide the SDK for RP to integrate Recluse easily. The SDK provides 3 functions: RP initial registration, processing of the user's login request and identity proof parsing. The Java SDK is implemented based on the Spring Boot framework with about 1100 lines JAVA code. The cryptographic computation is completed through Spring Security library. The user's login request contains Step 2, 3, 6, 7 and 12 in Figure 4; while identity proof parsing contains Step 18 in Figure 4.

Cross-Origin Resource Sharing (CORS). The chrome extension needs to construct cross-origin requests to communicate with the RP and IdP, which is forbidden by default by the same-origin security policy. Recluse adopts CORS to achieve this cross-origin communication. In details, we requires the RP and IdP to specify `chrome-extension://chrome-id` in the `Access-Control-Allow-Origin` field of its response header, which makes the request pass the permission checks at the browser. As `chrome-id` is unique assigned by the Google, no other (malicious) entity can perform the cross-origin communication.

B. Evaluation

We have compared the processing time of each user login in Recluse, with the original OIDC implementation (MITREid Connect) and SPRESSO which only hides the user's accessed RPs from IdP.

Environment. We run the evaluation on 3 physical machines connected in a separated 1Gbps network. A DELL OptiPlex 9020 PC (Intel Core i7-4770 CPU, 500GB SSD and 8GB RAM) with Window 10 prox64 works as the IdP. A ThinkCentre M9350z-D109 PC (Intel Core i7-4770s CPU, 128GB SSD and 8GB RAM) with Window 10 prox64 servers as RP. The user adopts Chrome v75.0.3770.100 as the user agent on the Acer VN7-591G-51SS Laptop (Intel Core i5-4210H CPU, 128GB SSD and 8GB RAM) with Windows 10 pro. For SPRESSO, the extra trusted entity FWD is deployed on the same machine as IdP. The monitor demonstrates that the calculation and network processing of the IdP does not become a bottleneck. For SPRESSO, one extra stronger host is adopted to deploy the user agent for better performance as analyzed later.

Performance. We have measured the processing time for 1000 login flows, the the results is demonstrated in Figure 5. The average time is 210 ms, 69ms and 295 ms for

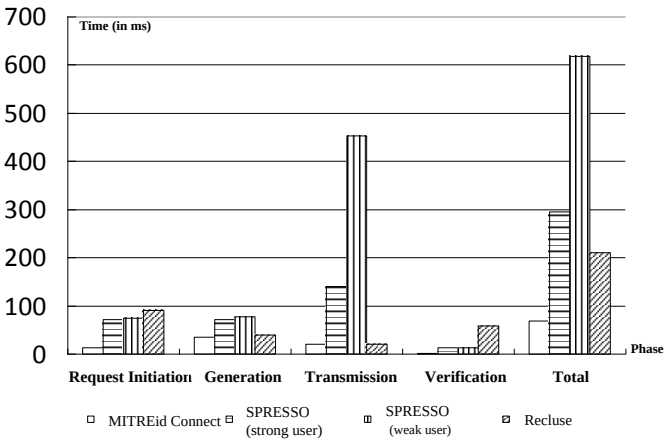

Fig. 5: The Evaluation.

Recluse, MITREid Connect and SPRESSO (stronger user) respectively. Compared to the original OIDC implementation (i.e., MITREid Connect), Recluse requires about the extra 135 ms processing time, which is composed of RP identifier transforming (about 49 ms), dynamic registration (42 ms), $PPID$ generation (2 ms), and $Acount$ calculation (54 ms). In details, the time for calculating t through Extended Euclidean algorithm needs 3 ms; the processing time of modular exponentiation, required in the calculation of $RPID_T$, and $PPID$, varies in the user (10 ms), IdP (2 ms) and RP (3 ms), as the execution of JavaScript implementation (at the user) requires more time than the Java implementations (at IdP and RP). Moreover, the modular exponentiation in $Account$ requires 54 ms, as the discrete logarithm t is no longer than 2048 bits, much longer than the n_{RP} and n_u (not longer than 256 bits) for $RPID_T$.

For better comparison, we further divide a SSO login flow into 4 phases: 1. **authentication request initiation**, the period which starts before the user sends the login request and ends after IdP has received the identity proof request; 2. **identity proof generation**, denoting the construction of identity proof at the IdP (excluding the user authentication); 3. **identity proof transmitting**, for transmitting the proof from the IdP to the RP with the user's help; and 4. **identity proof verification**, for the RP verifying and parsing the proof for the user's $Account$. The detailed comparison is shown in Figure 5.

In the authentication request initiation, MITREid Connect requires the shortest time (13 ms); SPRESSO needs 71 ms for RP to obtain the IdP's public information and encrypt its domain; Recluse needs 91 ms, for the $RPID_T$ calculation (1 modular exponentiation at the user and 2 at the RP) and dynamic registration.

For identity proof generation, MITREid Connect needs 35 ms (information constructing and signing); Recluse needs an extra 5 ms for the generation of $PPID$; while SPRESSO

requires 71 ms for a different format of identity proof.

For identity proof transmitting, IdP in MITREid Connect provides the proof as a fragment component (i.e., proof is preceded by #) to RP to avoid the reload of RP document; and RP uses the JavaScript code to send the proof to the background server; the total transmitting requires 20 ms. In Recluse, a chrome extension relays the identity proof from the IdP to RP, which needs 21 ms. The transmitting in SPRESSO is much complicated: The user's browser creates an iframe of the trusted entity (FWD), downloads the JavaScript from FWD, who obtains the RP's correct URL through a systematic decryption and communicates with the parent opener (also RP's document, but avoiding leaking RP to IdP) and RP's document through 3 postmessages. The time for transmitting identity proof in SPRESSO, relies on the performance of the user's host, 453 ms in our original user, and 140 ms in a stronger user.

In identity proof verification, the RP in MITREid Connect needs 1 ms for verifying the signature, SPRESSO requires 13 ms for a systematic decryption and signature verification, while Recluse needs 58 ms for calculation of $Account$ and signature verification.

VIII. Discussion

In this section, we provide some discussion about Recluse.

RP Certificate. In Recluse, the RP certificate $Cert_{RP}$ is used to provide the trusted binding between the $RPID_O$ and the RP's endpoint. RP certificate is compatible with the X.509 certificate. To integrate RP certificate in X.509 certificate, the CA generates the $RPID_O$ for the RP, and combines it in the subject filed (in detail, the common name) of the certificate while the endpoint is already contained. Instead of sending in Step 3 in Figure 4, $Cert_{RP}$ is sent to the user during the key agreement in TLS. Moreover, the mechanisms (e.g., the Certificate Transparency) to avoiding illegal certificate issued by the CA may be adopted to ensure the correctness of $RPID_O$, i.e., gloally unique and being the primitive root.

Platform. Recluse doesn't store any persistent information in the platform and may be implemented to be platform independent. Firstly, all the information (e.g., $Cert_{RP}$, $RPID_T$, n_u, $PPID$ and one-time endpoint) processed and cached in the user's platform is only correlated with the current session, which allows the user to login at any RP with a new platform without any synchronization. Secondly, in the current implementation of Recluse, a browser extension is adopted to capture the redirection from the RP and IdP, to reduce the modification at the RP and IdP. However, Recluse is able to be implemented without based on HTML5, avoiding the use of any no browser extensions, or plug-ins. The redirect URL is placed as one element in the response which triggers the JavaScript a the user to send a message to this URL. The functions at the user are processed in the JavaScript running in a Shadow DOM.

Support of authorization code flow. Recluse may be extended to hide the users' access trace in the authorization code flow. The RP obtains the authorization code in the same way as the identity proof in implicit protocol flow. However, the RPs needs to connect to the IdP directly, and uses this code with the RP identifier and secret for the identity proof. To avoid the IdP obtain the IP address from the connection, the anonymous network (e.g., Tor) may be used to establish the connection. While the RP's identifier and secret may be provided by the user who performs the dynamic registration described above.

DoS attack. The adversary may perform the DoS attack. The malicious RPs may try to exhaust the $RPID_O$ by applying the $Cert_{RP}$ frequently. However the large P provides a large set of $RPID_O$, and IdP may provide the offline check for $Cert_{RP}$ as it occurs only once for a RP (i.e., the initial registration). The malicious users may attempt to make the other users' $RPID_T$ be rejected at the IdP, by registering a large set of $RPID_T$s at IdP. However, the large P makes a huge number of dynamic registration required, and IdP may adopt existing DoS mitigation to limit the number of adversary's dynamic registrations.

Side channel attack. Recluse provides the security and privacy, under the assumption that IdP never leaks r of $RPID_O = g^r$ and UID, while RP and the user never leaks $n_u * n_{RP}$ and t. The malicious user may infer the r and UID by analyzing the timing difference of the responses. IdP may avoid this side channel attack by adding unpredicted or maximum needed delay. The curious IdP who fails to observe the global traffic, fails to perform the side attack on $n_u * n_{RP}$ and t, as it fails to measure the timing correlated with these two values.

IX. Related Works

SSO was first proposed in **. Now, the typical SSO standards include Kerberos, SAML, OAuth, OIDC and CAS, which has been adopted and implemented by Google, Facebook, and Twitter. 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.

Various attacks were proposed for the impersonation attack and identity injection, by breaking the confidentiality, integrity and binding of identity proof. *** Besides of OAuth 2.0 and OpenID Connect 1.0, Juraj et al. [27] 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.

The SSO is also adopted in the mobile application, for example, ** ** and ** provide the mobile SSO service. **How-**

ever, new attacks were found, as the mobile applications fails to ensure the confidentiality, integrity and binding. ****

The comprehensive formal security Analysis were performed on SAML, OAuth and OIDC. 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. Other security analysis [21] [22] [19] [24] [28] on SSO system concludes the rules SSO protocol must obey with different manners.

As suggested in NIST SP800-63C [5], user's privacy should be protected in SSO systems, which is partially achieved in the SSO standards, BrowerID and SPRESSO. The user's privacy protection [5] in SSO systems includes 1) the user should be able to control the range of the attributes exposed to the RP, 2) multiple RPs should fail to link the user through collusion, 3) IdP should fail to obtain the trace of RPs accessed by a user. The first property is satisfied in most SSO systems. For example, in OAuth and OIDC, IdP exhibits the attributes requested by the RP and sends the attributes to the RP only when the user has provided a clear consent, which may also minimize the exposed attributes as the user may disagree to provide partial attributes. BrowserID [12] [31] 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.

Anonymous SSO scheme is proposed to hide the user's identity to both the IdP and RPs, which can only be applied to the services that do not need the user's unique identifier. 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. [30], 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. [30] 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. In 2010, Han et al. [29] proposed a dynamic SSO system with digital signature to guarantee unforgeability. To protect user's privacy, it uses broadcast encryption to make sure only the designated service providers is able to check the validity of user's credential. User uses zero-knowledge proofs to show it is the owner of the valid credential. But in this system verifier is unable to find out the relevance of same user's different requests so that it cannot provide customization service to a user. So this system is not appropriate for current web applications. In 2013, Wang et al. proposed anonymous single sign-on schemes transformed from group signatures. In an ASSO scheme, a user gets credential from a trusted third party (same as IdP) once. Then user is able to authenticate itself to different service providers (same as RP) by generating a user proof via using the same credential. SPs can confirm the validity of each user but should not be able to trace the user's identity.

X. CONCLUSION

In this paper, we, for the first time, propose Recluse to preserve the users' privacy from the curious IdP and colluded RPs, without breaking the security of SSO systems. The identity proof is bound with a transformation of the original identifier, hiding the users' accessed RPs from the curious IdP. The user's account is independent for each RP, and unchanged to the destination RP who has the trapdoor, which prevents the colluded RPs from linking the users and allows the RP to provide the consecutive and individual services. The trusted user ensures the correct content and transmit of the identity proof with a self-verifying RP certificate. The evaluation demonstrates the efficiency of Recluse, about 200 ms for one user's login at a RP in our environment.

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] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.