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

Single sign-on (SSO) systems, such as OAuth, OpenID and SAML, have been widely adopted nowadays as a convenient web authentication mechanism. SSO delegates user authentication on websites to a third party, so-called identity providers (IdPs), so that users can access different services on cooperating sites, so-called relying parties (RPs), via a single authentication process. Using SSO, a user no long needs to maintain multiple credentials for different RPs, instead, she only maintains the credential for the IdP, which further provides identity proofs to RPs. For RPs, SSO shifts the burden of user authentication to IdPs and therefore reduces their own security risks and costs. As a result, SSO has been widely integrated with modern web systems. According to Alexa [**?**], 80 websites among the top-100 websites integrate SSO services. Meanwhile, many email and social networking providers such as Google, Facebook, Twitter, etc. have been actively serving as social identity providers to support social login.

However, the wide adoption of SSO also raises new privacy concerns [5]–[7]. In a typical SSO-based authentication session, such as the OpenID Connect (OIDC) example shown in Fig. 1, when a user attempts to log in to an RP, the authentication request is redirected from the RP to the IdP, who authenticates the user and creates an identify proof accordingly for the particular user and the particular RP. This identity proof contains information about both the user (e.g., user identifier) and the RP (e.g., RP identifier, URL, etc.). When a user leverages the identity issued by the IdP across multiple RPs, the IdP acquires a central role in web authentication, which enables it to collect information about the user's logins on different sites. We refer to this as *IdP-based access tracing*. Moreover, if user identifiers issued by the IsP for the same user across different RPs are unique or can be derived from each other, which is the case in most of the existing SSO systems [add citations], colluding RPs can not only track her online traces but also aggregate her attributes from these sites. We refer to this as *RP-based identity linkage*.

Both IdP-based access tracing and RP-based identity linkage can lead to more severe privacy risks such as web tracking, online user profiling, etc. Large IdPs, especially social IdPs like Google, Facebook, etc., are accused of being interested in collecting users' online behavioral information for various purposes (e.g., Screenwise Meter [9], Onavo [10]). Meanwhile, many IdPs are service providers themselves, hosting a variety of web services (i.e., RPs). By integrating data from their own services, they may obtain rich information to profile a user. On the contrary, unlike other privacy risks, such as user session re-identification that requires to compute the similarity between users' DNS queries [8], any interested IdP can easily discover the RPs accessed by a user since the information of the RP is necessary in the construction of identity proof, and associate a user's access traces by her unique identifier.

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 [13]–[15], 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 [16]. 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 [17], [18], and makes impersonation attack possible [19], [20]. 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 using the new id token as the identity proof. In details, id token includes the user's PPID, the RP's identifier (RPID), 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 RP previously registered endpoint. 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 RPID with the identifier stored locally. Figure 1 provides the details in the implicit flow of OIDC. The detailed processes are as follows:

- Step 1: User attempts to login at one RP.

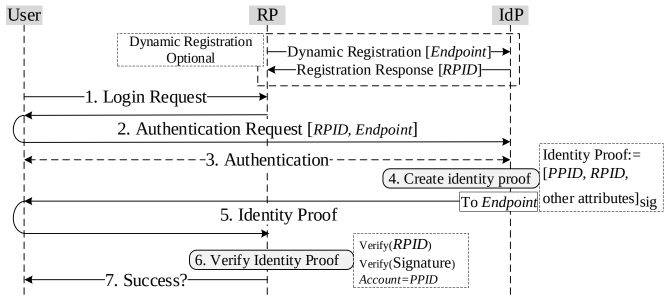

User
RP
IdP
Dynamic Registration Optional
Dynamic Registration [Endpoint]
Registration Response [RPID]
1. Login Request
2. Authentication Request [RPID, Endpoint]
3. Authentication
4. Create identity proof
Identity Proof:= [PPID, RPID, other attributes]sig
To Endpoint
5. Identity Proof
6. Verify Identity Proof
Verify(RPID)
Verify(Signature)
Account=PPID
7. Success?

Fig. 1: The implicit protocol flow of OIDC.

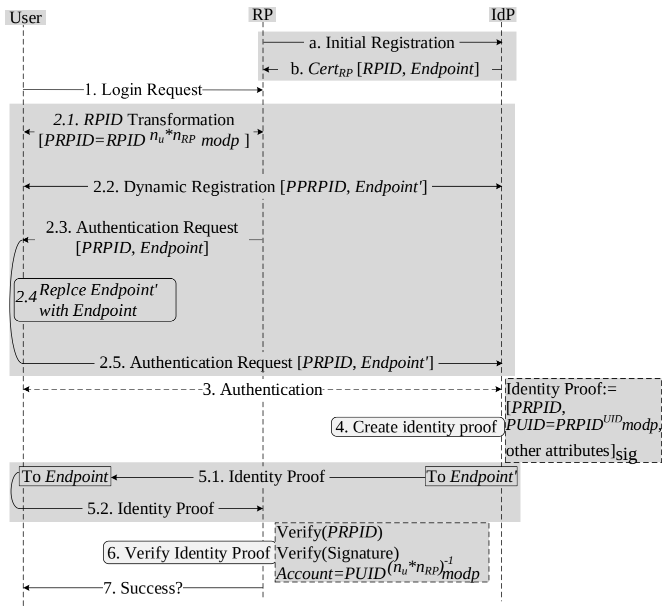

User
RP
IdP
a. Initial Registration
b. Cert_RP [RPID, Endpoint]
1. Login Request
2.1. RPID Transformation [PRPID=RPID^{n_u*n_RP} modp]
2.2. Dynamic Registration [PPRPID, Endpoint']
2.3. Authentication Request [PRPID, Endpoint]
2.4 Replce Endpoint' with Endpoint
2.5. Authentication Request [PRPID, Endpoint']
3. Authentication
Identity Proof:= [PRPID, PUID=PRPID^{UID} modp, other attributes]sig
4. Create identity proof
To Endpoint
5.1. Identity Proof
To Endpoint'
5.2. Identity Proof
6. Verify Identity Proof
Verify(PRPID)
Verify(Signature)
Account=PUID^{(n_u*n_RP)^{-1}} modp
7. Success?

Fig. 2: The Recluse.

- Step 2: The RP redirects the user to the corresponding IdP with a newly constructed request of identity proof. The request contains RPID, the registered endpoint, and the set of requested attributes.
- Step 3: If the user hasn't been authenticated, an extra authentication process is performed.
- Step 4, 5: The IdP generates the identity proof for the user who has been authenticated already and constructs the response with endpoint in request if it is the same with the registered one for the RP.
- Step 6, 7: The RP verifies the id token, identifies the user with PPID in the id token, and returns the authentication result to user.

Dynamic Registration. OIDC provides the dynamic registration [16] mechanism to register the RP for a new RPID, dynamically. After the first successful registration, RP obtains a registration token from the IdP, and is able to update its information (e.g., the endpoint) by a dynamic registration process with the registration token. One successful dynamic registration process will make the IdP assign a new unique RPID for this RP.

B. Basic Requirements of SSO

Based on the desiring function and the security analysis of SSO systems, we conclude the basic requirements of SSO. As our work is focused on the implicit flow of OIDC, the following requirements are illustrated based on the implicit flow of OIDC. The requirements are shown as follows:

1. **Identification**. Identification is the main feature of SSO system, which enables the RP to identify the user. Moreover, for web application, it is necessary to provide continuous and personalize service, which requires the RP to link the user with the specific account in RP. Therefore, the identity proof provided by the IdP must be linkable in the destination RP. That is, some of the existing SSO systems provide the pairwise identifier (e.g., PPID in OIDC) and others provide the unchanged identifier (e.g., email in SPRESSO) within the identity proof.
2. **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 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.
3. **Confidentiality**. The identity proof should only be obtained by the correct RP (and the user), no one else may get the identity proof [19]–[21]. 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.
4. **Integrity**. The valid identity proof should only be generated by the IdP, no one else may forge or modify a valid proof [20]. Otherwise, the adversary may replace the user's identifier in the proof for either impersonation attack and identity injection. The solution to ensure the integrity of identity proof 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 [22], [26].

III. Challenges and Solutions

In this section, we firstly describe the challenges in designing the privacy-preserving SSO system which fulfils the basic requirements and privacy at the same time. Then we propose and fulfil the enhanced requirements in Recluse which is based on the basic requirements and privacy issues.

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. To prevent the RPs's linkage of the single user, the user identifiers provided by IdP for one user should never be same or derivable to each other. However, it conflicts with the basic requirements on identity proof described in Section **??**:

- **No Identification.** 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 is only able to provide different PPIDs for user's multiple logins at the same RP, which makes RP fail to provide the consecutive and individual services.
- **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 [19]–[21].
- **No Confidentiality.** The confidentiality of identity proof is corrupted, and the leaked identity proof may result in the impersonation attacks [19]–[21]. 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.

Besides, as the **Integrity** is ensured by the signature generated by IdP, which is not related with the identity of

RP, it is not the challenge in designing privacy-preserving SSO system to avoid the identity proof to be tampered.

B. Recluse Overview

Recluse aims to hide the users' traces from both the IdP and colluded RPs, without violating the basic requirements of SSO systems. Therefore, we propose the enhanced requirements, the privacy-preserving version basic requirements of SSO systems and illustrate the methods to fulfil the requirements.

- **Trapdoor-existing Identification.** The trapdoor-existing identification requires that the user identifier generated by IdP should be never same in each authentication as the RP's identity is unknown to IdP and the identifier should be derivable to the specific account unchanged in RP 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.
- **Transformed Binding.** The transformed binding requires that the RP should provide the one-time transformed identifier, unlinkable to the RP's unique identifier without trapdoor and the identity proof should be bound with the one-time identifier. In Recluse, 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.
- **User-centric Confidentiality.** The user-centric confidentiality requires the checking of RP's endpoint and notification of RP's information should be shifted from IdP, who does not know the RP's identity to user. 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.

IV. Threat Model and Assumption

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

A. Threat Model

Honest but Curious IdP. All the action performed by IdP fulfils the requirements of Recluse and none of action undefined is conducted. But the IdP might try to achieve the additional information which is not defined to be exposed to IdP passively without breaking the rule of Recluse. 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. Moreover, 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.

Malicious User. The user might be under the full control of adversaries, so that the malicious user is able to conduct any action defined or not defined in the Recluse. 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 RP might be corrupted by the adversary or just built by the adversary, which is able to perform any action desired by the adversary for any purpose. 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. 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.

Moreover, the conclusion is also considered available in Recluse, of which the details are shown as follows:

Conclusion between RP and user. 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).

Conclusion among RPs. 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. But 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.

B. Assumptions

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 $g^r modP$, where P is a large prime and g is the primitive root. However, although IdPs are interested in the user's login trace, IdP will never collude with RPs.

V. RECLUSE

In this section, we firstly introduce the principle of Recluse, which introduces the goals in Recluse and the requirements. Then we describe the initialization of Recluse, including system initialization and initial registration of RP, following with the description of algorithms for calculating the RP identifier transformation, user identifier 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. Principle

Goals in Recluse. As analyzed in Section **??**, to enhanced requirements of SSO systems, Recluse has to allow only the exact RP to derive the user's unchanged account with a trapdoor, bind the proof with a transformation of RP's identifier, and achieve user-centric confidentiality of the identity proof. We further divide these three requirements into 5 goals and achieve these 5 goals using 5 phases which is formed as the process flow in Recluse:

- 1. Providing a self-verifying value for the user-centric check (for **User-centric Confidentiality**), to ensure the correct construction and transmission of identity proof (i.e., user-centric confidentiality), which is achieved in RP initial registration (step a, b in Figure **??**), applying for a RP certificate ($Cert_{RP}$), a self-verifying value;
- 2. Providing a RP identifier transformation (for **Transformed Binding**), 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, achieved in RP identifier transforming (step 2.1 in Figure **??**) generating transformation (denoted as $PRPID$) of RP's original identifier (denoted as $RPID$);
- 3. Checking the uniqueness of the transformation (for **Transformed Binding**), to avoid one identity proof being accepted by two or more RPs, which will harm the security of SSO systems, achieved in Dynamic registration (step 2.2 in Figure **??**), checking the global uniqueness of $RPID$;
- 4. Unlinkable PUID (for **Trapdoor-existing Identification**), for one user, the PUIDs for different RPs should vary, to prevent identity linkage, which is achieved in $PUID$ generation (step 4 in Figure **??**), generating PUID at the IdP, and transferring PUID to the RP;
- 5. Unchanged account (for **Trapdoor-existing Identification**), to allow only the destination RP (with the trapdoor) to derive the unchanged account for one user from varying PUIDs, which is necessary for providing the consecutive and individual services, achieved in $Account$ verifying (step 6 in Figure **??**), calculating the user's account in the RP.

To achieve the above goals, the main features in Recluse is the transformed RP identifier and pairwise user identifier generation which is described detailedly in Section V-C. $PRPID$ is the transformed RP identifier transmitted to IdP based on the $RPID$ (presenting the unique identity of RP), an one-time random number N as the large prime number p ($RPID$ is the primitive root of p) as $PRPID = RPID^N \ mod \ p$. $PUID$ is the user identifier provided by IdP to RP, which is generated as $PUID = PRPID^{UID} \ mod \ p$ where UID is

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.
$PUID$	User's privacy-preserving id in the identity proof.
$Account$	User's identifier at a RP.
$RPID$	RP's original identifier.
$PRPID$	The privacy-preserving $RPID$.
n_u	User-generated random nonce for $PRPID$.
n_{RP}	RP-generated random nonce for $PRPID$.
Y_{RP}	Public value for n_{RP}, $(RPID)^{n_{RP}}\ modP$.
t	A trapdoor, $t = (n_u * n_{RP})^{-1} mod\ (P-1)$.

the unique user identifier stored and only known by IdP. The non-derivability of $RPID$ and UID is based on the Discrete Logarithm Problem.

Discrete Logarithm Problem. 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 [27]), 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.

B. System Initialization and RP Initial Registration

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 **??** 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 $RPID$ 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 **??**, 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}.

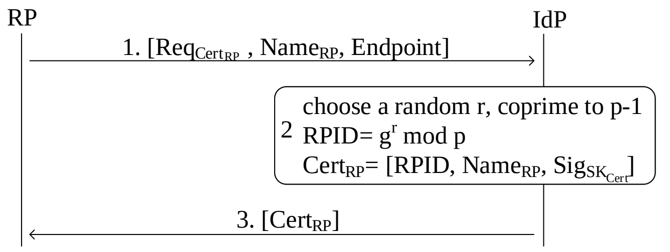

RP
IdP
1. [Req$_{Cert_{RP}}$, Name$_{RP}$, Endpoint]
2
choose a random r, coprime to p-1
RPID= g^r mod p
Cert$_{RP}$= [RPID, Name$_{RP}$, Sig$_{SK_{Cert}}$]
3. [Cert$_{RP}$]

Fig. 3: RP initial registration.

The IdP setup and following RP initial registration are based on the features of primitive root. 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 [28], [29].

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:

- 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 $RPID = g^r mod\ P$ 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 [$RPID, Name_{RP}$] using SK_{Cert}, and returns [$RPID, Name_{RP}, Sig_{SK_{Cert}}$] 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 $RPID$ for **Goal 4**:

- r should be coprime to $p-1$. This makes $RPID$ 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 $RPID$, obtain the same PUID for a user, which makes identity linkage possible.

The honest IdP is assumed to generate the correct $RPID$. However, we may perform an external check on $Cert_{RP}$ and $RPID$, 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 $RPID$. 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 $RPID$ and $RPID$ is a primitive root modulo p. 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 $PRPID$, $PUID$ and $Account$ separately, which is the foundation of each user's login process that described in Section V-D.

$PRPID$. Similar to Diffie-Hellman key Exchange [30], the RP and user generate the $PRPID$ cooperatively as follows:

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

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

$$PRPID = {Y_{RP}}^{n_u} = RPID^{n_u * n_{RP}} mod\ P \tag{1}$$

The generation ensures that $PRPID$ 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 $PRPID$ 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 $PRPID$ 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:

- $PRPID$ is a primite root modulo p, which prevents the RP from inferring UID from $PUID$ (**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:

$$(PRPID)^t = RPID\ mod\ P \tag{2}$$

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

$$PUID = PRPID^{UID}\ mod\ P \tag{3}$$

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

$$Account = (PRPID^{UID})^t = RPID^{UID} mod\ P \tag{4}$$

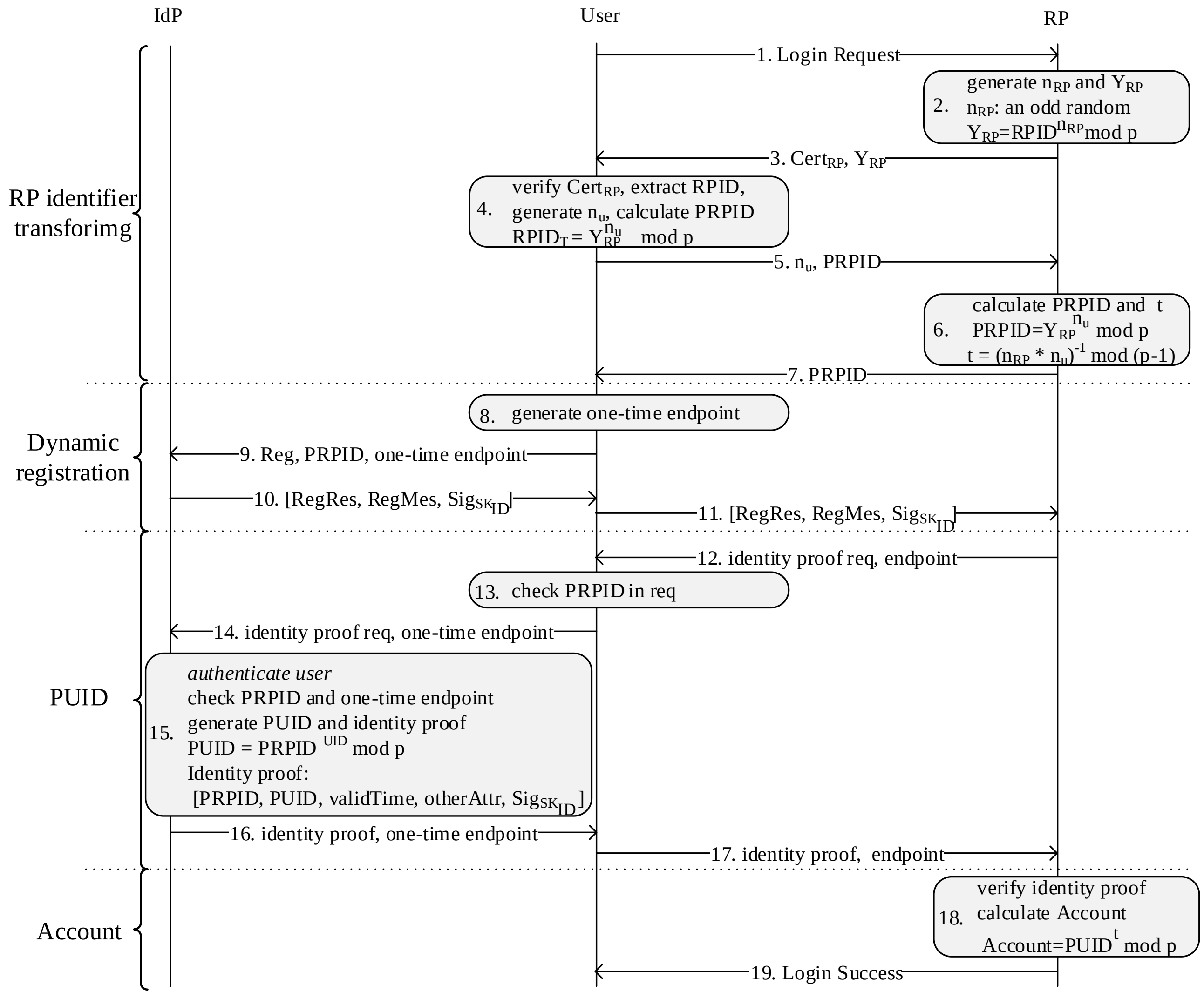
D. User Login Process

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 **??**.

For RP identifer transforming, the user and RP corporately process as follows. **(1)** The user sends a login request to trigger the negotiation of $PRPID$. **(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 $RPID$ from $Cert_{RP}$, and calculates $PRPID$ with a random chosen n_u as in Section V-C. **(5)** The user sends n_u and $PRPID$ to the RP. **(6)** RP calculates $PRPID$ using the received n_u with Y_{RP} as in Section V-C, and rejects the user's login request if the calculated $PRPID$ 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 $PRPID$ to the user, who will halt the login if the received $PRPID$ 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 $PRPID$ is accepted. **(9)** Then, the user registers the RP with the $PRPID$ and one-time endpoint. **(10)** If $PRPID$ 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 $PRPID$, issuing time 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 $PRPID$ within the expiration date. The user and RP will negotiate a new $PRPID$ if $RegRes$ is $FAIL$.

To acquire the PUID, the user corporates with the RP and IdP as follows. **(12)** RP constructs an identity proof request with the correctly registered $PRPID$ and the endpoint (the form of the request is detailed in Section V-E). **(13)** The user halts the login process if the received $PRPID$ 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 $PRPID$ and the one-time endpoint doesn't exist in the registered ones. Then, IdP generates the $PUID$ as in Section V-C, and constructs the identity proof with $PRPID$, $PUID$, 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



IdP
User
RP
RP identifier transforimg
Dynamic registration
PUID
Account
1. Login Request
2. generate n_RP and Y_RP; n_RP: an odd random; Y_RP=RPID^n_RP mod p
3. Cert_RP, Y_RP
4. verify Cert_RP, extract RPID, generate n_u, calculate PRPID; RPID_T = Y_RP^n_u mod p
5. n_u, PRPID
6. calculate PRPID and t; PRPID=Y_RP^n_u mod p; t = (n_RP * n_u)^-1 mod (p-1)
7. PRPID
8. generate one-time endpoint
9. Reg, PRPID, one-time endpoint
10. [RegRes, RegMes, Sig_SK_ID]
11. [RegRes, RegMes, Sig_SK_ID]
12. identity proof req, endpoint
13. check PRPID in req
14. identity proof req, one-time endpoint
15. authenticate user; check PRPID and one-time endpoint; generate PUID and identity proof; PUID = PRPID^UID mod p; Identity proof: [PRPID, PUID, validTime, otherAttr, Sig_SK_ID]
16. identity proof, one-time endpoint
17. identity proof, endpoint
18. verify identity proof; calculate Account; Account=PUID^t mod p
19. Login Success

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

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 $PUID$ as follows. **(18)** RP accepts the identity proof only when the signature is correctly verified with PK_{ID}, $PRPID$ 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. Compatibility 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 element of the identity proof request in OIDC is contained in Recluse as follows: the RP's identifier ($PRPID$ 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 ($PRPID$ in Recluse), the user's PUID, 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 $PRPID$ 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 $PRPID$ 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. To avoid the IdP to infer the RP's identity through the registration token in dynamic registration, the access control of the endpoint for dynamic registration is deleted in Recluse. As the registration response is transmitted through the user instead of server-to-server transmission between RP and IdP, the extra signature is required to guarantee the integrity of response. 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 firstly 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 [31].

A. Privacy

Curious IdP. The curious IdP might be interested in the user accessed RP or infer the correlation of RPs in two or more login flows by performing 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 might be interested in the RP's identity but fails to derive RP's identifying information (i.e., $RPID_O$ and correct endpoint) through a single login flow. IdP only receives $PRPID$ and one-time endpoint, and fails to infer the discrete logarithm $RPID$ from $PRPID$ 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 might try to infer the correlation of RPs in two or more login flows but fails achieve the relationship between the $PRPID$s. The secure random number generator ensures the random for generating $PRPID$ and the random string for one-time endpoint are independent in multiple login flows. Therefore, curious IdP fails to classify the RPs based on $PRPID$ 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 $PRPID$). 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 might try to find out the UID presenting the unchanged user identity in each RP but fails to infer the user's unique information (e.g., UID or other similar ones) in the passive way. The $PUID$ 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 $PUID$, 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 $PRPID$ 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 $PRPID$ (e.g., 1) be used for $PUID$, as the honest IdP only accepts a primitive root as the $PRPID$ in the dynamic registration. The RP also fails to change the accepted $PRPID$ in Step 11 in Figure 4, as the user checks it with the cached one.
- Two or more RPs might try to find out whether the $Account$s in each RP are belong to one user or not but fail to link the user in the passive way. The analysis can only be performed based on $Account$ and $PUID$.

The $Account$ is independent among RPs, as the $RPID$ chosen by honest IdP is random and unique. The $PUID$s are also independent due to the unrelated $PRPID$.

- Two or more RPs also might to lead IdP to generate the same $PUID$ or the $PUID$ derivable into same $Account$ in each RP. Since the $PUID$ is generated related with the $PRPID$, corrupted RPs might choose the related n_{RP} to correlate their $PRPID$, however, the $PRPID$ is also generated with the participation of n_U, so that RP does not have the ability to control the generation of $PRPID$. Moreover, corrupted RPs might choose the same $PPID$ to lead the IdP to generate the $PUID$ derivable into same $Account$, however, $RPID$ is verified by the user with through the $Cert_{RP}$, where the tampered $RPID$ is not acceptable to the honest user.

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 $RPID$ and the endpoint. In the second step, the user checks that the RPID in identity proof is a fresh $PRPID$ negotiated based on the $RPID$ 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 ($RPID$) 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 $PUID$. The correct RP checks the binding by comparing the $PRPID$ with the cached one, and provides the service to the $Account$ based on $PUID$.

Recluse binds the identity proof with $PRPID$, 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 $PRPID$, we only need to ensure one $PRPID$ (or its transformation) never be accepted by the other correct RP.

- $PRPID$ is unique in one IdP. The honest IdP checks the uniqueness of $PRPID$ in its scope during the dynamic registration, to avoid one $PRPID$ (in its generated identity proof) corresponding to two or more RPs.
- The mapping of $PRPID$ 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 $PRPID$ in different IdPs will be distinguished.
- The $PRPID$ 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 $PRPID$. For extra benefits, the adversary can only know or control one entity in the login flow (if controlling the two ends, no victim exists). The other correct one provides a random nonce (n_u or n_{RP}) for $PRPID$. The nonce is independent from the ones previously generated by itself and the ones generated by others, which prevents the adversary controlling the $PRPID$.

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 $PUID$ directly in the identity protected by SK_{ID}. Secondly, the malicious users and RPs fail to trigger the IdP generate a wanted $PUID$, as they cannot (1) obtain the uncontrolled user's $PUID$ at the correct RP; (2) infer the UID of any user from all the received information (e.g., $PUID$) and the calculated ones (e.g., $Account$); and (3) control the $PRPID$ 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 [21] 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 $PRPID$ 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 [21] 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 [32].

The IdP is implemented based on MITREid Connect [33]. an open-source OIDC Java implementation certificated by the OpenID Foundation [34]. 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 [35]. The Chrome extension clears the `referer` in the HTTP header, to avoid the RPs' URL leaked to the IdP. Moreover, 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, by adding the HTTP header `Access-Control-Allow-Origin` in the response of IdP and RP to accept only the request from the origin `chrome-extension://chrome-id` (`chrome-id` is unique assigned by the Google).

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.

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, 3.4GHz, 500GB SSD and 8GB RAM) with Window 10 prox64 works as the IdP. A ThinkCentre M9350z-D109 PC (Intel Core i7-4770s CPU, 3.1GHz, 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, 2.9GHz, 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 208 ms, 113 ms and 308 ms for Recluse, MITREid Connect and SPRESSO respectively. Compared to the original OIDC implementation (i.e., MITREid Connect), Recluse requires about the extra 95 ms processing time, which is composed of RP identifier transforming (about 46 ms), dynamic registration (about 52 ms), PID generation (6 ms), and $Acount$ calculation (about 44 ms), but reduces in identity proof transmission (about 43 ms) and minus the of authentication request initiation (about 10 ms). In details, the time for calculating t through Extended Euclidean algorithm needs about 3 ms; the processing time of modular exponentiation, required in the calculation of $PRPID$, and PID, varies in the user (about 22 ms), IdP (about 2 ms) and RP (about 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 about 46 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 $PRPID$.

For better comparison, we further divide a SSO login flow into 4 phases, which : 1. **authentication request initiation**

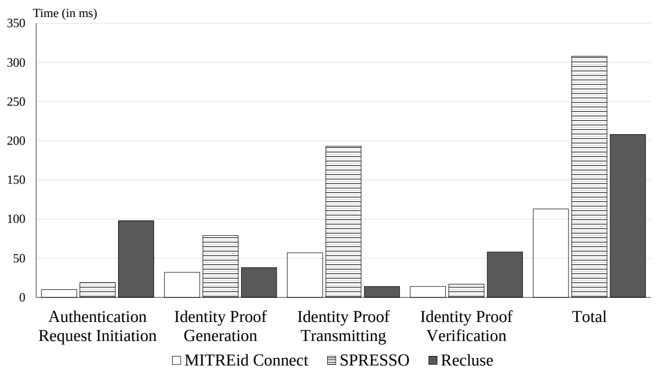


Fig. 5: The Evaluation.

(step 1-14 in Figure 4), the period which starts before the user sends the login request and ends after the user receive the identity proof request transmitted from itself. 2. **identity proof generation (step 15 in Figure 4)**, denoting the construction of identity proof at the IdP (excluding the user authentication); 3. **identity proof transmitting (step 16-17 in Figure 4)**, for transmitting the proof from the IdP to the RP with the user's help; and 4. **identity proof verification (step 18 in Figure 4)**, for the RP verifying and parsing the proof for the user's $Account$. However, the http transmission is consisted of the pairwise request and response, so that in the implementation of timer, the step 14 and 19 are counted as the identity proof transmitting. To avoid the time difference in each computer, we consistently anchor the time point at user agent where the time is always achieved from the user's PC. The detailed comparison is shown in Figure 5.

In the authentication request initiation, MITREid Connect requires the shortest time (10 ms); SPRESSO needs 19 ms for RP to obtain the IdP's public information and encrypt its domain; Recluse needs 98 ms, for the $PRPID$ calculation (1 modular exponentiation at the user and 2 at the RP) and dynamic registration.

For identity proof generation, MITREid Connect needs 32 ms (information constructing and signing); Recluse needs an extra 6 ms for the generation of $PPID$; SPRESSO requires 71 ms for a different format of identity proof, which is longer that the other two as the processing in SPRESSO is implemented with JavaScript while the others are using Java.

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 57 ms. In Recluse, a chrome extension relays the identity proof from the IdP to RP, which needs 14 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 post messages.

In identity proof verification, the RP in MITREid Connect needs 14 ms for verifying the signature, SPRESSO requires 17 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 $RPID$ 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 $RPID$ 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 $RPID$, 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., $Cert_{RP}$, $PRPID$, 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 $RPID$ by applying the $Cert_{RP}$ frequently. However the large P provides a large set of $RPID$, 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' $PRPID$ be rejected at the IdP, by registering a large set of $PRPID$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 $RPID = g^r$ 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.

Identity injection by malicious IdP. It has been discussed in [7] that even the impersonate attack by malicious IdP is not considered, the malicious might lead the user to access the RP as the identity of the adversary. SPRESSO requires that user should input his/her email at RP to avoid the identity injection, which is also available in Recluse by adding the extra user name (defined by user for each RP) input.

IX. Related Works

Various standards have been proposed to comfort the requirements of SSO systems, for example, OIDC is adopted by Google, OAuth 2.0 is adopted by Facebook and other standards such as SAML, CAS [37], Kerberos [36] and so on.

Various attacks were proposed for the impersonation attack and identity injection, by breaking the confidentiality, integrity and binding of identity proof, and extensive efforts have been devoted to the security considerations of SSO systems. In 2012, Juraj et al. [26] found XSW vulnerabilities which allows attackers insert malicious elements in 11 SAML frameworks and Wang et al. [22] proposed the traffic-guided analysis of SSO systems and found out several flaws in different systems. In 2014 Zhou et al. [23], in 2016 Wang et al. [20] and Yang et al. [25] built the automatic tester to analyse the implementations of existing applications and achieve the statistical result of these applications. Moreover, the malicious IdP is also considered, where in 2018 the Mohammad et al. [38] demonstrates the vulnerabilities and protect of IdP account hijack, which is the single failure in SSO systems and in 2016 and 2017 Christian et al. [39], [40] proposed the corrupted IdP might compromise the account associated with other IdPs. Besides, other analysis about SSO systems in various directions, such as in 2013 Armando et al. [41] issued the specific code injection in Google SSO system results in the impersonate attack, in 2014 Cao et al. [42] discussed about the security of communication channel between the RP and the IdP and in 2018 Yang et al. [43] analysed the SDK implementation of OAuth 2.0.

The SSO is also adopted in the mobile application, for example, Google, Facebook and other IdPs have already provided the mobile SSO service. However, new attacks were found, as the mobile applications fails to ensure the confidentiality, integrity and binding. In 2014 Chen et al. [19] generally demonstrated the difference between authentication and authorization and the challenges introduced by the migration of SSO systems from website to mobile application and in 2016 Mohsen et al. [44] proposed the security of SSO systems implemented through WebView, one of the most important Android components. Moreover, in 2016 Wang et al. [20] analysed the design and implementation of SSO systems for multiple platforms with the automatic testing. In 2015 Wnag et al. [45], in 2017 Yang et al. [46] and in 2019 Shi et al. [47] issued the new vulnerabilities in mobile SSO systems and conducted security assessments for the top Android applications and and achieve the statistical result of these applications.

The comprehensive formal security Analysis were performed on SAML, OAuth and OIDC. Firstly in 2008 Armando et al. [48] built the formal model of the protocol implemented in the SAML-based Google SSO system and revealed the flaws. In 2016 and 2017 Fett et al. [21], [31] conducted the formal analysis of the OAuth 2.0 and OpenID Connect standards using an expressive Dolev-Yao style model, and found the flaws in the implementation of SSO systems. Finally, it is proved that OAuth 2.0 and OpenID Connect satisfies the authorization and authentication requirements with the fixes. Besides, in 2015 Ye et al. [49] performed a formal analysis on the implementation of Android SSO systems and identified a major vulnerability in the existing Facebook Login implementation on Android system.

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 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 and employ technical measures, such as the use of pairwise pseudonymous to prevent the user linkage among multiple RPs. In 2014 Chen et al. [19] and in 2016 Yang et al. [25] illustrated the security and privacy consideration so OAuth 2.0 system about notification which immigrates the first 2 privacy issues. Similarly, the guideline of OIDC [4] requires the End-User consent for the release of the user's information. The guidelines of OIDC [4] and SAML [50] suggests that the IdP should provide the pairwise user identifier. However, the widely deployed SSO systems are all unable to prevent the IdP from tracing the user. To achieve the goal of protecting user from being tracked by IdP, in 2013 Mozilla proposed the Persona [6] based on the BrowserID protocol [13], which is now migrated to Firefox Accounts [51]. BrowserID enables the RP to identify the user through the login request signed by user's private key and the key is bound with user's email by IdP who need not know the RP's identity. In 2014 and 2015, Fett et al. [12], [13] performed the formal analysis on the BrowserID and finally found the flaw in it. In 2015 Fett et al. [7] proposed SPRESSO, the privacy-preserving SSO system, which enables the IdP to issue the identity proof for the encrypted RP identifier which does not expose RP's identity. However, no existing SSO systems protect user's login trace from both IdP tracking and RPs linking the user.

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. One of the earliest anonymous SSO system is proposed by Elmufti et al. [52] in 2008 for Global System for Mobile (GSM) communication. In 2013 Wang et al. [53] formalized the notion of anonymous single sign-on and proposed the anonymous based on group signatures. Moreover, in 2018 Lee et al. [54] proposed the anonymous SSO system based on Chebyshev chaotic-map-based assumptions and in 2018 Han et al. [55] proposed the system based on zero-knowledge proof. However, as the anonymous SSO system hides the user's identity from both IdP and RP, it is impossible for RP to provide personalize service to specific user.

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] "The top 500 sites on the web," https://www.alexa.com/topsites, Accessed July 30, 2019.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] Nat Sakimura, John Bradley, Mike Jones, Breno de Medeiros, and Chuck Mortimore, "Openid connect core 1.0 incorporating errata set 1," *The OpenID Foundation, specification*, 2014.

[10] 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.

[11] 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.

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

[13] Nat Sakimura, John Bradley, Mike Jones, Breno de Medeiros, and Chuck Mortimore, "Openid connect dynamic client registration 1.0 incorporating errata set 1," *The OpenID Foundation, specification*, 2014.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Xiaoyun Wang, Guangwu Xu, Mingqiang Wang, and Xianmeng Meng, *Mathematical foundations of public key cryptography*, CRC Press, 2015.

[22] Victor Shoup, "Searching for primitive roots in finite fields," *Mathematics of Computation*, vol. 58, no. 197, pp. 369–380, 1992.

[23] Wang Yuan, "On the least primitive root of a prime," in *Selected Papers Of Wang Yuan*, pp. 109–122. World Scientific, 2005.

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

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

[26] Elaine Barker, "Recommendation for key management part 1: General (revision 4)," *NIST special publication*, vol. 800, no. 57, pp. 1–160, 2016.

[27] "Mitreid connect /openid-connect-java-spring-server," https://github.com/mitreid-connect/OpenID-Connect-Java-Spring-Server, Accessed August 20, 2019.

[28] "Openid foundation," https://openid.net/certification/, Accessed August 20, 2019.

[29] "jsrsasign," https://kjur.github.io/jsrsasign/, Accessed August 20, 2019.

[30] Pascal Aubry, Vincent Mathieu, and Julien Marchal, "Esup-portail: open source single sign-on with cas (central authentication service)," *Proc. of EUNIS04–IT Innovation in a Changing World*, pp. 172–178, 2004.

[31] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller, "Kerberos: An authentication service for open network systems," in *Proceedings of the USENIX Winter Conference. Dallas, Texas, USA, January 1988*, 1988, pp. 191–202.

[32] 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.

[33] 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.

[34] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis, "O single sign-off, where art thou? an empirical analysis of single sign-on account hijacking and session management on the web," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018, pp. 1475–1492.

[35] Christian Mainka, Vladislav Mladenov, and Jörg Schwenk, "Do not trust me: Using malicious idps for analyzing and attacking single sign-on," in *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, 2016, pp. 321–336.

[36] Christian Mainka, Vladislav Mladenov, Jörg Schwenk, and Tobias Wich, "Sok: Single sign-on security - an evaluation of openid connect," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, 2017, pp. 251–266.

[37] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, Giancarlo Pellegrino, and Alessandro Sorniotti, "An authentication flaw in browser-based single sign-on protocols: Impact and remediations," *Computers & Security*, vol. 33, pp. 41–58, 2013.

[38] Yinzhi Cao, Yan Shoshitaishvili, Kevin Borgolte, Christopher Krügel, Giovanni Vigna, and Yan Chen, "Protecting web-based single sign-on protocols against relying party impersonation attacks through a dedicated bi-directional authenticated secure channel," in *Research in Attacks, Intrusions and Defenses - 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings*, 2014, pp. 276–298.

[39] Ronghai Yang, Wing Cheong Lau, Jiongyi Chen, and Kehuan Zhang, "Vetting single sign-on SDK implementations via symbolic reasoning," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018, pp. 1459–1474.

[40] Fadi Mohsen and Mohamed Shehab, "Hardening the oauth-webview implementations in android applications by re-factoring the chromium library," in *2nd IEEE International Conference on Collaboration and Internet Computing, CIC 2016, Pittsburgh, PA, USA, November 1-3, 2016*, 2016, pp. 196–205.

[41] 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.

[42] Ronghai Yang, Wing Cheong Lau, and Shangcheng Shi, "Breaking and fixing mobile app authentication with oauth2.0-based protocols," in *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, 2017, pp. 313–335.

[43] Shangcheng Shi, Xianbo Wang, and Wing Cheong Lau, "Mossot: An automated blackbox tester for single sign-on vulnerabilities in mobile applications," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, 2019, pp. 269–282.

[44] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, and M. Llanos Tobarra, "Formal analysis of SAML 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps," in *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering, FMSE 2008, Alexandria, VA, USA, October 27, 2008*, 2008, pp. 1–10.

[45] Quanqi Ye, Guangdong Bai, Kailong Wang, and Jin Song Dong, "Formal analysis of a single sign-on protocol implementation for android," in *20th International Conference on Engineering of Complex Computer Systems, ICECCS 2015, Gold Coast, Australia, December 9-12, 2015*, 2015, pp. 90–99.

[46] John Hughes, Scott Cantor, Jeff Hodges, Frederick Hirsch, Prateek Mishra, Rob Philpott, and Eve Maler, "Profiles for the oasis security assertion markup language (saml) v2. 0," *OASIS standard*, 2005, Accessed August 20, 2019.

[47] "About firefox accounts," https://mozilla.github.io/application-services/docs/accounts/welcome.html, Accessed August 20, 2019.

[48] 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.

[49] Kalid Elmufti, Dasun Weerasinghe, Muttukrishnan Rajarajan, and Veselin Rakocevic, "Anonymous authentication for mobile single sign-on to protect user privacy," *IJMC*, vol. 6, no. 6, pp. 760–769, 2008.

[50] Jingquan Wang, Guilin Wang, and Willy Susilo, "Anonymous single sign-on schemes transformed from group signatures," in *2013 5th International Conference on Intelligent Networking and Collaborative Systems, Xi'an city, Shaanxi province, China, September 9-11, 2013*, 2013, pp. 560–567.

[51] Tian-Fu Lee, "Provably secure anonymous single-sign-on authentication mechanisms using extended chebyshev chaotic maps for distributed computer networks," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1499–1505, 2018.

[52] 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.