UPRESSO: An Unlinkable Privacy-REspecting Single Sign-On System

***Abstract*—The Single Sign-On (SSO) service, provided by identity provider (IdP), is widely deployed and integrated to bring the convenience to both the relying party (RP) and the users. However, the privacy leakage is an obstacle to the users' adoption of SSO, as the curious IdP may track at which RPs users log in, while collusive RPs could link the user from a common or related identifer(s) issued by the IdP. Existing solutions preserve the user's privacy from either the curious IdP or the collusive RPs, but never from the both entities. In this paper, we provide an SSO system, named UPRESSO, to hide the user's accessed RPs from the curious IdP and prevent the identity linkage from the collusive RPs. In UPRESSO, IdP generates a different privacy-preserving ID (PID_U) for a user among RPs, and binds PID_U with a transformation of the RP identifier (PID_{RP}), without obtaining the real RP identifier. Each RP uses a trapdoor to derive the user's unique account from PID_U, while a user's accounts are different among the RPs. UPRESSO is compatible with OpenID Connect, a widely deployed and well analyzed SSO system; where dynamic registration is utilized to make PID_{RP} valid in the IdP. The analysis shows that user's privacy is preserved in UPRESSO without any degradation on the security of OpenID Connect. We have implemented a prototype of UPRESSO. The evaluation demonstrates that UPRESSO is efficient, and only needs 208 ms for a user to login at an RP in our environment.**

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

I. Introduction

Single sign-on (SSO) systems, such as OAuth [1], OpenID Connect [2] and SAML [3], have been widely adopted nowadays as a convenient web authentication mechanism. SSO delegates user authentication from websites, so-called relying parties (RPs), to a third party, so-called identity providers (IdPs), so that users can access different services at cooperating sites via a single authentication attempt. Using SSO, a user no longer needs to maintain multiple credentials for different RPs, instead, she maintains only the credential for the IdP, who in turn will generate corresponding *identity proofs* for those RPs. Moreover, SSO shifts the burden of user authentication from RPs to IdPs and reduces security risks and costs at RPs. As a result, SSO has been widely integrated with modern web systems. We analyze the Alexa top-100 websites [4] and find 80% of them support SSO service, and the analysis in [5] identifies SSO support on 6.30% of the Alexa top 1 million websites. 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.

SSO systems need to provide secure authentication [6], that is to ensure an honest user logs in to an honest RP under the correct identity (i.e., account). To achieve this, the identity proof should be valid only for the RP that the user requests to log in to (i.e., binding), and never leaked to other entities except this RP and the user (i.e., confidentiality); while this RP should never accept any information from the corrupted identity proof (i.e., integrity). However, various attacks are found to exploit the vulnerabilities in the SSO systems to break at least one of these three principles [7]–[14], and the adversary could impersonate the victim user at an RP or log in the browser of an honest user under an adversarys identity (i.e., identity injection). For example, Friendcaster was found to blindly accept any received identity proof [7], [15] (i.e., not checking binding), then a malicious RP could obtain the identity proof from a user who attempts to log in to this malicious RP, and use it to log in to Friendcaster as the victim user [14]; some RPs of Google ID SSO were found to accept the user's attributes unprotected in the identity proof (i.e., out scope of integrity protection), and therefore a malicious user could add incorrect attributes (e.g., the email address) in identity proof and then act as any user at the RP [9].

The wide adoption of SSO also raises new privacy concerns regarding online user tracking and profiling [16], [17]. Privacy leakage exists in all current SSO protocols and implementations. We adopt the SSO authentication session in the OpenID Connect (OIDC) as an example, to figure out the leakage. As shown in Fig. 1, on receiving a login request from a user (Step 1), the RP uses the RP's identifier to construct an authentication request and redirects it to IdP (Step 2); then, the IdP completes the user's authentication in Step 3, generates an identify proof for the user and binds it with the RP in Step 4, and redirects the identity proof to the RP confidentially in Step 5; finally, the RP verifies the binding and integrity of identity proof and sends the result to the user. From the authentication session, we find collusive RPs and curious IdP could break the user's privacy as follows.

- *RP-based identity linkage*. If the common (or derivable from others) identifiers are used in the identity proofs for a same user across different RPs, which is the case even in several widely deployed SSO systems [6], [18], collusive RPs could not only track her online traces but also correlate her attributes across the sites [16].
- *IdP-based access tracing*. IdP obtains the user's unique identifer in Step 3 and the identifers of the visited RPs in Step 2. Therefore, a curious IdP could easily discover all RPs accessed by a user and reconstruct her access traces.

Meanwhile, large IdPs, especially social IdPs like Google and Facebook, are known to be interested in collecting user-

TABLE I: Three functions in privacy-preserving SSO.

Solutions	$\mathcal{F}_{ID_U \mapsto PID_U}$	$\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$	$\mathcal{F}_{PID_U \mapsto Account}$
PPID	$Map[ID_U, ID_{RP}]$ (✓)	ID_{RP} (×)	PID_U (✓)
SPRESSO	ID_U (×)	$Enc(ID_{RP} \| \| nonce)$ (✓)	ID_U (×)
BrowserID	ID_U (×)	$\perp$ (✓)	ID_U (×)
UPRESSO	${PID_{RP}}^{ID_U}$ (✓)	${ID_{RP}}^{N_U * N_{RP}}$ (✓)	${PID_U}^t$ (✓)

s' online behavioral information for various purposes (e.g., Screenwise Meter [19], Onavo [20]). By simply serving the IdP role, these companies can easily collect a large amount of continuous data to reconstruct users' online traces. Moreover, many service providers are also hosting a variety of web services, which makes them easy to link the same user's multiple logins in each RP as the user's unique identifier is contained in the identity proof. Through internal integration, they could obtain rich information from SSO data to profile their clients.

While the privacy problems in SSO have been widely recognized [16], [17], only a few solutions have been proposed to protect user privacy [6], [21]. Among them, Pairwise Pseudonymous Identifier (PPID) [2], [22] is a most straightforward and commonly accepted solution to defend against RP-based identity linkage, which requires the IdP to create different identifiers for the user when she logs in to different RPs. In this way, even multiple malicious RPs collude with each other across the system, they cannot link the pairwise pseudonymous identifiers of the user and track which RPs she has visited. As a recommended practice by NIST [17], PPID has been specified in many widely adopted SSO standards including OIDC [2] and SAML [22]. However, PPID-based approaches cannot prevent the IdP-based access tracing, as the IdP still knows which RP the user visits.

To the best of our knowledge, there are only two schemes (i.e., BrowserID [18] and SPRESSO [6]) being proposed so far to prevent IdP-based access tracing. In BrowserID (and its prototype system known as Mozilla Persona [21] and Firefox Accounts [23]), IdP generates a user certificate to bind the user's unique identifier (i.e., email address) with a public key; while the user will use the corresponding private key to bind the identity proof (including the user certificate) with an RP, and send it to the correct RP confidentially. In SPRESSO, the RP chooses a third-party entity (named forwarder) as the proxy to receive the identity proof, and generates a pseudonymous identifier for itself on each login; IdP generates the identity proof for the user, binds it with the RP's pseudonymous identifier and sends the encrypted proof to the forwarder; while, the forwarder transmits the identity proof to the correct RP who performs the decryption and obtains the plain-text identity proof. In these two schemes, without the identifiers of the visiting RPs, the IdP needs to include the user's unique identifier (e.g., email address) in the identity proof, which is necessary for each RP to obtain a common account during the user's multiple logins. Then, the collusive RPs could perform RP-based identity linkage with the user's unique identifier.

As described above, none of existing SSO systems could prevent both the RP-based identity linkage and IdP-based access tracing. Here, we analyze these two privacy leakage problems formally. Each user has one identifier at the IdP and each RP respectively, which is denoted as ID_U at the IdP and $Account$ at the RP. Each RP has one global unique identifier (denoted as ID_{RP}), and a privacy-preserving identifier PID_{RP} (may be null) at IdP for each login. PID_{RP} is generated by the RP or the user, with the function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$. And, IdP generates a privacy-preserving user identifier (PID_U) with ID_U and PID_{RP}, based on the function $\mathcal{F}_{ID_U \mapsto PID_U}$. While, RP calculates $Account$ with the function $\mathcal{F}_{PID_U \mapsto Account}$. The three functions have to satisfy:

- To prevent RP-based identity linkage, $\mathcal{F}_{ID_U \mapsto PID_U}$ needs to ensure that PID_U are unlinkable among various RPs, while $\mathcal{F}_{PID_U \mapsto Account}$ needs to ensure that $Account$ are unlinkable among different RPs.
- To prevent IdP-based access tracing, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ needs to ensure that PID_{RP} are unlinkable for multiple logins at an RP.
- To ensure each RP obtains the unchanged $Account$ among the user's multiple logins, all the three functions need to be designed corporately, and therefore the output of $\mathcal{F}_{PID_U \mapsto Account}$ will be unchanged for a user's multiple logins at an RP.

Existing privacy-preserving schemes adopt either the unsatisfying $\mathcal{F}_{ID_U \mapsto PID_U}$ or $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, which will simplify the construction of $\mathcal{F}_{PID_U \mapsto Account}$, but fail to provide the complete user privacy. For example, in PPID [2], [22], $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ outputs ID_{RP} directly and $\mathcal{F}_{PID_U \mapsto Account}$ outputs PID_U as $Accout$, then IdP knows which RP the user visits; in BrowserID [18] and SPRESSO [6], $\mathcal{F}_{ID_U \mapsto PID_U}$ outputs ID_U directly and $\mathcal{F}_{PID_U \mapsto Account}$ uses ID_U as $Accout$ directly, then the collusive RPs could perform RP-based identity linkage.

In this paper, we propose UPRESSO, an Unlinkable Privacy-REspecting Single Sign-On system, which provides **all** the **satisfying** $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and $\mathcal{F}_{PID_U \mapsto Account}$ based on the discrete logarithm problem, and prevents both the RP-based identity linkage and IdP-based access tracing. In UPRESSO, for each login, a one-way trapdoor function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ is invoked with a randomly chosen trapdoor (t) to generate a random and unique PID_{RP} which is anonymously registered it at the IdP by the user; then IdP invokes a one-way function $\mathcal{F}_{ID_U \mapsto PID_U}$ to generate PID_U with PID_{RP} and ID_U, and issues the identity proof; finally, RP checks the binding and integrity of the identity proof, and invokes $\mathcal{F}_{PID_U \mapsto Account}$ with PID_U, ID_{RP}, PID_{RP} and the trapdoor t to obtain the unchanged $Accout$ for the user at this RP. Therefore, based on the discrete logarithm problem, UPRESSO ensures: (1) when a user logs in to an RP, the RP can derive an unchanged $Accout$ from different PID_U, but cannot derive ID_U; (2) when a user logs in to different RPs, various PID_Us and $Account$s are generated and collusive RPs cannot link the user's multiple logins; and (3) when an RP is visited during multiple logins, random PID_{RP}s are generated and the curious IdP cannot

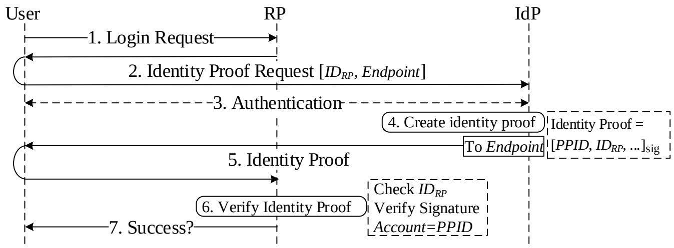

Fig. 1: The implicit protocol flow of OIDC.

infer ID_{RP} nor link these logins.

We have implemented a prototype of UPRESSO based on an open-source implementation of OIDC. UPRESSO inherits the security properties from OIDC by achieving the binding, integrity and confidentiality of identity proof, and only requires small modifications to add the three functions $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and $\mathcal{F}_{PID_U \mapsto Account}$ for privacy protection. Therefore, unlike BrowserID and SPRESSO which are the non-trivial re-designs of the existing SSO systems, UPRESSO is compatible with existing SSO systems, and doesn't require a completely new and comprehensive formal security analysis.

The main contributions of UPRESSO are as follows:

- We systematically analyze the privacy issues in SSO systems and propose a comprehensive protection solution to hide users' traces from both curious IdPs and collusive RPs, for the first time. We also provide a systematic analysis to show that UPRESSO achieves the same security level as existing SSO systems.
- We develop a prototype of UPRESSO that is compatible with OIDC and demonstrate its effectiveness and efficiency with experiment evaluations.

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 UPRESSO. 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 and Preliminary

UPRESSO is compatible with OIDC, and achieves the privacy protection based on the discrete logarithm problem. Here, we provide a brief introduction on OIDC and the discrete logarithm problem.

A. OpenID Connect

OIDC [2] is an extension of OAuth 2.0 to support user authentication, and becomes one of the most prominent SSO authentication protocols. Same as other SSO protocols [22], OIDC involves three entities, i.e., *users*, *identity provider (IdP)*, and *relying parties (RPs)*. Both users and RPs have to register at the IdP, the users register at the IdP to create credentials and identifiers (e.g. ID_U), while each RP registers at the IdP with its endpoint information to create its unique identifier (e.g., ID_{RP}) and the corresponding credential. IdP is assumed to securely maintain the attributes of users and RPs. Then, in the SSO authentication sessions, each user is responsible to start a login request at an RP, redirect the messages between RP and IdP, and check the scope of user's attributes provided to the RP; IdP authenticates the user, sets the $PPID$ for the user ID_U at the RP ID_{RP}, constructs the identity proof with $PPID$, ID_{RP} and the user's attributes consented by the user, and finally transmits the identity proof to the RP's registered endpoint (e.g., URL); each RP constructs an identity proof request with its identifier and the requested scope of user's attributes, sends an identity proof request to the IdP through the user, and parses the received identity proof to authenticate and authorize the user. Usually, the redirection and checking at the user are handled by a user-controlled software, called *user agent* (e.g., browser).

Implicit flow of user login. OIDC supports three processes for the SSO authentication session, known as *implicit flow*, *authorization code flow* and *hybrid flow* (i.e., a mix-up of the previous two). In the implicit flow of OIDC, a token, known as *id token*, is introduced as the identity proof, which contains user identifier (i.e., $PPID$), RP identifier (i.e., ID_{RP}), the issuer (i.e., IdP), issuing time, the validity period, and other requested attributes. The IdP signs the id token using its private key to ensure integrity, and sends it through the user to RP. In the authorization code flow, IdP binds an authorization code with the RP, and redirects this code to the RP; then RP establishes an HTTPS connection with IdP to ensure the integrity and confidentiality of the identity proof, and uses the authorization code with the RP's credential to obtain PPID and the user's other attributes.

UPRESSO is compatible to all the three flows. For brevity, we will present the application of UPRESSO in the implicit flow in details, and provide the integration with the authorization code flow briefly. Here, we first introduce the original processes in the implicit flow of OIDC.

As shown in Figure 1, the implicit flow of OIDC consists of 7 steps: when a user attempts to log in to an RP (Step 1), the RP constructs a request for identity proof, which is redirected by the user to the corresponding IdP (Step 2). The request contains ID_{RP}, RP's endpoint and a set of requested user attributes. If the user has not been authenticated yet, the IdP performs an authentication process (Step 3). If the RP's endpoint in the request matches the one registered at the IdP, it generates an identity proof (Step 4) and sends it back to the RP (Step 5). Otherwise, IdP generates a warning to notify the user about potential identity proof leakage. The RP verifies the id token (Step 6), extracts user identifier from the id token and returns the authentication result to the user (Step 7).

RP dynamic registration. OIDC provides a dynamic registration mechanism [24] for the RP to update its ID_{RP} dynamically. When an RP first registers at the IdP, it obtains a registration token, with which the RP can invoke the dynamic registration process to update its information (e.g., the endpoint). After each successful dynamic registration, the RP obtains a new unique ID_{RP} from the IdP. In UPRESSO, we slightly modify dynamic registration to register PID_{RP} at IdP.

B. Discrete Logarithm Problem

Discrete logarithm problem is adopted in UPRESSO for the construction of $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and $\mathcal{F}_{ID_U \mapsto PID_U}$, which generate privacy-preserving user identifier (e.g. PID_U) and RP identifier (e.g. PID_{RP}) respectively. Here, we provide a brief description of the discrete logarithm problem.

For $GF(p)$ where p is a large prime, a number g is called a generator of order q, if it can be used to construct a cyclic group of q elements by calculating $y = g^x \ mod \ p$. And, x is called the discrete logarithm of y modulo p. Given a large prime p, a generator g and a number y, it is computationally infeasible to derive the discrete logarithm (here x) of y (detailed in [25]), which is called discrete logarithm problem. The hardness of solving discrete logarithm has been used to construct several security primitives, including Diffie-Hellman key exchange and Digital Signature Algorithm (DSA).

III. The Privacy Dilemma in Single Sign-On

In this section, we describe the challenges for developing privacy-preserving SSO systems and provide an overview of the solutions proposed in UPRESSO.

A. Basic Security Requirements of SSO

Both the designs of SSO protocols and the implementations of SSO systems are challenging [6], and various vulnerabilities have been found in existing systems [9]–[14], [26]–[32]. With these vulnerabilities, the adversaries break the security of SSO systems and achieve the following two goals [6]:

- **Impersonation**: Adversary logs in to an honest RP as the victim user.
- **Identity injection**: A victim user logs in to an honest RP under the adversaries' identity.

We summarize basic requirements of SSO systems based on existing theoretical analysis [8], [33], [34] and practical attacks [9]–[14], [26]–[32] on SSO systems. These basic requirements focus on functions and security of SSO systems, and are as follows:

User identification. When a user logs in to a same RP multiple times, the RP should be able to associate these logins to provide a continuous and personalized service to that user.

Receiver designation. The receiver designation requires that the identity proof should be sent only to the RP (and user) that the user visits, and be bound to this RP so that it will accepted only by this RP.

Integrity and confidentiality. Only the IdP is able to generate a valid identity proof, no other entity should be able to modify or forge it [11] without being found. And, the confidentiality of the identity proof is ensured during the transmission among the IdP, user and the designated RP. While, the correct RP should only accept the valid identity proof.

These basic requirements are the minimum properties that an SSO system has to provide. User identification is necessary for all services except the anonymous systems which will be discussed in Section IX, therefore RPs should be able to identify the user with the help from IdP. While either the

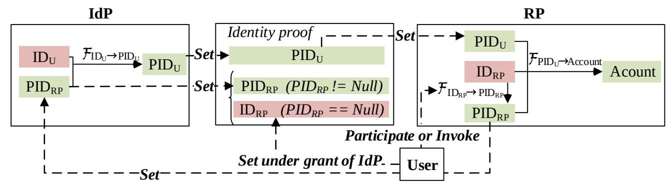

Fig. 2: ID transformations in a privacy-preserving SSO system.

receiver designation, integrity or confidentiality not satisfied, the impersonation and identity injection will exist in the SSO systems. For example, without the receiver designation, the adversaries cloud make the victim RP incorrectly accept the identity proof intended for other RP, which allows the adversary to perform impersonation directly or inject the identity proof in the web session between the victim user and honest RP with other web attacks (e.g., CSRF); and without integrity, the impersonation and identity injection attacks could be constructed easily as an adversary could directly modify user's identifier in the identity proof; while without confidentiality, the adversaries cloud use the leaked identity proof to impersonate the victim user at an honest RP [7], [8], [11].

B. The Privacy Dilemma and Existing Attempts

In addition to the basic requirements, a privacy-preserving SSO system should prevent the IdP-based access tracing and RP-based identity linkage. In details, the privacy-preserving SSO system should prevent the curious IdP from obtaining any information that could identify the user's accessed RP (for example, through RP's identifier and URL) or associate the logins at an RP, and prevent the collusive RPs from associating a user's logins at different RPs.

The privacy-preserving requirements need to be integrated into the basic requirements, to protect the user privacy under the prerequisite that the correct functions and security of SSO systems are ensured. The privacy-preserving user identification requires IdP to provide a user's identifier (PID_U) to help the RP in identifying the user locally, under the prerequisites that the curious IdP can never identify the visiting RP or classify logins based on RP and the collusive RPs cannot find the correlation between PID_Us for a user. The privacy-preserving receiver designation requires IdP (or the user) to bind an identity proof with an RP and send it only to this RP, while IdP can never identify the RP visited by the user or classify logins based on RP.

The above analysis demonstrates that the identifers of the user and RP need to be carefully processed in SSO systems. Here, we systematically analyze the forms of the user and RP identifiers, as required by privacy protection. As in Figure 2, IdP, who knows the user's globally unique identifer (ID_U), should only obtain a privacy-preserving identifer for an RP (PID_{RP}); each RP, who knows its globally unique identifer (ID_{RP}), should only receive a privacy-preserving identifer for a user (PID_U) from the identity proof. The PID_{RP} may be null, it means IdP has no information about RP, for example in BrowserID [18]. The PID_U can never be null, as RP has to derive the $Accout$ from it, which is the basic function of SSO systems. Here the privacy-preserving identifers in multiple sessions will never leak the globally unique identifer nor link the sessions from an entity (the RP and user). That is, PID_{RP}s in multiple sessions for a same RP should be independent from the view of the IdP; and for the collusive RPs, their obtained PID_Us for a same user should be independent.

Then, we analyze the generation and use of PID_U and PID_{RP}, considering the basic requirements of SSO system.

- IdP is the only trusted entity to control the generation of PID_U, as the user is not trusted by the RP and a malicious user may provide incorrect PID_U. Here, the "control" means, IdP may generate PID_U alone or with the corporation from the user, but it's the IdP who finally determines the value of PID_U. For clarity, we assume IdP generates PID_U by invoking $\mathcal{F}_{ID_U \mapsto PID_U}$ with ID_U and PID_{RP}, without loss of generality.
- The generation of PID_U and PID_{RP} must ensure the **user identification**, that is, the RP could derive a same $Account$ with the PID_U and PID_{RP} from different logins. We assume RP calculates $Account$ by invoking $\mathcal{F}_{PID_U \mapsto Account}$ with PID_U, ID_{RP} and PID_{RP}.
- Each PID_{RP} must be globally unique, i.e., only assigned to one RP, for achieving the **receiver designation**. The user and RP may generate the PID_{RP} separately or cooperatively, through the function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$. However, both the user and RP must check the uniqueness of PID_{RP} before accept and use it. If either the user or the RP doesn't perform the check, the adversary could make it accept a PID_{RP} same as an RP and then misuse the identity proof.
- The **receiver designation** further requires that PID_U is bound with either a non-null PID_{RP} or ID_{RP} in identity proof. When PID_{RP} is non-null, IdP builds the identity proof separately and the **integrity** is also ensured. When PID_{RP} is null, only the user cloud bind PID_U with an RP identifier (i.e., ID_{RP}) which is unique and checkable to the RP. In this case, the user who performs the binding, must have a publicly verifiable grant from the IdP, as required by **integrity**. The binding and integrity cloud be achieved by existing public key infrastructure.
- The **receiver designation** also requires the identity proof will only be sent to the correct RP. As IdP doesn't know ID_{RP}, the user or a third party trusted by the correct RP will ensure this.

Then, the dilemma in the design of a secure and privacy-preserving SSO system, could be transformed into finding three privacy-preserving ID transformation functions $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, $\mathcal{F}_{ID_U \mapsto PID_U}$, and $\mathcal{F}_{PID_U \mapsto Account}$, which satisfying:

- For an RP, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ generates PID_{RP}s in multiple logins, and these PID_{RP}s are independent to the IdP.
- For a user, $\mathcal{F}_{ID_U \mapsto PID_U}$ generates PID_Us in multiple logins at different RPs, and these PID_Us are independent to these RPs. $\mathcal{F}_{PID_U \mapsto Account}$ outputs $Account$s for a user at different RPs, and these $Account$ are independent to these RPs.
- For a user and an RP, $\mathcal{F}_{PID_U \mapsto Account}$ generates a unchanged $Account$ in multiple logins.

Various solutions [2], [6], [18], [22], are proposed, attempting to construct a secure and privacy-preserving SSO system. However, these scheme provide at most two satisfying functions, and therefore fail to prevent either the IdP-based access tracing or RP-based identity linkage.

- The traditional SSO systems provide no satisfying functions, and therefore fail to protect the user's privacy.
- SAML [22] and OIDC [2] provide only the satisfying $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$. The IdP obtains the ID_{RP} for an RP, and generates the unchanged PID_U for the same couple $< ID_U, ID_{RP} >$, while the PID_U are independent for different ID_{RP}s.
- BrowserID [18] and SPRESSO [6] provide only the satisfying $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$. In BrowserID, IdP obtains a null PID_{RP} and provides ID_U to the RP, therefore each RP obtains the unchanged $Accout$. In SPRESSO, each RP generates PID_{RP} by encrypting ID_{RP} padding by a random nonce, and IdP provides the unchanged ID_U for a user's multiple logins no matter which RP the user is visiting.

Obliviously, existing attempts fail to provide the complete privacy. The essential reason is that these schemes provide an unchanged value (e.g., PID_U in SAML [22] and OIDC [2], or ID_U in BrowserID [18] and SPRESSO [6]) for the user's multiple logins at an RP. To provide unchanged PID_U, IdP has to know ID_{RP} and then will be able to identity or link the logins at an RP. Providing ID_U to the RP, makes the collusive RPs easily link the user's logins at different RPs.

C. The Principles of UPRESSO

In this work, we present UPRESSO, a secure and privacy-preserving SSO system, which prevents the IdP-based access tracing and RP-based identity linkage under the prerequisites that the basic requirements of SSO systems are satisfied. UPRESSO adopts the public key infrastructure to ensure the integrity of the identity proof and TLS for its confidentiality, which is the same as other SSO systems. Therefore, we focus on how to provide the privacy-preserving user identification and receiver designation as follows:

Trapdoor user identification. UPRESSO breaks the implicit assumption in previous SSO systems, that an RP should obtain an unchanged value to identify a user in the multiple logins. A trapdoor identification is introduced, which allows the RP to derive the unchanged $Accout$ from different PID_Us. The trapdoor identification requires a cooperation of the three functions $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$. $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ is invoked with a trapdoor to generate $PID_{RP}s$ which are independent to IdP (preventing

IdP-based access tracing), and RP uses $\mathcal{F}_{PID_U \mapsto Account}$ with this trapdoor to derive the unchanged $Accout$ from PID_Us that are generated by IdP with $\mathcal{F}_{ID_U \mapsto PID_U}$.

Transformed receiver designation. UPRESSO splits the receiver designation into two steps: IdP designates the identity proof to a transformed RP identifer (i.e., PID_{RP}), while the user and RP cooperatively designate a fresh and unique PID_{RP} only to one ID_{RP}. Then, each RP only needs to check the designation based on PID_{RP}.

- In the first step, the IdP generates PID_U for PID_{RP} and achieves full privacy-preserving binding (i.e., PID_U with PID_{RP}). UPRESSO introduces an efficient one-way (trapdoor) function $\mathcal{F}_{ID_U \mapsto PID_U}$. It allows IdP to compute PID_U easily, avoiding the generation of PID_U to be the bottleneck at a high-throughput IdP; and also prevents the RP from finding any information about ID_U, which is required by preventing RP-based identity linkage.
- In the second step, the user and RP cooperatively generate a fresh PID_{RP} based on $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and check the uniqueness of the PID_{RP}, therefore a fresh and unique PID_{RP} is only mapped to one RP, when at least a correct user or correct RP exists. Moreover, the user needs to extract the correct endpoint of ID_{RP}, to ensure that the identity proof is sent to the only correct RP.

To meet the above two principles, we need to construct three satisfying functions $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$, design the protocols between the user, RP and IdP to avoid the privacy leakage during message transmission, and implement the processing at the user as required by the transformed receiver designation.

IV. Threat Model and Assumption

To be compatible the traditional SSO systems (e.g., SAML, OIDC), UPRESSO doesn't introduce any other entity, but only modifies the processes at existing entities, i.e, one IdP, multiple RPs and users, to provide the secure and privacy-preserving SSO service. Here, we introduce the threat model and assumptions in UPRESSO.

A. Threat Model

In UPRESSO, the IdP is assumed to be semi-honest, while the users and RPs could be controlled by the adversary and be malicious. The malicious users and RPs could behave arbitrarily and collude with each other for breaking the security and privacy of correct users. While, the IdP will follow the protocol correctly, and is only curious about the user's privacy. The details are as follows.

Semi-honest IdP. We assume the IdP is well-protected and will never leak any sensitive information. For example, the private key for generating the identity proof and RP certificate (used in Section V-B) will never be leaked, therefore the adversary fails to impersonate as the IdP to forge a valid identity proof or RP certificate. The honest IdP processes the requests of RP registration and identity proof correctly, and never colludes with others (e.g., malicious RPs and users). For example, IdP ensures the uniqueness of ID_{RP} and PID_{RP}, and generates the correct RP certificate, PID_U and identity proof. However, the curious IdP may attempt to break the user's privacy without violating the protocol. For example, the curious IdP may store and analyze the received messages, and perform the timing attacks, attempting to achieve the IdP-based linkage.

Malicious users. The adversary could control a set of users, for example through stealing the users' credentials [35], [36] or registering at the IdP and RPs directly. These malicious users aim to break the security of the SSO system. That is, they attempt to impersonate an uncontrolled user at the victim RP, and make a victim user log in at the correct RP under a controlled identity. To achieve this, they could behave arbitrarily [9], [13]. For example, the malicious users may forge the identity proof, modify the forwarding messages (requests of identity proof, identity proof, RP registration request and result, and etc.), and provide incorrect values for negotiating PID_{RP} (detailed in Section V-B).

Malicious RPs. The adversary could control a set of RPs, by registering an RP at the IdP or exploiting various vulneraries to attack RPs. These malicious RPs aim to break the security and privacy of the correct users, and could behave arbitrarily. For example, to break the security, the malicious RPs need to obtain an identity proof valid for other RP, and attempt to achieve this by behaving as follows: impersonating other RP at the user by providing the incorrect RP certificate, using incorrect values during the negotiation of PID_{RP} to make the generated PID_{RP} be same as the one for other RP, or constructing an incorrect request to trigger the IdP issuing an identity proof binding with other RP. Moreover, the malicious RPs may attempt to perform the RP-based identity linkage and break the user's privacy. To achieve this, the RPs could behave arbitrarily and collude with each other. For example, the RPs may attempt to derive the ID_U from PID_U by providing incorrect values to the IdP, and the colluded RPs may attempt to link the user's multiple logins, by providing correlated values (e.g., PID_{RP}) to the IdP.

Collusive users and RPs. The malicious users and RPs may collude and behave arbitrarily, attempting to break the security of UPRESSO. For example, the adversary may first act as a malicious RP, and make an incorrect identity proof generated for the visiting user, then act a malicious user, and use this identity proof to impersonate this victim user at another RP. The adversary could also first act as a user to login a correct RP and obtain an identity proof, then act a malicious RP to perform the identity injection attack, by injecting this identity proof to the session between the victim user and the correct RP with other web attacks (e.g., CSRF).

B. Assumption

In UPRESSO, we assume that the user agent deployed at the honest user is correctly implemented, and will transmit the messages to the correct destination. The TLS is also 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 a secure random number generator is adopted in UPRESSO to provide the unpredictable random numbers; and the adopted cryptographic algorithms, including the RSA and SHA-256, are secure and implemented correctly. Therefore, no one without private key can forge the signature, and the adversary fails to infer the private key during the computation. Moreover, we also assume the security of the discrete logarithm problem is ensured.

The collusive RPs may attempt to link a user based on the identifying attributes, such as the telephone number and credit number. Here, we assume that the users refuse to provide these attributes to the RPs, and the correct RPs never collect these attributes as required by privacy laws (e.g., GDPR). Moreover, the global network traffic analysis may be adopted to correlate the user's logins at different RPs. However, UPRESSO may integrate existing defenses to prevent this attack.

V. Design of UPRESSO

In this section, we provide designs of UPRESSO, a secure and privacy-preserving SSO system. First, we present the functions of privacy-preserving ID transformation which achieve the trapdoor user identification and transformed receiver designation. Then, we provide an overview of UPRESSO and describe the detailed protocol for providing the SSO service. Finally, we discuss the compatibility of UPRESSO with OIDC.

A. Functions of privacy-preserving ID transformation

The three functions $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$ are essential for the trapdoor user identification and transformed receiver designation. In UPRESSO, these functions are constructed based on discrete logarithm cryptography with the public parameters p, q, g and L, where p is a large prime defines the finite field $GF(p)$, L is the length of q, q ($2^{L-1} < q < 2^L$) is a prime divisor of $(p-1)$, and g is a generator of order q.

In UPRESSO, IdP assigns a unique random number as ID_U ($0 < ID_U < q$) at the user's registration, and a unique ID_{RP} at the RPs initial registration. The ID_{RP} is generated using Equation 1, where r is a random number ($1 < r < q$).

$$ID_{RP} = g^r mod\ p \tag{1}$$

For each login, the RP chooses a random number N_{RP} ($1 < N_{RP} < q$), the user chooses a random number N_U ($1 < N_U < q$). Then, the RP and user cooperatively generate PID_{RP} using the function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ as Equation 2. The function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ satisfies the requirements described in Section III-B. That is, the function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ is invoked to generate PID_{RP} for each login, while IdP fails to derive ID_{RP} from PID_{RP} and cannot find the relation among PID_{RP}s for a same RP, which is ensured by the discrete logarithm cryptography. Moreover, N_U and N_{RP} serves as the nonce which ensures that the PID_{RP} (also identity proof) is exactly constructed for this login, and the cooperation between the user and RP prevents the malicious user and RP from controlling the PID_{RP}. For example, the malicious user fails to make a correct RP accept a PID_{RP} used in another login, while the collusive RPs fail to use a same or correlated PID_{RP}s for different logins.

$$\mathcal{F}_{ID_{RP} \mapsto PID_{RP}} : {ID_{RP}}^{N_U * N_{RP}} mod\ p \tag{2}$$

For the user ID_U to login at an RP with a privacy-preserving identifer PID_{RP}, IdP calculates the user's privacy-preserving identifer PID_U using the function $\mathcal{F}_{ID_U \mapsto PID_U}$ as Equation 3. The function $\mathcal{F}_{ID_U \mapsto PID_U}$ satisfies the requirements described in Section III-B. Combining Equation 1, 2 and 3, we get that PID_U equals to $g^{r*N_U*N_{RP}*ID_U}\ mod\ p$. The discrete logarithm cryptography ensures that the RPs fail to derive ID_U from PID_U, nor link a user's PID_Us at different RPs who can never know r and ID_U.

$$\mathcal{F}_{ID_U \mapsto PID_U} : {PID_{RP}}^{ID_U}\ mod\ p \tag{3}$$

Finally, the RP derives $Account$ for the user with the function $\mathcal{F}_{PID_U \mapsto Account}$ as Equation 4. Here, the value $(N_U * N_{RP})^{-1} mod\ q$ is the trapdoor t. As q is a prime number, $1 < N_U < q$ and $1 < N_{RP} < q$, therefore q is coprime to $N_U * N_{RP}$, and the t that satisfies $t * (N_U * N_{RP}) = 1\ mod\ q$ always exists. The function $\mathcal{F}_{PID_U \mapsto Account}$ satisfies the requirements described in Section III-B. As shown in Equation 5, for a user's multiple logins at an RP, $\mathcal{F}_{PID_U \mapsto Account}$ outputs an unchanged $Acount$ which equals to ${ID_{RP}}^{ID_U} mod\ p$. Same as the analysis of PID_U, the collusive RPs fail to derive ID_U from $Account$ nor link a user's $Account$s due to the different and unknown rs.

$$\mathcal{F}_{PID_U \mapsto Account} : {PID_U}^{(n_U * n_{RP})^{-1} mod\ q} mod\ p \tag{4}$$

The **trapdoor user identification** is supported with these three functions. For a user's multiple logins, each PR obtains the different PID_Us and the corresponding ts , then derives the unchanged $Account$ as shown in Equation 5. The function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ prevents the curious IdP from linking the PID_{RP}s of different logins at an RP, and therefore avoids the IdP-based access tracing. The functions $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$ prevents the collusive RPs from linking a user's PID_Us and $Account$s at different RPs, and therefore avoids the RP-based identity linkage.

$$\begin{aligned} Account &= {PID_U}^t mod\ p \\ &= ({PID_{RP}}^{ID_U})^{(N_U * N_{RP})^{-1} mod\ q} mod\ p \\ &= {ID_{RP}}^{ID_U * N_U * N_{RP} * t\ mod\ q} = {ID_{RP}}^{ID_U} mod\ p \end{aligned} \tag{5}$$

The **transformed receiver designation** is also supported with the efficient functions $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and $\mathcal{F}_{ID_U \mapsto PID_U}$, together with a user-centric verification. The $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ ensures that the user and RP cooperatively generate a fresh PID_{RP} for a user's login, while $\mathcal{F}_{ID_U \mapsto PID_U}$ ensures that the IdP generates the exact PID_U for the ID_U who logins at PID_{RP}. The IdP will bind PID_U with PID_{RP} in the

TABLE II: The notations used in UPRESSO.

Notation	Definition	Attribute
p	A large prime.	Long-term
q	A large prime.	Long-term
L	Length of q.	Long-term
g	A generator of order q.	Long-term
ID_U	User's unique identifier.	Long-term
PID_U	User's privacy-preserving identifier.	One-time
$Account$	User's identifier at an RP.	Long-term
r	Secret value for $ID_{RP} = g^r \ mod \ p$.	Long-term
ID_{RP}	RP's original identifier.	Long-term
PID_{RP}	RP's privacy-preserving identifier.	One-time
N_U	User-generated random nonce for PID_{RP}.	One-time
N_{RP}	RP-generated random nonce for PID_{RP}.	One-time
Y_{RP}	Public value for n_{RP}, $(ID_{RP})^{N_{RP}} \ mod \ p$.	One-time
t	A trapdoor, $t = (N_U * N_{RP})^{-1} mod \ q$.	One-time
$Cert_{RP}$	An RP certificate.	Long-term
SK, PK	The private/public key of IdP.	Long-term

identity proof, which designates this identity proof to PID_{RP}. In the user-centric verification, both the user and RP checks the uniqueness of PID_{RP}, while the user further checks that PID_{RP} is exactly generated for the RP ID_{RP}, and then sends the identity proof only to this RP. Therefore, the PID_{RP} is designated to ID_{RP}. Finally, the transformed receiver designation is provided through the two-step designations.

B. UPRESSO Overview

UPRESSO contains four sub-protocols, i.e., system initialization, RP initial registration, user registration and SSO login. The system initialization is invoked by the IdP to initialize the SSO system and only needs to be invoked once for each SSO system. The RP initial registration is invoked by each RP to obtain the necessary parameters (a unique identifier ID_{RP} and an RP certificate $Cert_{RP}$) from the IdP and only needs to be invoked once for each RP. The user registration is only invoked once by each user to create a unique user identifier ID_U and the corresponding credential, where ID_U is generated by the IdP and only provided to the corresponding user. While, the SSO login is invoked once a user wants to log in an RP, and therefore will be invoked frequently. The process for user registration is the same as the one in the typical SSO systems, therefore, we focus on the processes in system initialization, RP initial registration and SSO login. For clarity, we list the used notations in Table II.

System initialization. The IdP chooses L, generates a large prime p, a prime q of L bits, and a generator g of order q as the parameters for the discrete logarithm cryptography [37], and generates one asymmetric key pair (SK denotes the private key and PK is the public key) for the generation of the identity proof and $Cert_{RP}$. The IdP keeps SK secretly, and provides p, q, g, L with PK as the public parameters. The values of p, q, g and L remain the same during the full lifecycle of an SSO system. While, the asymmetric key pair (SK, PK) will be updated when necessary. For example, when SK is leaked, IdP must update (SK,PK).

RP initial registration. The RP initial registration is invoked only once by an RP, to apply ID_{RP} and $Cert_{RP}$ from IdP. The detailed processes are as follows:

1) RP sends a request $Req_{Cert_{RP}}$ to the IdP. The $Req_{Cert_{RP}}$ contains the RP's endpoint (e.g., URL) for receiving the identity proof.
2) IdP chooses a unique and random r ($1 < r < q$), calculates ID_{RP} using Equation 1, generates the signature Sig_{SK} of $[ID_{RP}, endpoint]$ with SK, and returns $[ID_{RP}, endpoint, Sig_{SK}]$ as $Cert_{RP}$. Here, the r is never leaked.
3) The RP verifies $Cert_{RP}$ using PK, and stores ID_{RP} with the valid $Cert_{RP}$ for the further use.

SSO login. Once a user attempts to log in at an RP, the SSO login is invoked. We use the OIDC implicit protocol flow as an example, to demonstrate how to integrate the three functions $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and $\mathcal{F}_{PID_U \mapsto Account}$ into the typical SSO systems. As shown in Figure 4, the SSO login sub-protocol contains four phases, RP identifier transforming, RP identifier refreshing, PID_U generation and $Account$ calculation. In the RP identifier transforming, the user and RP negotiate PID_{RP} based on Diffie-Hellman key exchange [38], where PID_{RP} is calculated as in Equation 2. In the RP identifier refreshing, the user registerers the unique PID_{RP} at IdP. In the PID_U generation, IdP calculates PID_U with ID_U and PID_{RP} as in Equation 3. And in the $Account$ calculation, the RP derives the unchanged $Account$ as in Equation 4.

C. UPRESSO Protocol flow

In UPRESSO, the SSO login sub-protocol provides the secure SSO service and prevents both the IdP-based access tracing and RP-based identity linkage. The protocol, shown in Figure 3, prevents the curious IdP from obtaining the RP's identifying information during the interchanges, and avoids the adversary to break the security and user's privacy. Here we introduce the detailed processes for each step in Figure 3.

RP identifier transforming. In this phase, the user and RP cooperative to generate PID_{RP} as follows:

- The user sends a login request to trigger the negotiation of PID_{RP} (Step 1).
- The RP chooses a random N_{RP} ($1 < N_{RP} < q$), calculates $Y_{RP} = {ID_{RP}}^{N_{RP}} mod \ p$ (Step 2.1.1); and sends $Cert_{RP}$ with Y_{RP} to the user (Step 2.1.2).
- The user checks the $Cert_{RP}$, extracts ID_{RP} from the valid $Cert_{RP}$, chooses a random N_U ($1 < N_U < q$) to calculate $PID_{RP} = {Y_{RP}}^{N_U} mod \ p$ (Step 2.1.3); and sends N_U with PID_{RP} to the RP (Step 2.1.4).
- The RP calculates PID_{RP} with N_U and Y_{RP}, checks its consistency with the received one, derives the trapdoor $t = (N_U * N_{RP})^{-1} \ mod \ q$ (Step 2.1.5); and sends the calculated PID_{RP} to the user (Step 2.1.6).
- The user checks the consistency of the received PID_{RP} with the stored one.

During the process, the user will halt the login, if the $Cert_{RP}$ is invalid or the received PID_{RP} is different from the stored

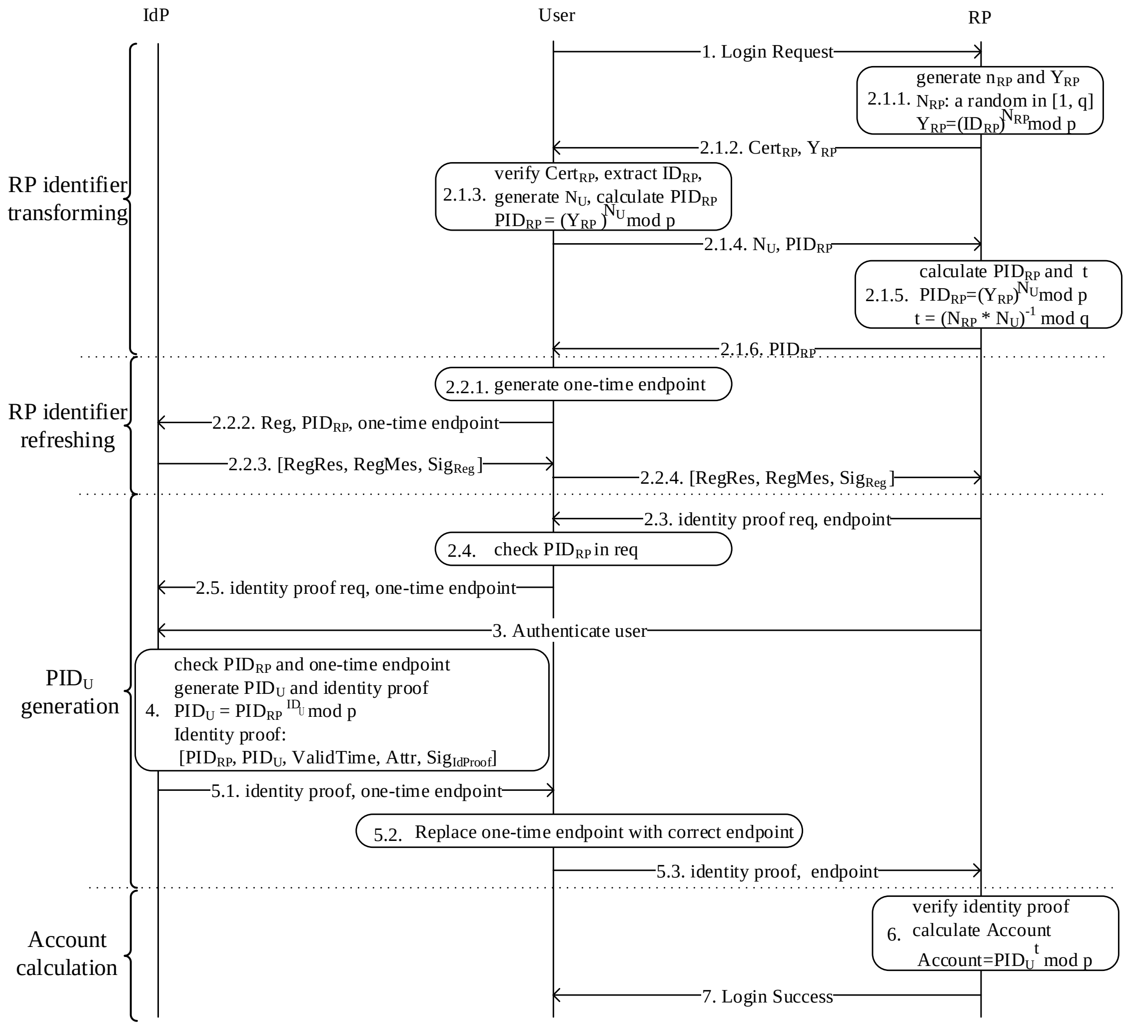

Fig. 3: Process for each user login.

one. The RP also halts the process if the PID_{RP} sent by the user is inconsistent with the calculated one.

RP identifier refreshing. The user registers PID_{RP} at the IdP as follows.

- The user generates an one-time endpoint to hide the RP's endpoint from IdP (Step 2.2.1), and sends the registering request [Reg, PID_{RP}, one-time endpoint] to the IdP (Step 2.2.2).
- The IdP checks PID_{RP}, and constructs the response [$RegRes$, $RegMes$, Sig_{Reg}] (Step 2.2.3). The $RegRes$ is registration result, and is set as OK only when PID_{RP} is never used before and is of order q module p. The $RegMes$ is the same as the dynamic registration response, and contains PID_{RP}, the issuing time and valid time. The Sig_{Reg} is the signature for $RegRes$ and $RegMes$ generated by the IdP with SK.
- The user accepts $RegRes$ directly due to the secure connection with IdP, and forwards the registration result to the RP (Step 2.2.4).
- The IdP checks Sig_{SK} and $RegMes$, and accepts $RegRes$ only when Sig_{Reg} is valid, PID_{RP} is the same as the negotiated one, and $RegMes$ is not expired.

If $RegRes$ is OK, the RP identifier refreshing completes. Otherwise, the user and RP will renegotiate the PID_{RP}.

$\mathbf{PID_U}$ generation. In this phase, the RP continues the process of the user's login and obtains the PID_U generated by the IdP. The processes are as follows.

- The RP uses PID_{RP} and the endpoint to construct an identity proof request, which is the same as the one in OIDC. (Step 2.3).
- The user checks the consistency of the received PID_{RP} with the negotiated one (Step 2.4); replaces the endpoint with the one-time endpoint generated in Step 2.2.1, and sends the modified identity proof request to the IdP (Step 2.5).
- The IdP authenticates the user if she hasn't been authenticated (Step 3); checks whether PID_{RP} and the one-time endpoint have been registered, calculates PID_U using Equation 3, constructs the identity proof [PID_{RP}, PID_U, $ValTime$, $Attr$,$Sig_{IdProof}$] where $ValTime$ is

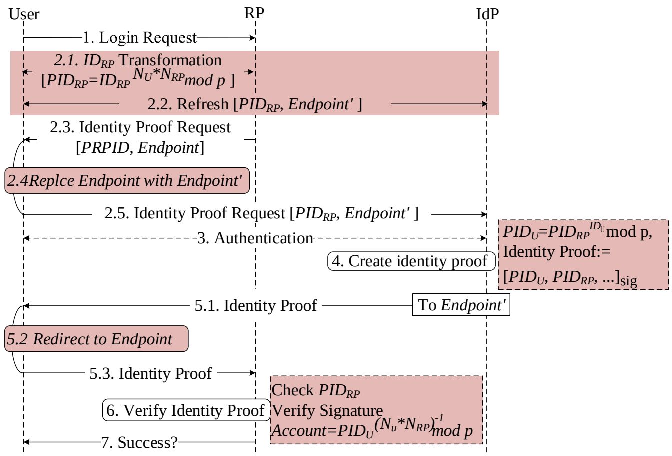

Fig. 4: UPRESSO compatibility with OIDC.

the valid period, $Attr$ contains the attributes that the user agrees to provide to the RP, $Sig_{IdProof}$ is the signature of the identity proof generated by IdP with SK (Step 4). Then, the IdP sends the identity proof with the one-time endpoint to the user (Step 5.1).

- The user finds the endpoint corresponding to the one-time endpoint (Step 5.2), and forwards the identity proof to the RP through this endpoint (Step 5.3).

The user halts the process if the PID_{RP} in the identity proof request is inconsistent with the negotiated one. The IdP rejects the identity proof request, if the PID_{RP} and the one-time endpoint have not been registered.

Account calculation. Finally, RP derives the user's $Account$ and completes the user's login as follows. The RP performs the checks on the identity proof, including the valid time, correctness of $Sig_{IdProof}$, and the consistency between PID_{RP} and the negotiated one. If all the checks pass, the RP extracts PID_U, and calculates $Accout$ according to Equation 4 (Step 6); and sends the $Success$ as the login result to the user (Step 7). If any check fails, the RP returns the $Fail$ to the user.

D. Compatibility with OIDC

UPRESSO could be integrated in the traditional SSO systems, to prevent the IdP-based access tracing and RP-based identity linkage. The integration doesn't degrade the security and only requires minimal modification. Here, we use the implicit protocol flow of OIDC as an example to demonstrate the compatibility of UPRESSO with the traditional SSO systems, as shown in Figure 4. The further analysis, such as integration with the authorization code flow of OIDC, is provided in Section VIII.

UPRESSO doesn't introduce any new role, nor change the security assumptions on each role (i.e., user, IdP and RP).

As shown in Figure 4, in UPRESSO, the SSO protocol for identity proof (Steps between 2.3 and 7) is the same as in OIDC (Steps between 2 and 7); the formats of identity proof and corresponding request are the same as in OIDC; the correctness checks on the identity proof request at the IdP (i.e., consistency of RP' identifier and endpoint with the registered one) are the same as in OIDC; the correctness checks on the identity proof (i.e., consistency of RP' identifier with the one in the request, integrity, validity time, freshness, and etc.) at the RP are the same as in OIDC.

However, UPRESSO achieves privacy preservation by integrating $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and $\mathcal{F}_{PID_U \mapsto Account}$, and introduces the following modifications on OIDC.

1) The identity proof is bound with PID_{RP} instead of ID_{RP}, which introduces the RP identifier transforming (Step 2.1) and RP identifier refreshing (Step 2.2).
2) The identity proof is designated to one-time endpoint instead of RP's identifying endpoint, which requires the user to register the one-time endpoint in Step 2.2 and replace it with the original endpoint in Step 5.2.
3) IdP generates PID_U based on (PID_{RP}, ID_U) instead of (ID_{RP}, ID_U).
4) The RP calculates $Account$ from the changing PID_U instead of an unchanged one.

Moreover, the RP identifier refreshing is compatible with the dynamic registration in OIDC, with the following modifications, that it is triggered by the user instead of the RP, adds PID_{RP} in the request and includes a signature Sig_{Res} in the response.

VI. Analysis

In this section, we analyze the security and privacy of UPRESSO.

A. Security

We prove that the basic requirements of SSO system, i.e., user identification, receiver designation, integrity and confidentiality, are still satisfied in UPRESSO with the modifications on OIDC, whose security has been formally analyzed in [34]. In the following, we analyze the affects of the modifications listed in Section V-D, respectively.

The first modification may affect the receiver designation, as the identity proof is bound with PID_{RP} instead of ID_{RP}. However, in UPRESSO, PID_{RP} provides the same binding as ID_{RP}, which is achieved by the **transformed receiver designation** through $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ with the protocols in RP identifier transforming and RP identifier refreshing. In OIDC, ID_{RP} is used to ensure that identity proof is only valid to the designated RP, as the correct IdP ensures that one ID_{RP} is only assigned to one RP, and the correct RP only accepts the identity proof which has a same ID_{RP} with the assigned one. In UPRESSO, the PID_{RP} is also unique [1] and one PID_{RP} is only assigned to one RP when at least a correct user or correct RP exists, then identity proof bound with a PID_{RP} is only valid to this RP. The detailed proofs are:

[1]This has no impact on scalability. In practice, we only need to ensure all PID_{RP}s are different among the unexpired identity proof (the number denoted as n). We calculates the probability that at least one PID_{RP} is obtained more than once in RP identifier transforming. The probability is $1 - \prod_{i=0}^{n-1}(1 - i/q)$, increases with n. For a 256-bit q, the probability is smaller than 2^{-183}, when $n = 2^{36}$ which means IdP's throughput is about $2 * 10^8$ req/s when valid period is 5 minutes.

- The correct IdP ensures the the uniqueness of PID_{RP}, while the correct user checks this uniqueness through the RP identifier refreshing directly, and the correct RP checks it based on the user-redirected RP identifier refreshing result, which is signed by the IdP.
- The correct user and RP check the freshness of PID_{RP} based on the nonce N_{RP} and N_U respectively, which avoids the replay attack by reusing the unique PID_{RP} incorrectly.
- The cooperation between the user and RP in function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and the protocol in RP identifier transforming, prevent the adversary from controlling the generation of PID_{RP}. For example, the adversary may negotiate with two correct entities (either user or RP) and attempt to make one same PID_{RP} generated. This is prevented, as the RP chooses N_{RP} before obtaining N_U to calculate $PID_{RP} = {ID_{RP}}^{N_{RP} * N_u}\ mod\ p$, while the user calculates PID_{RP} by ${Y_{RP}}_u^N\ mod\ p$ and fails to derive N_{RP} from the Y_{RP}.

The second modification may also affect the receiver designation, while UPRESSO uses $Cert_{RP}$ to achieve **transformed receiver designation**. In OIDC, the endpoint is used to ensure that the correct user sends the identity proof only to the designated RP, while the correct mapping between the endpoint and ID_{RP} is ensured by the IdP. In UPRESSO, the correct user obtains the correct endpoint for ID_{RP} from $Cert_{RP}$. While, $Cert_{RP}$ is generated by the IdP to bind RP's endpoint with the ID_{RP}, and can never be forged or modified by others due to the digital signature.

The last two modifications affect user identification, which is still ensured in UPRESSO by the **trapdoor user identification** provided by $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$. In OIDC, the RP uniquely identifies a user based on the identifier from the IdP, who provides a unique and unchanged identifier for a user ID_U at an RP. In UPRESSO, the correct RP computes an unchanged value $Accout = {PID_U}^t\ mod\ p = {ID_{RP}}^{ID_U}\ mod\ p$ for a user's multiple logins, as shown in Equation 5; and one $Accout$ is only assigned to one user at an RP, as IdP ensures that one ID_U is only assigned to one user. Moreover, the calculation can never be tampered by the adversary, as PID_U is provided by the IdP and protected in the identity proof, while t is stored at the RP itself, and the calculation is performed at the RP.

The above analysis demonstrates that (1) integrity and confidentiality are not affected by the modifications in UPRESSO and could be guaranteed by the mechanisms (i.e., digital signature and TLS) inherited from OIDC; and (2) these modifications on OIDC introduce no security degradation on user identification and receiver designation. Therefore, UPRESSO provides the secure SSO service.

B. Privacy

In this section, we prove that UPRESSO prevents the IdP-based access tracing and RP-based identity linkage.

IdP-based access tracing prevention. UPRESSO prevents the IdP-based access tracing, the curious IdP cannot derive RP's identifying information from one login, nor associate the logins based on which RP is visited. The detailed proofs are as follows.

The IdP cannot derive RP's identifying information from any login. UPRESSO prevents the leakage of RP's identifying information (Step 2 in Figure 1 in OIDC), as the user provides the IdP a random string as the one-time endpoint instead of the RP's exact endpoint, and sends PID_{RP} instead of ID_{RP}. From any PID_{RP}, the IdP cannot derive ID_{RP}, as the IdP doesn't know $N_U * N_{RP}$ and cannot determine which ID_{RP} corresponds to this PID_{RP}. That is because, for any given PID_{RP}, all the already-assigned ID_{RP}s could be the one corresponding to it, as for arbitrary ID_{RP} there always exists N_U and N_{RP} making $PID_{RP} = {ID_{RP}}^{N_U * N_{RP}\ mod\ q}\ mod\ p$. We prove it in two steps.

- First, for an arbitrary PID_{RP} (denoted as $g^{r_1 * N_1\ mod\ q}\ mod\ p$, $N_1 = N_{U1} * N_{RP1}\ mod\ q$) and an arbitrary ID_{RP} (denoted as $g^{r_2}\ mod\ p$, $r_2 \neq r_1$), there always exists N_2 satisfying $r_2 * N_2 = r_1 * N_1\ mod\ q$. That's because q is a prime and co-prime to any r_2, then there always exists $N_2^{'}$ making $r_2 * N_2^{'} = 1\ mod\ q$, and $N_2 = (r_1 * N_1) * N_2^{'}\ mod\ q$ making the equality hold.
- Second, for the derived N_2, there always exists two numbers N_{U2} and N_{RP2} satisfying $N_2 = N_{U2} * N_{RP2}\ mod\ q$. That's because, q is a prime and co-prime to any chosen N_{U2}, there always exists a number $N_{RP2}^{'}$ making $N_{U2} * N_{RP2}^{'} = 1\ mod\ q$, and then exists $N_{RP2} = N_{RP2}^{'} * N_2\ mod\ q$ making $N_{U2} * N_{RP2} = N_2\ mod\ q$.

IdP cannot to determine whether two or more logins are for a same RP. The only information that can be used for this classification is one-time endpoint and PID_{RP}. However, both one-time endpoints and PID_{RP}s are independent among the logins, guaranteed by the secure random number generators that used to generate one-time endpoints and N_Us at the correct user, and N_{RP}s at the correct RPs.

RP-based identity linkage prevention. UPRESSO prevents the RP-based identity linkage, any malicious RPs cannot derive the user's identifying information (i.e., ID_U) from PID_U and $Account$, nor associate a user's logins at different RPs. The detailed proofs are as follows.

An RP cannot derive ID_U from PID_U and $Account$ in one login, due to the one-way function $\mathcal{F}_{ID_U \mapsto PID_U}$.

- For PID_U, it equals to ${PID_{RP}}^{ID_U}\ mod\ p$ according to $\mathcal{F}_{ID_U \mapsto PID_U}$, and further transformed to $g^{r * N_U * N_{RP} * ID_U\ mod\ q}\ mod\ p$ by combining $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ with Equation 1. Here, p, q and g are public parameters, PID_{RP}, N_{RP} and N_U are known to the RP, while r is secretly maintained by the IdP and never leaked to the RP. Then, it is computational infeasible to compute ID_U from PID_U with all the known values (e.g., PID_{RP}, g and etc.) due to the discrete logarithm problem.
- For $Account$, it equals to ${ID_{RP}}^{ID_U}\ mod\ p$ according to Equation 5, and further transformed to $g^{r * ID_U\ mod\ q}\ mod\ p$ with Equation 1. Same as the above

analysis, it is also computational infeasible to compute ID_U from $Account$ with all the known values (e.g., ID_{RP}, g and etc.).

- The RP cannot infer ID_U by combining $Account$ and PID_U. $Account$ and PID_U are both generated from ID_U, however $Accout = {PID_U}^t\ mod\ p$ where t is a random value known to t and independent with ID_U.

An RP cannot derive ID_U from PID_Us and $Account$s obtained in multiple logins. All these $Account$s are equal, while any PID_U (e.g., PID_{U1}) can be computed from any other PID_U (e.g., PID_{U2}), PID_{U1}=$PID_{U2} * Account^{N_{U1}*N_{RP1}-N_{U2}*N_{RP2}}$, where N_{U2}, N_{RP2}, N_{U2} and N_{RP2} are values known to the RP and independent with ID_U.

The collusive RPs may attempt to associate a user's $Account$s by checking whether the equality $Account_2$=$(Account_1)^{r_2/r_1}\ mod\ p$ holds for $Account_1$ at an RP $g^{r_1}\ mod\ p$ and $Account_2$ at $g^{r_2}\ mod\ p$. But, the associating always fails, as RPs cannot derive r (and therefore r_2/r_1) from ID_{RP} due to the discrete logarithm problem. The collusive RPs cannot associate a user's PID_Us either, due to the unknown rs.

A malicious RP may attempt to manipulate N_{RP}s in one or multiple logins to make the generated PID_Us or $Account$s be vulnerable for deriving ID_U, and the collusive RPs may attempt to manipulate N_{RP}s cooperatively to make a user's PID_Us or $Account$s be correlated at these RPs and then to associate a user's multiple logins. Here, N_{RP}s are the only values controlled by the RPs. However, the manipulation on N_{RP} is masked by N_U in PID_U due to cooperative function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, and has no effect on $Account$ as shown in Equation 5.

- For PID_U, it equals to ${PID_{RP}}^{ID_U}\ mod\ p$ and $g^{r*N_U*N_{RP}*ID_U\ mod\ q}\ mod\ p$. The RP cannot control PID_{RP} as it generates N_{RP} before obtaining N_U and cannot change N_{RP} after obtaining N_U. The random and independent N_U prevents the RPs from controlling PID_U.
- For $Account$, it equals to ${ID_{RP}}^{ID_U}\ mod\ p$ and $g^{r*ID_U\ mod\ q}\ mod\ p$. Obviously, $Account$ is independent with N_{RP} and cannot be controlled by any RP.

The malicious RPs may collude with malicious users and attempt to associate a victim user's $Account$s at the different RPs based on the relation among the $Account$s of the malicious user and victim user. For example, at ID_{RP1} and ID_{RP2}, the victim user's accounts are $Account_{v1}$ and $Account_{v2}$, while the malicious user's ones are $Account_{m1}$ and $Account_{m2}$, then the adversary may attempt to find whether exists a value ID_{m-v} satisfying both $Account_{m1}/Account_{v1} = {ID_{RP1}}^{ID_{m-v}}\ mod\ p$ and $Account_{m2}/Account_{v2} = {ID_{RP2}}^{ID_{m-v}}\ mod\ p$. However, as ID_Us are independent while ID_U is only known to the IdP and the corresponding user, the adversary cannot derive the victim user's ID_U (and then ID_{m-v}) for this association.

The malicious RPs may collude with malicious users and manipulate N_{RP}s in the sessions with the victim users for linking the victim user's logins at different RPs. However, same as the above analysis, malicious RPs cannot control the victim user's PID_Us and $Account$s due to the independent N_Us from the victim user.

VII. Implementation and Performance Evaluation

We have implemented a prototype of UPRESSO, and evaluated its performance by comparing with the original OIDC and a privacy-preserving SSO, SPRESSO.

A. Implementation

We adopt SHA-256 for digest generation, and RSA-2048 for the signature generation. We randomly choose a 2048-bit prime as p, a 256-bit prime as q, and a q-order generator as g. The N_U, N_{RP} and ID_U are 256-bit random numbers. Then, the discrete logarithm cryptography provides equivalent security strength (i.e., 112 bits) as RSA-2048 [39]. UPRESSO includes the processing at the IdP, user and RP. The implementations at each entity are as follows.

The implementation of IdP only needs small modifications on existing OIDC implementation. The IdP is implemented based on MITREid Connect [40], an open-source OIDC Java implementation certificated by the OpenID Foundation [41]. We add 3 lines Java code for generation of PID_U, 26 lines for converting the dynamic registration into RP identifier refreshing, i.e., checking PID_{RP} provided by the RP and adding a signature Sig_{Reg} in the response. The calculations of ID_{RP}, PID_U and RSA signature are implemented based on Java built-in cryptographic libraries (e.g., BigInteger).

The user-side processing is implemented as a Chrome extension with about 330 lines JavaScript code, to provide the functions in Steps 2.1.3, 2.2.1, 2.4 and 5.2. The cryptographic computation, e.g., $Cert_{RP}$ verification and PID_{RP} negotiation, is implemented based on jsrsasign [42], an efficient JavaScript cryptographic library. This chrome extension requires permissions to read chrome tab information, send HTTPS request/reply and hijack the HTTPS responses, to obtain the RP's URL and communicate with IdP and RP. Here, the cross-origin HTTPS requests sent by this chrome extension to the RP and IdP, will be blocked by Chrome due to the default same-origin security policy. To avoid this block, UPRESSO modifies the IdP and RP, and sets `chrome-extension://chrome-id` (`chrome-id` is uniquely assigned by Google) in the HTTPS header `Access-Control-Allow-Origin` of the IdP's and RP's responses.

We provide a Java SDK for an RP to integrate UPRESSO. The SDK provides 2 functions to encapsulate RP's processings: one for *RP identifier transforming* and *RP identifier refreshing* phases, and the other for *Account calculation* phase. The SDK is implemented based on the Spring Boot framework with about 1100 lines code, and cryptographic computations are implemented based on Spring Security library. An RP only needs to invoke these two functions for the integration.

B. Performance Evaluation

We have compared the processing time of each user login in UPRESSO, with the original OIDC implementation (MITREid

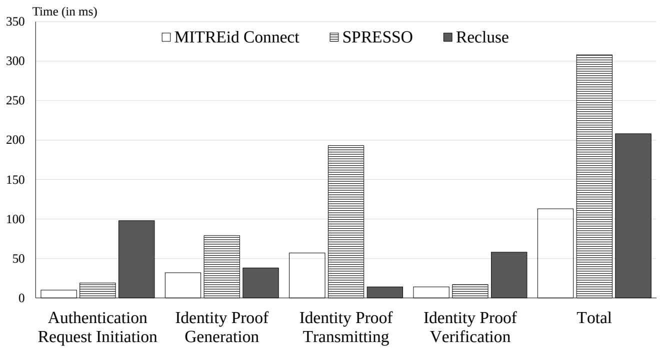

Fig. 5: The Evaluation.

Connect) and SPRESSO which only hides the user's accessed RPs from IdP.

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 prox64. 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 (the load of CPU and network is in the moderate level).

We have measured the processing time for 1000 login flows, and the the results is demonstrated in Figure 5. The average time is 208 ms, 113 ms and 308 ms for UPRESSO, MITREid Connect and SPRESSO respectively. The result shows that UPRESSO proved the privacy protection without introducing prominent overhead.

For better comparison, we further divide a SSO login flow into 4 phases, which : 1. **Authentication request initiation** (Steps 1-2.5 in Figure 3), 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 4 in Figure 3), denoting the construction of identity proof at the IdP (excluding the user authentication); 3. **Identity proof transmition** (Steps 5.1-5.3 in Figure 3), for transmitting the proof from the IdP to the RP with the user's help; and 4. **Identity proof verification** (Steps 6 in Figure 3), for the RP verifying and parsing the proof for the user's $Account$.

In the authentication request initiation, as UPRESSO need negotiate the PID_{RP} (containing 1 modular exponentiation at the user and 2 at the RP) and renew the RP identifier at IdP, it should require the longest time cost. And SPRESSO has to obtain the public information from IdP and encrypt its domain (as the RP identifier), it should result the slight overhead compared with MITREid Connect. Finally, the evaluation shows that MITREid Connect requires the shortest time (10 ms), UPRESSO needs 98 ms and SPRESSO needs 19 ms.

For identity proof generation, UPRESSO need the extra time cost for the generation of PID_U compared with MITREid Connect and SPRESSO. However, the evaluation shows that SPRESSO requires the longest time in this stage, and finally the time cost is found caused by the signature generation as SPRESSO is implemented with JavaScript while the others are using Java. In this stage, MITREid Connect needs 32 ms, UPRESSO needs an extra 6 ms and SPRESSO requires 71 ms.

For identity proof transmission, UPRESSO should need the shortest time among these competitors, as it only need the chrome extension to relay the identity proof from the IdP to RP. 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, which result that the time cost should be at a moderate level. 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. The evaluation result shows that MITREid Connect needs 57 ms, UPRESSO needs only 14 ms and SPRESSO needs about 193 ms.

In identity proof verification, UPRESSO needs the extra time for calculation of $Account$ and signature verification compared with MITREid and SPRESSO. Therefore, the evaluation result is that MITREid Connect needs 14 ms, UPRESSO needs 58 ms and SPRESSO requires 17 ms.

VIII. DISCUSSION

In this section, we provide some discussion about UPRESSO.

Authorization code flow. UPRESSO 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 use this code with the RP identifier and secret for the `id token`. To avoid the IdP obtaining the IP address from the connection, the anonymous network (e.g., Tor) may be used to establish the connection. And the RP's identifier and secret are issued by the IdP in the dynamic registration described forementioned.

Multi-platform user agent. UPRESSO doesn't store any persistent information in the platform and may be implemented to be platform independent. Firstly, all the information (e.g., $Cert_{RP}$, PID_{RP}, N_U, ID_{RP} and one-time endpoint) processed and cached in the user's platform is only correlated with the current session, which allows the user to log in to any RP with a new platform without any synchronization. Secondly, in the current implementation of UPRESSO, a browser extension is adopted to capture the redirection from the RP and IdP, to reduce the modification at the RP and IdP. However, to comfort the requirement of using UPRESSO in multiple platforms (e.g., mobile phones), UPRESSO is able to be implemented based on HTML5, without the use of any browser extensions, or plug-ins. The assumption for secure cross-platform user agent is the IdP must always be honest without providing any malicious JavaScript code which is similar with it in SPRESSO (requiring the honest entity, FWD). But it is required the code should be trustful, which is ensured to be correct and unmodifiable by any adversary. As the IdP is considered honest, it could take the responsibility for providing the same trustful JavaScript code as chrome extension to accomplish the PID_{RP} negotiation and other missions. While the code has been already loaded from IdP, if the code is honest (without any prior inserted malicious code) it cannot be modified or monitored. Moreover, the new mechanism called SRI (subresource integrity) under development enables the opener of an iframe to require the hash of document loaded in it to equal with the one set by opener, which ensures the code cannot be malicious even the IdP try to insert the malicious code. For each start, RP opens the iframe with the SRT hash (of correct user agent code) and the iframe downloads the code from IdP, so that, as the RP will never collude with the IdP, the code cannot be malicious.

DoS attack. The adversary may perform the DoS attack. The malicious RPs may try to exhaust the ID_{RP} by applying the $Cert_{RP}$ frequently. However the large p provides a large set of ID_{RP}, 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' PID_{RP} be rejected at the IdP, by registering a large set of PID_{RP}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. Moreover, for IdP's dynamic registration storage, the data contains RP's client_id (no more than 256-bit length) and redirect_uri (tens-Byte length). We consider that each dynamic registration data cost no more than 100 Bytes storage. And for each client_id IdP can set the lifetime of validity. It is assumed that for each client_id its lifetime is 5 minutes and during 5 minutes there are 1 billion requests for dynamic registration. So IdP need to offer about 100 GB storage for dynamic registration. The extra cost of storage can be ignored.

Identity injection by malicious IdP. It has been discussed in [6] that even the impersonate attack by malicious IdP is not considered, the malicious IdP might lead the user to access the RP as the identity of the adversary (identity injection). That is the IdP might generate an identity proof representing the adversary's identity while an honest user log in to an honest RP. However, in SPRESSO, the user is required to send her email to RP at the very beginning of authentication and IdP must provide the relevant identity proof. It is also available in UPRESSO that user upload her extra user name (defined by user for each RP) before the login to the RP.

RP certificate. The honest IdP is assumed to generate the correct r and ID_{RP}. However, based on the idea of certificate transparency [43], an external check may be performed to ensure that no two valid $Cert_{RP}$ assigned to a same ID_{RP} and ID_{RP} is a primitive root modulo p. The external check needs to be performed by a third party instead of RP, as the RP will benefit from incorrect ID_{RP}, e.g., linking the user among RPs with the same ID_{RP}. In UPRESSO, the RP certificate $Cert_{RP}$ is used to provide the trusted binding between the ID_{RP} and the RP's endpoint. RP certificate could be compatible with the X.509 certificate. To integrate RP certificate in X.509 certificate, the CA generates the ID_{RP} 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 2.1.2 in Figure 3, $Cert_{RP}$ is sent to the user during the key agreement in TLS. Moreover, the mechanisms (e.g., the Certificate Transparency) to avoid illegal certificate issued by the CA being adopted to ensure the correctness of ID_{RP}, i.e., globally unique and being the primitive root.

IX. Related Works

Various SSO standards have been proposed and widely deployed. For example, OIDC is adopted by Google, OAuth 2.0 is deployed in Facebook, SAML is implemented in the Shibboleth project [44], and Central Authentication Service (CAS) [45] is widely adopted by Java applications. Kerberos [46], proposed by MIT, is now replaced by the SSO standards (e.g., OIDC, OAuth) who provide better privacy, as the users in Kerberos fail to control on the releasing of their private information.

A. Security consideration about SSO systems.

Analysis on SSO designing and implementation. Even the user's account at IdP not compromised, various vulnerabilities in the SSO implementations were exploited for the impersonation attack and identity injection, by breaking at least one of the requirements. (1) To break the confidentiality of identity proof, Wang et al. [9] performed a traffic analysis/manipulation on SSO implementations provided by Google and Facebook; [10]–[12] exploited the vulnerability at the RP's implementations of OAuth, i.e., the publicly accessible information is misused as the identity proof; Armando et al. [26] exploited the vulnerability at the user agent, to transmit the identity proof to the malicious RP. (2) The integrity is broken [9]–[13], [28], [29] in the implementations of SAML, OAuth and OIDC. For example, [13] exploited XML Signature wrapping (XSW) vulnerabilities to modify the identity proof without being found by RPs; the incomplete verification at the client allows the modification of the identity proof [10]–[12]; ID spoofing and key confusion make the identity proof issued by the adversary be accepted by the victim RPs [28], [29]. (3) The designation is also broken [10]–[12], [30], as the RP may misuse the bearer token as the identity proof [10]–[12], and IdP may not bind the refresh/access token with RP which allows the refresh/access token injection [30]. Cao et al. [47] attempts to improve the confidentiality and integrity, by modifying the architecture of IdP and RP to build a dedicated, authenticated, bidirectional, secure channel between them.

Analysis on mobile platform SSO systems. Compared to web SSO systems, new vulnerabilities were found in the mobile SSO systems, due to the lack of trusted user agent (e.g., the browser) [14], [27]. The confidentiality of the identity proof may be broken due to the untrusted transmission. For example, the WebView is adopted to send the identity proof, however, the malicious application who integrates this WebView may steal the identity proof [14]; the lack of authentication between mobile applications may also make the identity proof (or index) be leaked to the malicious applications [27]. Various automatic tester were proposed to analyze the mobile SSO systems [11], [14], [27], [31], [32], for the traditional vulnerabilities (e.g., inadequate transmission protection [27], token replacement [32]) and new ones in mobile platforms (webview [14], application logic error [31]).

Formal analysis on SSO systems. The comprehensive formal security Analysis were performed on SAML, OAuth and OIDC. Armando et al. [33] built the formal model for the Google's implementation of SAML, and found that malicious RP might reuse the identity proof to impersonate the victim user at other RP, i.e., breaking the binding. Fett et al. [8], [34] conducted the formal analysis of the OAuth 2.0 and OpenID Connect standards using an expressive Dolev-Yao style model, and proposed the 307 redirect attack and IdP Mix-Up attack. The 307 redirect attack makes the browser expose the user's credential to RP. IdP Mix-Up attack allows the malicious IdP to receive the identity proof issued by the correct IdP for the correct RP (who integrates the malicious IdP), which breaks the confidentiality. Fett et al. [8], [34] proved that OAuth 2.0 and OIDC satisfy the authorization and authentication requirements, as the two bugs are fixed in the revisions of OAuth and OIDC. Ye et al. [48] performed a formal analysis on the implementation of Android SSO systems, and found a vulnerability in the existing Facebook Login implementation on Android system, as the session cookie between the user and Facebook may be obtained by the malicious RP application.

Analysis on malicious IdP. One concern of SSO is that, the adversary controls the user's accounts at the correlated RPs, once the user's account at IdP is compromised. A backwards-compatible extension (single sign-off) is proposed for OIDC, which revokes the adversary's access to the RPs [5].

The requirements of security authentication are summarized based on the previous work about SSO security. Moreover, as UPRESSO is compatible with OIDC, the protection schemes against existing attacks are also available in UPRESSO.

B. Privacy consideration about SSO systems.

Privacy is the another concern of SSO systems. As suggested in NIST SP800-63C [17], the user's privacy protection in SSO systems includes, 1) the user's control on the attributes exposed to the RP, 2) prevention of identity linkage, and 3) avoiding of IdP-based access tracing.

Privacy-preserving SSO systems. OAuth and OIDC provide the user notification to achieve the user's control on its private information [7], [12]. The pairwise user identifier is proposed to avoid the identity linkage performed by collusive RPs in SAML and OIDC [2], [3]. In SPRESSO [6] and BrowserID [18] (adopted in Persona [21] and its new version Firefox Accounts [23]), IdP doesn't know which RP the user is accessing, however the user's email address is sent to the RP, which introduces the risk of identity linkage performed by the collusive RPs. Fett et al. [18], [49] performed a formal analysis on the implementation of BrowserID and found that IdP may still know which RP is accessed by the user.

However, none of existing SSO protocols are able to protect user from being tracked by both the collusive RPs and IdP at the same time. Compared with the existing schemes that only protect user's privacy in one side, UPRESSO is able to prevent user from being traces in both sides (being tracked by RPs and IdP). Moreover, UPRESSO is not the simple combining of existing schemes but the completely novel solution based on the OIDC standard.

Anonymous SSO systems. Anonymous SSO scheme is proposed to hide the user's identity to both the IdP and RPs, which may only be applied to the anonymous services that do not identify the user. One of the earliest anonymous SSO system is proposed for Global System for Mobile (GSM) communication in 2008 [50]. In 2013, the notion of anonymous single sign-on is formalized [51]. Then, the various cryptographic primitives, e.g., group signatures and zero-knowledge proof, are adopted to build anonymous SSO scheme [51], [52].

However, the anonymous SSO systems enable the user access the service provided by RP without providing her identity to both RP and IdP which avoids user being traced, therefore, RP is unable to distinguish whether multiple accesses are from the same user or not. For most web service providers, it means the personalized service for users are not available, which results the anonymous schemes are not useful. Compared with anonymous SSO schemes, in UPRESSO RP is able to transform the user's PID_U into the constant $Account$, based on which the RP can distinguish the same user in multiple requests.

X. CONCLUSION

In this paper, we, for the first time, propose UPRESSO to protect the users' privacy from both the curious IdP and collusive 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 collusiveh RPs from linking the users and allows the RP to provide the consecutive and individual services. The trusted user ensures the correct content and transmission of the identity proof with a self-verifying RP certificate. The evaluation demonstrates the efficiency of UPRESSO, about 200 ms for one user's login at a RP in our environment.

REFERENCES

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

[2] 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.

[3] 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.

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

[5] 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*, 2018, pp. 1475–1492.

[6] 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*, 2015, pp. 1358–1369.

[7] 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*, 2014, pp. 892–903.

[8] 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*, 2016, pp. 1204–1215.

[9] 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, San Francisco, California, USA*, 2012, pp. 365–379.

[10] 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*, 2014, pp. 495–510.

[11] 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*, 2016, pp. 167–176.

[12] 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*, 2016, pp. 651–662.

[13] 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*, 2012, pp. 397–412.

[14] 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*, 2016, pp. 196–205.

[15] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich, "Explicating SDKs: Uncovering assumptions underlying secure authentication and authorization," in *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, 2013, pp. 399–314.

[16] Eve Maler and Drummond Reed, "The venn of identity: Options and issues in federated identity management," *IEEE Security & Privacy*, vol. 6, no. 2, pp. 16–23, 2008.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

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

[24] 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.

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

[26] 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.

[27] Hui Wang, Yuanyuan Zhang, Juanru Li, Hui Liu, Wenbo Yang, Bodong Li, and Dawu Gu, "Vulnerability assessment of oauth implementations in android applications," in *Proceedings of the 31st Annual Computer Security Applications Conference, Los Angeles, CA, USA*, 2015, pp. 61–70.

[28] 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*, 2016, pp. 321–336.

[29] 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*, 2017, pp. 251–266.

[30] 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*, 2018, pp. 1459–1474.

[31] 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, Proceedings*, 2017, pp. 313–335.

[32] 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*, 2019, pp. 269–282.

[33] 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*, 2008, pp. 1–10.

[34] 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*, 2017, pp. 189–202.

[35] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang, "Targeted online password guessing: An underestimated threat," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, 2016, pp. 1242–1254.

[36] Hung-Min Sun, Yao-Hsin Chen, and Yue-Hsun Lin, "opass: A user authentication protocol resistant to password stealing and password reuse attacks," *IEEE Trans. Information Forensics and Security*, vol. 7, no. 2, pp. 651–663, 2012.

[37] Patrick Gallagher, "Digital signature standard (dss)," *Federal Information Processing Standards Publications, volume FIPS*, pp. 186–3, 2013.

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

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

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

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

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

[43] Ben Laurie, Adam Langley, and Emilia Käsper, "Certificate transparency," *RFC*, vol. 6962, pp. 1–27, 2013.

[44] "The shibboleth project," https://www.shibboleth.net, Accessed July 30, 2019.

[45] 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.

[46] 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*, 1988, pp. 191–202.

[47] 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. Proceedings*, 2014, pp. 276–298.

[48] 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*, 2015, pp. 90–99.

[49] Daniel Fett, Ralf Küsters, and Guido Schmitz, "An expressive model for the web infrastructure: Definition and application to the browserid SSO system," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA*, 2014, pp. 673–688.

[50] 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.

[51] 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*, 2013, pp. 560–567.

[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, Proceedings, Part I*, 2018, pp. 470–490.