UPPRESSO: An Unlinkable Privacy-PREserving Single Sign-On System

***Abstract*—Single sign-on (SSO) services are widely provided in the Internet by identity providers (IdPs) as the identity management and authentication infrastructure. After being authenticated by the IdP, a user is allowed to log into relying parties (RPs) by submitting an *identity proof* (i.e., id token of OpenID Connect or SAML assertion). However, SSO introduces the potential leakage of user privacy, as (*a*) a curious IdP could track a user's all visits to any RP and (*b*) collusive RPs could link the user identity across different RPs, to learn the user's detailed activity profile. Existing privacy-preserving SSO solutions protect the users' activity profiles against either the curious IdP or the collusive RPs, but never prevent both of these threats.**

In this paper, we propose an SSO system, called *UPRESSO*, to protect a user's activity profile of RP visits from both the curious IdP and the collusive RPs. In each login process of UPRESSO, the IdP that is aware of all users' identities, computes the privacy-preserving pseudo-identity (PID_U) for a user, based on the user's identity and the pseudo-identity (PID_{RP}) of the visited RP. PID_{RP} bound along with PID_U in the identity proof, is transformed cooperatively by the user and the RP, and then the IdP does not know the visited RPs. The visited RP thats obtain a trapdoor from the transformation of PID_{RP}, is able to use this trapdoor to derive the user's unique account from PID_U, while a user's accounts are different across the RPs. The login process of UPRESSO adopts the communication pattern of OpenID Connect, a widely deployed SSO system. The analysis demonstrates that UPRESSO protects user privacy well, without any degradation on the security of OpenID Connect. We have implemented the prototype of UPRESSO and the experimental evaluation shows that UPRESSO is efficient and it takes only 208 ms for a user to log into an RP.

***Index Terms*—Single sign-on, security, privacy.**

I. Introduction

Single sign-on (SSO) systems, such as OAuth [1], OpenID Connect [2] and SAML [3], have been widely adopted as the identity management and authentication infrastructure in the Internet. SSO enables a website, called the *relying party* (RP), to delegate its user authentication to a trusted third party called the *identity provider* (IdP). Thus, a user can visit multiple RPs with only a single explicit authentication attempt on the IdP. With the help of SSO, a user no longer needs to remember multiple credentials for different RPs; instead, she maintains only the credential for the IdP, which will generate *identity proofs* for her visits to these RPs. As a result, SSO has been widely integrated with many application services. For example, we find that 80% of the Alexa Top-100 websites [4] support SSO, and the analysis on the Alexa Top-1M websites [5] identifies 6.30% with the SSO support. Meanwhile, many email and social network providers (such as Google, Facebook, Twitter, etc.) are serving the IdP roles.

The primary goal of SSO services is to implement secure user authentication [6], i.e., to ensure that an honest user can always log in to an honest RP under the correct account. To achieve this, an identity proof generated by the IdP should explicitly specify the authenticated user (i.e., *user identification*) and the RP to which the user attempts to log in (i.e., *receiver designation*). The identify proof should be received only by that RP and user but not by any other entities (i.e., *confidentiality*) and verified by the receiver (i.e., *integrity*).

However, various attacks exploit vulnerabilities in different SSO systems to break at least one of these requirements [7]–[14], where the adversaries attempt to either impersonate the victim user at an honest RP or log in to honest RPs under the adversary's identity. For example, Friendcaster was found to accept any received identity proof [7], [15] (i.e., a violation of receiver designation) so that a malicious RP could log in to Friendcaster as the victim user by replaying the identity proof received from the user to Friendcaster [14]. [9] reported that some RPs of Google ID SSO accepted user attributes that were not tied to the identity proof (i.e., a potential violation of integrity). As a result, a malicious user could insert arbitrary attributes (e.g., an email address) into the identity proof to impersonate another user at the RP.

Moreover, the adoption of SSO also raises several privacy concerns regarding online user tracking and profiling [16], [17]. User privacy leaks in all existing SSO protocols and implementations. Taking a widely used SSO protocol, OpenID Connect (OIDC), as an example, we explain its login process and the privacy leakage risk. As shown in Fig. 1, upon receiving a user login request (Step 1), the RP constructs an authentication request with its identity and redirects it to the IdP (Step 2). After authenticating the user, the IdP generates an identify proof containing the identities of the user and the RP in Step 4, which is forwarded to the RP by the user in Step 5. Finally, the RP verifies the identity proof and allows the user to log in. In such authentication sessions, any curious IdP or multiple collusive RPs can easily break users' privacy as follows.

- *IdP-based visit tracing*. The IdP knows the identities of the RP and user in a single authentication session in Step 2 and Step 3, respectively. As a result, a curious IdP could easily discover all the RPs that the victim user attempts to visit and profile her online activities.
- *RP-based identity linkage*. The RP learns user's identity from the identify proof. When the IdP generates identity proofs for a user, if a same user identifier is used in identity proofs generated for different RPs, which is the

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	${PID_{RP}}^{ID_U}$ (✓)	${ID_{RP}}^{N_U * N_{RP}}$ (✓)	${PID_U}^{t}$ (✓)

case of several widely deployed SSO systems [6], [18], malicious RPs can collude to not only link the user's login activities at different RPs for online tracking but also correlate her attributes across multiple RPs [16].

Large IdPs, especially social IdPs such as Google and Facebook, are known to be interested in collecting users' 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. On the other hand, in today's Internet, many service providers host a variety of web services, which provides them an advantaged position to link a user's multiple logins at different RPs. Through an internal information integration, rich information can be obtained from the SSO data for user profiling.

While the privacy problems in SSO have been widely recognized [16], [17], only a few solutions were proposed to protect user privacy [6], [21]. Among them, Pairwise Pseudonymous Identifier (PPID) [2], [22] is a most straightforward and commonly adopted solution to defend against RP-based identity linkage. It requires the IdP to create different identifiers for the user when she logs in to different RPs, so that the pairwise pseudonymous identifiers of the same user cannot be used to link user logins at different RPs no matter they collude or not. 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 attacks, as the IdP still knows which RP the user visits.

To the best of our knowledge, only two schemes (i.e., BrowserID [18] and SPRESSO [6]) have been proposed so far to defend against 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) to a public key. With the corresponding private key, the user binds the receiving RP's identifier to the identity proof and sends it to that RP. In this way, the IdP does not need to know the RP's identity in the generation of identify proofs. SPRESSO requires the RP to create a pseudonymous identifier at each login for the IdP to generate the identity proof and therefore hides its real identifier from the IdP. The RP employs a third-party entity (named *forwarder*), which works as a proxy to receive the identity proof from the IdP and relay the identity proof to the corresponding RP. While the RPs' identifiers are protected from the IdP in both schemes, the IdP has to know the unique identifier (e.g., email address) of the user and includes it in the identity proof so that a same user can be recognized by a same RP in her multiple logins. As a result, both schemes are vulnerable to RP-based linkage attacks.

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 multiple RPs, while $\mathcal{F}_{PID_U \mapsto Account}$ needs to ensure that $Account$ are also 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 a certain 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 a certain 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

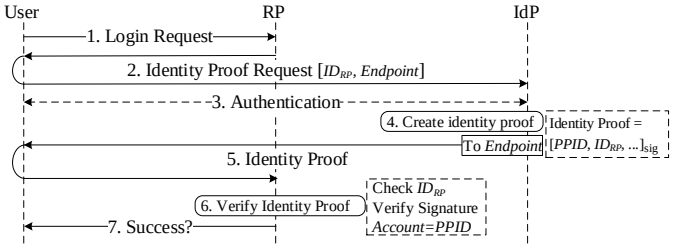

Fig. 1: The implicit protocol flow of OIDC.

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 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 **??** 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

UPPRESSO 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.

UPPRESSO is compatible to all the three flows. For brevity, we will present the application of UPPRESSO 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 dy-

namic 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 UPPRESSO, we slightly modify dynamic registration to register PID_{RP} at IdP.

B. Discrete Logarithm Problem

Discrete logarithm problem is adopted in UPPRESSO 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 Identity-Transformation Approach of UPPRESSO

In this section, we analyze the challenges to design secure privacy-preserving SSO systems, and provide an overview of the identity-transformation approach proposed in UPPRESSO.

A. Security Requirements of SSO

We summarize the basic requirements of SSO systems based on existing theoretical analyses [8], [26], [27] and practical attacks [9]–[14], [28]–[34]. These requirements enable an SSO system to provide qualified authentication services for RPs, through identity proofs.

- **User identification.** When a user logs into a certain RP for multiple times by submitting identity proofs, the RP extracts the identical user identifier from these identity proofs, to provide personalized services for this user.
- **RP designation.** The designated receiver (or RP) is specified in an identity proof, so that this identity proof is accepted by the visited RP only.
- **Integrity and confidentiality.** Only the IdP is trusted to generate identity proofs, RPs do not accept an identity proof with any modification or a forged one. Meanwhile, a valid identity proof is transmitted only to the user and the designated RP.

First of all, user identification is necessary for common SSO systems to help the RPs to receive the user's identifier, except the anonymous services. Any violation of these requirements [9]–[14], [28]–[34] result in *impersonation attacks* (i.e., the adversaries log into an honest RP as a victim user) or *identity injection attacks* (i.e., a victim user logs into an honest RP under some attacker's identity). If the designated RP is not well specified or verified in identity proofs, the adversaries could deceive an RP to accept the identity proofs generated

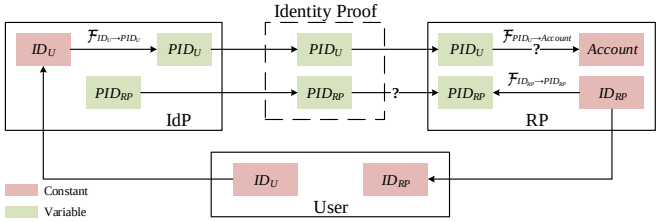

Fig. 2: Identity transformations in privacy-preserving SSO.

for other RPs, so that the adversaries would (*a*) impersonate some victim user, by colluding with a malicious RP to obtain such an identity proof and submitting it to the RP, or (*b*) inject such identity proofs in the communications between the victim user and some RP. Impersonation and identity injection attacks would be successfully launched, if the attackers could arbitrarily modify the user identifiers in identity proofs. Or, the adversaries could impersonate the victim user by submitting any leaked identity proof to the RP [7], [8], [11], if confidentiality is not ensured.

B. The Identity Dilemma of Privacy-Preserving SSO

As mentioned in Section I, the user privacy is leaked through IdP-based visit tracing or RP-based identity linkage. Thus, a privacy-preserving SSO system shall prevent these two kinds of user privacy leakage, while satisfying four basic security requirements of SSO services. Since integrity and confidentiality are ensured by digital signatures of the IdP and TLS communications in the system [2], we focus only user identification and RP designation in the remainder.

The pseudo-identifiers of users and RPs (denoted as PID_U and PID_{RP}, respectively) are introduced, as we consider two privacy threats. To prevent IdP-based visit tracing, the IdP shall be aware of PID_{RP} at most and then ID_{RP} is not bound in identity proofs; otherwise, if ID_{RP} is disclosed to the IdP, it enable IdP-based visit tracing because the user is authenticated by the IdP and ID_U is automatically disclosed. In order to prevent RP-based identity linkage, only PID_U but not ID_U is enclosed in the identity proofs; otherwise, collusive RPs will link a user's login activities based on ID_U in received identity proofs. Finally, only the pseudo-identifiers of users and RPs (i.e., at most PID_U and PID_{RP}, but not ID_U or ID_{RP}) are bound by the IdP in identity proofs.

Then, when we consider both the requirements of security and privacy, it brings the identity dilemma as follows: when the IdP is unaware of the visited RP, it (*a*) binds PID_{RP} in each identity proof that still enables an RP to verify the specified receiver, to ensure RP designation, and (*b*) generates PID_Us in the identity proofs for a user in multiple logins at an RP, which enable the RP to extract the identical user identifier (i.e., $Account$), to ensure user identification. Moreover, for a certain user, PID_{RP}s are different in the multiple logins at an RP, to prevent IdP-based visit tracing, PID_Us are different when the user visits different RPs, to prevent RP-based identity linkage, and $Account$s are RP-specific (i.e., distinct for different RPs).

We explain the relationship between the identifiers, the pseudo-identifiers and the identity proof in Figure 2. In every login process, the user knows both ID_U and ID_{RP}. The IdP is aware of ID_U after it authenticates the user. Variable PID_{RP} and PID_U are generated for the identity proof, satisfying that PID_{RP} enables the verification of RP designation and PID_U enables the derivation of identical $Account$. It is worthy noting that when user privacy is not well considered, this relationship is simplified. For example, in commonly-used OIDC [2], ID_U is directly assigned to PID_U, or constant RP-specific PID_U is adopted.

C. The Principles of UPPRESSO

As demonstrated in Figure 2, to design secure privacy-preserving SSO solutions (i.e., solve the identity dilemma of privacy-preserving SSO), we need to find out satisfying functions to compute PID_U, PID_{RP} and $Account$, which are denoted as $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, and $\mathcal{F}_{PID_U \mapsto Account}$, respectively.

First of all, UPPRESSO adopts the communication pattern of OpenID Connect, and then *integrity* and *confidentiality* of identity proofs are also implemented by digital signatures of the IdP and TLS communications in the system.

In order to prevent ***IdP-based visit tracing***, the IdP of UPPRESSO is aware of PID_{RP} but not ID_{RP}, and it is computationally infeasible for the IdP to derive (any information about) ID_{RP} from PID_{RP} and PID_U; to prevent ***RP-based identity linkage***, an RP accepts identity proofs including PID_U but not ID_U, to derive $Account$ distinct among RPs, and it is computationally infeasible for the RP to derive (any information about) ID_U from PID_U, PID_{RP} and $Account$.

Finally, we attempt to achieve ***user identification*** and ***RP designation***, by the comprehensive design of identity transformations (i.e., $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, and $\mathcal{F}_{PID_U \mapsto Account}$) as follows.

- $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$ work together to ensure user identification, i.e., derive identical $Account$ for a user in multiple logins at an RP.
- $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ ensures RP designation, i.e., PID_{RP} is enough to specify a certain RP.

UPPRESSO designs three pseudo-identifiers (i.e., PID_U, PID_{RP}, and $Account$) in a *dynamical* and *comprehensive* way, based on *static* ID_U and ID_{RP}. That is, for a certain user, in each login process at an RP, PID_U and PID_{RP} vary, to satisfying the requirements of privacy; but PID_U and PID_{RP} vary synchronously so that identical $Account$s are derived. In particular, the user and the RP negotiate PID_{RP} based on ID_{RP} in each login process, and then PID_{RP} is transmitted to the IdP. The IdP generates PID_U based on ID_U and also PID_{RP}. Moreover, the RP obtains a private trapdoor in the negotiation, which is used to compute $Account$ from PID_U.

Trapdoor user identification. Existing SSO solutions always depend on constant ID_U in all identity proofs, or RP-specific PID_U that keeps constant for an RP, to identify an account in the RP. UPPRESSO introduces trapdoor user identification, while an RP holds a trapdoor to derive the identical $Accout$ from dynamic PID_Us in identity proofs. Intuitively, this trapdoor also works in the generations of PID_{RP} and PID_U directly or indirectly.

Transformed RP designation. To bind the dynamic PID_{RP} in identity proofs signed by the IdP, the user firstly cooperates with the RP to generate PID_{RP} based on ID_{RP} and then registers this transformed RP identifier (i.e., PID_{RP}) in the IdP. The identity transformation of RP is completely kept secret to the IdP. Then, the one-time PID_{RP} is bound with PID_U in the identity proof. PID_{RP} is computed based on ID_{RP} and the trapdoor, so that the RP holding the trapdoor is able to verify the specified receiver of identity proofs with transformed PID_{RP} but not ID_{RP}.

D. Existing SSO Solutions

We use the framework of identity transformation in Figure 2 to explain the designs of different SSO solutions. First of all, when $PID_U = ID_U$ and $PID_{RP} = ID_{RP}$, this framework describes commonly-adopted OpenID Connect. It is also applicable to explain the approaches of existing privacy-preserving SSO solutions, including PPID [2], BrowserID [18] and SPRESSO [6].

PPID prevents only RP-based identity linkage but not IdP-based visit tracing, where $PID_{RP} = ID_{RP}$. The IdP by itself maintains the deterministic mapping from ID_U to PID_U distinct for different RPs, and $Account = PID_U$.

SPRESSO and BrowserID prevent only IdP-based visit tracing but not RP-based identity linkage. In each login process of SPRESSO, the RP generates PID_{RP} by encrypting ID_{RP} padded with a nonce (i.e., $PID_{RP} = Enc(RP_ID||nonce)$), and PID_{RP} is forwarded by the user to the IdP, so that ID_{RP} is kept unknown to the IdP and PID_{RP} enables the verification of RP designation. The IdP of SPRESSO provides constant ID_U in identity proofs for a user's multiple logins no matter which RP the user is visiting, and $Account = ID_U$.

The IdP of BrowserID signs identity proofs binding only ID_U but no PID_{RP} or ID_{RP} (i.e., $PID_{RP} = \perp$), and IdP-based visit tracing is prevented. On the other hand, in order to ensure RP designation with null PID_{RP}, ID_{RP} is bound in the *subsidiary* identity proof signed by the user instead of the IdP. The identity proof signed by the IdP authorizes the user to sign subsidiary identity proofs, and subsidiary identity proofs are kept confidential to the IdP. ID_U is bound in each identity proof of BrowserID, and $Account = ID_U$.

As analyzed above, none of these solutions provides protections against both IdP-based access tracing and RP-based identity linkage. One of the reasons is that they do not explicitly clarify three pseudo-identifiers (i.e., PID_U, PID_{RP}, and $Account$) and then design $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, and $\mathcal{F}_{PID_U \mapsto Account}$ comprehensively.

IV. Threat Model and Assumption

UPPRESS follows the same service mode as traditional SSO systems (e.g., SAML and OIDC), and it consists of an IdP, a

number of RPs and users. The IdP provides user authentication services for all RPs. In this section, we describe the threat model and assumptions of these entities in UPPRESSO.

A. Threat Model

The IdP is assumed to be curious-but-honest, while some malicious users and RPs could be completely compromised by adversaries. Malicious users and RPs behave arbitrarily and may collude with each other, to break the guarantees of security and privacy for other correct users.

Curious-but-honest IdP. The IdP strictly follows its specification, but is curious about the user privacy, especially the login activities at different RPs. The IdP is well-protected and never leak any sensitive information. For example, the private key to sign identity proofs and RP certificates (see Section V-B for details) is held by the IdP always, so adversaries cannot forge an identity proof or RP certificate. The honest IdP follows the designed protocols to process the requests from users and RPs, and never colludes with any others (e.g., malicious RPs or users). For example, IdP ensures the uniqueness of ID_{RP} and ID_U when an RP or a user registers, and computes the pseudo-identifier as the UPPRESSO protocol specifies.

However, the curious IdP attempts to break the user's privacy without violating the protocol. For example, the curious IdP may store all messages received and sent, and tries to conduct attacks of visit tracing or identity linkage by analyzing the relationship among ID_U, ID_{RP}, PID_U and PID_{RP}.

Malicious users. The adversaries could compromise a set of users, by stealing the users' credentials [35], [36] or registering sybil users at the IdP and RPs directly. These malicious users aim to break the security of UPPRESSO. That is, they attempt to impersonate an uncompromised user at some correct RP, or trick a victim user to log into an correct RP under the identity of a compromised user. For example, the malicious users may modify, inject, drop and replay any messages, and deviate arbitrarily from the specification when processing ID_{RP}, PID_{RP} and identity proofs.

Malicious RPs. The adversary could compromised a set of RPs, by registering an RP at the IdP or exploiting software vulneraries to intrude RPs. These malicious RPs aim to break the security and privacy of correct users, and behave arbitrarily. For example, malicious RPs attempt to obtain an identity proof valid for another RP, to allow some user to log into this target RP: a malicious RP manipulates PID_{RP} when a user is logging in, to receive an identity proof that will be accepted by the target RP verifying PID_{RP} but not ID_{RP}. Or, the malicious RPs may collude to perform RP-based identity linkage to break user privacy. For example, the RPs may attempt to derive ID_U from PID_U and $Account$ by manipulating PID_{RP} to the IdP, to link the user's multiple logins at different RPs.

Collusive users and RPs. Malicious users and RPs may collude and behave arbitrarily, attempting to break the guarantees of security and privacy. For example, collusive malicious users and RPs may conduct impersonation or identity injection attacks, by manipulating PID_U and PID_{RP} in an identity proof.

B. Assumption

We assume that the user agent deployed at an honest user is correctly implemented, and transmit messages to the destination correctly. TLS is also correctly implemented at the IdP, (correct) RPs and users, which ensures confidentiality and integrity of the communications among correct entities.

The cryptographic algorithms (such as RSA and SHA-256) and building blocks (such as random number generators and the discrete logarithm problem) used in UPPRESSO, are assumed to be secure and correctly implemented.

UPPRESSO considers the RP-based identity linkage based on the user identifiers at RPs, so other RP-based identity linkage based on the distinctive user attributes at RPs (e.g., telephone number, address and driver license) are out of the scope of this paper. We prevent the IdP-based visit tracing based on the SSO protocols, and we do not consider other network attacks (e.g., the network traffic analysis to correlate a user's logins at different RPs).

V. The Design of UPPRESSO

This section provides the design of UPPRESSO. We firstly present the detailed functions of identifier transformation, for trapdoor user identification and transformed RP designation. Then, we provide an overview of UPPRESSO and describe the protocols. Finally, we discuss the compatibility of UPPRESSO with OIDC.

A. Functions of Identifier Transformation

As mentioned in Section III, the functions of identifier transformation are essential for privacy-preserving SSO systems. In UPPRESSO, $\mathcal{F}_{ID_{RP}\mapsto PID_{RP}}$, $\mathcal{F}_{ID_U\mapsto PID_U}$ and $\mathcal{F}_{PID_U\mapsto Account}$ are constructed based on the discrete logarithm with public parameters p, q, and g, where p is a large prime defining the finite field $GF(p)$, q is a prime factor of $(p-1)$, and g is a generator of order q in $GF(p)$.

The IdP assigns a unique random number as ID_U ($1 < ID_U < q$) to each user, and a unique ID_{RP} at the RP's initial registration. ID_{RP} is computed as follows, where r is a random number ($1 < r < q$) generated by the IdP.

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

In each login, the user and the visited RP negotiate PID_{RP} as follows. The RP chooses a random number N_{RP} ($1 < N_{RP} < q$), and the user chooses another random number N_U ($1 < N_U < q$). Then, they cooperatively compute PID_{RP} as in Equation 2.

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

$\mathcal{F}_{ID_{RP}\mapsto PID_{RP}}$ satisfies the following requirements. First, it is computationally infeasible for the IdP to derive ID_{RP} from PID_{RP} due to the discrete logarithm problem. N_U and N_{RP} serves as nonces to ensure that (*a*) PID_{RP} is valid only

for this login as well as the identity proof, and (*b*) the IdP cannot correlate multiple PID_{RP}s for a same RP. Finally, the cooperation by the user and the RP prevents a single malicious entity from manipulating PID_{RP} in the identity proof.

On receiving a request of ID_U and PID_{RP} from an authenticated user, the IdP calculates PID_U as Equation 3, and binds it in the identity proof.

$$\mathcal{F}_{ID_U \mapsto PID_U} : PID_U = {PID_{RP}}^{ID_U} \bmod p \quad (3)$$

We have $PID_U = {ID_{RP}}^{N_U N_{RP} ID_U} = g^{r N_U N_{RP} ID_U} \bmod p$ from Equations 1, 2 and 3. The discrete logarithm problem ensures that the RP cannot derive ID_U from PID_U. Moreover, provided that r is kept secret to the RP, collusive RPs cannot link a user's PID_Us at different RPs.

Finally, the RP derives $Account$ for the user as Equation 4. Here, we define $T = (N_U N_{RP})^{-1} \bmod q$ as the trapdoor at the RP. As q is a prime number and $1 < N_U, N_{RP} < q$, q is coprime to $N_U N_{RP}$, and then T that satisfies $T(N_U N_{RP}) = 1 \bmod q$ always exists.

$$\mathcal{F}_{PID_U \mapsto Account} : Account = {PID_U}^{T} \bmod p \quad (4)$$

We have $Account = {ID_{RP}}^{ID_U} \bmod p$ as Equation 5 shows, from Equations and 2, 3, and 4. So for a user's multiple logins at an RP, the RP extracts an identical $Acount$.

$$\begin{aligned} Account &= {PID_U}^{T} = ({PID_{RP}}^{ID_U})^{(N_U N_{RP})^{-1} \bmod q} \\ &= {ID_{RP}}^{ID_U N_U N_{RP} T \bmod q} = {ID_{RP}}^{ID_U} \bmod p \end{aligned} \quad (5)$$

$\mathcal{F}_{PID_U \mapsto Account}$ satisfies the following requirements. Similar to the analysis of PID_U, the RP cannot derive ID_U from PID_U, and collusive RPs cannot link a user's PID_Us at different RPs.

Trapdoor user identification. In a user's multiple logins, the RP expresses different PID_Us and have corresponding Ts, so that always derives the identical $Account$. The comprehensive design of identifier transformation functions prevents collusive RPs from linking a user's PID_Us and $Account$s at different RPs, and therefore prevents RP-based identity linkage.

Transformed RP designation. 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 identity proof, which designates this identity proof to PID_{RP}. Therefore, the PID_{RP} is designated to ID_{RP}. Finally, the transformed RP designation is provided through the two-step designations. 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.

B. UPPRESSO Overview

In addition to the functions of identifier transformation, UPPRESSO needs to introduce more steps at the user to facilitate these identifier transformations. It is worthy noting that, in order to protect user privacy against both the IdP and the visited RP, these steps have to be conducted at the user. Firstly, because the IdP is unaware of the visited RP and also the RP's endpoint to receive the identity proof, this endpoint shall be queried by the user from the trusted IdP indirectly. In UPPRESSO this is implemented as an RP certificate signed by the IdP, which is composed of ID_{RP}, the RP's endpoint and other supplementary information. The user handles the endpoint by itself. Secondly, after the negotiation of PID_{RP}, it is registered at the IdP by the authenticated user. This cannot be finished by the RP; otherwise, the IdP will correlate PID_{RP} and ID_{RP}.

TABLE II: The notations used in UPPRESSO.

Notation	Definition	Attribute
p	A large prime.	Long-term
q	A large prime factor of $(p-1)$.	Long-term
g	A generator of order q in $GF(p)$.	Long-term
SK, PK	The private/public key of IdP.	Long-term
ID_{RP}	RP's unique identifier.	Long-term
r	value for $ID_{RP} = g^r \bmod p$.	Secret, Long-term
$Cert_{RP}$	An RP certificate.	Long-term
ID_U	User's unique identifier.	Long-term
$Account$	User's identifier at an RP.	Long-term
PID_{RP}	RP's privacy-preserving identifier.	One-time
PID_U	User'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}, $(ID_{RP})^{N_{RP}} \bmod p$.	One-time
T	A trapdoor, $T = (N_U N_{RP})^{-1} \bmod q$.	One-time

UPPRESSO runs with four procedures, including system initialization, RP initial registration, user registration and SSO login. The system initialization is conducted only once by the IdP to establish the system. The RP initial registration is launched by each RP to obtain the necessary configurations (a unique identifier ID_{RP} and its RP certificate $Cert_{RP}$) from the IdP before it provides services for users, and each RP launches this procedure only once. The user registration is launched only once by each user to set up a unique user identifier ID_U and the corresponding credential. Finally, the SSO login is launched when a user attempts to log in an RP, and it is designed based on the functions of identifier transformation.

The procedure for user registration is the same as that in typical SSO systems. Therefore, we focus on the procedures of system initialization, RP initial registration and SSO login. For clarity, we list the notations in Table II.

System initialization. The IdP generates a large prime p, a prime factor q of $p-1$ and a generator g of order q as the parameters for the discrete logarithm problem. The IdP generates one key pair (SK, PK) to sign identity proofs and RP certificates. The lengths of p, q and (SK, PK) shall satisfy the requirements of security strength.

The IdP keeps SK as secrets, while p, q, g and PK are public parameters.

RP initial registration. An RP registers itself at the IdP to request ID_{RP} and $Cert_{RP}$ 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 $Cert_{RP} = [ID_{RP}, endpoint]_{SK}$, where $[\cdot]_{SK}$ means a message signed using SK.
3) The RP verifies $Cert_{RP}$ using PK, and then accepts ID_{RP} and $Cert_{RP}$.

Note that, ID_{RP} cannot be chosen by the RP, and it must be chosen by the IdP AND Here, the r is never leaked. ID_U can be selected by the user or the IdP.

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 [37], 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. UPPRESSO Protocol flow

In UPPRESSO, The protocol, shown in Figure 3, 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 $Y_{RP} = {ID_{RP}}^{N_{RP}} \bmod p$ (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 $PID_{RP} = {Y_{RP}}^{N_U} \bmod p$ (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 $T = (N_U N_{RP})^{-1} \bmod q$ (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 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 refreshing 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 the 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 necessary (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 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

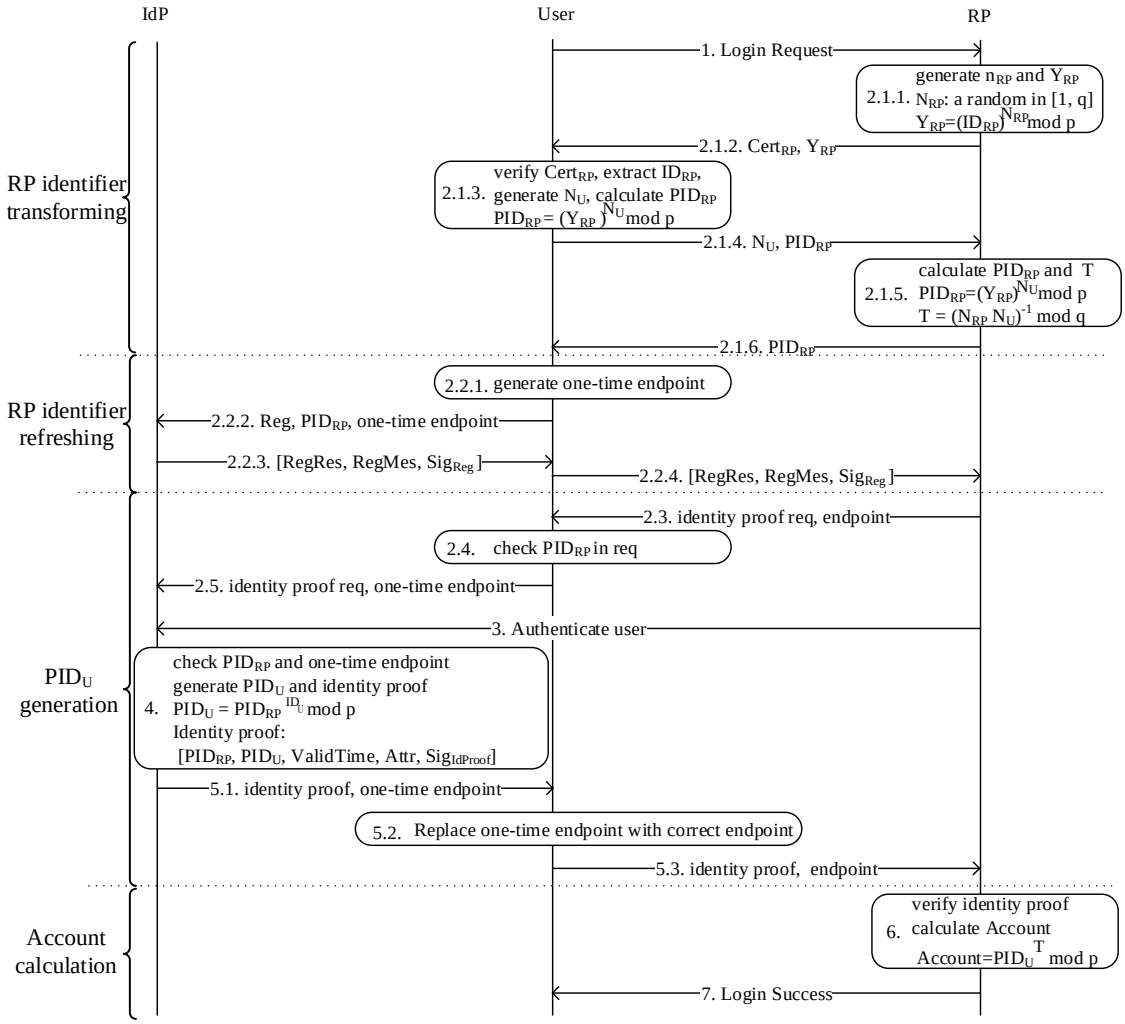

Fig. 3: Process for each user login.

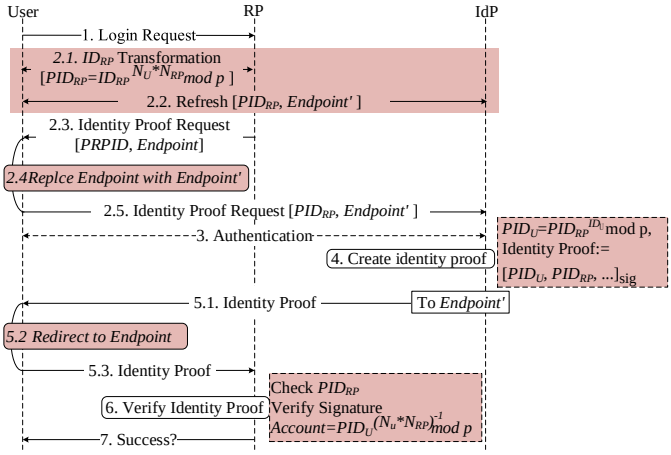

Fig. 4: UPPRESSO compatibility with OIDC.

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

UPPRESSO 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 UPPRESSO 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.

UPPRESSO 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 UPPRESSO, 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, UPPRESSO 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.

The above modifications could be completed automatically for each login, without affecting other communication pattern. The user automatically invokes the JavaScript functions to complete RP identifier transforming, one-time endpoint generating/replacing and RP identifier refreshing for each login. While, the RP server and IdP server provide the corresponding web service to complete the processing automatically. The protocol of RP identifier transformation is based Diffie-Hellman key exchange [37], while N_U is provided to RP for computing the trapdoor. The protocol of RP identifier refreshing is based on the dynamic registration in OIDC, while 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

Here, we analyze the security and privacy of UPPRESSO.

A. Security

We prove that the basic requirements of SSO system, i.e., user identification, RP designation, integrity and confidentiality, are still satisfied in UPPRESSO with the modifications on OIDC, whose security has been formally analyzed in [27]. In the following, we analyze the affects of the modifications listed in Section V-D, respectively.

The first modification may affect the RP designation, as the identity proof is bound with PID_{RP} instead of ID_{RP}. However, in UPPRESSO, PID_{RP} provides the same binding as ID_{RP}, which is achieved by the **transformed RP 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 UPPRESSO, the PID_{RP} is also unique 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:

- 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 collusive RPs and users cannot make one specified (e.g., duplicated) PID_{RP} be generated in the negotiation with a correct user or RP. For example, the malicious RP may attempt to misuse the received identity proof at another RP by acting as a user to negotiate a same PID_{RP} with this target RP. However, this is prevented and the adversary cannot control the values of PID_{RP} due to the cooperated function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and the protocol in RP identifier transforming. The RP chooses N_{RP} before obtaining N_U and cannot change N_{RP} after receiving N_U, while the user calculates PID_{RP} by ${Y_{RP}}^{N_u} \bmod p$ and cannot to derive N_{RP} from the Y_{RP}. As PID_{RP} is calculated based on two random numbers N_U and N_{RP}, a same PID_{RP} may indeed be generated in two negotiations, however the probability is $1/q$ ($\leqslant 2^{-255}$ when $L = 256$), which is negligible.

The second modification makes the user obtain one-time endpoint instead of RP's correct endpoint from the IdP, and may affect the transmission of identity proof between the user and RP. In UPPRESSO, $Cert_{RP}$ is introduced to ensure that the correct user sends the identity proof only to the correct endpoint of the designated RP. 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 UPPRESSO, 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 UPPRESSO 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 identical identifier for a user ID_U at an RP. In UPPRESSO, the correct RP computes an identical value $Accout = {PID_U}^T = {ID_{RP}}^{ID_U} \bmod p$ 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 UPPRESSO 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 RP designation. Therefore, UPPRESSO

provides the secure SSO service.

B. Privacy

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

IdP-based visit tracing prevention. UPPRESSO prevents the IdP-based visit 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. UPPRESSO 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 $PID_{RP} = {ID_{RP}}^{N_U N_{RP} \bmod q} \bmod p$. 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. UPPRESSO 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.

Any RP cannot derive ID_U from PID_U and Account in one login directly, 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 RP and independent with ID_U.

Any RP cannot directly derive ID_U from PID_Us and $Accounts$ 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}=$PID_{U2} * Account^{N_{U1} * N_{RP1} - N_{U2} * N_{RP2}} \ mod \ p$, where N_{U2}, N_{RP2}, N_{U2} and N_{RP2} are values known to the RP and independent with ID_U.

The collusive RPs cannot directly associate a user's $Accounts$ and PID_Us. The collusive RPs may attempt to link a user's accounts by checking whether the equality $Account_2$=$(Account_1)^{r_2/r_1} \ mod \ p$ holds for $Account_1$ at an RP $ID\_RP1 = g^{r_1} \ mod \ p$ and $Account_2$ at another RP $ID\_RP2 = g^{r_2} \ mod \ p$. 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.

Any malicious RP cannot derive ID_U and collusive RPs cannot correlate PID_Us (Accounts), by manipulating N_{RP}s. 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 collusive RPs and users cannot associate the victim user. The RPs may collude with the 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_{Δ} satisfying both $Account_{m1}/Account_{v1} = {ID_{RP1}}^{ID_{\Delta}} \ mod \ p$ and $Account_{m2}/Account_{v2} = {ID_{RP2}}^{ID_{\Delta}} \ mod \ p$. 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_{Δ}) for this association.

The malicious RPs and users cannot associate the victim user by manipulating N_{RP}s or N_Us. The malicious RPs may manipulate N_{RP}s while the colluded users may manipulate N_Us, attempting to link 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 UPPRESSO, and evaluated its performance by comparing with the original OIDC and SPRESSO which only prevents IdP-based access tracing.

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 [38]. UPPRESSO 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 [39], an open-source OIDC Java implementation certificated by the OpenID Foundation [40]. 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 [41], an efficient JavaScript cryptographic library. This chrome extension requires permissions *chrome.tabs* and *chrome.windows* to obtain RP's URL from the browser's tab, and *chrome.webRequest* to intercept, block, modify requests to the IdP/RP [42]. 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, UPPRESSO modifies the IdP and RP, and sets `chrome-extension://chrome-id` (`chrome-id` is uniquely assigned by Google) in HTTPS header `Access-Control-Allow-Origin` of the IdP's and RP's responses.

We provide a Java SDK for RPs to integrate UPPRESSO. 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

Environment. The evaluation was performed on 3 machines, one (3.4GHz CPU, 8GB RAM, 500GB SSD, Windows 10) as IdP, one (3.1GHz CPU, 8GB RAM, 128GB SSD, Windows 10) as an RP, and the last one (2.9GHz CPU, 8GB RAM, 128GB SSD, Windows 10) as a user. The user agent is Chrome v75.0.3770.100. And the machines are connected by an isolated 1Gbps network.

Setting. We compare UPPRESSO with MITREid Connect [39] and SPRESSO [6], where MITREid Connect provides open-source Java implementations [39] of IdP and RP's SDK, and SPRESSO provides the JavaScript implementations based on node.js for all entities [6]. We implemented an Java RP based on Spring Boot framework for UPPRESSO and MITREid Connect respectively, by integrating the corresponding SDK. The RPs in all the three schemes provide the same function, i.e., extracting the user's account from the identity proof. We have measured the time for a user's login at an RP, and calculated the average values of 1000 measurements. For better analysis, we divide a login into 4 phases according to the lifecycle of identity proof: **Identity proof requesting** (Steps 1-2.5 in Figure 3), RP (and user) constructing and transmitting the request to IdP; **Identity proof generation** (Step 4 in Figure 3), IdP generating identity proof (no user authentication); **Identity proof extraction** (Steps 5.1-5.3 in Figure 3), RP server extracts the identity proof from the IdP; and **Identity proof verification** (Steps 6 in Figure 3), RP verifying and parsing the identity proof.

Results. The evaluation results are provided in Figure 5. The overall processing times are 113 ms, 308 ms and 208 ms for MITREid Connect, SPRESSO and UPPRESSO, respectively. The details are as follows.

In the requesting, UPPRESSO requires the user and RP to perform 1 and 2 modular exponentiations respectively for PID_{RP} transforming and cooperatively complete PID_{RP} refreshing at the IdP, which totally needs 98 ms; SPRESSO needs 19 ms for the RP to obtain IdP's public key and encrypt its domain; while MITREid Connect only needs 10 ms.

In the generation, UPPRESSO needs an extra 6 ms for computing PID_U, compared to MITREid Connect which only needs 32 ms. SPRESSO requires 71 ms, as it implements the IdP based on node.js and therefore can only adopt a JavaScript cryptographic library, while others adopt a more efficient Java library. As the processing in SPRESSO and MITREid Connect

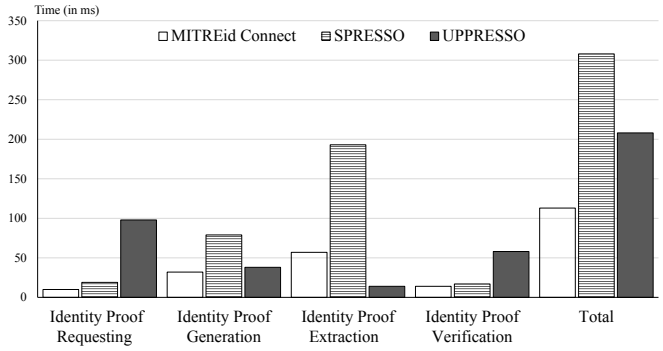

Fig. 5: The Evaluation.

is the same, the processing time in SPRESSO may be reduced to 32 ms. And, then the overall time in SPRESSO will be 269 ms, still larger than 208 ms in UPPRESSO.

In the extraction, UPPRESSO only needs 14 ms where the Chrome extension relays the identity proof to RP server directly for the extraction. MITREid Connect requires the IdP to send the identity proof to RP's web page which then sends the proof to RP server through a JavaScript function, and needs 57 ms. SPRESSO needs the longest time (193 ms) due to a complicated processing at the user's browser, which needs the browser to obtain identity proof from IdP, download the JavaScript program from a trusted entity (named FWD), execute the program to decrypt RP's endpoint, send this endpoint to RP's web page who finally transmits the proof to RP server. In the evaluation, FWD and IdP are deployed in one machine, which doesn't introduce performance degradation based on the observation.

In the verification, UPPRESSO needs an extra calculation for $Account$, which then requires 58 ms, compared to 14 ms in MITREid Connect and 17 ms in SPRESSO.

VIII. Discussions and Future works

We first discuss the scalability, security against DoS attack and OIDC authorization code flow support in UPPRESSO, and then present the extensions for UPPRESSO.

Scalability. The adversary cannot exhaust ID_{RP} and PID_{RP}. For ID_{RP}, it is generated only in RP's initial registration. For PID_{RP}, in practice, we only need to ensure all PID_{RP}s are different among the unexpired identity proof (the number denoted as n). We assume that IdP doesn't perform the uniqueness check, and then calculate the probability that at least two PID_{RP}s are equal in these n ones. The probability is $1-\prod_{i=0}^{n-1}(1-i/q)$ which increases with n. For an IdP with throughput $2*10^8$ req/s and valid period of identity proof set as 5 minutes, n is less than 2^{36}, then the probability is less than 2^{-183} for 256-bit q. Moreover, as this probability is negligible, the uniqueness check of PID_{RP}, i.e., the RP identifier refreshing, could be removed in the SSO login process, and this optimization can be adopted when this negligible probability is acceptable by the users and RPs.

Security against DoS attack. The adversary may attempt to perform DoS attack on the IdP and RP. For example, the adversary may act as a user to invoke the RP identifier refreshing (Step 2.2.3) and identity proof generation (Step 4) at the IdP, which requires the IdP to perform two signature generations and one modular exponentiation. However, as the user has already been authenticated at the IdP, the IdP cloud identify the malicious users based on audit, in addition to the existing DoS mitigation schemes. The adversary may act as a user requesting to log into an RP, and make the RP perform two modular exponentiations. The RP could previously computed a set of Y_{RP}s to mitigate this attack.

OIDC authorization code flow support. The privacy-preserving functions $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ and $\mathcal{F}_{PID_U \mapsto Account}$ can be integrated into OIDC authorization code flow directly, therefore RP-based identity linkage and IdP-based access tracing are still prevented during the construction and parsing of identity proof. The only privacy leakage is introduced by the transmission, as RP servers obtain the identity proof directly from the IdP in this flow, which allows the IdP to obtain RP's network information (e.g., IP address). UPPRESSO could integrate anonymous networks (e.g., Tor) to prevent this leakage.

Platform independent. Our current implementation only requires the user to install a Chrome extension and doesn't need to store any persistent data at the user's machine. Moreover, the implementation could be further extended to remove the Chrome extension, whose JavaScript program is then fetched from the honest IdP. The processing is similar as SPRESSO. That is, 1) the RP's window (window A) opens a new iframe (window B) to visit the RP's web page, while the RP's web page redirects window B to the IdP; 2) window B downloads the JavaScript program from IdP and performs the processing in Steps 2.1.3, 2.2.1, 2.4 and 5.2; 3) then postMessages are adopted to exchange messages between window A and B for Steps 2.1.4, 2.1.6, 2.2.4, 2.3 and 5.3. The opener handle of window B is preserved (i.e., window A) for the postMessage, as window A opens window B with a web page from the RP; and window B is redirected to the IdP with *noreferrer* attribute set, to prevent the browser from sending RP's URL in the Referrer header to the IdP.

Malicious IdP mitigation. The IdP is assumed to assign unique ID_{RP} in $Cert_{RP}$ for each RP and generate the correct PID_U for each login. The malicious IdP may attempt to provide incorrect ID_{RP} and PID_U, which could be prevented by integrating certificate transparency [43] and user's identifier check [6]. With certificate transparency [43], the monitors checks the uniqueness of ID_{RP} among all the certificates stored in the log server. To prevent the malicious IdP from injecting a incorrect PID_U, the correct user could provide a nickname to the correct RP for an extra check as in SPRESSO [6].

IX. Related Works

Various SSO protocols have been proposed, such as, OIDC, OAuth 2.0, SAML, Central Authentication Service (CAS) [44]

and Kerberos [45]. These protocols are widely adopted in Google, Facebook, Shibboleth project [46], Java applications and etc. And, plenty of works have been conducted on security analysis and privacy protection for SSO systems.

A. Security analysis of SSO systems.

Analysis on SSO implementations. Various vulnerabilities were found in SSO implementations, and then exploited for impersonation and identity injection attacks by breaking the confidentiality, integrity or designation of identity proof. Wang et al. [9] analyzed the SSO implementations of Google and Facebook from the view of the browser relayed traffic, and found the adversary could manipulate the traffic to steal the identity proof. The browser's vulnerabilities could also be exploited by malicious RPs to break the confidentiality of identity proof [28]. The integrity has been tampered with in SAML, OAuth and OIDC systems [9]–[13], [30], [31], due to various vulnerabilities, such as XML Signature wrapping (XSW) [13], RP's incomplete verification [10]–[12] and etc. And, a dedicated, bidirectional authenticated secure channel was proposed to improve the confidentiality and integrity of identity proof [47]. The vulnerabilities were also found to break the designation, such as the incorrect binding at IdPs [32] and insufficient verification at RPs [10]–[12].

Analysis on mobile SSO systems. In mobile SSO systems, a new challenge is introduced in the identity proof transmission between mobile applications of RP and IdP. No unique identifier (such as DNS name in Web SSO) exist to designate the receiving mobile application, as the malicious application may register a same identifier with the victim at mobile OSes, to steal identity proof [29]. The IdP may provide an encapsulated WebView to be integrated in the RP's mobile application to simplify this transmission. However vulnerabilities were found in WebView [48], and exploited by malicious RP's application to tamper with the communication between the WebView and IdP server [14]. Automatic analyzing tools were proposed for mobile SSO systems, and plenty of vulnerabilities were found in the top Android applications to break the confidentiality and binding of identity proof [14], [29], [33], [34].

Formal analysis on SSO systems. The SSO standards (e.g., SAML, OAuth and OIDC) and the typical implementations have been formally analyzed. Fett et al. [8], [27] conducted the formal analysis on OAuth 2.0 and OIDC standards and proposed two new attacks, i.e., 307 redirect attack and IdP Mix-Up attack. When IdP misuses HTTP 307 status code for redirection, the sensitive information (e.g., credentials) entered at the IdP will be leaked to the RP by the user's browser. While, IdP Mix-Up attack confuses the RP about which IdP is used and makes the victim RP send the identity proof to the malicious IdP. Fett et al. [8], [27] proved that OAuth 2.0 and OIDC are secure once these two attacks prevented. UPPRESSO could be integrated into OIDC, which simplifies its security analysis. The formal analysis of Google's SAML implementation [26] found a vulnerability which could be exploited by the malicious RPs to reuse the identity proof at other RPs. Ye et al. [49] performed a formal analysis of SSO implementations for Android, and found a vulnerability of Facebook Login which leaked the Facebook's session cookie to the malicious RP applications.

Single sign-off. In SSO systems, once a user's IdP account is compromised, the adversary could hijack all her RPs' accounts. A backwards-compatible extension, named single sign-off, is proposed for OIDC. The single sign-off allows the user to revoke all her identity proof and notify all RPs to freeze her accounts [5]. The single sign-off could also be achieved in UPPRESSO, where the correct user needs to revoke the identity proofs at all RPs, as the IdP doesn't know which RPs the user visits.

B. Privacy protection for SSO systems.

Privacy-preserving SSO systems. As suggested in NIST SP800-63C [17], SSO systems should prevent both RP-based identity linkage and IdP-based access tracing. The pairwise user identifier is adopted in SAML [3] and OIDC [2], and only prevents RP–based identity linkage; while SPRESSO [6] and BrowserID [18] only prevent IdP-based access tracing. BrowserID is adopted in Persona [21] and Firefox Accounts [23], however a formal analysis on Persona, found IdP-based accessing tracing could still succeed [18], [50]. UPPRESSO prevents both the RP-based identity linkage and IdP-based accessing tracing, and could be integrated into OIDC which has been formally analyzed [27]. Moreover, OAuth and OIDC allow users to determine whether to provide an attribute to the RPs or not [7], [12].

Anonymous SSO systems. Anonymous SSO schemes are designed to allow users to access a service (i.e. RP) protected by a verifier (i.e., IdP) without revealing their identity. One of the earliest anonymous SSO systems is proposed for Global System for Mobile (GSM) communication in 2008 [51]. The notion of anonymous SSO was formalized [52] in 2013. And, various cryptographic primitives, such as group signature, zero-knowledge proof and etc., were adopted to design anonymous SSO schemes [52], [53]. Anonymous SSO schemes are designed for the anonymous services, and not applicable to common services.

X. CONCLUSION

In this paper, we propose an unlinkable privacy-preserving single sign-on system, named UPPRESSO, which, for the first time, protects a user's activity profile of RP visits from both the curious IdP and the collusive RPs. UPPRESSO provides three functions, $\mathcal{F}_{ID_{RP}\mapsto PID_{RP}}$ to prevent curious IdP from obtaining the identifer of the visiting RP, $\mathcal{F}_{ID_U\mapsto PID_U}$ to prevent collusive RPs from linking a user, and $\mathcal{F}_{PID_U\mapsto Account}$ allowing each RP to derive a unchanged account for a user's multiple logins. These three functions could be integrated into existing SSO protocols, such as OIDC, to protect the user's privacy, without degrading the security. Moreover, these functions are efficient, the evaluation demonstrates it takes only 208 ms for a user to log into an RP.

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, 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

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

[39] "MITREid connect /openid-connect-java-spring-server," https://github.com/mitreid-connect/OpenID-Connect-Java-Spring-Server, Accessed August 20, 2019.
[40] "Openid foundation," https://openid.net/certification/, Accessed August 20, 2019.
[41] "jsrsasign," https://kjur.github.io/jsrsasign/, Accessed August 20, 2019.
[42] Google, "Declare Permissions," https://developer.chrome.com/extensions/declare_permissions.
[43] Ben Laurie, Adam Langley, and Emilia Käsper, "Certificate transparency," *RFC*, vol. 6962, pp. 1–27, 2013.
[44] 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.
[45] 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.
[46] "The shibboleth project," https://www.shibboleth.net, Accessed July 30, 2019.
[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] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin, "Attacks on webview in the android system," in *Twenty-Seventh Annual Computer Security Applications Conference, ACSAC 2011, Orlando, FL, USA, 5-9 December 2011*, 2011, pp. 343–352.
[49] 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.
[50] 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.
[51] 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.
[52] 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.
[53] 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.