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 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's identities across different RPs, to learn the user's 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 *UPPRESSO*, to protect a user's activity profile of RP visits against both the curious IdP and the collusive RPs. In the login process of UPPRESSO, the IdP that is aware of all users' identities, calculates a privacy-preserving pseudo-identifier (PID_U) for a user, based on the user's identity and the pseudo-identifier (PID_{RP}) of the visited RP. PID_{RP} bound along with PID_U in the identity proof, is transformed from the RP's identity cooperatively by the user and the RP, and then the IdP does not know the visited RPs. The visited RP obtain a trapdoor from the transformation of PID_{RP}, and is able to use this trapdoor to derive the user's account from PID_U, while a user's accounts are different across the RPs. The login process of UPPRESSO follows the communication pattern of OpenID Connect, a widely deployed SSO system. The analysis demonstrates that UPPRESSO protects user privacy well, without any degradation on the security guarantee of OpenID Connect. We have implemented the prototype of UPPRESSO and the experimental evaluation shows that UPPRESSO is efficient and it takes only 254 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 OpenID Connect [1], OAuth [2] and SAML [3], have been widely deployed 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 visits multiple RPs with only a single explicit authentication attempt at 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. 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 in the Internet.

The adoption of SSO also raises several privacy concerns regarding online user tracking and profiling [6], [7]. 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 risk of privacy leakage. On receiving a user's login request, the RP constructs a request of identity proof with its identity and redirects it to the IdP. After authenticating the user, the IdP generates an identify proof containing the identities of the user and the RP, which is forwarded to the RP by the user. Finally, the RP verifies the identity proof and allows the user to log in. From such login instances, any curious IdP or multiple collusive RPs could break the users' privacy as follows.

- *IdP-based login tracing*. The IdP knows the identities of the RP and user in each single login instance, to generate the identity proof. As a result, a curious IdP could discover all the RPs that the victim user attempts to visit and profile her online activities.
- *RP-based identity linkage*. The RP learns a user's identity from the identify proof. When the IdP generates identity proofs for a user, if the same user identifier is used in identity proofs generated for different RPs, which is the case of several widely deployed SSO systems [8], [9], malicious RPs could collude to not only link the user's login activities at different RPs for online tracking but also associate her attributes across multiple RPs [6].

Large IdPs, especially social IdPs such as Google and Facebook, are interested in collecting users' online behavioral information for various purposes (e.g., Screenwise Meter [10] and Onavo [11]). By simply serving the IdP role, these companies can easily collect a large amount of data to reconstruct users' online traces. On the other hand, in the Internet, many service providers host a variety of web services and therefore take an advantaged position to link a user's multiple logins at different RPs. Through the internal information integration, rich information will be obtained from the SSO data for user profiling. Meanwhile, the technologies of privacy-preserving record linkage [12] and private set intersection [13] allow multiple organizations (e.g., RPs) to share and link the data without direct sharing their clients' data, which has paved the path for cross-organizational RP-based identity linkage.

While the privacy problems in SSO have been widely recognized [6], [7], only a few solutions were proposed to protect user privacy [9], [14]. Among them, Pairwise Pseudonymous Identifier (PPID) [1], [15] is a 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 into different RPs, so that the pseudo-identifiers of the same user cannot be used to link the user's logins at different RPs even if they collude. As a recommended practice by NIST [7], PPID has been specified in many widely adopted SSO standards including OIDC [1] and SAML [15]. However, PPID-based approaches cannot prevent the IdP-based login tracing attacks, as the IdP still knows which RP the user visits.

To the best of our knowledge, only two schemes (i.e., BrowserID [8] and SPRESSO [9]) have been proposed so far to defend against IdP-based login tracing. In BrowserID (and its prototypes known as Mozilla Persona [14] and Firefox Accounts [16]), the IdP generates a special identity proof to bind the user's unique identifier (i.e., email address) to a public key. With the corresponding private key, the user signs an extra subsidiary identity proof to bind the visited RP's identity and its identity, and sends this pair of identity proofs to that RP. In this way, the IdP does not know the RP's identity when generating identify proofs. SPRESSO requires the RP to create a one-time pseudo-identifier at each login for the IdP to generate an identity proof, and then hides the RP's real identity from the IdP. The RP employs a third-party entity called forwarder, which works as a proxy to relay the identity proof from the IdP to the corresponding RP. In both schemes, the RPs' identifiers are protected from the IdP; however, the IdP has to know the user's unique identifier (e.g., email address) and includes it in identity proofs so that the visit RP can recognize the user in her multiple logins. As a result, both schemes are still vulnerable to RP-based identity linkage.

As discussed above, none of the existing SSO systems defend against both IdP-based login tracing and RP-based identity linkage at the same time. Before presenting our solution, we first formally analyze the privacy problems and solutions in SSO. Let us denote the user's and the visited RP's identities as ID_U and ID_{RP}, respectively. To protect user privacy against RP-based identity linkage, ID_U should not be explicitly included in the identity proof which will be received by the RP. Instead, a privacy-preserving pseudo-identifier PID_U should be used (as in the PPID-based approaches [1], [15]), which can be viewed as the output of a one-way identifier-transformation function $\mathcal{F}_{ID_U \mapsto PID_U}$ at the IdP, which authenticates the user and then know ID_U. Similarly, to prevent IdP-based login tracing, ID_{RP} should not be explicitly included in the identity proof but replaced by a pseudo-identifier PID_{RP} (as in SPRESSO [9] and BrowserID [8]), which is generated by another one-way function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$ at the RP. However, if both PID_U and PID_{RP} are used in identity proofs to replace ID_U and ID_{RP} at the same time, assuming they can be securely exchanged between the IdP and the RP in an SSO login process, the RP will allow the user to log in as PID_U, which will be different in the user's multiple logins at a same RP. As a result, the RP has no clues to derive the real account of the user but treats her as a one-time user every time when she logs in. This violates the basic requirements of SSO services.

In this paper, we propose an Unlinkable Privacy-PREserving Single Sign-On (UPPRESSO) system to provide comprehensive protections against both IdP-based login tracing and RP-based identity linkage. The key idea of UPPRESSO is to design a special identifier-transformation function $\mathcal{F}_{PID_U \mapsto Account}$, which maps *all* PID_Us of a user to a unique $Account$ at the RP in all logins, where $Account$ is created when the user logs into the RP for the first time. Since in every login instance PID_U and PID_{RP} are separately generated by the IdP and the RP, respectively, we have to design matched one-way identifier-transformation functions $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, so that three identifier-transformation functions work cooperatively to ensure: (*a*) when a user logs into an RP for multiple times, the RP always maps PID_Us to an identical $Account$ without knowing the user's identity; moreover, when a user logs into multiple RPs, (*a*) a curious IdP cannot link multiple PID_{RP}s to a particular RP or associate them together, and collusive RPs (*c*) cannot link PID_Us to a particular user or associate them together, (*d*) nor link $Account$s of a same user at different RPs.

To achieve these goals, we design three one-way identifier-transformation functions based on the discrete logarithm problem. First, we design a one-way trapdoor function $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}(ID_{RP}, T)$ for an RP to generate a random PID_{RP} based on a randomly chosen trapdoor T, and a one-way function $\mathcal{F}_{ID_U \mapsto PID_U}(ID_U, PID_{RP})$ for the IdP to generate PID_U based on PID_{RP}. With PID_{RP}, PID_U and the trapdoor T, the RP applies $\mathcal{F}_{PID_U \mapsto Account}(PID_U, PID_{RP}, T)$ to derive the unique $Account$. We summarize our contributions as follows.

- We formally analyze the privacy problems in SSO as an identifier-transformation problem, and propose the first comprehensive solution to hide the users' login traces from both the curious IdP and malicious collusive RPs. To the best of our knowledge, UPPRESSO is the first SSO system that provides secure SSO services against IdP-based login tracing and RP-based identity linkage.
- We systematically analyze the security of UPPRESSO and show that it achieves the same security level as existing SSO systems, while the users' login traces are well protected.
- We have implemented a prototype of UPPRESSO based on an open-source implementation of OIDC, which requires only small modifications to support three identifier-transformation functions for privacy protections. Unlike BrowserID and SPRESSO, UPPRESSO does not require non-trivial re-designs of SSO services, which makes it more compatible with existing SSO systems.
- We compare the performance of the UPPRESSO prototype with the state-of-the-art SSO systems (i.e., OIDC and SPRESSO), and demonstrate its efficiency.

The rest of the paper is organized as follows. We first introduce the background and preliminaries in Section II. Then, we describe the identifier-transformation based approach, the threat model, and our UPPRESSO design in Sections III, IV and V, followed by a systematical analysis of security and privacy in Section VI. We provide the implementation specifics and experiment evaluation in Section VII, discuss the related works in Section IX, and conclude our work in Section X.

II. Background and Preliminaries

UPPRESSO is compatible with OIDC and provides privacy protections based on the discrete logarithm problem. Here, we provide a brief introduction about OIDC and the discrete logarithm problem.

A. OpenID Connect

As an extension of OAuth 2.0 to support user authentication, OIDC [1] is one of the most popular SSO protocols. Same as other SSO protocols [15], OIDC involves three entities, i.e., *users*, the *identity provider (IdP)*, and *relying parties (RPs)*. Both users and RPs register at the IdP with identifiers (ID_U, ID_{RP} and PID_U in some schemes), and the necessary information such as credentials, RP endpoints (e.g., URLs to receive the identity proofs), etc. The IdP is assumed to maintain these attributes securely. In an OIDC login process, a user is responsible for initiating a login request at an RP, redirecting the SSO messages between the RP and IdP, and checking the scope of user attributes in the identify proof generated by the IdP for the visited RP. Usually, the redirection and checking actions are handled by a user-controlled software, known as *user agent* (e.g., browser). Once receiving a user login request, the RP constructs an identity proof request with its identifier and the requested scope of user attributes, sends the identity proof request to the IdP through the user, and parses the received identity proof to authenticate the user. The IdP authenticates the user based on her ID_U and credential, maps ID_U to $PPID$ (i.e., privacy-preserving pseudo-identifier) based on the RP identity (ID_{RP}), generates an identity proof containing $PPID$, ID_{RP} and requested user attributes, and returns the identity proof to the endpoint registered by the RP.

OIDC Implicit Flow. OIDC supports three different user login flows, which are the *implicit flow*, *authorization code flow* and *hybrid flow* (i.e., a mix-up of the previous two). In the implicit flow, an *id token* is generated as the identity proof, which contains a user identifier, an RP identifier, the issuer (i.e., IdP), the validity period, and other requested attributes. The IdP signs the id token using its private key to ensure integrity, and sends it to RP through the user. In the authorization code flow, the IdP binds an authorization code with the RP, and sends it to the RP through the user; then, the RP establishes an HTTPS connection to the IdP and uses the authorization code with the RP's credential to obtain the user's identifier and other attributes. UPPRESSO is compatible with all three flows. For brevity, we will present our design and implementation of UPPRESSO on top of the implicit flow of OIDC in details, and discuss the extension to support the authorization code flow in Section VIII.

The original OIDC implicit flow is shown in Figure 1. When a user attempts to log into an RP, the RP constructs an identity proof request and returns it to the user, which gets redirected

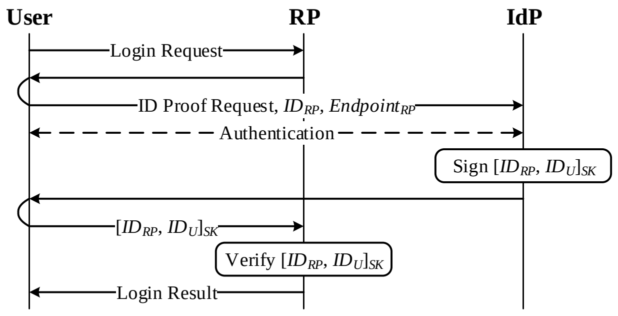
User
RP
IdP
Login Request
ID Proof Request, ID_{RP}, $Endpoint_{RP}$
Authentication
Sign $[ID_{RP}, ID_U]_{SK}$
$[ID_{RP}, ID_U]_{SK}$
Verify $[ID_{RP}, ID_U]_{SK}$
Login Result

Fig. 1: The implicit flow of OIDC.

to the IdP. The request contains ID_{RP}, the RP endpoint to receive the identity proof, and the scope of requested user attributes. If the user has not yet been authenticated, the IdP initiates an authentication process to authenticate her. For a successfully authenticated user, the IdP generates an id token and returns it to the RP endpoint. If the endpoint is not registered for that RP, the IdP will return a warning to notify the user about potential identity proof leakage. Once the RP receives the identity proof, it makes the authentication decision after verifying the validity.

RP Dynamic Registration. The RP dynamic registration [17] of OIDC allows an RP to update its information at the IdP. When an RP first registers at the IdP, it obtains a registration token, with which the RP can initiate a dynamic registration process to update its information (e.g., the endpoint). After a successful dynamic registration, the RP obtains a new unique ID_{RP} from the IdP. UPPRESSO leverages this function and slightly modify the dynamic registration process to enable *RP pseudo-identifier registration*, which allows an RP to generate different privacy-preserving identifiers (PID_{RP}s) and register them at the IdP.

B. Discrete Logarithm Problem

Based on the discrete logarithm problem, UPPRESSO design the identifier-transformation functions. Here, we briefly review the discrete logarithm problem.

For the finite field $GF(p)$ where p is a large prime, a number g is called a generator of order q, if it constructs 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 solve the discrete logarithm (i.e., x) of y [18], which is called the discrete logarithm problem. The hardness of solving discrete logarithms is utilized to design several secure cryptographic primitives, including Diffie-Hellman key exchange and the digital signature algorithm (DSA).

III. The Identifier-Transformation Approach of UPPRESSO

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

A. *Security Requirements of SSO*

The primary goal of SSO services is to implement secure user authentication [9], i.e., to ensure that a valid user can always log in to an honest RP under his/her unique account. To achieve this, every identity proof generated by the IdP should explicitly specify the user authenticated by the IdP (i.e., ***user identification***) and the RP to which the user attempts to log in (i.e., ***RP designation***). User identification also requires an RP to be able to identify and recognize every authenticated user under her unique account registered at that RP during multiple logins. Moreover, the identify proof should be generated by the trusted IdP and correctly transmitted to the dedicated RP (and the user in some schemes) but not anyone else (i.e., ***confidentiality***), and the identify proof should not be modified or forged (i.e., ***integrity***). These are the basic security requirements of SSO systems summarized based on existing theoretical analyses [19]–[21] and practical attacks [22]–[34].

Many SSO attacks exploit vulnerabilities in SSO design or implementation to break at least one of these security requirements, so that the adversaries can log in to an honest RP as a victim user (called *impersonation attacks*), or allure a victim user to log in to an honest RP under the attacker's account (called *identity injection attacks*). For example, Friendcaster used to accept every received identity proof (i.e., a violation of RP designation) [35], [36], so a malicious RP could replay the identity proof received from the user for itself and log in to Friendcaster as the victim user [30]. In [20], [27], [36], the adversaries could impersonate the victim user with a leaked identity proof (i.e., a violation of confidentiality). It was also 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) [23], as a result, an adversary could insert arbitrary attributes (e.g., an email address) into the identity proof to impersonate another user at the RP.

B. *The Privacy Dilemma in SSO Identity Proofs*

As discussed in Section I, existing SSO systems are vulnerable to IdP-based login tracing and RP-based identity linkage privacy leakage. We argue that a secure and privacy-preserving SSO system should prevent *both* types of privacy leakage while satisfying *all* four basic security requirements. However, meeting the security and privacy requirements at the same time incurs a dilemma in identity proof generation.

In an SSO authentication session, the identity proof is generated by the IdP about the authenticated user and the requesting RP. Let us denote the long-term unique identifiers of the user and the RP as ID_U and ID_{RP}, respectively. First, to prevent IdP-based login tracing, the identity proof request should not disclose ID_{RP} to the IdP, since the IdP already knows ID_U after authenticating the user. However, to ensure RP designation, the IdP should bind each identify proof with a one-time pseudo-identifier of the RP (denoted as PID_{RP}) generated by the user or the RP, or together. But, it should be computationally infeasible for the IdP to derive ID_{RP} from PID_{RP}. Meanwhile, to prevent RP-based identity linkage, the identity proof should not directly contain ID_U or disclose it in any means. This requires the IdP to generate a one-time pseudo-identifier of the user (denoted as PID_U) and bind the identity proof with it. However, to ensure user identification, the requesting RP should be able to recognize the user and link PID_U to her unique account at the RP (denoted as $Account$). But, it should be computationally infeasible for the RP to derive ID_U from PID_U or $Account$.

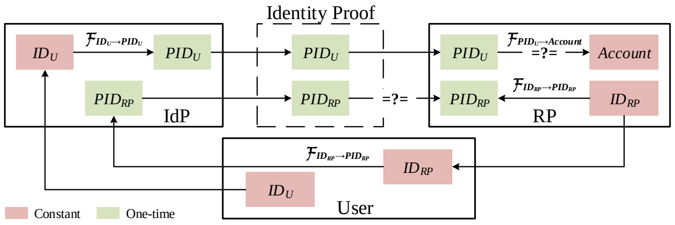
Identity Proof
ID_U
$\mathcal{F}_{ID_U \mapsto PID_U}$
PID_U
PID_{RP}
IdP
PID_U
PID_{RP}
=?=
PID_U
$\mathcal{F}_{PID_U \mapsto Account}$
=?=
$Account$
PID_{RP}
$\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$
ID_{RP}
RP
$\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$
ID_{RP}
ID_U
User
Constant
One-time

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

We illustrate the relationships between the identifiers, pseudo-identifiers, and the identity proof in Fig. 2, where red and green blocks respectively represent long-term identifiers and one-time pseudo-identifiers known to each entity, and the arrows denote how the pseudo-identifiers are obtained. Obviously, a dilemma exists in the SSO login session, that is, the IdP is expected to generate a PID_U for the authenticated user ID_U that can be linked to her $Account$ at the requesting RP, without even knowing the RP's identity.

C. *The Identifier-transformation Framework and UPPRESSO*

As shown in Fig. 2, we convert the secure SSO login problem to an identifier-transformation problem, which aims to design three identifier-transformation functions $\mathcal{F}_{ID_U \mapsto PID_U}$, $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, and $\mathcal{F}_{PID_U \mapsto Account}$ appropriately, to compute PID_U, PID_{RP} and $Account$ that satisfy the desired security and privacy requirements. To solve the dilemma, the key is to pass some information about the user's $Account$ at an RP to the IdP so that all PID_Us dynamically generated for the user during her multiple logins at that RP can correctly link to her $Account$. Meanwhile, such information should not provide additional knowledge for the IdP to infer the RP's real identity (i.e., ID_{RP}).

To achieve this goal, UPPRESSO constructs three transformation functions in an integrated way such that $\mathcal{F}_{ID_U \mapsto PID_U}$ allows the IdP to generate PID_U based on ID_U and PID_{RP} and $\mathcal{F}_{PID_U \mapsto Account}$ allows the RP to derive $Account$ from PID_U and PID_{RP}, where (i) in each login session, PID_{RP} helps to link PID_U to $Account$; (ii) in different logins, different PID_Us and PID_{RP}s are generated to protect privacy; and (iii) PID_Us and PID_{RP}s are generated cooperatively so that an invariant $Account$ can always be derived for a same user and RP pair.

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 T to derive the identical

TABLE I: Identifier-transformation in privacy-preserving SSO.

Solution	PID_U	PID_{RP}	$Account$
PPID	$\mathcal{F}(ID_U, ID_{RP})$	ID_{RP}	PID_U
SPRESSO	ID_U	$Enc(ID_{RP}\|nonce)$	ID_U
BrowserID[†]	ID_U	$\perp$	ID_U
UPPRESSO	$\mathcal{F}(ID_U, PID_{RP})$	$\mathcal{F}(ID_{RP}, T)$	$\mathcal{F}(PID_U, T)$

†: BrowserID binds null PID_{RP} in the identity proofs by the IdP, but ID_{RP} is bound in the *subsidiary* identity proof signed by the user.

$Accout$ from dynamic PID_Us in identity proofs. Intuitively, the trapdoor T also plays a part 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 identifier 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 calculated 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 Privacy-Preserving SSO Solutions

We use the framework of identifier transformation in Figure 2 to explain the designs of existing SSO solutions. First of all, when $PID_U = ID_U$ and $PID_{RP} = ID_{RP}$, this framework describes the basic SSO services. It is also applicable to explain the approaches of privacy-preserving SSO systems, including PPID [1], BrowserID [8] and SPRESSO [9].

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

SPRESSO and BrowserID prevent only IdP-based login tracing but not RP-based identity linkage. In a login instance 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 binds 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 login 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 then $Account = ID_U$.

As analyzed above, none of these solutions provides protections against both IdP-based login 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, attempting 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 calculates 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 login 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 [37], [38] 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 login tracing based on the SSO protocols, and we do not consider other network attacks (e.g., the network traffic analysis to associate 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 calculated as follows, where r is a random number ($1 < r < q$) generated by the IdP.

$$ID_{RP} = g^r \bmod p \qquad (1)$$

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 calculate PID_{RP} as in Equation 2.

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

$\mathcal{F}_{ID_{RP}\mapsto PID_{RP}}$ 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 associate multiple PID_{RP}s for a same RP. Finally, the cooperation by the user and the RP prevents a single malicious entity from manipulating the value of PID_{RP}.

On receiving an identity proof 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.

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

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. If r is known to the RP, two collusive RPs might attempt to associate a user's PID_Us by checking whether the equality ${PID_{U_1}}^{r_2 N_{U_2} N_{RP_2}} = {PID_{U_2}}^{r_1 N_{U_1} N_{RP_1}} \bmod p$ holds or not, because $PID_{U_1} = g^{r_1 N_{U_1} N_{RP_1} ID_{U_1}} \bmod p$ and $PID_{U_2} = g^{r_2 N_{U_2} N_{RP_2} ID_{U_2}} \bmod p$.

Finally, the RP derives $Account$ for the user as Equation 4. Here, we define $T = (N_U N_{RP})^{-1} \bmod q$ as the RP's trapdoor. 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.

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

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

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

$\mathcal{F}_{PID_U\mapsto Account}$ satisfies the following requirements. Similar to the analysis of PID_U, the RP cannot derive ID_U from $Account$, and collusive RPs cannot link a user's $Account$s 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} in each 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}. Finally, the transformed RP designation is provided through the two-phase 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 login tracing.

B. UPPRESSO Overview

In addition to the identifier-transformation functions, 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 to ensure confidentiality; otherwise, an incorrect endpoint leaks the identity proofs. 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. Then, the user determines the endpoint by itself, while in commonly-used OIDC systems, the endpoint is configured by the IdP. Secondly, we handle PID_{RP} in two phases: it is registered at the trusted IdP with a one-time endpoint, and then PID_{RP} is included in the identity proof as common identity proofs. Therefore, they are implemented as two phases compatible with OIDC. Finally, 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 associate PID_{RP} and ID_{RP}.

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 including 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 user 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 clear understand, 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 secret, while p, q, g and PK are public parameters.

TABLE II: The notations used in UPPRESSO.

Notation	Definition	Attribute
p	A large prime.	Long-term constant
q	A large prime factor of $(p-1)$.	Long-term constant
g	A generator of order q in $GF(p)$.	Long-term constant
SK, PK	The private/public key of IdP.	Long-term constant
ID_{RP}	$ID_{RP} = g^r \bmod p$, an RP's unique identity.	Long-term constant
$Cert_{RP}$	An RP certificate, containing the RP's identity and endpoint.	Long-term constant
ID_U	A user's unique identity.	Long-term constant
$Account$	$Account = {ID_{RP}}^{ID_U} \bmod p$, a user's identifier at an RP.	Long-term constant
PID_{RP}	$PID_{RP} = {ID_{RP}}^{N_U N_{RP}} \bmod p$, an RP's pseudo-identifier.	One-time variable
PID_U	$PID_U = {PID_{RP}}^{ID_U} \bmod p$, a user's pseudo-identifier.	One-time variable
N_U	A user-generated nonce for PID_{RP}.	One-time variable
N_{RP}	An RP-generated nonce for PID_{RP}.	One-time variable
Y_{RP}	$Y_{RP} = {ID_{RP}}^{N_{RP}} \bmod p$, the public value for N_{RP}.	One-time variable
T	$T = (N_U N_{RP})^{-1} \bmod q$, the trapdoor to derive $Account$.	One-time variable

RP Initial Registration. An RP registers itself at the IdP to request ID_{RP} and $Cert_{RP}$ as follows:

- The RP sends a registration request to the IdP, including the RP's endpoint (e.g., URL) for receiving the identity proof.
- The IdP chooses a unique random number r ($1 < r < q$), calculates $ID_{RP} = g^r \bmod p$, signs $[ID_{RP}, Endpoint_{RP}, *]$ using SK, where $*$ is supplementary information such as the RP's common name, and returns $Cert_{RP} = [ID_{RP}, Endpoint_{RP}, *]_{SK}$ to the RP, where $[\cdot]_{SK}$ means a message signed using SK.
- 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 r is kept unknown to the RP. On the contrary, ID_U may be chosen by the user or the IdP, provided that it is unique for each user.

SSO Login. Once a user attempts to log into an RP, the SSO login is initiated. As shown in Figure 3, the SSO login consists of four phases, RP identifier transformation, PID_{RP} registration, identity proof generation and $Account$ calculation. In the RP identifier transforming, the user and the RP negotiate $PID_{RP} = {ID_{RP}}^{N_U N_{RP}} \bmod p$. Then, the user registerers PID_{RP} at the IdP. The user requests the identity proof from the IdP, and the IdP calculates $PID_U = {PID_{RP}}^{ID_U} \bmod p$ and signs the identity proof. Finally, in the $Account$ calculation, the RP derives $Account = {PID_U}^{(N_U N_{RP})^{-1} \bmod q} \bmod p$ after verifying the identity proof, and allows the user to log in as $Account$.

C. SSO Login Flow of UPPRESSO

Figure 3 shows the detailed SSO login protocol of UPPRESSO. We describe the detailed processes as follows.

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

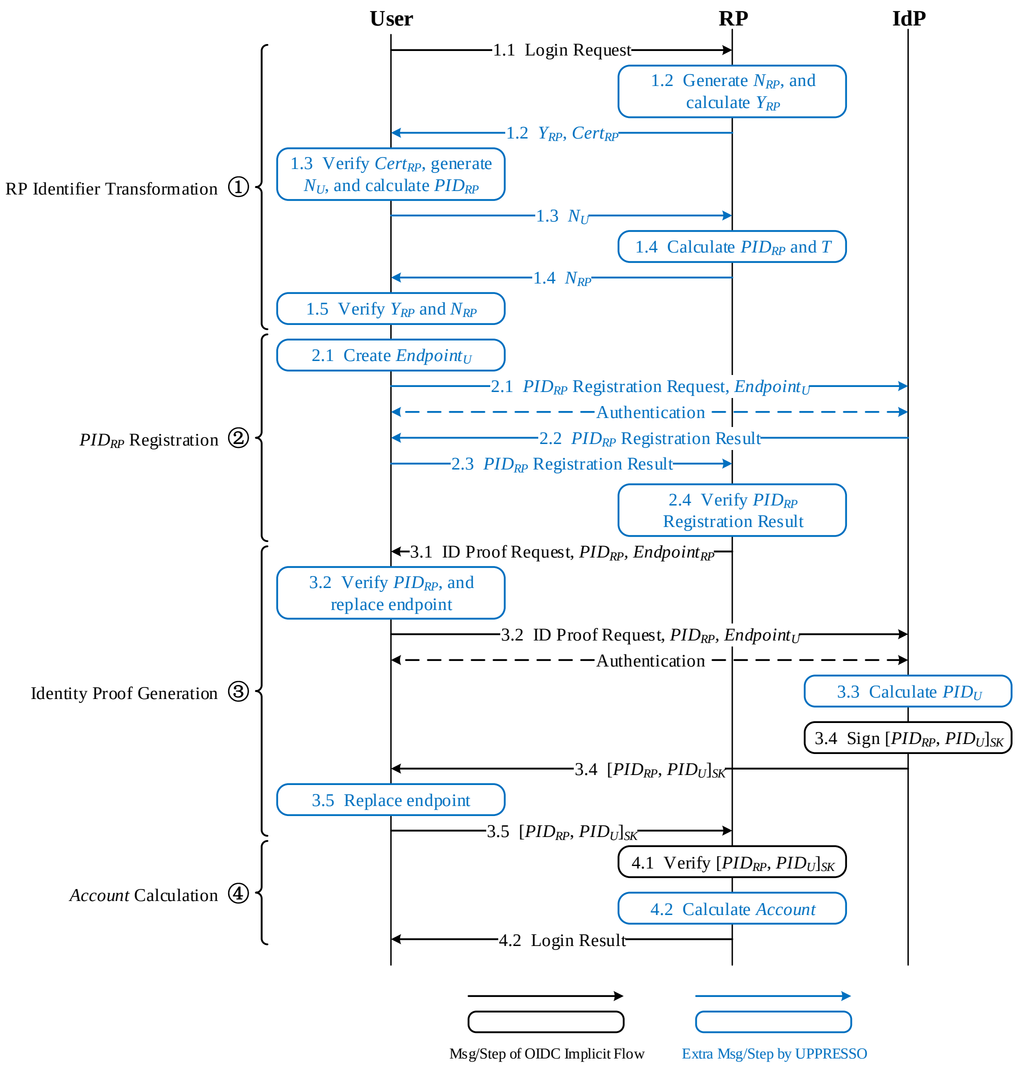
User
RP
IdP
RP Identifier Transformation ①
1.1 Login Request
1.2 Generate N_{RP}, and calculate Y_{RP}
1.2 Y_{RP}, $Cert_{RP}$
1.3 Verify $Cert_{RP}$, generate N_U, and calculate PID_{RP}
1.3 N_U
1.4 Calculate PID_{RP} and T
1.4 N_{RP}
1.5 Verify Y_{RP} and N_{RP}
PID_{RP} Registration ②
2.1 Create $Endpoint_U$
2.1 PID_{RP} Registration Request, $Endpoint_U$
Authentication
2.2 PID_{RP} Registration Result
2.3 PID_{RP} Registration Result
2.4 Verify PID_{RP} Registration Result
Identity Proof Generation ③
3.1 ID Proof Request, PID_{RP}, $Endpoint_{RP}$
3.2 Verify PID_{RP}, and replace endpoint
3.2 ID Proof Request, PID_{RP}, $Endpoint_U$
Authentication
3.3 Calculate PID_U
3.4 Sign $[PID_{RP}, PID_U]_{SK}$
3.4 $[PID_{RP}, PID_U]_{SK}$
3.5 Replace endpoint
3.5 $[PID_{RP}, PID_U]_{SK}$
Account Calculation ④
4.1 Verify $[PID_{RP}, PID_U]_{SK}$
4.2 Calculate *Account*
4.2 Login Result
Msg/Step of OIDC Implicit Flow
Extra Msg/Step by UPPRESSO

Fig. 3: Process for each user login.

1.1 The user sends a login request to trigger the negotiation of PID_{RP}.
1.2 The RP chooses a random N_{RP} $(1 < N_{RP} < q)$, calculates $Y_{RP} = {ID_{RP}}^{N_{RP}} \bmod p$; and sends Y_{RP} with $Cert_{RP}$ to the user.
1.3 The user verifies $Cert_{RP}$, extracts ID_{RP} from the valid $Cert_{RP}$, chooses a random N_U $(1 < N_U < q)$ to calculate $PID_{RP} = {Y_{RP}}^{N_U} \bmod p$, and sends N_U to the RP.
1.4 The RP verifies $N_U \neq 0 \bmod q$, calculates PID_{RP} with N_U and Y_{RP}, derives the trapdoor $T = (N_U N_{RP})^{-1} \bmod q$; and acknowledges the negotiation by responding with N_{RP}.
1.5 The user verifies that $N_{RP} \neq 0 \bmod q$ and $Y_{RP} = {ID_{RP}}^{N_{RP}} \bmod p$.

The user will halt the negotiation, if $Cert_{RP}$ is invalid, $N_{RP} = 0 \bmod q$, or $Y_{RP} \neq {ID_{RP}}^{N_{RP}} \bmod p$. The verification of Y_{RP} and N_{RP} ensures the order of Y_{RP} (and also PID_{RP}) is q, and prevents a malicious RP from choosing an arbitrary Y_{RP} (then PID_{RP}) of order less than q, which makes it less difficult for the RP to derive ID_U from PID_U.

$\mathbf{PID_{RP}}$ Registration. The user registers PID_{RP} at the IdP as follows.

2.1 The user creates an one-time endpoint to hide the RP's endpoint from the IdP, and sends the PID_{RP} registration request $[PID_{RP}, Hash(N_{RP}, N_U), Endpoint_U]$ to the IdP.
2.2 The IdP authenticates the user if she has not been authenticated yet. The IdP verifies that PID_{RP} is unique among unexpired PID_{RP}s, and then signs the response $[PID_{RP}, Hash(N_{RP}, N_U), Validity]_{SK}$, where $Validity$ is the validity period. The IdP returns the signed response to the user.
2.3 The user forwards the registration result to the RP.
2.4 The RP verifies the IdP's signature, and accepts it only if PID_{RP} and $Hash(N_{RP}, N_U)$ match those in the negotiation and it is in the validity period.

$Hash(N_{RP}, N_U)$ is attached as the nonce to avoid the result is accepted by two or more RPs, which have different ID_{RP}s but generate a same PID_{RP} with a negligible possibility. The IdP ensures PID_{RP} is unique among unexpired ones; otherwise, one identity proof for one PID_{RP} might be accepted by other RPs. More details are analyzed in Section VI.

ID Proof Generation. In this phase, the RP continues the process of the user login and obtains PID_U generated by the IdP. The processes are as follows.

3.1 The RP uses PID_{RP} and $Endpoint_{RP}$ to construct an identity proof request for a set of user's attributes.
3.2 The user first confirms the scope of the requested attributes and verifies PID_{RP} with the negotiated one. The user replaces the endpoint with the registered one-time $Endpoint_U$, and sends the modified identity proof request to the IdP.
3.3 The IdP verifies whether PID_{RP} and $Endpoint_U$ have been registered and unexpired, and calculates $PID_U = {PID_{RP}}^{ID_U} \bmod p$ for the authenticated user.
3.4 The IdP constructs and signs the identity proof $[PID_{RP}, PID_U, Iss, ValTime, Attr]_{SK}$, where Iss is the identifier of the IdP, $ValTime$ is the validity period, $Attr$ contains the requested attributes. Then, the IdP sends the identity proof to the one-time endpoint at the user.
3.5 The user extracts the RP endpoint in $Cert_{RP}$, and forwards the identity proof to the RP through this endpoint.

The user halts the process if PID_{RP} in the identity proof request is inconsistent with the negotiated one. The IdP rejects the identity proof request, if the pair of PID_{RP} and $Endpoint_U$ has not been registered.

Account calculation. Finally, RP derives the user's $Account$ and completes the user login as follows.

4.1 The RP verifies the identity proof, including the signature, the validity period, and the consistency between PID_{RP} and the negotiated one.
4.2 The RP extracts PID_U, and calculates $Accout = {PID_U}^T \bmod p$; and sends the $Success$ as the login result to the user. If any fails, the RP rejects this login.

D. Compatibility with OIDC

The design of UPPRESSO can be integrated in the traditional SSO systems, and it only requires small modifications to implement the SSO login flow of UPPRESSO (i.e., the most important protocol of UPPRESSO). Next, we compare it with OIDC to demonstrate the compatibility of UPPRESSO.

As mentioned above, the SSO login flow of UPPRESSO is composed of four phases, namely, RP identifier transforming, PID_{RP} registration, identity proof generation, and $Account$ calculation. RP identifier transforming is conducted between a user and the RP, which is specific to UPPRESSO, while the others adopt the communication patterns of some OIDC flows.

First of all, UPPRESSO doesn't introduce any new role, nor change the security assumptions on each role (i.e., user, IdP and RP). PID_{RP} registration can be viewed as the RP dynamic registration flow of OIDC [17], where an entity registers its identity and endpoint at the IdP. In UPPRESSO, this process is launched by any authenticated user who provides a candidate of the identifer, the registration response includes a signature from the IdP, and the registration will become invalid after a validity period. These differences will bring very small modifications.

Identity proof generation and $Account$ calculation of UPPRESSO, finish the same steps as the implicit protocol flow of OIDC, except some parameters are modified as follows. ID_U is transformed to PID_U by the IdP, and this modification is actually allowed in OIDC; for example, PPID allows various designs to calculate PID_U from ID_U. The calculation of $Account$ from PID_U bound in the identity proof, can be viewed as a step by the RP to derive its user account after the implicit protocol flow of OIDC ends. The final modification is the replacement of endpoint by the user, to forward the identity proof to the RP. In fact, this message forwarding is common when an application-layer network proxy is deployed. So identity proof generation and $Account$ calculation of UPPRESSO, can be viewed as a particular but compatible implementation of the implicit protocol flow of OIDC. Finally, identity proof generation and $Account$ calculation of UPPRESSO, can be also implemented as the authorization code flow of OIDC with small modifications, which is discussed in Section VIII.

VI. Analysis

This section presents the analysis of 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 [21]. In the following, we analyze the affects of the modifications listed in Section V-D, respectively.

The first modification is that the identity proof is bound with PID_{RP}, which is negotiated between the user and the RP in RP identifier transformation and then registered at the IdP by the user in PID_{RP} registration. This may affect RP designation, as the identity proof binds PID_{RP} instead of ID_{RP}. In OIDC ID_{RP} is used to ensure that an identity proof is only valid to the designated RP, as the IdP ensures ID_{RP} is unique and the RP only accepts identity proofs binding ID_{RP}. In UPPRESSO PID_{RP} provides the same binding as ID_{RP} as follows, which is achieved by **transformed RP designation** through $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$.

- The IdP ensures the uniqueness of PID_{RP} in PID_{RP} registration (Step 2.2). And the PID_{RP} registration result is signed by the IdP, and verified by the user and the RP before the protocol moves on.
- Collusive RPs and users cannot deceive an RP to accept a fake signed PID_{RP} registration result, because $Hash(N_{RP}, N_U)$ is included in PID_{RP} registration as

nonce and the RP accepts the PID_{RP} registration result only when $Hash(N_{RP}, N_U)$ matches.

The calculations of PID_U and $Account$ in UPPRESSO ensure user identification, by **trapdoor user identification** through $\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$, $\mathcal{F}_{ID_U \mapsto PID_U}$ and $\mathcal{F}_{PID_U \mapsto Account}$. In OIDC, the RP uniquely identifies a user based on the identifier from the trusted IdP. In UPPRESSO, the RP derives an identical $Accout = {PID_U}^T = {ID_{RP}}^{ID_U} \bmod p$ for a user's multiple logins, while both ID_U and ID_{RP} are uniquely assigned by the trusted IdP. Moreover, the calculation can never be tampered by adversaries, as PID_U is provided by the IdP and protected in the identity proof, while T is stored and the calculation is performed at the RP.

In OIDC, the endpoint to receive identity proofs is configured by the IdP, guaranteeing that it is sent to the designated RP, to ensure confidentiality. In UPPRESSO the endpoint is replaced by the user to forward identity proofs to RPs, but confidentiality is still ensured because we introduced $Cert_{RP}$ which is signed by the trusted IdP to guarantee that the user obtains the correct endpoint for ID_{RP}.

The above analysis demonstrates that (1) *integrity* and *confidentiality* are not affected by the modifications in UPPRESSO; and (2) the modifications introduce no security degradation on user identification and RP designation. Therefore, UPPRESSO provides the secure SSO services.

B. Privacy

UPPRESSO prevents both IdP-based login tracing and RP-based identity linkage.

IdP-based login tracing prevention. *The IdP cannot obtain any information about the visited RP from any login,* 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}. For a given PID_{RP}, any $ID_{RP} = g^r$ could find N satisfying $PID_{RP} = {ID_{RP}}^N$ mod p as follows.

Let's assume $PID_{RP} = g^{r_1 N_1 \bmod q} \bmod p$ and $N_1 = N_{U1} N_{RP1} \bmod q$. Then, for any $ID_{RP} = g^{r_2} \bmod p$, $r_2 \neq r_1$), there always exists N_2 satisfying $r_2 N_2 = r_1 N_1 \bmod q$. Because q is a prime and co-prime to any r_2, there always exists $N_2^{'}$ satisfying $r_2 N_2^{'} = 1 \; mod \; q$, and $N_2 = (r_1 N_1) N_2^{'}$ mod q making the equality hold.

IdP cannot to determine whether two or more logins are at 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. *Any RP cannot derive ID_U from one PID_U and Account without manipulating Y_{RP}s.*

- For $PID_U = {PID_{RP}}^{ID_U} = g^{r N_U N_{RP} ID_U \bmod q}$ mod p. 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 calculate ID_U from PID_U due to the discrete logarithm problem.
- For each $Account = {ID_{RP}}^{ID_U} = g^{r ID_U \bmod q} \bmod p$, it is also computational infeasible to calculate 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 $Accout = {PID_U}^T \bmod p$ where T is a random value known to RP and independent with ID_U when the RP doesn't manipulate Y_{RP}.

Any RP cannot derive ID_U from multiple PID_Us and Accounts without manipulating Y_{RP}s. All these $Account$s are equal, while any PID_U (e.g., PID_{U1}) can be calculated from any other PID_U (e.g., PID_{U2}) for the user at this RP, PID_{U1}=$PID_{U2} Account^{N_{U1} N_{RP1} - N_{U2} N_{RP2}} \bmod p$, where N_{U2}, N_{RP2}, N_{U2} and N_{RP2} are values known to the RP and independent with ID_U when the the RP doesn't manipulate Y_{RP}s.

The collusive RPs cannot associate a user's Accounts and PID_Us without manipulating Y_{RP}s. The collusive RPs may attempt to link a user's accounts by checking whether the equality $Account_2$=$(Account_1)^{r_2/r_1} \bmod p$ holds for $Account_1$ at an RP $ID_{RP1} = g^{r_1} \bmod p$ and $Account_2$ at another RP $ID_{RP2} = g^{r_2} \bmod p$. 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 associate PID_Us (Accounts), with manipulating Y_{RP}s. A malicious RP may attempt to manipulate Y_{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 Y_{RP}s cooperatively to make a user's PID_Us or $Account$s be associated at these RPs and then to associate a user's multiple logins. Here, Y_{RP}s are the only values controlled by the RPs. The Y_{RP} must be in the form of ${ID_{RP}}^{N_{RP}} \bmod p$, which is checked by the correct user with the provided N_{RP}. Then, the malicious RPs can only manipulate N_{RP}s. 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 ${PID_{RP}}^{ID_U} \bmod p$ and $g^{r N_U N_{RP} ID_U \bmod q} \bmod p$. 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 ${ID_{RP}}^{ID_U} \bmod p$ and $g^{r ID_U \bmod q} \bmod p$. Obviously, $Account$ is independent with N_{RP} and cannot be controlled by any RP.

The collusive RPs and users cannot associate the victim user, with manipulating Y_{RP}s. 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 $Account_{m1}/Account_{v1} = {ID_{RP1}}^{ID_{\Delta}} \bmod p$ and $Account_{m2}/Account_{v2} = {ID_{RP2}}^{ID_{\Delta}} \bmod p$. 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.

VII. Implementation and Performance Evaluation

We have implemented the UPPRESSO prototype, and evaluated its performance by comparing with the original OIDC which only prevents RP-based identity linkage, and SPRESSO which only prevents IdP-based login tracing.

A. Implementation

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

The implementation of IdP only needs small modifications on the existing OIDC implementation. The UPPRESSO IdP is implemented based on MITREid Connect [40], an open-source OIDC Java implementation certificated by the OpenID Foundation [41]. We add 3 lines of Java code to calculate PID_U, 26 lines to the function of dynamic registration to support PID_{RP} registration, i.e., checking PID_{RP} and adding a signature and validity period in the response. The calculations of PID_{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 of JavaScript code, to provide the functions in Steps 1.3, 1.5, 2.1, 3.2 and 3.5. The cryptographic computation, e.g., $Cert_{RP}$ verification and PID_{RP} negotiation, is implemented based on jsrsasign [42], an efficient JavaScript cryptographic library. This chrome extension requires permissions *chrome.tabs* and *chrome.windows* to obtain the RP's URL from the browser's tab, and *chrome.webRequest* to intercept, block, modify requests to the IdP or RP [43]. 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 `Access-Control-Allow-Origin` header 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 PID_{RP} registration, while the other for $Account$ calculation. 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 [40] and SPRESSO [9], where MITREid Connect provides open-source Java implementations [40] of IdP and RP's SDK, and SPRESSO provides the JavaScript implementations based on node.js for all entities [9]. We implemented a Java RP based on Spring Boot framework for UPPRESSO and MITREid Connect, by integrating the corresponding SDK respectively. 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 1.1-3.2 in Figure 3), RP (and user) constructing and transmitting the request to IdP; **Identity proof generation** (Steps 3.3 and 3.4-generation in Figure 3), IdP generating identity proof (no user authentication); **Identity proof extraction** (Steps 3.4-transmission and 3.5 in Figure 3), RP server extracts the identity proof from the IdP; and **Identity proof verification** (Steps 4.1 and 4.2 in Figure 3), RP verifying and parsing the identity proof.

Results. The evaluation results are provided in Figure 4. The overall processing times are 113 ms, 308 ms and 254 ms for MITREid Connect, SPRESSO and UPPRESSO, respectively. The details are as follows.

In the requesting, UPPRESSO requires the user and RP to perform 2 and 2 modular exponentiations respectively for RP identifier transformation and complete PID_{RP} registration at the IdP, which totally needs 144 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

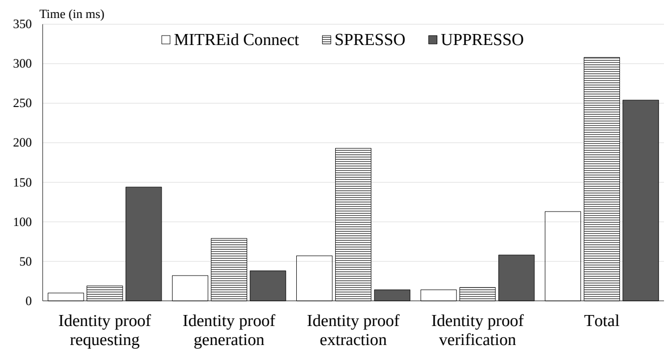

Fig. 4: The Evaluation.

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 254 ms in UPPRESSO.

In the identity proof extraction, UPPRESSO only needs 14 ms where the Chrome extension relays the identity proof to the RP server directly. MITREid Connect requires the IdP to send the identity proof to the RP's web page which then sends the proof to the 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 proofs from the 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

This section lists extended discussions of UPPRESSO and future works.

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 PID_{RP} registration (Step 2.1) and identity proof generation (Step 3.2) 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 calculated 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 login 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 needs to integrate existing 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 1.3, 1.5, 2.1, 3.2 and 3.5; 3) then postMessages are adopted to exchange messages between window A and B for Steps 1.2, 1.3, 1.4, 2.3, 3.1 and 3.5. 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 [44] and user's identifier check [9]. With certificate transparency [44], 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 [9].

IX. Related Works

Various SSO protocols have been proposed, such as, OIDC, OAuth 2.0, SAML, Central Authentication Service (CAS) [45] and Kerberos [46]. These protocols are widely adopted in Google, Facebook, Shibboleth project [47], Java applications and etc. And, plenty of works have been conducted on privacy protection and security analysis for SSO systems.

A. Privacy protection for SSO systems.

Privacy-preserving SSO systems. As suggested by NIST [7], SSO systems should prevent both RP-based identity linkage and IdP-based login tracing. The pairwise user identifier is adopted in SAML [3] and OIDC [1], and only prevents RP–based identity linkage; while SPRESSO [9] and BrowserID [8] only prevent IdP-based login tracing. BrowserID is adopted in Persona [14] and Firefox Accounts [16], however an analysis on Persona, found IdP-based accessing tracing could still succeed [8], [48]. UPPRESSO prevents both the RP-based identity linkage and IdP-based login tracing, and could be integrated into OIDC which has been formally analyzed [21]. Moreover, OAuth and OIDC allow users to determine the scope of attributes exposed to the RP [28], [36].

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 [49]. The notion of anonymous SSO was formalized [50] in 2013. And, various cryptographic primitives, such as group signature, zero-knowledge proof and etc., were adopted to design anonymous SSO schemes [50], [51]. Anonymous SSO schemes are designed for the anonymous services, and not applicable to common services which need user identification.

B. Security analysis of SSO systems.

Formal analysis on SSO standards. The SSO standards (e.g., SAML, OAuth and OIDC) have been formally analyzed. Fett et al. [20], [21] conducted the formal analysis on OAuth 2.0 and OIDC standards based on an expressive Dolev-Yao style model [48], and proposed two new attacks, i.e., 307 redirect attack and IdP Mix-Up attack. When the 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, which breaks the confidentiality of the identity proof. Fett et al. [20], [21] 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. [19] formally analyzed SAML and its variant proposed by Google, and found that Google's variant of SAML doesn't set RP's identifier in the identity proof, which breaks RP designation.

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.

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. [23] analyzed the SSO implementations of Google and Facebook from the view of the browser relayed traffic, and found logic flaws in IdPs and RPs to break the confidentiality and integrity of identity proof. An authentication flaw found in Google Apps [24], allowed a malicious RP to hijack a user's authentication attempt and inject the malicious code to steal the cookie (or identity proof) for the targeted RP, breaking the confidentiality. The integrity has been tampered with in SAML, OAuth and OIDC systems [22], [23], [27]–[29], [31], due to various vulnerabilities, such as XML Signature wrapping (XSW) [22], RP's incomplete verification [23], [27], [28] and etc. And, a dedicated, bidirectional authenticated secure channel was proposed to improve the confidentiality and integrity of identity proof [52]. The vulnerabilities were also found to break the RP designation, such as the incorrect binding at IdPs [32], insufficient verification at RPs [27], [28]. An automatical tool, SSOScan [25], has been designed to detect vulnerabilities at RPs, which could be exploited to break the confidentiality and RP designation of identity proof.

Analysis on mobile SSO systems. In mobile SSO systems, the IdP App, IdP SDK (i.e. an encapsulated WebView) or system browser are adopted to redirect identity proof from IdP App to RP App. However, none of them was trusted to ensure the identity proof only be sent to the designated RP, as WebView and system browser cannot authenticate RP App while the IdP App may be repackaged in Android [26]. Ye et al. [53] performed an 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. 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 RP designation of identity proof [26], [30], [33], [34].

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 254 ms for a user to log into an RP.

References

[1] 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.

[2] Dick Hardt, "The OAuth 2.0 Authorization Framework," *RFC*, vol. 6749, pp. 1–76, 2012.

[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] 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.

[7] Paul A Grassi, M Garcia, and J Fenton, "NIST special publication 800-63c digital identity guidelines: Federation and assertions," *National Institute of Standards and Technology, Los Altos, CA*, 2017.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Rakesh Agrawal, Alexandre V. Evfimievski, and Ramakrishnan Srikant, "Information sharing across private databases," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, 2003, pp. 86–97.

[13] Emiliano De Cristofaro and Gene Tsudik, "Practical private set intersection protocols with linear complexity," in *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, 2010, pp. 143–159.

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

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

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

[17] 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.

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

[19] 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.

[20] Daniel Fett, Ralf Küsters, and Guido Schmitz, "A comprehensive formal security analysis of OAuth 2.0," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria*, 2016, pp. 1204–1215.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[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] 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.

[36] 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.

[37] 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.

[38] 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.

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

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

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

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

[43] Google, "Declare Permissions," https://developer.chrome.com/extensions/declare_permissions.

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

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

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

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

[48] 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.

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

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

[51] 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.

[52] 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.

[53] 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.