

# 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, computes 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 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 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 identity 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. In such login processes, 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 process, 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 identity 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 IdP and RP through the user in an SSO login process, the RP will allow the user to log in as the account  $PID_U$ , 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*  $PID_U$ s 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 process  $PID_U$  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  $PID_U$ s 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  $PID_{RPs}$  to a particular RP or associate them together, and collusive RPs (c) cannot link  $PID_U$ s to a particular user or associate them together, (d) nor link *Accounts* 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 (OIDC)

As an extension of OAuth 2.0 to support user authentication, OIDC [1] is one of the most prominent SSO authentication protocols. Same as other SSO protocols [15], OIDC involves three entities, i.e., *users*, *identity provider (IdP)*, and *relying parties (RPs)*. Both users and RPs have to register at the IdP with identifiers such as  $ID_U$  and  $ID_{RP}$  (or  $PID_{RP}$  in some schemes) and the corresponding account information such as credential, RP's endpoint (e.g., URLs), etc. IdP is assumed to maintain these attributes securely. In an OIDC authentication session, 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 the user attributes in the identify proof generated by the IdP for the 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'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. The IdP authenticates the user based on her  $ID_U$  and credential, sets the  $PPID$  (i.e., a privacy-preserving pseudo-identifier) for the user-RP pair (denoted by  $ID_U$  and  $ID_{RP}$ ), generates an identity proof containing  $PPID$ ,  $ID_{RP}$  and the user attributes, and returns the identity proof to the endpoint registered by the RP.

**OIDC implicit flow.** OIDC supports three different SSO authentication processes (called 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 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 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 user's  $PPID$  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 protocol flow of the original OIDC implicit flow is shown in Figure 1, consisting of 7 steps: when a user attempts

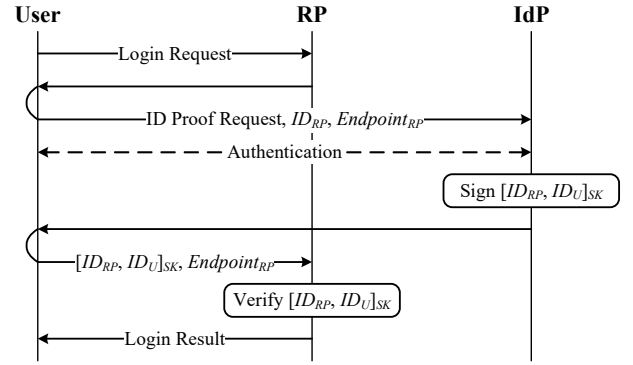


Fig. 1: The implicit flow of OIDC.

to log in to an RP (Step 1), the RP constructs an identity proof request and returns it to the user, which gets redirected to the IdP (Step 2). The request contains  $ID_{RP}$ , RP's endpoint and a set of requested user attributes. If the user has not yet been authenticated, the IdP initiates an authentication process (Step 3) to authenticate the user. For a successfully authenticated user, the IdP generates an id token (Step 4) and returns it to the registered endpoint of the RP (Step 5). If no registered endpoint is found for that RP, the IdP will generate a warning to notify the user about potential identity proof leakage. Once the RP receives the id token, it verifies its signature (Step 6) and makes the authentication decision (Step 7).

**RP dynamic registration.** OIDC has a dynamic registration mechanism [17] that allows an RP to update its information at the IdP dynamically. When an RP first registers at the IdP, it obtains a registration token, with which the RP can invoke a dynamic registration process to update its information (e.g., the endpoint). After a successful dynamic registration, the RP obtains a new and unique  $ID_{RP}$  from the IdP. UPPRESSO leverages this function and slightly modify the dynamic registration process to enable *RP identifier refreshing*, which allows each RP to generate different privacy-preserving identifiers ( $PID_{RPs}$ ) and register them at the IdP.

### B. Discrete Logarithm Problem

Based on the discrete logarithm problem, UPPRESSO design the identifier-transformation functions  $\mathcal{F}_{ID_{RP} \rightarrow PID_{RP}}$  and  $\mathcal{F}_{ID_U \rightarrow PID_U}$  to generate privacy-preserving user identifier (e.g.  $PID_U$ ) and RP identifier (e.g.  $PID_{RP}$ ), respectively. Here, we briefly review 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  $y = g^x \bmod p$ . 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 [18]), which is called the 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).

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 [1],  $ID_U$  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 login 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 identifier 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  $Accounts$  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  $Account$  from dynamic  $PID_U$ s 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

TABLE I: Three functions in privacy-preserving SSO.

Solution	$\mathcal{F}_{ID_U \mapsto PID_U}$	$\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$	$\mathcal{F}_{PID_U \mapsto Account}$
PPID	$Map[ID_U, ID_{RP}]$ (✓)	$ID_{RP}$ (×)	$PID_U$ (✓)
SPRESSO	$ID_U$ (×)	$Enc(ID_{RP}    nonce)$ (✓)	$ID_U$ (×)
BrowserID	$ID_U$ (×)	$\perp$ (✓)	$ID_U$ (×)
UPPRESSO	$PID_{RP}^{ID_U}$ (✓)	$ID_{RP}^{N_U N_{RP}}$ (✓)	$PID_U^T$ (✓)

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 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 identifier 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 [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 for different RPs, and  $Account = PID_U$ .

SPRESSO and BrowserID prevent only IdP-based login 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 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  $Account = ID_U$ .

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

UPPRESSO 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 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 vulnerabilities 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 computed as follows, where  $r$  is a random number ( $1 < r < q$ ) generated by the IdP.

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

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.

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

$\mathcal{F}_{ID_{RP} \mapsto PID_{RP}}$  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  $PID_{RP}$  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  $PID_U$ s 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 2, 3, and 4. So for a user's multiple logins at an RP, the RP extracts an identical  $Account$ .

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

$\mathcal{F}_{PID_U \mapsto Account}$  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  $PID_U$ s at different RPs.

**Trapdoor user identification.** In a user's multiple logins, the RP expresses different  $PID_U$ s and have corresponding  $T$ s, so that always derives the identical  $Account$ . The comprehensive design of identifier-transformation functions prevents collusive RPs from linking a user's  $PID_U$ s and  $Accounts$  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 login 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

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
$ID_{RP}$	RP's unique identifier, $ID_{RP} = g^r \bmod p$ .	Long-term constant
$Cert_{RP}$	An RP certificate.	Long-term constant
$ID_U$	User's unique identifier.	Long-term constant
$Account$	User's identifier at an RP, $Account = ID_{RP}^{ID_U} \bmod p$ .	Long-term constant
$PID_{RP}$	RP's privacy-preserving identifier, $PID_{RP} = ID_{RP}^{N_U N_{RP}} \bmod p$ .	One-time
$PID_U$	User's privacy-preserving identifier, $PID_U = PID_{RP}^{ID_U} \bmod p$ .	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 $N_{RP}$ , $Y_{RP} = ID_{RP}^{N_{RP}} \bmod p$ .	One-time
$T$	A trapdoor, $T = (N_U N_{RP})^{-1} \bmod q$ .	One-time

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 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 (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  $SK$  secret, 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:

- RP sends a registration request to the IdP, including the RP's endpoint (e.g., URL) for receiving the identity proof.
- IdP chooses a unique random number  $r$  ( $1 < r < q$ ), computes  $ID_{RP} = g^r \mod p$ , signs  $[ID_{RP}, endpoint, *]$  using  $SK$ , where  $*$  is supplementary information such as the RP's common name, and returns  $Cert_{RP} = [ID_{RP}, endpoint]_{SK}$  to the RP, where  $[ ]_{SK}$  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 in at an RP, the SSO login is initiated. As shown in Figure ??, the SSO login consists of four phases, RP identifier transformation, RP identifier refreshing,  $PID_U$  generation and  $Account$  calculation. In the RP identifier transforming, the user and RP negotiate  $PID_{RP} = ID_{RP}^{N_U N_{RP}} \mod p$ . In the RP identifier refreshing, the user registers  $PID_{RP}$  at the IdP. In the  $PID_U$  generation, the IdP calculates  $PID_U = PID_{RP}^{ID_U} \mod p$  and signs the identity proof. And in the  $Account$  calculation, the RP derives  $Account = PID_U^{(N_U N_{RP})^{-1} \mod q} \mod p$  after verifying the identity proof, and allows the user to log in as  $Account$ .

### C. UPPRESSO Protocol flow

Figure 3 shows the SSO login protocol of UPPRESSO. 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:

- 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  $Y_{RP} = ID_{RP}^{N_{RP}} \mod p$ ; and sends  $Y_{RP}$  with  $Cert_{RP}$  to the user.
- 1.3 The user checks the  $Cert_{RP}$ , extracts  $ID_{RP}$  from the valid  $Cert_{RP}$ , chooses a random  $N_U$  ( $1 < N_U < q$ ) to calculate  $PID_{RP} = Y_{RP}^{N_U} \mod p$  (Step 2.1.3); and sends  $N_U$  to the RP (Step 2.1.4).
- 1.4 The RP calculates  $PID_{RP}$  with  $N_U$  and  $Y_{RP}$ , derives the trapdoor  $T = (N_U N_{RP})^{-1} \mod q$  (Step 2.1.5); and acknowledges the negotiation.

During the process, the user will halt the login, if the  $Cert_{RP}$  is invalid. The user verifies that  $PID_{RP} \neq g^0 \mod p$ ; if

$PID_{RP} = g^0 \mod p$ , it means that  $N_U = 0 \mod q$  or  $N_{RP} = 0 \mod q$  and then  $PID_U = g^{0 \cdot ID_U}$  is constant for all users. This case appears only if  $N_U = q$  or  $N_{RP} = q$ .

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

- 2.1 The user creates a one-time endpoint to hide the RP's endpoint from the IdP, and sends the RP identifier refreshing request  $[Reg, PID_{RP}, Hash(Y_{RP}, N_U), \text{one-time endpoint}]$  to the IdP.
- 2.2 The IdP authenticates the user if necessary (Step 3); 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 not used with other valid 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  $SK$ .
- 2.3 The user accepts  $RegRes$  if  $PID_{RP}$  is correct and the valid signature by the IdP, and forwards the registration result to the RP (Step 2.2.4).
- 2.4 The RP checks  $Sig_{SK}$  and  $RegMes$ , and accepts  $RegRes$  only when  $Sig_{Reg}$  is valid,  $Hash(Y_{RP}, N_U)$  and  $PID_{RP}$  are 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  $PID_{RP}$ . Here,  $Hash(Y_{RP}, N_U)$  is attached as

The IdP check  $PID_{RP}$  is unique in unexpired ones, otherwise, this is might be accepted by other RP (see Section xxx for details).

The RP checks if  $Hash(Y_{RP}, N_U)$  matches, to ensure this is signed for it (not for other RPs). More details are in xxxxx

**ID proof 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.

- 3.1 The RP uses  $PID_{RP}$  and the endpoint to construct an identity proof request.
- 3.2 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).
- 3.3 The IdP authenticates the user if necessary (Step 3); The IdP checks whether  $PID_{RP}$  and the one-time endpoint have been registered and unexpired, calculates  $PID_U = PID_{RP}^{ID_U} \mod p$ .
- 3.4 The IdP 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.



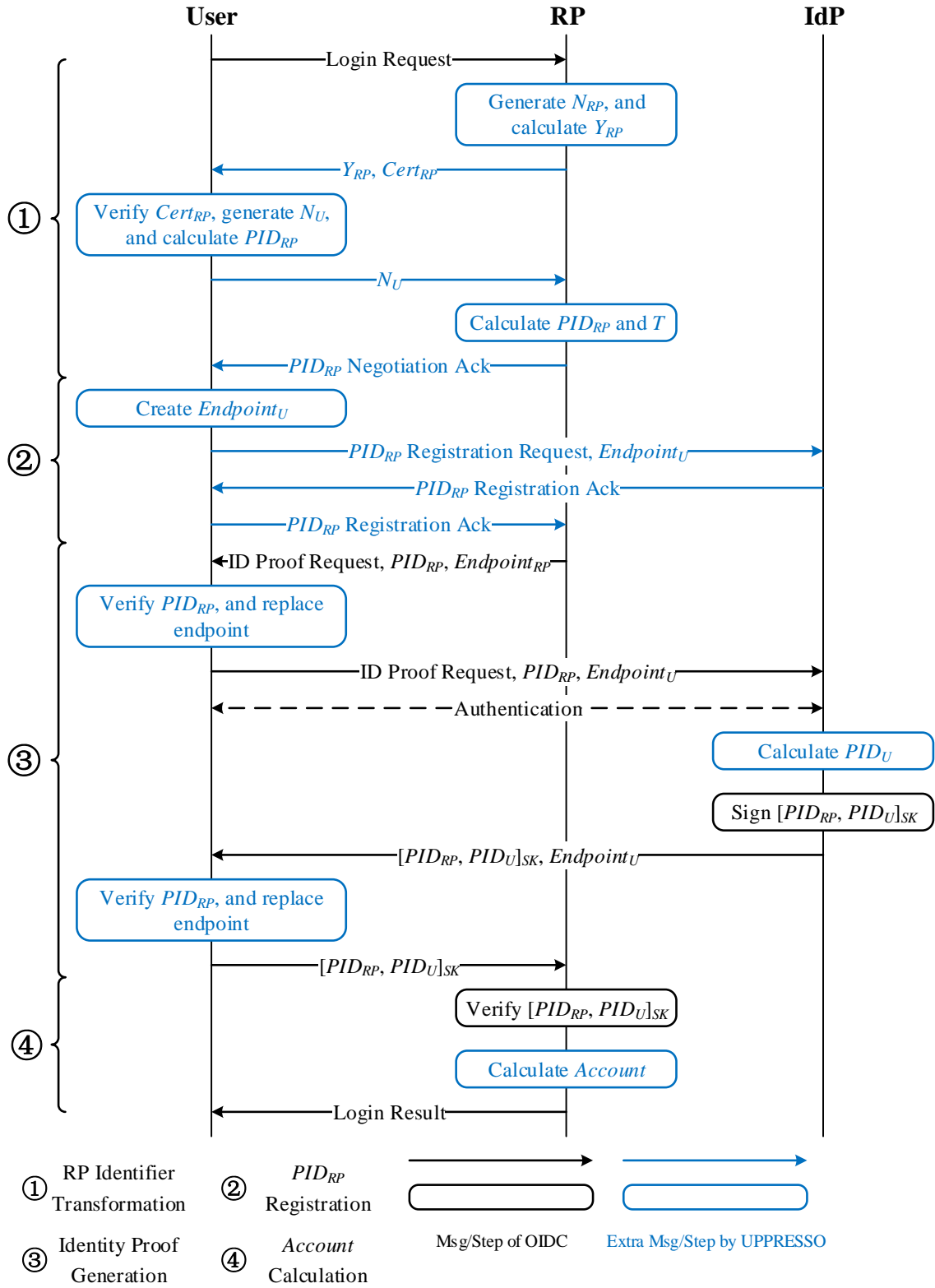


Fig. 3: Process for each user login.

- 3.5 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.

- 4.1 The RP performs the checks on the identity proof, including the valid time, correctness of  $Sig_{IdPProof}$ , and the consistency between  $PID_{RP}$  and the negotiated one.
- 4.2 If all the checks pass, the RP extracts  $PID_U$ , and calculates  $Accout = PID_U^T \bmod p$  (Step 6); and sends the *Success* as the login result to the user (Step 7). If any check fails, the RP returns the *Fail* to the user.

#### D. Compatibility with OIDC

UPPRESSO could be integrated in the traditional SSO systems, to prevent the IdP-based login 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 3. The further analysis, such as integration with the authorization code flow of OIDC, is provided in Section VIII.

First of all, 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 3, 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 [39], 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 [21]. 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-redirectioned 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  $1/q$  ( $\leq 2^{-255}$  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  $Account = PID_U^T = ID_{RP}^{ID_U} \bmod p$  for a user's multiple logins, as shown in Equation 5; and one  $Account$  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 login tracing and RP-based identity linkage.

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

*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  $PID_{RPs}$  are independent among the logins, guaranteed by the secure random number generators that used to generate one-time endpoints and  $N_U$ s at the correct user, and  $N_{RPs}$  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  $PID_{RP}^{ID_U} \bmod p$  according to  $\mathcal{F}_{ID_U \mapsto PID_U}$ , and further transformed to  $g^{r * N_U * N_{RP} * ID_U \bmod q} \bmod p$  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  $ID_{RP}^{ID_U} \bmod p$  according to Equation 5, and further transformed to  $g^{r * ID_U \bmod q} \bmod p$  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  $Account = PID_U^T \bmod p$  where  $T$  is a random value known to RP and independent with  $ID_U$ .

*Any RP cannot directly derive  $ID_U$  from  $PID_U$ s and  $Accounts$  obtained in multiple logins.* All these  $Accounts$  are equal, while any  $PID_U$  (e.g.,  $PID_{U1}$ ) can be computed from any other  $PID_U$  (e.g.,  $PID_{U2}$ ),  $PID_{U1} = PID_{U2} *$

$Account^{N_{U1} * N_{RP1} - N_{U2} * N_{RP2}} \bmod p$ , where  $N_{U2}$ ,  $N_{RP2}$ ,  $N_{U1}$  and  $N_{RP1}$  are values known to the RP and independent with  $ID_U$ .

The collusive RPs cannot directly associate a user's Accounts and  $PID_U$ 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  $PID_U$ s either, due to the unknown  $rs$ .

Any malicious RP cannot derive  $ID_U$  and collusive RPs cannot associate  $PID_U$ s (Accounts), by manipulating  $N_{RPs}$ . A malicious RP may attempt to manipulate  $N_{RPs}$  in one or multiple logins to make the generated  $PID_U$ s or Accounts be vulnerable for deriving  $ID_U$ , and the collusive RPs may attempt to manipulate  $N_{RPs}$  cooperatively to make a user's  $PID_U$ s or Accounts be associated at these RPs and then to associate a user's multiple logins. Here,  $N_{RPs}$  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  $\mathcal{F}_{ID_{RP} \rightarrow PID_{RP}}$ , 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. The RPs may collude with the users and attempt to associate a victim user's Accounts at the different RPs based on the relation among the Accounts 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  $ID_\Delta$  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  $ID_U$ s 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  $ID_\Delta$ ) for this association.

The malicious RPs and users cannot associate the victim user by manipulating  $N_{RPs}$  or  $N_U$ s. The malicious RPs may manipulate  $N_{RPs}$  while the colluded users may manipulate  $N_U$ s, 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  $PID_U$ s and Accounts due to the independent  $N_U$ s 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 login 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 [40]. 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 [41], an open-source OIDC Java implementation certificated by the OpenID Foundation [42]. 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 jsrsasn [43], 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 [44]. 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,

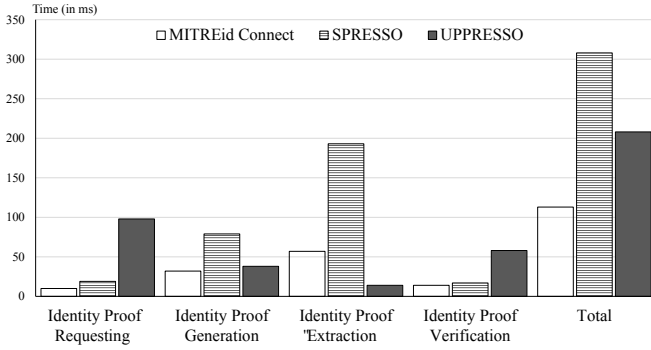


Fig. 4: The Evaluation.

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 [41] and SPRESSO [9], where MITREid Connect provides open-source Java implementations [41] of IdP and RP’s SDK, and SPRESSO provides the JavaScript implementations based on node.js for all entities [9]. 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 4. 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 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  $PID_{RPs}$  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  $PID_{RPs}$  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  $Y_{RPs}$  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 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 [45] and user's identifier check [9]. With certificate transparency [45], 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) [46] and Kerberos [47]. These protocols are widely adopted in Google, Facebook, Shibboleth project [48], 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. [23] 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 [24]. The integrity has been tampered with in

SAML, OAuth and OIDC systems [22], [23], [25], [27]–[29], [31], due to various vulnerabilities, such as XML Signature wrapping (XSW) [22], RP's incomplete verification [25], [27], [28] and etc. And, a dedicated, bidirectional authenticated secure channel was proposed to improve the confidentiality and integrity of identity proof [49]. The vulnerabilities were also found to break the designation, such as the incorrect binding at IdPs [32] and insufficient verification at RPs [25], [27], [28].

**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 [26]. 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 [50], and exploited by malicious RP's application to tamper with the communication between the WebView and IdP server [30]. 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 [26], [30], [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. [20], [21] 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. [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. The formal analysis of Google's SAML implementation [19] found a vulnerability which could be exploited by the malicious RPs to reuse the identity proof at other RPs. Ye et al. [51] 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 [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 a formal analysis on Persona, found IdP-based accessing tracing could still succeed [8], [52]. UPRESSO prevents both the RP-based identity linkage and IdP-based accessing tracing, and could be integrated into OIDC which has been formally analyzed [21]. Moreover, OAuth and OIDC allow users to determine whether to provide an attribute to the RPs or not [28], [35].

**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 [53]. The notion of anonymous SSO was formalized [54] in 2013. And, various cryptographic primitives, such as group signature, zero-knowledge proof and etc., were adopted to design anonymous SSO schemes [54], [55]. 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,  $\mathcal{F}_{ID_{RP} \rightarrow PID_{RP}}$  to prevent curious IdP from obtaining the identifier of the visiting RP,  $\mathcal{F}_{ID_U \rightarrow PID_U}$  to prevent collusive RPs from linking a user, and  $\mathcal{F}_{PID_U \rightarrow Account}$  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] 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, "Draft nist special publication 800-63c 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] 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.
- [36] 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.
- [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] Whitfield Diffie and Martin E. Hellman, “New directions in cryptography,” *IEEE Trans. Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
- [40] Elaine Barker, “Recommendation for key management part 1: General (revision 4),” *NIST special publication*, vol. 800, no. 57, pp. 1–160, 2016.
- [41] “MITREid connect /openid-connect-java-spring-server,” <https://github.com/mitreid-connect/OpenID-Connect-Java-Spring-Server>, Accessed August 20, 2019.
- [42] “Openid foundation,” <https://openid.net/certification/>, Accessed August 20, 2019.
- [43] “jsrsasign,” <https://kjur.github.io/jsrsasign/>, Accessed August 20, 2019.
- [44] Google, “Declare Permissions,” [https://developer.chrome.com/extensions/declare\\_permissions](https://developer.chrome.com/extensions/declare_permissions).
- [45] Ben Laurie, Adam Langley, and Emilia Käsper, “Certificate transparency,” *RFC*, vol. 6962, pp. 1–27, 2013.
- [46] 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.
- [47] 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.
- [48] “The shibboleth project,” <https://www.shibboleth.net>, Accessed July 30, 2019.
- [49] 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.
- [50] 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.
- [51] 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.
- [52] 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.
- [53] 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.
- [54] 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.
- [55] Jinguang Han, Liquan 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.