UPPRESSO: Untraceable and Unlinkable Privacy-PREserving Single Sign-On Services

***Abstract*—Single sign-on (SSO) protocols such as OpenID Connect (OIDC), allow a user to maintain only the credential for an identity provider (IdP) to log into multiple relying parties (RPs). However, SSO introduces privacy threats, as (*a*) a curious IdP could trace a user's visits to RPs, and (*b*) colluding RPs could learn a user's online profile by linking her identities across these RPs. This paper presents a privacy-preserving SSO scheme, called UPPRESSO, to protect a user's online profile against both (*a*) an honest-but-curious IdP and (*b*) malicious RPs colluding with other users. An identity-transformation approach is proposed to generate untraceable ephemeral pseudo-identities for an RP and a user in a login, from which the visited RP derives a permanent account for the user, while these identity transformations provide unlinkability among the logins visiting different RPs. We built the UPPRESSO prototype providing OIDC-compatible SSO services, accessed from a commercial-off-the-shelf browser without plug-ins or extensions. The extensive evaluations show that it fulfills the security and privacy requirements of SSO with reasonable overheads.**

I. Introduction

Single sign-on (SSO) protocols such as OpenID Connect (OIDC) [1], OAuth 2.0 [2], and SAML [3], [4], are widely deployed for identity management and authentication. SSO allows a user to log into a website, known as the *relying party* (RP), using her account registered at another trusted web service, known as the *identity provider* (IdP). An RP delegates its user identification and authentication to the IdP, which issues an *identity token* (such as "id token" in OIDC and "identity assertion" in SAML) for a user to visit the RP. For instance, when an OIDC user initiates a login to visit an RP, the target RP constructs an identity-token request with its identity (denoted as ID_{RP}) and redirects it to a trusted IdP. After authenticating this user, the IdP issues an identity token binding the identities of the authenticated user and the target RP (i.e., both ID_U and ID_{RP}), which is returned to the user and then forwarded to the RP. Finally, the RP verifies the identity token to determine whether the token holder is allowed to log in. Thus, a user maintains only one credential for user authentication to the IdP, instead of multiple credentials for visiting different RPs.

The wide adoption of SSO raises concerns on user privacy, because it facilitates the tracking of a user's login activities by interested parties [5], [6], [7], [8]. To issue identity tokens, an IdP should know the target RP to be visited by a user. So a curious IdP could trace a user's all login activities over time [7], [6], called *IdP-based login tracing* in this paper. Another privacy threat arises from the fact that RPs learn a user's identity from the identity tokens they receive. If an identical user identity is enclosed in the tokens for a user to visit different RPs, colluding RPs could link the logins across these RPs to learn the user's online profile [8], [9]. This threat is called *RP-based identity linkage*.

While preventing different privacy threats (i.e., IdP-based login tracing, RP-based identity linkage, or both), privacy-preserving SSO schemes [8], [5], [7], [10], [6], [11], [12] aim to provide services for users with a commercial-off-the-shelf (COTS) browser without plug-ins or extensions. This requires that, except the credentials for user authentication to the IdP, there is no long-term secret processed in the browser across different logins. We analyze existing privacy-preserving SSO solutions in Section II, and compare them with privacy-preserving identity federation [13], [14], [15], [16], [17], [18] where more privacy threats are eliminated at the cost of a browser plug-in or extension.

In this paper, we present UPPRESSO, an Untraceable and Unlinkable Privacy-PREserving Single Sign-On protocol. It proposes *identity transformations* in SSO, and integrates them in popular OIDC services accessed from COTS browsers. In UPPRESSO, an RP and a user transform ID_{RP} into ephemeral PID_{RP}, which is sent to a trusted IdP to transform ID_U into an ephemeral user pseudo-identity PID_U. Then, the identity token issued by the IdP binds only PID_U and PID_{RP}, instead of ID_U and ID_{RP}. On receiving the token, the RP transforms PID_U into an account unique at each RP but identical across multiple logins visiting this RP.

UPPRESSO prevents both IdP-based login tracing and RP-based identity linkage, while existing privacy-preserving SSO schemes address only one of them [7], [6], [5], [10], [12] or need another fully-trusted server in addition to the IdP [11], [6]. That is, it offers an easy-to-deploy option to enhance the protections of user privacy for SSO services.

Meanwhile, compared with privacy-preserving identity federation [19], [13], [16], [14], [17], [15], [18] where a browser plug-in or extension is required to handle a long-term user secret, UPPRESSO works with COTS browsers without plug-ins or extensions, but collusive attacks by the IdP and RPs still break user privacy. Our strategy provides a convenient choice for some users not installing browser plug-ins or extensions.

Our contributions are as follows.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.[23|24]xxxx
www.ndss-symposium.org

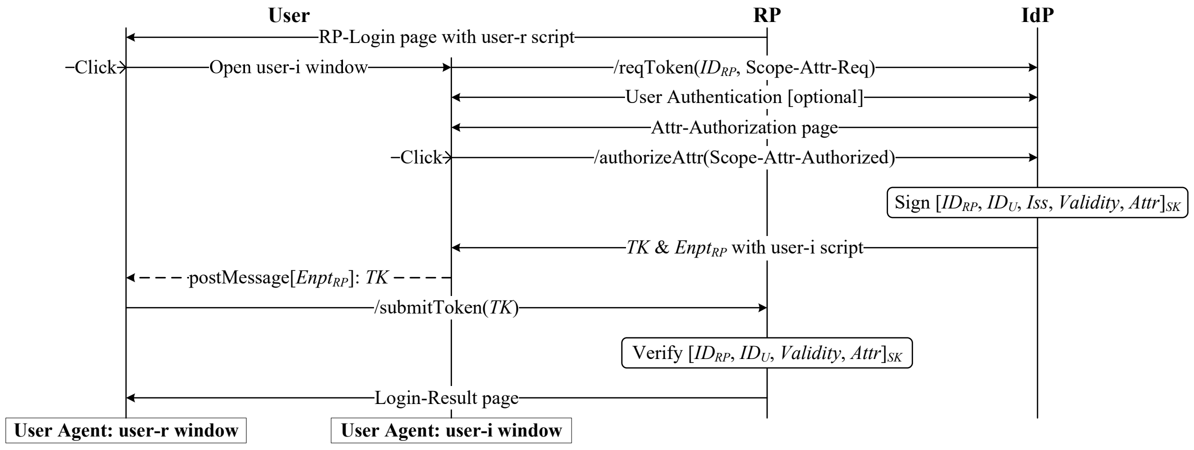

User
RP
IdP
RP-Login page with user-r script
Click
Open user-i window
/reqToken(ID_{RP}, Scope-Attr-Req)
User Authentication [optional]
Attr-Authorization page
Click
/authorizeAttr(Scope-Attr-Authorized)
Sign [ID_{RP}, ID_U, Iss, $Validity$, $Attr$]$_{SK}$
TK & $Enpt_{RP}$ with user-i script
postMessage[$Enpt_{RP}$]: TK
/submitToken(TK)
Verify [ID_{RP}, ID_U, $Validity$, $Attr$]$_{SK}$
Login-Result page
User Agent: user-r window
User Agent: user-i window

Fig. 1. The implicit SSO login flow of OIDC with pop-up UX

- We proposed a novel identity-transformation approach for privacy-preserving SSO and designed identity-transformation algorithms with desirable properties.
- We developed the UPPRESSO protocol based on the identity transformations in an OIDC-compatible way, and proved that it satisfies the security and privacy requirements of SSO services.
- We implemented a prototype of UPPRESSO on top of an open-source OIDC implementation. Through performance evaluations, we confirmed that UPPRESSO introduces reasonable overheads.

We explain the privacy threats in SSO and present related works in Section II. The identity-transformation approach is proposed in Section III, followed by the detailed designs of UPPRESSO in Section IV. Security and privacy are analyzed in Section V. We present the prototype implementation and evaluations in Section VI, and discuss extended issues in Section VII. Section VIII concludes this work.

II. Background and Related Work

A. OpenID Connect

OIDC is one of the most popular SSO protocols. It supports different login flows: implicit flow, authorization code flow, and hybrid flow (a mix of the other two). These flows differ in the steps for requesting, receiving and forwarding identity tokens, but have common security requirements for tokens. We present our designs in the implicit flow and discuss the support for the authorization code flow in Section VII.

Users and RPs register at an OIDC IdP with their identities and other information such as user credentials (e.g., passwords) and RP endpoints (i.e., the URLs to receive tokens). User operations in the login flows are conducted through a user agent (or browser typically). Figure 1 shows the implicit SSO login flow of OIDC with *pop-up UX* [20], [21], [22], which is recommended to provide an in-context user experience. When a user clicks the SSO login button in a browser window visiting the target RP (called the *user-r* window in this paper), an identity-token request is constructed with the RP's identity and the scope of requested user attributes. It pops up another window (i.e., the *user-i* window), and this request is sent to the IdP. After authenticating the user, the IdP issues an identity token that encloses the (pseudo-)identities of the user and the target RP, the authorized attributes, a validity period, etc. The identity token is then forwarded to the RP's endpoint via the user-i window. Finally, the RP verifies the token and allows the holder to log in as the enclosed user (pseudo-)identity. There are scripts in two browser windows, responsible for communicating with the respective origin servers and also the cross-origin communications using the `postMessage` HTML5 API within the COTS browser.

Alternatively, if a browser works as the user agent of OIDC with only one window, user attributes are authorized in this window when redirected to the IdP [1], [2], which is called *redirect UX.*

B. Privacy-Preserving SSO and Identity Federation

To provide secure SSO services, an IdP signs identity tokens binding the identities of the authenticated user and the target RP [1], [2]. However, a user's login activities are leaked due to these identities: the curious IdP learns the visited RP from the RP identity enclosed in an identity-token request (i.e., IdP-based login tracing), while colluding RPs link a user's logins visiting these RPs by the user identity in the tokens (i.e., RP-based identity linkage).

Privacy-preserving SSO. Existing solutions propose various mechanisms to hide (*a*) RP identities from the IdP or/and (*b*) user identities from the visited RPs. Table I compares these solutions, and only UPPRESSO provides authentication services for RPs accessed from COTS browsers while preventing both of the two privacy threats and not introducing extra trusted servers in addition to the honest IdP.

Pairwise pseudonymous identifiers (PPIDs) are specified [1], [4] and recommended [5] in OIDC to protect user privacy against colluding RPs. An IdP assigns a unique PPID for a user to log into every RP and encloses it in identity tokens, so colluding RPs cannot link the user by PPIDs. It does not

TABLE I
PRIVACY-PRESERVING SOLUTIONS OF SSO AND IDENTITY FEDERATION

	Solution	Privacy Threat♯		Extra Trusted Server†	Browser Plug-in or Extension‡
		IdP-based Login Tracing	RP-based Identity Linkage		
SSO	OIDC w/ PPID [5]	○	●	●	●
	BrowserID [7]	●	○	●	○
	SPRESSO [6]	●	○	◐^{1}	●
	POIDC [12], [10]	●	○^{2}	●	●
	MISO [11]	●	●	○	●
	UP-SSO [23]	●	●	●	○
Identity Federation	PRIMA [19]	●	○	●	○
	PseudoID [13]	●	●	○	○
	Opaak [16]	●	●	●	○
	PP-IDF [14], [17], [15]	●	●	●	○
	Fabric Idemix [18]	●	●	●	○
SSO	UPPRESSO	●	●	●	●

♯ **Privacy Threat**: ● prevented, ○ not prevented. † **Extra Trusted Server**: ● no, ○ required. ‡ **Browser Plug-in or Extension**: ● no, ○ required.

1. SPRESSO assumes a malicious IdP (but honest IdPs in others), and then a trusted forwarder server is needed to decrypt the RP identity and forward tokens to the RP.
2. A variation of POIDC [12] proposes to also hide a user's identity in the commitment and prove this to the IdP in zero-knowledge, but it takes seconds to generate such a zero-knowledge proof (ZKP) even for a powerful server [24], [25], [26], which is impracticable for a user agent in SSO systems.

prevent IdP-based login tracing because the IdP needs the RP's identity to assign PPIDs.

Other privacy-preserving SSO schemes prevent IdP-based login tracing but leave users vulnerable to RP-based identity linkage, due to the unique user identities enclosed in identity tokens. For example, in BrowserID [7] after authenticating a user, an IdP issues a "user certificate" binding the user's identity to an ephemeral public key. The user then uses the private key to sign a subsidiary "identity assertion" that binds the target RP's identity and sends both of them to the RP. In SPRESSO an RP creates a one-time tag (or ephemeral pseudo-identity) for each login [6], and in POIDC [12], [10] a user sends an identity-token request with a hash commitment on the target RP's identity (but not the RP identity), which are enclosed in identity tokens along with the user's unique identity.

MISO [11] decouples the calculation of PPIDs from an honest IdP, to an extra fully-trusted mixer server that calculates a user's PPID based on ID_U, ID_{RP} and a secret after it receives the authenticated user's identity from the IdP. MISO prevents both RP-based identity linkage for the RPs receives only PPIDs, and IdP-based login tracing for ID_{RP} is disclosed to the mixer but not the IdP. It protects a user's online profile against even collusive attacks by the IdP and RPs, but the mixer server could track a user's all login activities. UP-SSO [23] runs a fully-trusted Intel SGX enclave on the user side, which is remotely attested by the IdP and then receives the secret to generate PPIDs for a user, and then both of two privacy threats are eliminated.

Privacy-preserving identity federation. Identity federation enables a user registered at a trusted IdP to be accepted by other parties, potentially with different accounts, but browser plug-ins or extensions are always needed to access such services. Thus, although the term "single sign-on (SSO)" was used in some schemes [13], [16], [14], [27], [28], [29], this paper refers to them as *identity federation* to emphasize this difference.

In PRIMA [19], an IdP signs a credential that binds user attributes and a verification key. Using the signing key, the user authenticates herself and provides selected attributes to RPs. This verification key works as the user's identity and exposes her to RP-based identity linkage. PseudoID [13] introduces another trusted server in addition to the IdP, to blindly sign [30] a token that binds a long-term user secret and a pseudonym. The user then unblinds this token, and then authenticates herself to an RP using the secret.

Several schemes [16], [18], [17], [15], [14] are designed based on anonymous credentials [31], [32], [33]. For instance, the IdP signs anonymous credentials in Opaak [16], UnlimitID [15], U-Prove [17], and EL PASSO [14], each of which binds a long-term user secret. Then a user proves ownership of the anonymous credentials using her secret. Similarly, Fabric [18] integrates Idemix anonymous credentials [32] for unlinkable pseudonyms.

Some privacy-preserving identity federation solutions [13], [14], [15], [16], [17], [18] prevent both IdP-based login tracing and RP-based identity linkage, for (a) the RP's identity is not enclosed in the anonymous credentials and (b) the user selects different pseudonyms to visit different RPs. They even protect user privacy against collusive attacks by the IdP and RPs, as these pseudonyms cannot be linked through anonymous credentials [31], [32], [33] when the ownership of these credentials is proved to colluding RPs. This privacy protection depends on the long-term user secrets to mask the relationship of these pseudonyms (or accounts), so that a browser plug-in or extension is always needed to handle the user secrets. In Section VII we particularly discuss the collusion of the IdP and RPs in privacy-preserving SSO.

Anonymous identity federation. Such approaches offer the strongest privacy, where a user visits RPs with pseudonyms that cannot be used to link any two actions. Anonymous identity federation was formalized [27] and implemented using cryptographic primitives such as group signature and ZKP [27], [29], [28]. Extended features including proxy re-verification [29], designated verification [28] and distributed IdP servers [34], are also considered.

These completely-anonymous authentication services only work for special applications and do not support user identification (or account uniqueness) at each RP, a common requirement in most applications.

C. OPRF-based Applications

PrivacyPass and TrustToken [35], [36] adopted the oblivious pseudo-random function (OPRF) protocol of Hash(EC)DH [37], [38] to generate anonymous tokens. A user generates a random number e to blind an unsigned token T into $[e]T$. After authenticating the user, a token server signs $[e]T$ using its private key k and returns $[ke]T$. The user then uses e to convert it into $(T, [k]T)$, which is redeemed when she accesses protected resources (see [35], [36] for details). In this procedure, the token server obliviously calculates a pseudo-random output as an *anonymous* token, because $(T, [k]T)$ and $([e]T, [ke]T)$ cannot be linked.

UPPRESSO employs a similar cryptographic technique.[1] A user selects t to transform ID_{RP} into $PID_{RP} = [t]ID_{RP}$. Then, the IdP uses the user's permanent identity u to calculate $PID_U = [u]PID_{RP} = [ut]ID_{RP}$, which is similar to token signing by the token server in PrivacyPass/TrustToken. Hence, *IdP untraceability* in UPPRESSO, i.e., the IdP cannot link ID_{RP} and $[t]ID_{RP}$, roughly corresponds to *unlinkability of token signing-redemption* in PrivacyPass/TrustToken, i.e., the token server cannot link T and $[e]T$.

UPPRESSO leverages this cryptographic technique in very different ways. First, it works among three parties, unlike the two-party PrivacyPass/TrustToken protocols. UPPRESSO shares the user-selected random number t with the target RP, enabling it to independently derive the user's permanent account, i.e., $Acct = [t^{-1}]PID_U$. This account is "obliviously" determined by the IdP when it calculates the user's pseudo-identity. Second, u and ID_{RP}, which roughly correspond to k and T in PrivacyPass/TrustToken, are assigned as the permanent identities of the user and the RP, respectively, to establish the relationship among identities and accounts. This is very different from existing OPRF-based systems [35], [36], [39], [40], [41], [42], [43], [44], [45], [46] that always use k as a server's secret key. Finally, UPPRESSO leverages randomness of Hash(EC)DH OPRFs to provide *RP unlinkability*, and this property is not utilized in PrivacyPass/TrustToken. It ensures colluding RPs cannot link any logins across RPs, even by sharing their knowledge about the pseudo-identities and permanent accounts in UPPRESSO. This property offers desirable indistinguishability of *different users* visiting colluding RPs. It roughly corresponds to the indistinguishability of *different private keys* for signing anonymous tokens, not considered in PrivacyPass/TrustToken.

[1]We designed and implemented UPPRESSO in 2020, and in 2023 someone reminded us that the identity transformations are very similar to the Hash(EC)DH OPRF.

Although UPPRESSO mathematically employs the same technique in the identity transformations as the Hash(EC)DH OPRF [37], [38], we require more properties of these algorithms. UPPRESSO essentially depends on the *deterministicness* property of *pseudo-random* functions to derive a permanent account for any t, *obliviousness* to ensure IdP untraceability, and *pseudo-randomness* to provide RP unlinkability. In contrast, PrivacyPass and TrustToken [35], [36] depend on only the properties of deterministicness and obliviousness. Moreover, UPPRESSO requires *more* properties for secure SSO services (i.e., user identification and RP designation; see Theorems 1 and 2 in Section V-B), which are not discussed in OPRFs [47], [37], [38]. So an OPRF protocol is not always ready to work as the identity transformations in UPPRESSO.

D. Security and Privacy Analysis of SSO Protocols

Dolev-Yao style models are developed to analyze the communications among entities in an SSO system [6], [7], [48], to ensure that all messages including identity tokens are delivered as expected in the system and then to prove security and privacy of services based on only traditional public-key and symmetric cryptographic algorithms (e.g., RSA and AES in Spresso [6]). On the contrary, in this paper security and privacy of SSO services are analyzed based on more complicated cryptographic primitives, while secure communications among entities (i.e., authenticity, confidentiality and integrity of identity tokens) are ensured by common security mechanisms within a COTS browser.

III. The Identity-Transformation Approach

In this section, we firstly list the security requirements of SSO, explain the identity dilemma in identity tokens, and finally present the identity-transformation approach.

A. Security Requirements for SSO Services

Non-anonymous SSO services are designed to allow a user to log into an honest RP as her unique account at this RP, by presenting identity tokens issued by a trusted IdP [1], [2], [3], [4], [5]. It makes no sense to discuss the login results at a malicious RP. This security goal is achieved through the following properties of identity tokens.

First of all, *authenticity*, *confidentiality*, and *integrity* of identity tokens are necessary to prevent forging, eavesdropping and tampering. In SSO services, identity tokens are signed by the trusted IdP, transmitted over HTTPS/TLS [1], [2], [3], and finally forwarded to the target RPs by the authenticated user. Common security mechanisms within a browser (e.g., the mechanisms of HTTP session and `postMessage` targetOrigin) are also required for confidentiality of tokens [21], [49], [48], [7] (see Section IV-D for details).

Meanwhile, in each login the IdP issues an identity token that (*a*) specifies the target RP (i.e., *RP designation*) and (*b*) identifies the authenticated user who requests this token (i.e., *user identification*), by the enclosed (pseudo-)identities of RP and user [1], [2], [3] (see Section V-B for details). RP designation prevents malicious RPs from replaying received

TABLE II
THE (PSEUDO-)IDENTITIES IN PRIVACY-PRESERVING SSO

Notation	Description	Lifecycle
ID_{U_i}	The i-th user's unique identity at the IdP.	Permanent
ID_{RP_j}	The j-th RP's unique identity at the IdP.	Permanent
$Acct_{i,j}$	The i-th user's account at the j-th RP.	Permanent
$PID^l_{U_i,j}$	The i-th user's pseudo-identity in her l-th login visiting the j-th RP.	Ephemeral
$PID^l_{RP_j}$	The j-th RP's pseudo-identity in the user's l-th login visiting this RP.	Ephemeral

identity tokens to gain illegal access to other honest RPs as victim users. Therefore, an honest RP checks the RP (pseudo-)identity enclosed in a token before accepting it [1], [7], [6], [5], [12], [10], [23], [11] and then allows the token holder to log in as the specified (pseudo-)identity (i.e., account).

B. Identity Transformations in SSO

As analyzed in Section II-B, existing privacy-preserving solutions of SSO hide only RP identities or user identities (i.e., only IdP-based login tracing or RP-based identity linkage is prevented), or introduce an extra fully-trusted server to process the identities of RP and user before an identity token is issued.

On the contrary, UPPRESSO attempts to *simultaneously* transform the identities of RP and user in each login, without extra trusted servers more than the honest-but-curious IdP, to provide untraceable and unlinkable SSO service. Table II lists the notations, and a sub/super-script (i.e., i, j, or l) may be omitted if it does not cause ambiguity.

First of all, each user owns her account unique at an RP. $\mathcal{F}_{Acct*}(ID_{U_i}, ID_{RP_j}) = Acct_{i,j}$ is proposed to determine a user's account at an RP. An account belonging to some user is *meaningful*, while *meaningless* accounts at an RP do not belong to any user.

To provide IdP untraceability (or prevent IdP-based login tracing), a user initiates a login by negotiating an *ephemeral* pseudo-identity $PID^l_{RP_j}$ with the target RP and sending an identity-token request for $PID^l_{RP_j}$ (but not ID_{RP_j}) to an IdP.

After authenticating the initiating user as ID_{U_i}, the IdP calculates an *ephemeral* $PID^l_{U_i,j}$ based on ID_{U_i} and $PID^l_{RP_j}$, and then issues an identity token that binds $PID^l_{U_i,j}$ and $PID^l_{RP_j}$. To provide RP unlinkability (or prevent RP-based identity linkage), ID_{U_i} is not enclosed in the identity token.

On receiving a verified token, the RP calculates the user's *permanent* account (i.e., $Acct_{i,j}$) based on $PID^l_{U_i,j}$ and $PID^l_{RP_j}$, and allows the token holder to log in.

Therefore, in addition to $\mathcal{F}_{Acct*}(ID_U, ID_{RP}) = Acct$, we propose the following identity transformations in a login initiated by a user to visit an RP in privacy-preserving SSO:

- $\mathcal{F}_{PID_{RP}}(ID_{RP}) = PID_{RP}$, calculated by the initiating user, where ID_{RP} is the target RP's identity. In the IdP's view, $\mathcal{F}_{PID_{RP}}()$ is a one-way function and PID_{RP} is *indistinguishable* from random variables.
- $\mathcal{F}_{PID_U}(ID_U, PID_{RP}) = PID_U$, calculated by the IdP, where ID_U identifies the user requesting tokens. In the target RP's view, $\mathcal{F}_{PID_U}()$ is a one-way function and PID_U is *indistinguishable* from random variables.
- $\mathcal{F}_{Acct}(PID_U, PID_{RP}) = Acct$, calculated by the target RP. In the user's multiple logins visiting the RP, $Acct = \mathcal{F}_{Acct*}(ID_U, ID_{RP})$ is always derived.

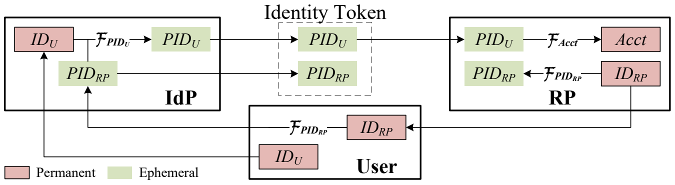


Fig. 2. Identity transformations in UPPRESSO

The relationships among the (pseudo-)identities are depicted in Figure 2. Red and green blocks represent *permanent* identities and *ephemeral* pseudo-identities, respectively, and labeled arrows denote the transformations of (pseudo-)identities. PID_{RP} and PID_U are ephemeral; otherwise, the IdP could learn the visited RP. $PID_{U,j}$ for visiting different RPs are independent of each other; otherwise, colluding RPs could learn something on ID_U from the user pseudo-identities and link the logins.

In privacy-preserving SSO services with the identity transformations, an identity token explicitly binds $PID_{RP_j} = \mathcal{F}_{PID_{RP}}(ID_{RP_j})$ but implicitly designates only the RP with ID_{RP_j}. The following properties are required for the identity transformations:

- **Account Uniqueness.** Each user owns an account locally unique at an RP, determined by $\mathcal{F}_{Acct*}(ID_U, ID_{RP})$.
- **User Identification.** Based on an identity token, the designated honest RP derives only the account exactly identifying the user requesting this token.
- **RP Designation.** An identity token is rejected, at any honest RPs other than the designated one.
- **IdP Untraceability.** The IdP learns nothing on the visited RP from the identity-token requests.
- **RP Unlinkability.** Colluding RPs cannot link any login initiated by an honest user visiting an RP, to any logins visiting any other colluding RPs by honest users.

In addition to the basic requirement of account uniqueness, user identification and RP designation ensure security of SSO services, while IdP untraceability and RP unlinkability protect user privacy (see Section V for the formal proofs).

IV. THE DESIGNS OF UPPRESSO

A. Threat Model

The system consists of an honest-but-curious IdP, as well as several honest or malicious RPs and users. This threat model is in line with widely-used SSO services in the Internet [1], [2], [3], [4].

Honest-but-curious IdP. The IdP strictly follows the protocols, while remaining interested in learning about a user's login activities. For example, it could store received messages to infer the relationship among ID_U, ID_{RP}, PID_U, and PID_{RP}. It never actively violates the protocols.

The IdP maintains the private key well for signing identity tokens and RP certificates, and adversaries cannot forge such signed messages.

Malicious users. Adversaries could control a set of users by stealing their credentials or registering Sybil users in UPPRESSO. Their objective is to (*a*) impersonate a victim user at honest RPs or (*b*) entice an honest user to log into an honest RP under another user's account [6], [48]. A malicious user could modify, insert, drop, or replay messages or even behave arbitrarily.

Malicious RPs. Adversaries could control a set of RPs by registering at the IdP as an RP or exploiting vulnerabilities to compromise some RPs. Malicious RPs could behave arbitrarily, attempting to compromise the security or privacy guarantees of UPPRESSO. For example, they could manipulate PID_{RP} and t in a login, attempting to (*a*) entice honest users to return an identity token that could be accepted by some honest RP or (*b*) affect the generation of PID_U to further analyze the relationship between ID_U and PID_U.

Colluding users and RPs. Malicious users and RPs could collude, attempting to break the security or privacy guarantees for honest users and RPs; for example, to impersonate an honest victim user at honest RPs or link an honest user's logins visiting colluding RPs.

B. Assumptions

We assume secure communications between honest entities, and the cryptographic primitives are secure. The software stack of an honest entity (e.g., a browser) is correctly implemented to transmit messages to receivers as expected. These communication assumptions are realized by the commonly-used mechanisms in popular SSO services (e.g., HTTPS/TLS, HTTP session maintenance, and the `postMessage` HTML5 API in COTS browsers).

While PID_{RP} but not ID_{RP} is processed by the IdP, it requires that no other information about the visited RP is leaked to the IdP, in the requesting, receiving and forwarding of identity tokens. We realize this assumption by specific designs in Section IV-D. Moreover, our work focuses only on the privacy threats introduced by SSO protocols, and does not consider the tracking of user activities by network traffic analysis or crafted web pages, as they can be prevented by other defenses.

UPPRESSO is designed for users who value privacy, or provides an effective option for such users. So privacy leakages due to re-identification by distinctive attributes across RPs are out of our scope. In UPPRESSO a user never authorizes the IdP to enclose distinctive attributes, such as telephone number, Email address, etc., in tokens or sets such attributes at any RP.

C. Identity-Transformation Algorithms

We propose the identity transformations on an elliptic curve $\mathbb{E}$, and Table III lists the notations. It is easy to shift these transformations into a finite field $\mathbb{F}_q$.

TABLE III
NOTATIONS USED IN THE UPPRESSO PROTOCOL

Notation	Description
$\mathbb{E}$, G, n	$\mathbb{E}$ is an elliptic curve over a finite field $\mathbb{F}_q$. G is a base point (or generator) on $\mathbb{E}$, and the order of G is a prime number n.
ID_{U_i}	$ID_U = u \in \mathbb{Z}_n$ is the i-th user's unique identity at the IdP, which is known only to the IdP.
ID_{RP_j}	$ID_{RP} = [r]G$ is the j-th RP's unique identity, which is publicly known; $r \in \mathbb{Z}_n$ is known to *nobody*.
t	$t \in \mathbb{Z}_n$ is a user-selected random integer in each login; t is shared with the target RP and kept unknown to the IdP.
$PID^l_{RP_j}$	$PID_{RP} = [t]ID_{RP} = [tr]G$ is the j-th RP's pseudo-identity, in the user's l-th login visiting this RP.
$PID^l_{U_i,j}$	$PID_U = [ID_U]PID_{RP} = [utr]G$ is the i-th user's pseudo-identity, in the user's l-th login visiting the j-th RP.
$Acct_{i,j}$	$Acct = [t^{-1} \bmod n]PID_U = [ID_U]ID_{RP} = [ur]G$ is the i-th user's locally-unique account at the j-th RP, publicly known.
SK, PK	The IdP's private key and public key, used to sign and verify identity tokens and RP certificates.
$Enpt_{RP_j}$	The j-th RP's endpoint for receiving the identity tokens.
$Cert_{RP_j}$	The RP certificate binding $Enpt_{RP_j}$ and ID_{RP_j}, signed by the IdP.

***ID_U*, *ID_{RP}* and *Acct*.** The IdP assigns a unique random integer $u \in \mathbb{Z}_n$ to a user (i.e., $ID_U = u$), and randomly selects unique $ID_{RP} = [r]G$ for an RP. Here, G is a base point on $\mathbb{E}$ of order n, and $[r]G$ denotes the addition of G on the curve r times. $Acct_{i,j} = \mathcal{F}_{Acct*}(ID_{U_i}, ID_{RP_j}) = [ID_{U_i}]ID_{RP_j} = [u_i r_j]G$ is automatically assigned to a user at every RP.

A user's accounts at different RPs are inherently different and unlinkable. A user's identity $ID_U = u$ is unknown to all entities except the honest IdP; otherwise, colluding RPs could calculate $[u]ID_{RP_j}$s for any known u and link these accounts. Meanwhile, $ID_{RP} = [r]G$ and $Acct = [ID_U]ID_{RP}$ are publicly-known, but r is always kept secret; otherwise, two colluding RPs with $ID_{RP_j} = [r]G$ and $ID_{RP_{j'}} = [r']G$ could link a user's accounts by checking whether $[r']Acct_j$ is equal to $[r]Acct_{j'}$ or not.

In UPPRESSO r is processed only once by the IdP, while u is used only by the IdP internally and not enclosed in any message; see Section IV-E for details.

***ID_{RP}-PID_{RP}* Transformation.** When visiting an RP, a user selects a random number $t \in \mathbb{Z}_n$ and calculates PID_{RP} as below. She also sends t to the RP.

$$PID_{RP} = \mathcal{F}_{PID_{RP}}(ID_{RP}, t) = [t]ID_{RP} = [tr]G \qquad (1)$$

***ID_U-PID_U* Transformation.** On receiving an identity-token request for PID_{RP} from a user authenticated as ID_U, the IdP calculates PID_U as below.

$$PID_U = \mathcal{F}_{PID_U}(ID_U, PID_{RP}) = [ID_U]PID_{RP} = [utr]G \qquad (2)$$

***PID_U-Acct* Transformation.** After verifying a signed identity token, the RP calculates $Acct$ as follows.

$$Acct = \mathcal{F}_{Acct}(PID_U, t) = [t^{-1} \bmod n]PID_U \qquad (3)$$

The above identity transformations ensure *account uniqueness*: At an RP, every user owns a unique account, because

$ID_{RP} = [r]G$ is also a generator of $\mathbb{E}$ and then $Acct = \mathcal{F}_{Acct*}() = [ur]G$ is unique at this RP.

Meanwhile, in a login involving no adversary, the visited RP always derives the account belonging to the user who requests the identity token, by calculating $\mathcal{F}_{Acct}(PID_U, PID_{RP})$. From Equations 1, 2, and 3, it is derived that

$$Acct = [t^{-1}utr]G = [ur]G = \mathcal{F}_{Acct*}(ID_U, ID_{RP}) \qquad (4)$$

In Section V we formally prove the other required properties (i.e., user identification, RP designation, IdP untraceability, and RP unlinkability) of the identity transformations, under adversarial scenarios.

D. The Designs Specific for Web Applications

When the required properties of identity transformations are satisfied, in UPPRESSO we need to additionally ensure:

- Authenticity, confidentiality, and integrity of identity tokens, by the common mechanisms in widely-deployed SSO services [1], [2], [3], [21], [49], [48], [7], [22],
- $PID_{RP} = [t]ID_{RP}$ is calculated based on the target RP's identity (but not any other RPs') and t is known only to the target RP and the initiating user (but not the honest IdP; otherwise, the IdP is able to calculate ID_{RP} by itself),
- No extra privacy leakage in the requesting, receiving and forwarding of identity tokens (e.g., no information about the RP server is leaked to the IdP).

In an original OIDC system with pop-up UX, as shown in Figure 1, a user requests a token by sending the target RP's identity (i.e., ID_{RP}) via the user-i window, and the IdP replies with the signed token and also the RP's endpoint to receive tokens (i.e., $Enpt_{RP}$). Then, after receiving the token, the user-i script forwards it to the user-r script by `postMessage`, which is restricted by the origin of $Enpt_{RP}$ [6], [50], [7], [21], [49], [1]. The `postMessage` targetOrigin mechanism [51] restricts the recipient, so this ensures confidentiality of identity tokens because only scripts downloaded from the origin of $Enpt_{RP}$ are legitimate recipients of the forwarded tokens as the relationship between $Enpt_{RP}$ and ID_{RP} is maintained at the honest IdP server.

In UPPRESSO the IdP receives PID_{RP} (but not ID_{RP}), so it cannot reply with $Enpt_{RP}$ to the user-i script. Thus, in each login, as shown in Figure 3, when the user-i window is popped up, the RP certificate (i.e., $Cert_{RP}$) and the scope of request user attributes are also carried with the HTTP request to the IdP, which is generated in the user-r window. This HTTP request downloads the user-i script, which in turn verifies $Cert_{RP}$ to extract ID_{RP} and $Enpt_{RP}$, and calculates $PID_{RP} = [t]ID_{RP}$. Then, after the IdP signs the token, the user-i script forwards it to the user-r script by `postMessage`, which is also restricted by the origin of $Enpt_{RP}$. This ensures that (*a*) $PID_{RP} = [t]ID_{RP}$ is calculated based on the target RP's identity and (*b*) the identity token is forwarded to the corresponding RP server, because $Enpt_{RP}$ and ID_{RP} are signed in $Cert_{RP}$ by the IdP.

It is worth noting that, in the request to pop up the user-i window, $Cert_{RP}$ and the scope of requested user attributes are carried as *fragments* (but not parameters), which are identified by `#` instead of `?`, so $Cert_{RP}$ is not sent to the IdP web server. Meanwhile, we save $Enpt_{RP}$ and t using the `sessionStorage` HTML object, because the user-i window is refreshed to finish user authentication and attribute authorization. Therefore, $Cert_{RP}$, $Enpt_{RP}$ and t are processed locally within the COTS browsers.

In UPPRESSO a user configures nothing locally for the IdP's public key is set in the user-i script to verify $Cert_{RP}$, like in other popular SSO systems [1], [2], [3], [4]. So, on receiving an identity-token request, the IdP web server checks the included HTTP `origin` header to ensure it is sent by the user-i script.

Compared with the original OIDC protocol in Figure 1, UPPRESSO adopts the common mechanisms for secure communications, except that the relationship between ID_{RP} and $Enpt_{RP}$ is maintained as $Cert_{RP}$ (i.e., in an offline way but still by the IdP). Therefore, authenticity, confidentiality, and integrity of identity tokens are ensured in UPPRESSO. Moreover, in UPPRESSO the IdP's token endpoint works in a way compatible with original OIDC protocols [2], [1]: When receiving a token request for an unknown ID_{RP} (i.e., PID_{RP}), the IdP issuers a token but replies with an "empty" $Enpt_{RP}$, and the calculation of PID_U is considered as a special way to assign PPIDs.

Finally, to prevent referer leakage in the HTTP request to pop up the user-i window, we need to ensure this HTTP request does not carry a `referer` header, which reveals the visited RP's domain to the IdP server.[2] This is achieved by setting `<meta name="referrer" content="no-referrer">` in the RP's SSO login page. This method is specified by W3C [52] and widely supported. We have tested it in various browsers such as Chrome, Safari, Edge, Opera, and Firefox, and confirmed no referer leakage.

E. The UPPRESSO protocol

System Initialization. $\mathbb{E}$, G and n are set up and publicly published. An IdP generates a key pair (SK, PK) to sign and verify identity tokens and RP certificates.

RP Registration. Each RP registers itself at the IdP to obtain ID_{RP} and its RP certificate $Cert_{RP}$ as follows. This registration may be conducted by face-to-face means.

1. An RP pre-installs PK by trusted means. It submits a registration request, including the endpoint to receive identity tokens and other information.
2. After examining the request, the IdP randomly selects $r \in \mathbb{Z}_n$ and assigns a *unique* point $[r]G$ to the RP as its identity. Note that r is not processed any more and then known to nobody due to the elliptic curve discrete logarithm problem (ECDLP). The IdP then signs

[2]In original OIDC services this leakage commonly exists, but does not matter because such services are not designed to prevent IdP-based login tracing.

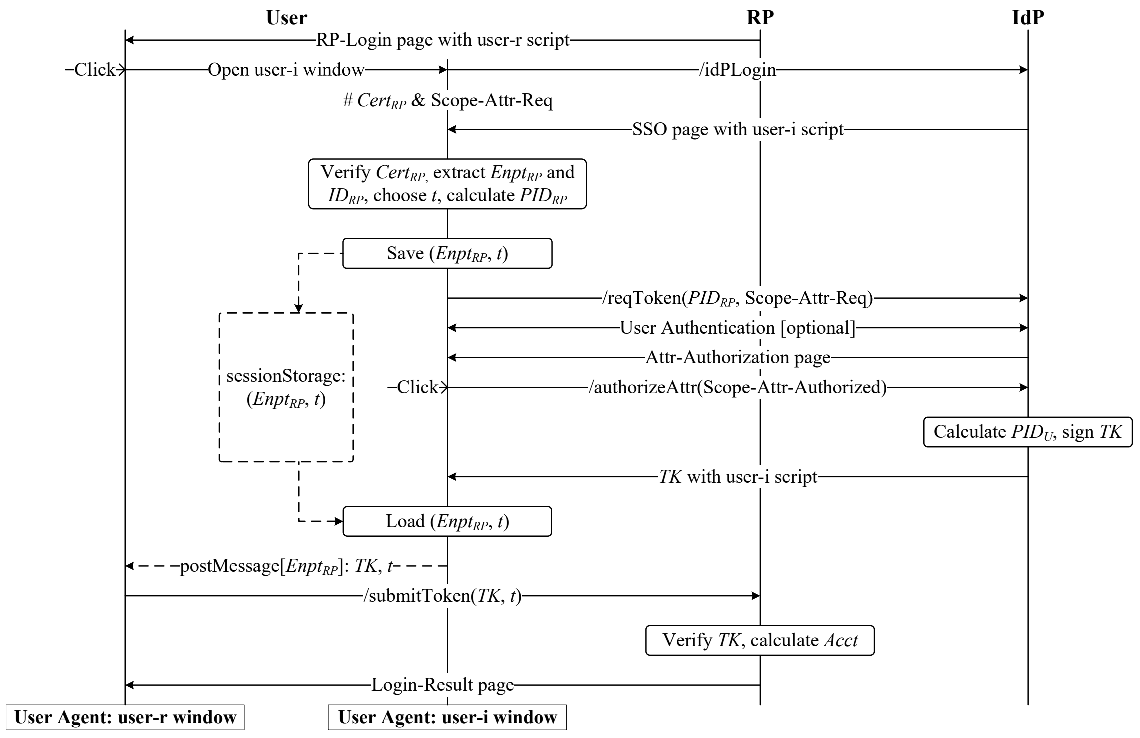

User
RP
IdP
RP-Login page with user-r script
Click
Open user-i window
/idPLogin
# $Cert_{RP}$ & Scope-Attr-Req
SSO page with user-i script
Verify $Cert_{RP}$, extract $Enpt_{RP}$ and ID_{RP}, choose t, calculate PID_{RP}
Save ($Enpt_{RP}$, t)
sessionStorage: ($Enpt_{RP}$, t)
/reqToken(PID_{RP}, Scope-Attr-Req)
User Authentication [optional]
Attr-Authorization page
Click
/authorizeAttr(Scope-Attr-Authorized)
Calculate PID_U, sign TK
TK with user-i script
Load ($Enpt_{RP}$, t)
postMessage[$Enpt_{RP}$]: TK, t
/submitToken(TK, t)
Verify TK, calculate $Acct$
Login-Result page
User Agent: user-r window
User Agent: user-i window

Fig. 3. The SSO login flow of UPPRESSO

$Cert_{RP} = [ID_{RP}, Enpt_{RP}, *]_{SK}$, where $[\cdot]_{SK}$ is a message signed using SK and $*$ is supplementary information such as the RP's common name.

3. The RP verifies $Cert_{RP}$ using PK, and accepts ID_{RP} and $Cert_{RP}$.

User Registration. Each user sets up her unique username and the corresponding credential for the IdP. The IdP assigns a unique random identity $ID_U = u \in \mathbb{Z}_n$ to the user.

It requires that ID_U is known *only* to the IdP. In UPPRESSO ID_U is used only by the IdP *internally*, not enclosed in any message. For example, a user's identity is generated and always restored by hashing her username concatenated with the IdP's private key. Then, ID_U is never stored in hard disks, and protected almost the same as the IdP's private key because it is *only* used to calculate PID_U as the IdP is signing an identity token binding $PID_U = [ID_U]PID_{RP}$.

Account Synchronization. Every RP regularly synchronizes all accounts at it from the IdP, and maintains an up-to-date local list of meaningful accounts. We discuss account synchronization in Section V-E.

SSO Login. A login flow involves three steps: identity-token requesting, generation, and acceptance. In Figure 3, the IdP's and RP's operations are connected by two vertical lines, respectively. The user operations are split into two groups in different browser windows by vertical lines, one communicating with the IdP and the other with the RP. Solid horizontal lines indicate messages exchanged between a user and the IdP (or the RP), while dotted lines represent the mechanisms within a browser: (*a*) `postMessage` invocations between two scripts (or browser windows) or (*b*) `sessionStorage` objects accessed by different pages of an HTTP session.

1. *Identity-Token Requesting.* The user requests an identity token for $PID_{RP} = [t]ID_{RP}$.

1.1 The target RP provides an SSO login page with the user-r script, and $Cert_{RP}$ is set in this script. The user clicks the SSO login button, to pop up a new browser window responsible for communicating with the IdP.

1.2 The user-i window downloads the user-i script, which processes $Cert_{RP}$ and the scope of requested attributes sent from the user-r window. It verifies $Cert_{RP}$, extracts ID_{RP} and $Enpt_{RP}$ from $Cert_{RP}$, chooses a random number $t \in \mathbb{Z}_n$, and calculates $PID_{RP} = [t]ID_{RP}$.

1.3 The user-i script uses the `sessionStorage` HTML5 object to save $Enpt_{RP}$ and t, and then requests an identity token by sending PID_{RP} and the scope of requested attributes.

2. *Identity-Token Generation.* The IdP calculates $PID_U = [ID_U]PID_{RP}$ and signs an identity token as follows.

2.1 On receiving an identity-token request, the IdP authenticates the initiating user as ID_U, if not authenticated yet. Then, it obtains the user's authorization to enclose attributes in the token.

2.2 The IdP calculates $PID_U = [ID_U]PID_{RP}$, and signs $TK = [PID_{RP}, PID_U, Iss, Validity, Attr]_{SK}$, where Iss identifies the IdP, $Validity$ indicates the validity period, and $Attr$ contains the authorized user attributes.

2.3 The IdP replies with the identity token to the user-i window, as well as the user-i script to forward this token.

3. *Identity-Token Acceptance.* The RP receives the identity token and allows the user to log in.

3.1 From the `sessionStorage` object the user-i script loads $Enpt_{RP}$ and t, and forwards the identity token and t to the user-r window by `postMessage` which is restricted by $Enpt_{RP}$.

3.2 Then, the user-r script submits it to the RP. The RP verifies the signature and the validity period of the token and calculates $Acct = [t^{-1}]PID_U$.

3.3 Only if it is meaningful in its account list, the RP allows the user to log in as $Acct$.

If any verification fails, this login flow will be terminated immediately. For example, the user halts it when receiving an invalid $Cert_{RP}$. The IdP rejects an identity-token request in Step 3.3 if the received PID_{RP} is not a point on $\mathbb{E}$, and the RP rejects the token holder if the identity token is invalid or the derived account is meaningless.

V. Security and Privacy Analysis

As account uniqueness has been analyzed in Section IV-C, in this section we prove security (i.e., user identification and RP designation) and privacy (i.e., IdP untraceability and RP unlinkability) of UPPRESSO.

A. Adversarial Scenarios

Based on our design goals and the potential adversaries discussed in Section IV-A, we consider three adversarial scenarios as below and the security and privacy guarantees are proved against different adversaries.

Security (i.e., user identification and RP designation) is proved against malicious RPs and users. Malicious RPs and users could collude [48], [7], [6], attempting to (*a*) impersonate an honest user to log into some honest RP or (*b*) entice an honest user to log into an honest RP under another's account.

Provided that the security properties of UPPRESSO are satisfied, we analyze the privacy properties *only* for successful logins because no meaningful account is derived in an unsuccessful login where the initiating user or the target RP deviates from the specifications. It make nonsense to track login activities resulting in meaningless accounts. Therefore, an *honest-but-curious* is considered in the proof of IdP untraceability, while RP unlinkability is analyzed against *honest-but-colluding* RPs and users.

B. Security

In secure SSO systems, an identity token denoted as TK, which is requested by a user to visit an RP, *enables only this user to log into only the honest target RP as her account at this RP*. It makes no sense to discuss the login results at malicious RPs. TK is signed by the IdP to bind PID_{RP} and PID_U, where (*a*) the RP with ID_{RP} is designated if $PID_{RP} = [t]ID_{RP}$ is calculated and (*b*) $PID_U = [ID_U]PID_{RP}$ is calculated based on the initiating user's identity ID_U.

When authenticity, confidentiality and integrity of identity tokens are ensured, we summarize the following sufficient conditions of secure SSO services [48], [7], [6]:

User Identification. From TK the designated honest RP derives *only* the meaningful account belonging to the user requesting TK.

RP Designation. From TK, *only* the designated honest RP derives meaningful accounts.

As mentioned in Section IV-E, an (honest) RP derives accounts based on *any* signed identity tokens. The following proofs of user identification and RP designation assume *malicious* RPs colluding with users, which attempt to break the security guarantees of UPPRESSO, by arbitrarily manipulating t and/or sending TK to an RP not designated.

Let's assume totally s users and p RPs in UPPRESSO, whose identities are denoted as $\mathbb{ID}_U = \mathbb{u} = \{u_{i;1 \leq i \leq s}\}$ and $\mathbb{ID}_{RP} = \{[r_{j;1 \leq j \leq p}]G\}$, respectively. There are meaningful accounts $Acct_{i,j} = [u_i]ID_{RP_j} = [u_i r_j]G$ at each RP. $Acct_{i,j}$ and ID_{RP_j} are publicly-known, but u_i and r_j are kept unknown to adversaries.

We prove user identification by showing that, at any RP designated by TK, a meaningless account not belonging to the initiating user will never be derived.

Theorem *1 (User Identification):* Given unknown $\mathbb{ID}_U$ and known $\mathbb{A}cct$, for any known $ID_{RP} \in \mathbb{ID}_{RP}$, malicious adversaries cannot find $\check{t}$ and TK binding $PID_{RP} = \mathcal{F}_{PID_{RP}}(ID_{RP}, \hat{t})$ and $PID_{\hat{U}} = \mathcal{F}_{PID_U}(ID_{\hat{U}}, PID_{RP})$ satisfying that $\mathcal{F}_{Acct}(PID_{\hat{U}}, \check{t}) = \mathcal{F}_{Acct*}(ID_{\check{U}}, ID_{RP})$, where $ID_{\hat{U}} \neq ID_{\check{U}}$ and $ID_{\hat{U}}, ID_{\check{U}} \in \mathbb{ID}_U$.

Proof. It requires that, for any ID_{RP}, adversaries cannot find $\hat{t}$ and $\check{t}$ satisfying that $\mathcal{F}_{Acct}(PID_{\hat{U}}, \check{t}) = [\check{t}^{-1}]PID_{\hat{U}} = [\check{t}^{-1}ID_{\hat{U}}]PID_{RP} = [\check{t}^{-1}\hat{t}ID_{\hat{U}}]ID_{RP} = [ID_{\check{U}}]ID_{RP} = \mathcal{F}_{Acct*}(ID_{\check{U}}, ID_{RP})$, where $ID_{\hat{U}} \neq ID_{\check{U}}$.

When u_i and r_j are kept unknown, this is equivalent to the elliptic curve discrete logarithm problem (ECDLP) of $\check{t}^{-1}\hat{t} = \log_{[\hat{u}r]G}[\check{u}r]G$. □

Next, we prove RP designation by showing that, at any honest RPs other than the designated one, no meaningful account will be derived from TK.

Theorem *2 (RP Designation):* Given known $\mathbb{ID}_{RP}$, known $\mathbb{A}cct$ and unknown $\mathbb{ID}_U$, malicious adversaries cannot find j_1, j_2, i_1, i_2, t_1, and t_2 which satisfy that $\mathcal{F}_{Acct}(\mathcal{F}_{PID_U}(ID_{U_{i_1}}, PID_{RP}), t_2) = \mathcal{F}_{Acct*}(ID_{U_{i_2}}, ID_{RP_{j_2}})$, where PID_{RP} may be calculated as $\mathcal{F}_{PID_{RP}}(ID_{RP_{j_1}}, t_1)$ or arbitrarily generated (i.e., RP_{j_1} or no RP is designated), $j_1 \neq j_2$, $1 \leq j_1, j_2 \leq p$, and $1 \leq i_1, i_2 \leq s$.

Proof. If adversaries could find j_1, j_2, i_1, i_2, t_1 and t_2 which satisfy that $[{t_2}^{-1}t_1 ID_{U_{i_1}}]ID_{RP_{j_1}} = [ID_{U_{i_2}}]ID_{RP_{j_2}}$ where $j_1 \neq j_2$, the ECDLP of ${t_2}^{-1}t_1 = \log_{[u_{i_1} r_{j_1}]G}[u_{i_2} r_{j_2}]G$ would be solved.

Or, the adversaries attempt to find PID_{RP}, j_2, i_1, i_2 and t_2 satisfying that $[{t_2}^{-1}ID_{U_{i_1}}]PID_{RP} = [ID_{U_{i_2}}]ID_{RP_{j_2}}$, but this cannot be solved due to unknown ID_{U_i}. □

C. Privacy against IdP-based Login Tracing

The honest IdP would attempt to infer the visited RPs, by collecting information from the identity-token requests. It does not obtain any information about the target RP in a login (e.g.,

ID_{RP}, $Enpt_{RP}$ or $Cert_{RP}$), except its *ephemeral* pseudo-identity PID_{RP}.

We prove that, PID_{RP} is *indistinguishable* from a random variable on $\mathbb{E}$ to the IdP, so it cannot (*a*) link multiple logins visiting an RP, or (*b*) distinguish a login initiated by a user to visit any RP from those logins visiting other RPs by this user.

THEOREM 3 *(IdP Untraceability): The IdP cannot distinguish $PID_{RP} = [t]ID_{RP}$ from a random variable on $\mathbb{E}$, where t is random in $\mathbb{Z}_n$ and kept unknown to the IdP.*

PROOF. As G is a generator on $\mathbb{E}$ of order n, $ID_{RP} = [r]G$ is also a generator of order n. As t is uniformly-random in $\mathbb{Z}_n$ and kept unknown, $PID_{RP} = [t]ID_{RP}$ is *indistinguishable* from a point that is uniformly-random on $\mathbb{E}$. □

This property of obliviousness has been proved in the Hash(EC)DH OPRF [37], [38], where the OPRF server works similarly to the IdP and learns nothing about a client's inputs or outputs of the evaluated pseudo-random function (i.e., ID_{RP} or $Acct$ in UPPRESSO).

D. Privacy against RP-based Identity Linkage

RPs could collude with some non-honest users, to infer the identities of honest users or link an honest user's accounts across these RPs.

In each login, an RP obtains *only* a random number t and an identity token enclosing PID_{RP} and PID_U. From the token, it learns only an *ephemeral* pseudo-identity $PID_U = [ID_U]PID_{RP}$ of the initiating user, from which it derives a *permanent* locally-unique identifier (or account) $Acct = [ID_U]ID_{RP}$. It cannot directly calculate ID_U from PID_U or $Acct$ due to the ECDLP. Next, in Theorem 4 we prove that, even if RPs collude with each other and some non-honest users by sharing PID_Us and other information observed in all their logins, they still cannot link any login visiting an RP by an honest user to any other logins visiting other colluding RPs by honest users.

With the trapdoor t, PID_{RP} and PID_U can be transformed into ID_{RP} and $Acct$, respectively, and vice versa. So we denote all the information that an RP learns in a login as a tuple $L = (ID_{RP}, t, Acct) = ([r]G, t, [ur]G)$.

As c RPs collude with each other, they create a shared view of all their logins, denoted as $\mathbb{L}$. When they collude further with v non-honest users, the logins initiated by these users are picked out of $\mathbb{L}$ and linked together as $\mathfrak{L}^m = \begin{Bmatrix} L^m_{1,1}, & L^m_{1,2}, & \cdots, & L^m_{1,c} \\ L^m_{2,1}, & L^m_{2,2}, & \cdots, & L^m_{2,c} \\ \cdots, & \cdots, & L^m_{i,j}, & \cdots \\ L^m_{v,1}, & L^m_{v,2}, & \cdots, & L^m_{v,c} \end{Bmatrix}$, where $L^m_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G) \in \mathbb{L}$ for $1 \le i \le v$ and $1 \le j \le c$. Any login in $\mathbb{L}$ but not linked in $\mathfrak{L}^m$ is initiated by an honest user to visit one of the c colluding RPs.

THEOREM 4 *(RP Unlinkability): Given $\mathbb{L}$ and $\mathfrak{L}^m$, c RPs colluding with v non-honest users cannot link any login visiting some of these RPs by an honest user to any subset of logins visiting any other RPs by honest users.*

PROOF. Out of $\mathbb{L}$ we randomly choose a login $L' \neq L^m_{i,j}$ ($1 \le i \le v, 1 \le j \le c$), which is initiated by an (unknown) honest

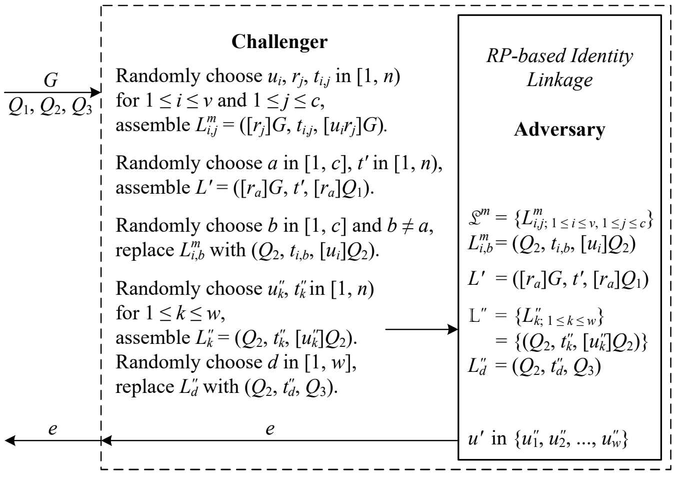

Challenger
G
Q_1, Q_2, Q_3
Randomly choose u_i, r_j, $t_{i,j}$ in $[1, n)$ for $1 \le i \le v$ and $1 \le j \le c$, assemble $L^m_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G)$.
Randomly choose a in $[1, c]$, t' in $[1, n)$, assemble $L' = ([r_a]G, t', [r_a]Q_1)$.
Randomly choose b in $[1, c]$ and $b \neq a$, replace $L^m_{i,b}$ with $(Q_2, t_{i,b}, [u_i]Q_2)$.
Randomly choose u''_k, t''_k in $[1, n)$ for $1 \le k \le w$, assemble $L''_k = (Q_2, t''_k, [u''_k]Q_2)$.
Randomly choose d in $[1, w]$, replace L''_d with (Q_2, t''_d, Q_3).
e
e
RP-based Identity Linkage
Adversary
$\mathfrak{L}^m = \{L^m_{i,j;\, 1 \le i \le v,\, 1 \le j \le c}\}$
$L^m_{i,b} = (Q_2, t_{i,b}, [u_i]Q_2)$
$L' = ([r_a]G, t', [r_a]Q_1)$
$\mathbb{L}'' = \{L''_{k;\, 1 \le k \le w}\} = \{(Q_2, t''_k, [u''_k]Q_2)\}$
$L''_d = (Q_2, t''_d, Q_3)$
u' in $\{u''_1, u''_2, ..., u''_w\}$

Fig. 4. The PPT algorithm $\mathcal{D}^*_R$ constructed based on the RP-based identity linkage game to solve the ECDDH problem.

user with $ID_{U'} = u'$ to a malicious RP_a where $a \in [1, c]$. Then, we randomly choose another malicious RP_b, where $b \in [1, c]$ and $b \neq a$. Consider any subset $\mathbb{L}'' \subset \mathbb{L}$ of w ($1 \le w < s - v$) logins visiting RP_b by unknown honest users, and denote the identities of the users initiating these logins as $\mathbb{u}_w = \{u''_1, u''_2, \cdots, u''_w\} \subset \mathbb{u}$.

Next, we prove that the colluding adversaries cannot distinguish $u' \in \mathbb{u}_w$ from u' randomly selected in the universal set (i.e., $u' \in \mathbb{Z}_n$). This indicates the adversaries cannot link L' to another login (or a subset of logins) visiting RP_b.

We define an RP-based identity linkage game $\mathcal{G}_R$ between an adversary and a challenger, to describe this login linkage: the adversary receives $\mathfrak{L}^m$, L', and $\mathbb{L}''$ from the challenger and outputs e, where (*a*) $e = 1$ if it decides u' is in $\mathbb{u}_w$ or (*b*) $e = 0$ if it believes u' is randomly chosen from $\mathbb{Z}_n$. Thus, the adversary succeeds in $\mathcal{G}_R$ with an advantage $\mathbf{Adv}$:

$$\begin{aligned} \Pr{}_1 &= \Pr(\mathcal{G}_R(\mathfrak{L}^m, L', \mathbb{L}'') = 1 \mid u' \in \mathbb{u}_w) \\ \Pr{}_2 &= \Pr(\mathcal{G}_R(\mathfrak{L}^m, L', \mathbb{L}'') = 1 \mid u' \in \mathbb{Z}_n) \\ \mathbf{Adv} &= |\Pr{}_1 - \Pr{}_2| \end{aligned}$$

As depicted in Figure 4, we then design a PPT algorithm $\mathcal{D}^*_R$ based on $\mathcal{G}_R$ to solve the elliptic curve decisional Diffie-Hellman (ECDDH) problem: given $(G, [x]G, [y]G, [z]G)$, decide whether z is equal to xy or randomly chosen in $\mathbb{Z}_n$, where G is a point on an elliptic curve $\mathbb{E}$ of order n, and x and y are integers randomly and independently chosen in $\mathbb{Z}_n$.

The algorithm $\mathcal{D}^*_R$ works as follows. (1) On receiving an input $(G, Q_1 = [x]G, Q_2 = [y]G, Q_3 = [z]G)$, the challenger chooses random numbers in $\mathbb{Z}_n$ to construct $\{u_i\}$, $\{r_j\}$, and $\{t_{i,j}\}$ for $1 \le i \le v$ and $1 \le j \le c$, with which it assembles $L^m_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G)$. It ensures $[r_j]G \neq Q_2$ (or $r_j \neq y$) in this procedure. (2) It randomly chooses $a \in [1, c]$ and $t' \in \mathbb{Z}_n$, to assemble $L' = ([r_a]G, t', [r_a]Q_1) = ([r_a]G, t', [x r_a]G)$. (3) Next, the challenger randomly chooses $b \in [1, c]$ but $b \neq a$, and replaces ID_{RP_b} with $Q_2 = [y]G$. Hence, for $1 \le i \le v$, the challenger replaces $L^m_{i,b} = ([r_b]G, t_{i,b}, [u_i r_b]G)$ with $(Q_2, t_{i,b}, [u_i]Q_2) = ([y]G, t_{i,b}, [u_i y]G)$, and finally constructs $\mathfrak{L}^m$. (4) The challenger chooses random numbers in $\mathbb{Z}_n$ to construct $\{u''_k\}$ and $\{t''_k\}$ for $1 \leq k \leq w$, with which it assembles $\mathbb{L}'' = \{L''_{k;1\leq k\leq w}\} = \{(Q_2, t''_k, [u''_k]Q_2)\} = \{([y]G, t''_k, [u''_k y]G)\}$. It ensures $[u''_k]G \neq Q_1$ (i.e., $u''_k \neq x$) and $u''_k \neq u_i$, for $1 \leq i \leq v$ and $1 \leq k \leq w$. Finally, it randomly chooses $d \in [1, w]$ and replaces L''_d with $(Q_2, t''_d, Q_3) = ([y]G, t''_d, [z]G)$. Thus, $\mathbb{L}'' = \{L''_{k;1\leq k\leq w}\}$ represents the logins initiated by w honest users, i.e., $\mathbb{u}_w = \{u''_1, u''_2, \cdots, u''_{d-1}, z/y, u''_{d+1}, \cdots, u''_w\}$. (5) When the adversary of $\mathcal{G}_R$ receives $\mathfrak{L}^m$, L', and $\mathbb{L}''$ from the challenger, it returns e which is also the output of $\mathcal{D}^*_R$.

According to the above construction, x is embedded as $ID_{U'}$ of the login L' visiting the RP with $ID_{RP_a} = [r_a]G$, and z/y is embedded as $ID_{U''_d}$ of $\mathbb{L}''$ visiting the RP with $ID_{RP_b} = [y]G$, together with $\{u''_1, \cdots, u''_{d-1}, u''_{d+1}, \cdots, u''_w\}$. Meanwhile, $[r_a]G$ and $[y]G$ are two malicious RPs' identities in $\mathfrak{L}^m$. Because $x \neq u''_{k;1\leq k\leq w,k\neq d}$ and then x is not in $\{u''_1, \cdots, u''_{d-1}, u''_{d+1}, \cdots, u''_w\}$, the adversary outputs $s = 1$ and succeeds in the game *only if* $x = z/y$. Therefore, using $\mathcal{D}^*_R$ to solve the ECDDH problem, we have an advantage $\mathbf{Adv}^* = |\Pr^*_1 - \Pr^*_2|$, where

$$\begin{aligned}
&\Pr^*_1 = \Pr(\mathcal{D}^*_R(G, [x]G, [y]G, [xy]G) = 1)\\
=&\Pr(\mathcal{G}_R(\mathfrak{L}^m, L', \mathbb{L}'') = 1 \mid u' \in \mathbb{u}_w) = \Pr_1\\
&\Pr^*_2 = \Pr(\mathcal{D}^*_R(G, [x]G, [y]G, [z]G) = 1)\\
=&\Pr(\mathcal{G}_R(\mathfrak{L}^m, L', \mathbb{L}'') = 1 \mid u' \in \mathbb{Z}_n) = \Pr_2\\
&\mathbf{Adv}^* = |\Pr^*_1 - \Pr^*_2| = |\Pr_1 - \Pr_2| = \mathbf{Adv}
\end{aligned}$$

If in $\mathcal{G}_R$ the adversary has a non-negligible advantage, then $\mathbf{Adv}^* = \mathbf{Adv}$ is also non-negligible regardless of the security parameter λ. This violates the ECDDH assumption.

Therefore, the adversary has no advantage in $\mathcal{G}_R$ and cannot decide whether L' is initiated by some honest user with an identity in $\mathbb{u}_w$ or not. Because RP_b is any malicious RP, this proof can be easily extended from RP_b to more colluding malicious RPs. □

E. Account Synchronization

If new users are allowed to register after the initialization, in order to distinguish meaningful accounts from meaningless ones, an RP needs to regularly synchronize its accounts from the IdP. For example, when an account is derived but not in the visited RP's local account list, the RP contacts the IdP to synchronize accounts. Then, after the instant account synchronization, the token holder will be rejected if the derived account is still considered as meaningless (i.e., not in the RP's local account list).

An RP cannot imperceptively treat the token holder with an account not in the list as a newly-registered user. Although a malicious user cannot log into this RP as any meaningful account belonging to other users according to Theorems 1 and 2, she could log into such an RP as a meaningless but identical account in multiple logins. For example, both $\hat{t}$ and $\check{t}$ ($\check{t} \neq \hat{t}$) are reused in these logins, as analyzed in the proof of Theorem 1. Then, this malicious user actually owns multiple accounts at one RP.

The instant account synchronization can be replaced by conditional PID_{RP} checking as below. If an RP derives some account not in its local account list, it additionally checks PID_{RP} in this case: If PID_{RP} enclosed in the signed token is equal to $[t]ID_{RP}$, this account is asserted to belong to a newly-registered user[3] and the RP updates its account list locally; otherwise, it rejects this token.

VI. Implementation and Evaluation

We implemented a prototype of UPPRESSO[4] and conducted experimental comparisons with two open-source SSO systems: (*a*) MITREid Connect [50], a PPID-enhanced OIDC system with redirect UX, preventing only RP-based identity linkage, and (*b*) SPRESSO [6], which prevents only IdP-based login tracing. All these solutions work with COTS browsers as user agents.

A. Prototype Implementation

The UPPRESSO prototype implemented identity transformations on the NIST P256 elliptic curve where $n \approx 2^{256}$, and the IdP was developed on top of MITREid Connect [50], with minimal code modifications. In the prototype, the scripts of user-i and user-r consist of about 160 and 140 lines of JavaScript code, respectively. The cryptographic computations such as $Cert_{RP}$ verification and PID_{RP} negotiation are conducted based on jsrsasign [53], an open-source JavaScript library.

We developed a Java-based RP SDK with about 500 lines of code on the Spring Boot framework. Two functions encapsulate the RP operations of UPPRESSO: one for requesting identity tokens and the other for deriving accounts. The cryptographic computations are finished using the Spring Security library. Then, an RP can invoke necessary functions by adding less than 10 lines of Java code, to access the services provided by UPPRESSO.

SPRESSO implements all entities by JavaScript based on node.js, while MITREid Connect provides Java implementations of IdP and RP SDK. Thus, for UPPRESSO and MITREid Connect, we implemented RPs based on Spring Boot by integrating the respective SDKs. In all three schemes, the RPs provide the same function of obtaining the user's account from verified identity tokens.

For fair comparisons, MITREid Connect and the UPPRESSO prototype implement the implicit flow of OIDC, while SPRESSO implements a similar flow to forward identity tokens to an RP. All systems employ RSA-2048 and SHA-256 to generate tokens.

B. Performance Evaluation

Experiment setting. Of all solutions, the IdP and RP servers were deployed on Alibaba Cloud Elastic Compute Service,

[3]In this case, adversaries cannot manipulate t and then the derived account is meaningful.

[4]The prototype is open-sourced at https://github.com/uppresso/.

each of which ran Windows 10 with 8 vCPUs and 32GB RAM. The extra forwarder server of SPRESSO ran Ubuntu 20 with 16 vCPUs, also on the cloud platform, helping to forward identity tokens to RP servers.

We conducted experiments in two settings: (*a*) a browser, Chrome 104.0.5112.81, ran on a virtual machine on the cloud platform with 8 vCPUs and 32 GB memory, and (*b*) the browser running locally on a PC with Core i7-8700 CPU and 32 GB memory, remotely accessed the servers. All entities except the local browser, were deployed in the same virtual private cloud and connected to one vSwitch, which minimized the impact of network delays.

Comparisons. We split the login flow into three phases for detailed comparisons: (1) *identity-token requesting* (Steps 1 and 2 in Figure 3), to construct an identity-token request and send it to the IdP server; (2) *identity-token generation* (Step 3 of UPPRESSO), to generate an identity token at the IdP server, while the user authentication and the user-attribute authorization are excluded; and (3) *identity-token acceptance* (Step 4), where the RP receives, verifies, and parses the identity token.

In the identity-token requesting phase of UPPRESSO a browser downloads the two scripts, as described in Section IV-D. As mentioned in Section II-A, to process the token retrieved from the IdP, in MITREid Connect and SPRESSO a user-r script is downloaded during the phase of token generation. SPRESSO needs another script from the forwarder server in the token acceptance phase [6].

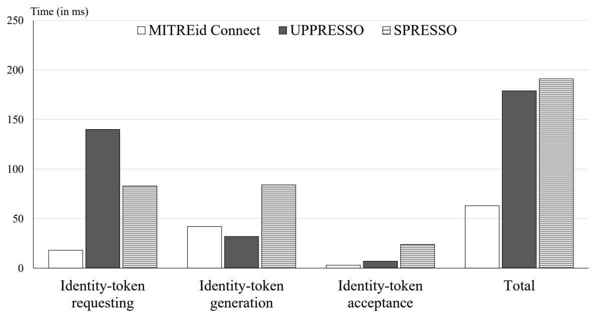
We compared the average time required for an SSO login in three schemes based on 1,000 measurements. As shown in Figure 5, MITREid Connect, UPPRESSO, and SPRESSO require (*a*) 63 ms, 179 ms, and 190 ms, respectively when all entities were deployed on the cloud platform, or (*b*) 312 ms, 471 ms, and 510 ms, respectively when the user browser ran locally to visit the cloud servers.

Regarding identity-token requesting, the RP of MITREid Connect immediately constructs an identity-token request. UPPRESSO incurs overheads in opening a new browser window and downloading the scripts.[5] In SPRESSO the RP needs to obtain information about the IdP and encrypt its domain using an ephemeral key, resulting in extra overheads.

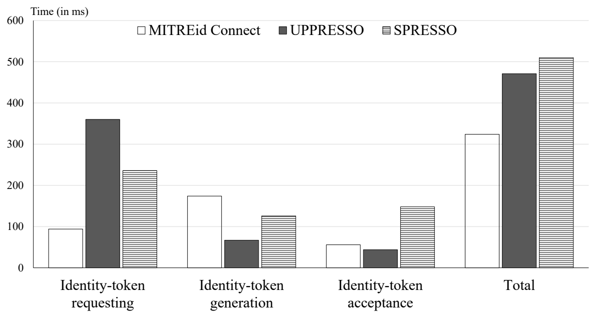
UPPRESSO requires the least time for generating identity tokens for it receives the token from the IdP without any additional processing. MITREid Connect and SPRESSO require extra time as the browser downloads a script from the RP in this phase. Moreover, SPRESSO takes slightly more time to generate an identity token, as it implements the IdP using node.js and uses a JavaScript cryptographic library that is a little less efficient than the Java library used in the others.

In the identity-token acceptance phase, MITREid Connect and UPPRESSO take similar amounts of time for the RP to receive a token and accept it. In contrast, SPRESSO takes the longest time due to its complex processing at the user agent. After receiving an identity token from the IdP, the browser downloads another script from the forwarder server, to decrypt the RP endpoint and sends the token to this endpoint.

[5]This pop-up overhead can be reduced by browser extensions. We have implemented such a browser extension while keeping the IdP and RPs unmodified, and the supplementary experiments showed a reduction of (*a*) about 90 ms in the virtual private cloud setting and (*b*) 260 ms when accessed remotely.



(a) In a virtual private cloud



(b) With a remotely-visiting browser

Fig. 5. The time costs of SSO login in MITREid Connect, UPPRESSO, and SPRESSO

VII. Discussions

IdP-RP collusive attacks. In this work, we prefer to support COTS browsers rather than prevent the IdP-RP collusive attacks. Even when the IdP strictly follows the protocols but shares information with RPs, a user would complete her logins *entirely* with colluding entities, and then the IdP and RPs could always link a user's (pseudo-)identities and accounts (i.e., both IdP untraceability and RP unlinkability are broken), unless a long-term secret, unknown to the colluding IdP and RPs, is introduced to mask the relationship of these (pseudo-)identities and accounts. That is, regardless of which $\mathcal{F}_{Acct*}()$ is adopted to determine $Acct_{i,j} = \mathcal{F}_{Acct*}(ID_{U_i}, ID_{RP_j})$, the IdP colluding with RPs could always track a user's login activities by analyzing the relationship between derived accounts and the user's identity, unless $Acct_{i,j} = \mathcal{F}_{Acct*}(s_i, ID_{U_i}, ID_{RP_j})$ is adopted and s_i is a long-term secret known to only the user. In this case, a browser extension or plug-in is required to handle such a user secret, and privacy-preserving identity federation [14], [15], [32], [13], [17] prefer the strategy. Besides, if the user secret is lost or leaked, the user has to notify all RPs to update her accounts derived from this secret, or additional revocation checking will be needed [14], [15]. Alternatively, MISO [11] introduces an *extra fully-trusted mixer server* other than the IdP to keep such a long-term secret, to calculate user pseudo-identities (or accounts) in each login. Then, in MISO the mixer server can track all users' login activities, but the IdP cannot even when colluding with RPs.

On the contrary, UPPRESSO is designed to provide privacy-preserving SSO services accessed from COTS browsers without extra trusted servers, at the cost of user privacy vulnerable to IdP-RP collusive attacks. This strategy offers an options for some users.

Support for the authorization code flow. In the authorization code flow of OIDC [1], the IdP does not directly return identity tokens. Instead, it generates an authorization code, which is forwarded to the target RP. The RP uses this code to retrieve identity tokens from the IdP.

$\mathcal{F}_{Acct*}()$, $\mathcal{F}_{PID_U}()$, $\mathcal{F}_{PID_{RP}}()$ and $\mathcal{F}_{Acct}()$, can be also integrated into the authorization code flow of OIDC to transform (pseudo-)identities in signed identity tokens. Then, the user-i script will forward an authorization code (but not the token) to the user-r script, and to the RP finally. Then, this code serves as an index to retrieve tokens by the RP, not disclosing any information about the user. On receiving an authorization code, the RP uses it as well as a credential issued by the IdP during the initial registration [1], to retrieve identity tokens.

Meanwhile, to hide RP identities from the IdP in the retrieval of identity tokens, privacy-preserving credentials (e.g., ring signatures [54] or privacy passes [35], [36]) and anonymous communications (e.g., oblivious proxies [55] or Tor [56]) need to be adopted for RPs. Otherwise, if the visited RP is not anonymously authenticated to the IdP in the retrieval of tokens, IdP untraceability is broken.

Alternatives for generating ID_{RP} and binding $Enpt_{RP}$. In UPPRESSO the IdP generates random ID_{RP} and signs an RP certificate to bind ID_{RP} and $Enpt_{RP}$, which is verified by the user-i script. This ensures the relationship between the RP designated in an identity token and the endpoint to receive this token. It also guarantees that the target RP has already registered itself at the IdP and prevents unauthorized RPs from utilizing the IdP's services [1], [10].

An alternative method for binding ID_{RP} and $Enpt_{RP}$ is to design a *deterministic* scheme to calculate unique ID_{RP} based on the RP's unambiguous name such as its domain. This can be achieved by encoding the domain with a hashing-to-elliptic-curves function [57], which provides collision resistance but not revealing the discrete logarithm of the output. It generates a point on the elliptic curve $\mathbb{E}$ as ID_{RP}, ensuring the *uniqueness* of $ID_{RP} = [r]G$ while keeping r unknown. Then, any RP can be visited through the SSO services, either authorized by the IdP or not.

In this case, the user-r script sends only the endpoint but not its RP certificate in Step 1.2, and the user-i script calculates ID_{RP} by itself based on the RP's unambiguous name. However, if the RP changes its domain, for example, from `https://theRP.com` to `https://RP.com`, the accounts (i.e., $Acct = [ID_U]ID_{RP}$) will inevitably change. Thus, a user is required to perform special operations to migrate her account to the updated RP system. It is worth noting that the user operations cannot be eliminated in the migration to the updated one; otherwise, it implies two colluding RPs could link a user's accounts across these RPs.

Restriction of the user-r script's origin. The user-i script forwards identity tokens to the user-r script, and the `postMessage` targetOrigin mechanism [51] restricts the recipient of the forwarded identity tokens, to ensure that the tokens will be sent to the intended $Enpt_{RP}$, as specified in the RP certificate. The targetOrigin is specified as a combination of protocol, port (if not present, 80 for `http` and 443 for `https`), and domain (e.g., `RP.com`). The user-r script's origin must accurately match the targetOrigin to receive tokens.

The targetOrigin mechanism does not check the whole URL path in $Enpt_{RP}$, but introduces no *additional* risk. Consider two RPs in one domain but receiving identity tokens through different endpoints, e.g., `https://RP.com/honest/tk` and `https://RP.com/malicious/tk`. This mechanism cannot distinguish them. Because a browser controls access to web resources following the same-origin policy (SOP) [58], a user's resources in the honest RP server is accessible to the malicious server. For example, it could steal cookies by `window.open('https://RP.com/honest').document.cookie`, even if the honest RP restricts only the HTTP requests to specific paths are allowed to access its cookies. So this risk is caused by the SOP model of browsers but not our designs, and commonly exists in SSO services accessed from COTS browsers [2], [6], [50], [21], [22].

VIII. Conclusion

This paper presents UPPRESSO, an untraceable and unlinkable privacy-preserving SSO system for protecting a user's online profile across RPs against both a curious IdP and colluding RPs. We propose an identity-transformation approach and design algorithms satisfying the requirements of security and privacy: (*a*) $\mathcal{F}_{Acct*}()$ determines a user's account, unique at an RP and unlinkable across RPs, (*b*) $\mathcal{F}_{PID_{RP}}()$ protects a visited RP's identity from the curious IdP, (*c*) $\mathcal{F}_{PID_U}()$ prevents colluding RPs from linking a user's multiple logins visiting different RPs, and (*d*) $\mathcal{F}_{Acct}()$ derives a user's permanent account from ephemeral pseudo-identities. The identity transformations are integrated into the widely-adopted OIDC protocol, maintaining user convenience and security guarantees of SSO services. Our experimental evaluations of the UPPRESSO prototype demonstrate its efficiency, with an average login taking 174 ms when the IdP, the visited RP, and user browsers are deployed in a virtual private cloud, or 421 ms when a user visits remotely.

References

[1] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, *OpenID Connect core 1.0 incorporating errata set 1*, The OpenID Foundation, 2014.

[2] D. Hardt, *RFC 6749: The OAuth 2.0 authorization framework*, Internet Engineering Task Force, 2012.

[3] J. Hughes, S. Cantor, J. Hodges, F. Hirsch, P. Mishra, R. Philpott, and E. Maler, *Profiles for the OASIS security assertion markup language (SAML) v2.0*, OASIS, 2005.

[4] T. Hardjono and S. Cantor, *SAML V2.0 subject identifier attributes profile version 1.0*, OASIS, 2018.

[5] P. Grassi, E. Nadeau, J. Richer, S. Squire, J. Fenton, N. Lefkovitz, J. Danker, Y.-Y. Choong, K. Greene, and M. Theofanos, *SP 800-63C: Digital identity guidelines: Federation and assertions*, National Institute of Standards and Technology (NIST), 2017.

[6] D. Fett, R. Küsters, and G. Schmitz, "SPRESSO: A secure, privacy-respecting single sign-on system for the Web," in *22nd ACM Conference on Computer and Communications Security (CCS)*, 2015, pp. 1358–1369.

[7] ——, "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.

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

[9] F. A. Services, "About Firefox Accounts," https://mozilla.github.io/application-services/docs/accounts/welcome.html, accessed August 20, 2019.

[10] M. Kroschewski and A. Lehmann, "Save the implicit flow? Enabling privacy-preserving RP authentication in OpenID Connect," *Privacy Enhancing Technologies*, vol. 2023, no. 4, pp. 96–116, 2023.

[11] R. Xu, S. Yang, F. Zhang, and Z. Fang, "MISO: Legacy-compatible privacy-preserving single sign-on using trusted execution environments," in *8th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2023.

[12] S. Hammann, R. Sasse, and D. Basin, "Privacy-preserving OpenID Connect," in *15th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2020, pp. 277–289.

[13] A. Dey and S. Weis, "PseudoID: Enhancing privacy for federated login," in *3rd Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 2010.

[14] Z. Zhang, M. Król, A. Sonnino, L. Zhang, and E. Rivière, "EL PASSO: Efficient and lightweight privacy-preserving single sign on," *Privacy Enhancing Technologies*, vol. 2021, no. 2, pp. 70–87, 2021.

[15] M. Isaakidis, H. Halpin, and G. Danezis, "UnlimitID: Privacy-preserving federated identity management using algebraic MACs," in *15th ACM Workshop on Privacy in the Electronic Society (WPES)*, 2016, pp. 139–142.

[16] G. Maganis, E. Shi, H. Chen, and D. Song, "Opaak: Using mobile phones to limit anonymous identities online," in *10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[17] C. Paquin, *U-Prove technology overview v1.1*, Microsoft Corporation, 2013.

[18] H. Fabric, "MSP implementation with Identity Mixer," https://hyperledger-fabric.readthedocs.io/en/release-2.2/idemix.html, accessed July 20, 2022.

[19] M. Asghar, M. Backes, and M. Simeonovski, "PRIMA: Privacy-preserving identity and access management at Internet-scale," in *52nd IEEE International Conference on Communications (ICC)*, 2018.

[20] W. Li and C. Mitchell, "Analysing the security of Google's implementation of OpenID Connect," in *13th International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

[21] Google for Developers, "Google Identity: Integration considerations," https://developers.google.com/identity/gsi/web/guides/integrate/, accessed January 13, 2025.

[22] Uber Developers, "OIDC Web SDK," https://developer.uber.com/docs/consumer-identity/oidc/web, accessed April 10, 2025.

[23] C. Guo, F. Lang, Q. Wang, and J. Lin, "UP-SSO: Enhancing the user privacy of SSO by integrating PPID and SGX," in *International Conference on Advanced Computing and Endogenous Security (ICACES)*, 2021.

[24] B. Diamond and J. Posen, "Succinct arguments over towers of binary fields," https://eprint.iacr.org/2023/1784, 2024.

[25] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, and M. Orru, "zk-Bench: A toolset for comparative evaluation and performance benchmarking of SNARKs," in *14th International Conference on Security and Cryptography for Networks (SCN)*, 2024.

[26] W. Ma, Q. Xiong, X. Shi, X. Ma, H. Jin, H. Kuang, M. Gao, Y. Zhang, H. Shen, and W. Hu, "GZKP: A GPU accelerated zero-knowledge proof system," in *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 340–353.

[27] J. Wang, G. Wang, and W. Susilo, "Anonymous single sign-on schemes transformed from group signatures," in *5th International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, 2013.

[28] J. Han, L. Chen, S. Schneider, H. Treharne, and S. Wesemeyer, "Anonymous single-sign-on for n designated services with traceability," in *23rd European Symposium on Research in Computer Security (ESORICS)*, 2018.

[29] J. Han, L. Chen, S. Schneider, H. Treharne, S. Wesemeyer, and N. Wilson, "Anonymous single sign-on with proxy re-verification," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 223–236, 2020.

[30] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, 1982, pp. 199–203.

[31] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT*, 2001.

[32] J. Camenisch and E. Herreweghen, "Design and implementation of the Idemix anonymous credential system," in *9th ACM Conference on Computer and Communications Security (CCS)*, 2002.

[33] M. Chase, S. Meiklejohn, and G. Zaverucha, "Algebraic MACs and keyed-verification anonymous credentials," in *21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[34] Z. Zhang, C. Xu, C. Jiang, and K. Chen, "TSAPP: Threshold single-sign-on authentication preserving privacy," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 1515–1527, 2024.

[35] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "PrivacyPass: Bypassing Internet challenges anonymously," *Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 164–180, 2018.

[36] Web Incubator CG, "TrustToken API," https://github.com/WICG/trust-token-api, accessed July 20, 2022.

[37] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *Journal of the ACM*, vol. 51, no. 2, pp. 231–262, 2004.

[38] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and T-PAKE in the password-only model," in *AsiaCrypt*, 2014.

[39] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *2nd Theory of Cryptography Conference (TCC)*, 2005.

[40] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu, "Highly-efficient and composable password-protected secret sharing (or: How to protect your Bitcoin wallet online)," in *1st IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, pp. 276–291.

[41] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "PESTO: Proactively secure distributed single sign-on, or how to trust a hacked server," in *5th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2020.

[42] S. Jarecki and X. Liu, "Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection," in *6th Theory of Cryptography Conference (TCC)*, 2009, pp. 577–594.

[43] A. Bagherzandi, S. Jarecki, Y. Lu, and N. Saxena, "Password-protected secret sharing," in *18th ACM Conference on Computer and Communications Security (CCS)*, 2011, pp. 433–444.

[44] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *28th USENIX Security Symposium*, 2019.

[45] S. Jarecki, H. Krawczyk, and J. Resch, "Updatable oblivious key management for storage systems," in *26th ACM Conference on Computer and Communications Security (CCS)*, 2019, pp. 379–393.

[46] J. Camenisch, A. de Caro, E. Ghosh, and A. Sorniotti, "Oblivious PRF on committed vector inputs and application to deduplication of encrypted data," in *23rd International Conference on Financial Cryptography and Data Security (FC)*, 2019, pp. 337–356.

[47] S. Casacuberta, J. Hesse, and A. Lehmann, "SoK: Oblivious pseudorandom functions," in *7th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2022.

[48] D. Fett, R. Küsters, and G. Schmitz, "An expressive model for the web infrastructure: Definition and application to the BrowserID SSO system," in *35th IEEE Symposium on Security and Privacy (S&P)*, 2014.

[49] B. de Medeiros, M. Scurtescu, P. Tarjan, and M. Jones, *OAuth 2.0 multiple response type encoding practices*, The OpenID Foundation, 2014.

[50] J. Richer, "MITREid Connect v1.3.3," http://mitreid-connect.github.io/index.html, accessed August 20, 2021.

[51] The WHATWG Community, "HTML living standard: 9.3 cross-document messaging," https://html.spec.whatwg.org/multipage/web-messaging.html, accessed June 7, 2022.

[52] J. Eisinger and E. Stark, *W3C candidate recommendation: Referrer policy*, World Wide Web Consortium (W3C), 2017.

[53] K. Urushima, "jsrsasign (RSA-Sign JavaScript Library)," https://kjur.github.io/jsrsasign/, accessed August 20, 2019.

[54] A. Bender, J. Katz, and R. Morselli, "Ring signatures: Stronger definitions, and constructions without random oracles," in *3rd Theory of Cryptography Conference (TCC)*, 2006, pp. 60–79.

[55] M. Thomson and C. Wood, *RFC 9458: Oblivious HTTP*, Internet Engineering Task Force, 2024.

[56] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *13th USENIX Security Symposium*, 2004, pp. 303–320.

[57] A. Faz-Hernandez, S. Scott, N. Sullivan, R. Wahby, and C. Wood, *draft-irtf-cfrg-hash-to-curve-16: Hashing to elliptic curves*, Internet Engineering Task Force, 2022.

[58] W3C Web Security, "Same origin policy," https://www.w3.org/Security/wiki/Same_Origin_Policy, accessed June 7, 2022.