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


## Abstract

Single sign-on (SSO) allows 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 track a user's all 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 the transformations provide *unlinkability*. This approach protects the identities of a user and the visited RP, while enabling SSO services accessed from a commercial-off-the-shelf browser without plug-ins or extensions. We built a prototype of UPPRESSO on top of MITREid Connect, an open-source SSO system. The extensive evaluations show that it fulfills the security and privacy requirements of SSO with reasonable overheads.


## 1 Introduction

Single sign-on (SSO) protocols such as OpenID Connect (OIDC) [47], OAuth 2.0 [31], and SAML [30, 32], 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 a trusted web service, known as the *identity provider* (IdP). An RP delegates 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, an OIDC user requests to log into an RP, which constructs an identity-token request with its identity (denoted as $ID_{RP}$) and posts (or redirects) it to a trusted IdP. After authenticating the user, the IdP issues an identity token binding the identities of the user and the target RP (i.e., $ID_U$ and $ID_{RP}$), which is returned to the user and then forwarded to the RP. Finally, the RP verifies the identity token to determine if the token holder is allowed to log in. Thus, a user maintains only one credential for the IdP, instead of multiple credentials for different RPs.

OIDC also provides comprehensive services of identity management, by enabling an IdP to enclose more user attributes in identity tokens, along with the authenticated user's identity. The attributes (e.g., age and hobby) are maintained at the IdP and enclosed in identity tokens after a user's authorization [31, 47].

The wide adoption of SSO raises concerns on user privacy, because it facilitates the tracking of a user's login activities by interested parties [21, 22, 26, 43]. To issue identity tokens, an IdP should know the target RP to be visited by a user and the login time. So a curious IdP could potentially track a user's all login activities over time [21, 22], called *IdP-based login tracing* in this paper. Another privacy risk arises from the fact that RPs learn the user's identity from the identity tokens they receive. If an identical user identity is enclosed in tokens for a user to visit different RPs, colluding RPs could link the logins across these RPs to learn the user's online profile [43, 48]. This risk 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 [21, 22, 26, 27, 43, 57] aim to implement authentication and identity management with the following features: (*a*) *User authentication only to the IdP*, which eliminates the need for authentication between a user and any RP, and then requires a user to maintain only the credential for the IdP, (*b*) *User identification at each RP*, which enables an RP to recognize each user by an account unique within the RP to provide customized services across multiple logins, and (*c*) *IdP-confirmed attribute provision*, where a user's attributes are maintained at a trusted IdP and provided to RPs after the user's authorization. In Section 2, we analyze existing privacy-preserving solutions for SSO and also identity federation.

We present UPPRESSO, an Untraceable and Unlinkable Privacy-PREserving Single Sign-On protocol. It proposes *identity transformations* and integrates them in popular OIDC services. 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 $PID_U$. 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 to this RP.

UPPRESSO prevents both IdP-based login tracing and RP-based identity linkage, while existing privacy-preserving SSO solutions address only one of them [21, 22, 26, 27, 39] or need another fully-trusted server in addition to the IdP [57]. Meanwhile, UPPRESSO implements SSO services accessed from commercial-off-the-shelf (COTS) browsers without any plug-ins or extensions, and provides all desirable features of SSO protocols as listed above. In contrast, privacy-preserving identity federation [12, 17, 33, 42, 45, 58] supports only some but not all of these features, and always requires a browser plug-in or extension.

Our contributions are as follows.

- We proposed a novel identity-transformation approach for privacy-preserving SSO and designed identity-transformation algorithms with desirable properties.



*Conference'17, Washington, DC, USA*



ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- We developed the UPPRESSO protocol based on the identity transformations with several designs specific to web applications, 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 present the background and related works in Section 2 and the identity-transformation approach in Section 3, followed by the detailed designs in Section 4. Security and privacy are analyzed in Section 5. We explain the prototype implementation and evaluations in Section 6, and discuss extended issues in Section 7. Section 8 concludes this work.

## 2 Background and Related Work

### 2.1 OIDC Services

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 tokens, but have common security requirements for identity tokens. We present our designs in the implicit flow and discuss the support for the authorization code flow in Section 7.

Users and RPs register at an OIDC IdP with their identities and other information such as user credentials and RP endpoints. As shown in Figure 1, on receiving a login request, an RP constructs an identity-token request with its identity and the scope of requested user attributes. This request is posted (or redirected) 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 requested attributes, a validity period, etc. The user then forwards the identity token to the RP's endpoint. The RP verifies the token and allows the holder to log in as the user (pseudo-)identity enclosed.

User operations in OIDC are conducted by a user agent (or browser typically). To process tokens retrieved from the IdP, which is carried with a URL following the fragment identifier # instead of ? due to security considerations [11], a COTS browser needs to download a web script from the visited RP, called the *user-r* script. Further, another *user-i* script downloaded from the IdP server, is recommended to provide an in-context user experience [25, 40, 51], which is called *pop-up UX*, for the user's attribute authorization. Then, two scripts are responsible for communicating with the respective origin servers (e.g., token requesting, receiving and forwarding) and also the cross-origin communications using the `postMessage`

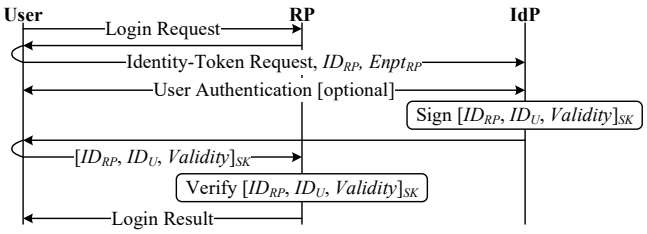



**Figure 1: The implicit SSO login flow of OIDC**

HTML5 API within a browser. Alternatively, if a browser works as the user agent with *only* the user-r script, user attributes are authorized in the redirected IdP window. This is called *redirect UX*.

The following features are desired in SSO services and supported by OIDC and other popular SSO protocols [26, 30–32, 47].

**User authentication only to the IdP.** RPs only verify the identity tokens issued by an IdP, and the authentication between a user and the IdP is conducted *independently* of the steps where identity tokens are generated. This offers advantages. First, the IdP authenticates users by any appropriate means such as passwords, one-time passwords, or multi-factor authentication (MFA). Meanwhile, a user only maintains her credential for the IdP. If it is lost or leaked, the user needs to renew it only at the IdP.

**User identification at each RP.** An RP recognizes each user by an identity (or account) *unique* within the RP to provide customized services across multiple logins. Such non-anonymous SSO systems are more desirable in various applications than anonymous services.

**IdP-confirmed attribute provision.** An IdP may enclose also user attributes in identity tokens along with user (pseudo-)identities [31, 47]. A user maintains her attributes at the trusted IdP, which provides only pre-selected ones to RPs or obtains the user's authorization before enclosing attributes.

### 2.2 Privacy-Preserving Solutions for SSO and Identity Federation

SSO allows a user to log into an RP, without by herself maintaining an account at the RP or holding a long-term secret verified by the RP, so that SSO services are accessible from COTS browsers. On the contrary, identity federation enables a user registered at a trusted IdP to be accepted by other parties, potentially with different accounts, but extra *authentication operations between the user and RPs* are involved, so plug-ins or extensions are needed to access such services from a browser. Thus, although the term "single sign-on (SSO)" was used in some schemes [12, 28, 29, 42, 54, 58], this paper refers to them as *identity federation* to emphasize this difference.

Table 1 compares privacy-preserving schemes of SSO and identity federation, by analyzing whether a service feature is supported and whether a privacy threat is prevented or not. This table also shows whether an extra trusted server is required, in addition to the IdP. Among them only UPPRESSO satisfies all desirable properties.

**Privacy-preserving SSO.** Some schemes [21, 22, 26] prevent either IdP-based login tracing or RP-based identity linkage, but not both. First of all, pairwise pseudonymous identifiers (PPIDs) are specified [30, 47] and recommended [26] in OIDC to protect user privacy against colluding RPs. An IdP assigns a unique PPID for a user to log into some RP and encloses it in identity tokens, so colluding RPs cannot link the user. It does not 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 globally-unique user identities enclosed in identity tokens. For example, in BrowserID [21] after authenticating a user, an IdP issues a "user certificate" binding the user's identity but no attribute 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

**Table 1: Privacy-preserving solutions of SSO and identity federation**

| | Solution | Service Feature† | | | Privacy Threat♯ | | Extra Trusted Server‡ |
|---|---|---|---|---|---|---|---|
| | | User Authentication Only to the IdP | User Identification at Each RP | IdP-confirmed Attribute Provision | IdP-based Login Tracing | RP-based Identity Linkage | |
| SSO | OIDC w/ PPID [26] | ● | ● | ● | ○ | ● | ● |
| | BrowserID [21] | ● | ● | ○ | ● | ○ | ● |
| | SPRESSO [22] | ● | ● | ○ | ● | ○ | ○[1] |
| | POIDC [27, 39] | ● | ● | ● | ● | ○ | ● |
| | MISO [57] | ● | ● | ● | ● | ● | ○ |
| Identity Federation | PRIMA [1] | ○ | ● | ● | ● | ○ | ● |
| | PseudoID [12] | ○ | ● | ○ | ● | ● | ○ |
| | Opaak [42] | ○ | ◐[2] | ○ | ● | ● | ● |
| | PP-IDF [33, 45, 58] | ○ | ● | ●[3] | ● | ● | ● |
| | Fabric Idemix [17] | ○ | ◐[4] | ● | ● | ● | ● |
| SSO | UPPRESSO | ● | ● | ● | ● | ● | ● |

†. **Service Feature**: ● supported, ○ unsupported, ◐ partially supported. ♯. **Privacy Threat**: ● prevented, ○ not prevented.
‡. **Extra Trusted Server**: ● without, ○ required.
1. SPRESSO assumes a *malicious* IdP (but honest IdPs in others), and a forwarder server is trusted to decrypt the RP identity and forward tokens to the RP.
2. Opaak supports two exclusive pseudonym options: (*a*) linkable within an RP but unlinkable across multiple RPs and (*b*) unlinkable for any pair of actions.
3. Different from [33, 58], a credential in U-Prove [45] may contain some attributes that are *invisible* to the IdP, in addition to the ones confirmed by the IdP.
4. In the original design of Idemix [5], every user logs into an RP with a unique account, but Fabric Idemix [17] implements completely-anonymous services.

a one-time tag (or pseudo-identity) for each login [22], or a user sends an identity-token request with a hash commitment on the target RP's identity in POIDC [27, 39], which are enclosed in identity tokens along with the user's unique identity. As in SPRESSO the IdP is assumed to be malicious, there is no IdP-confirmed attribute provision. Besides, a variation of POIDC [27] 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 [13, 16, 41], which is impracticable for a user agent in SSO systems.

MISO [57] decouples the calculation of PPIDs from an honest IdP in OIDC, to *an extra 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.

**Privacy-preserving identity federation.** First of all, in these schemes [1, 12, 17, 33, 42, 45, 58], additional authentication operations between the user and RPs are involved.

In PRIMA [1], an IdP signs a credential that binds user attributes and a verification key. Using the signing key, the user provides selected attributes to RPs. This verification key works as the user's identity and exposes her to RP-based identity linkage. PseudoID [12] introduces another trusted server in addition to the IdP, to blindly sign [9] an access token that binds a user secret, a pseudonym but no other user attributes. The user then unblinds this token, authenticates herself to an RP using the secret, and log into it.

Several schemes [17, 33, 42, 45, 58] are designed based on anonymous credentials [5, 6, 8]. For instance, the IdP signs anonymous credentials in Opaak [42], UnlimitID [33], U-Prove [45], and EL PASSO [58], each of which binds a long-term user secret. Then a user proves ownership of the anonymous credentials using her secret, and discloses IdP-confirmed attributes in the credentials in most schemes except Opaak. Similarly, Fabric [17] integrates Idemix anonymous credentials [5] for completely-unlinkable pseudonyms and IdP-confirmed attribute provision.

Some privacy-preserving identity federation solutions [12, 17, 33, 42, 45, 58] 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 [5, 6, 8] when the ownership of these credentials is proved to colluding RPs. However, this protection brings extra complexity to users, and such services cannot be accessed from COTS browsers. In Section 7 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 [54] and implemented using cryptographic primitives such as group signature and ZKP [28, 29, 54]. Extended designs including proxy re-verification [29], designated verification [28] and distributed IdP servers [59], are also considered.

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

# 3 The Identity-Transformation Approach

## 3.1 Security Requirements for SSO Services

Non-anonymous SSO services [26, 30–32, 47] are designed to allow a user to log into an honest RP with her account at this RP, by presenting *identity tokens* issued by a trusted IdP.

To achieve this goal, the IdP issues an identity token that specifies the visited RP (i.e., *RP designation*) and identifies the authenticated user (i.e., *user identification*) by (pseudo-)identities [31, 32, 47]. An

**Table 2: The (pseudo-)identities in UPPRESSO**

| 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 |
| $PID^l_{U_{i,j}}$ | 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 |
| $Acct_{i,j}$ | The $i$-th user's account at the $j$-th RP. | Permanent |

honest RP checks the RP's identity enclosed in a token before accepting it and allows the token holder to log in as the specified account. This prevents malicious RPs from replaying received tokens to gain illegal access to other honest RPs as victim users.

*Authenticity*, *confidentiality*, and *integrity* of identity tokens are necessary to prevent forging, eavesdropping and tampering. Identity tokens are signed by the trusted IdP, transmitted over HTTPS [31, 32, 47], and finally forwarded to target RPs by the authenticated user. Regular security mechanisms in a browser (e.g., HTTP session and `postMessage` targetOrigin) are also required for confidentiality of tokens [11, 20, 21, 25] (see Section 4.4 for details).

## 3.2 Identity Transformation

UPPRESSO implements privacy-preserving SSO that ensures security as above, while preventing both IdP-based login tracing and RP-based identity linkage. The requirements of security and privacy are satisfied through *transformed identities* in the identity tokens. Table 2 lists the notations, and a sub/super-script (i.e., $i$, $j$, or $l$) may be omitted if it does not cause ambiguity.

A user initiates a login by negotiating an *ephemeral* pseudo-identity $PID_{RP}$ with the target RP and sending an identity-token request for $PID_{RP}$ to an IdP. After successfully authenticating the user as $ID_U$, the IdP calculates an *ephemeral* $PID_U$ based on $ID_U$ and $PID_{RP}$ and then issues an identity token that binds $PID_U$ and $PID_{RP}$. On receiving a verified token, the RP calculates the user's *permanent* $Acct$ and allows the token holder to log in. 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.

For RP designation, $PID_{RP}$ should be associated *uniquely* with the target RP. For user identification, with *ephemeral* $PID^l_U$ in each login, the designated RP should be able to derive a locally-unique *permanent* account (i.e., $Acct$) for the user at this RP. In order to prevent IdP-based login tracing, it is essential to ensure that the IdP does not obtain any information about $ID_{RP}$ from any $PID^l_{RP}$. Thus, in a user's multiple logins visiting an RP, *independent* $PID^l_{RP}$s

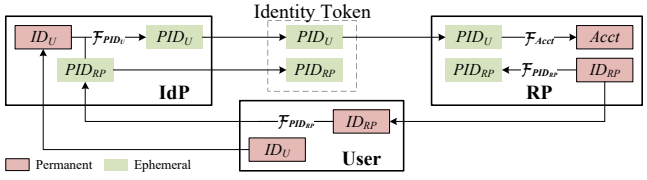



**Figure 2: Identity transformations in UPPRESSO**

and *independent* $PID^l_U$s should be generated: The IdP should not be able to link multiple logins visiting a given RP, while the RP's identity is kept unknown to the IdP. If $PID^l_U$ is not completely independent of each other, it is possible for the IdP to link multiple logins visiting a certain RP. Finally, to prevent RP-based identity linkage, the RP should not obtain any information about $ID_U$ from any $PID_{U,j}$, which implies that $PID_{U,j}$ for visiting different RPs should also be independent of each other.

We propose the following identity transformations in a login initiated by a user to visit an RP in privacy-preserving SSO:

- $\mathcal{F}_{Acct*}(ID_U, ID_{RP}) = Acct$, determining a user' account at an RP. Given $ID_U$ and $ID_{RP}$, $Acct$ is *unique* at this RP.
- $\mathcal{F}_{PID_{RP}}(ID_{RP}) = PID_{RP}$, calculated by the user. 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. 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. That is, in the user's multiple logins visiting the RP, $Acct = \mathcal{F}_{Acct*}(ID_U, ID_{RP})$ is always derived.

# 4 The Designs of UPPRESSO

## 4.1 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 [30–32, 47].

**Honest-but-curious IdP.** The IdP strictly follows the protocols, while remaining interested in learning about user activities. For example, it could store received messages to infer the relationship between $ID_U$, $ID_{RP}$, $PID_U$, and $PID_{RP}$. It never actively violates the protocols. We discuss malicious IdPs in Section 7.

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

**Malicious users.** Adversaries could control a set of users by stealing their credentials or registering Sybil users in UPPRESSO. Their objective [20, 22] 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. A malicious user could modify, insert, drop, or replay messages or even behave arbitrarily in login flows.

**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.

## 4.2 Assumptions

We assume secure communications between honest entities (e.g., HTTPS), and the cryptographic primitives are secure. The software

**Table 3: 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 IdP-signed RP certificate binding $ID_{RP_j}$ and $Enpt_{RP_j}$. |

stack of an honest entity (e.g., a browser) is correctly implemented to transmit messages to receivers as expected.

UPPRESSO is designed for users who value privacy, or provides an 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. Meanwhile, our work focuses only on the privacy threats introduced by SSO protocols, and does not consider the tracking of user activities by traffic analysis or crafted web pages, as they can be prevented by other defenses. For example, FedCM [19] proposes to disable iframe and third-party cookies in SSO, which could be exploited to track users.

## 4.3 Identity-Transformation Algorithms

We design identity transformations on an elliptic curve $\mathbb{E}$, and Table 3 lists the notations.

***$ID_U$, $ID_{RP}$ 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 a registered 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.

$Acct = \mathcal{F}_{Acct*}(ID_U, ID_{RP_j}) = [ID_U]ID_{RP_j} = [ur_j]G$ is automatically assigned to a user at every RP, and a user's accounts at different RPs are inherently different and unlinkable.

***$ID_{RP}$-$PID_{RP}$* Transformation.** In each login, a user selects a random number $t \in \mathbb{Z}_n$ to calculate $PID_{RP}$.

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

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

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

***$PID_U$-$Acct$* Transformation.** The user sends $t$ to the target RP as a trapdoor to derive her account. After verifying a token that encloses $PID_U$ and $PID_{RP}$, it calculates $Acct$ as follows.

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

From Equations 1, 2, and 3, it is derived that

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

With the help of $t$, the visited RP derives an identical account for a user in her different logins. It is exactly the user's *permanent* account at this RP.

A user's identity $u$ 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.

## 4.4 The Designs Specific for Web Applications

In commonly-used SSO protocols [30–32, 47], an IdP needs to know the visited RP to ensure confidentiality of identity tokens. For instance, in OIDC services with redirect UX, an RP's endpoint to receive tokens is stored as the `redirect_uri` parameter at the IdP. The IdP employs HTTP 302 redirection to send tokens to the RP, by setting this parameter as the target URL in the HTTP response to a user's identity-token request [47]. Alternatively, in an OIDC system with pop-up UX [25, 40, 51], the user-i script confirms the target RP's origin with the IdP, and then forwards tokens to the user-r script restricted by this origin.

In UPPRESSO the IdP does not know about the visited RP, requiring a user agent by itself to calculate $PID_{RP}$ and forward identity tokens to the RP. We implement UPPRESSO compatibly in OIDC services with pop-up UX [25, 51], and the user-agent functions are also implemented by the two scripts, downloaded from the IdP and the visited RP, respectively. The user-i script is necessary in UPPRESSO and then it cannot be integrated in OIDC services with redirect UX, because the user-r script could leak its origin to the IdP web server (i.e., break IdP untraceability) due to the automatic inclusion of an HTTP `referer` header in HTTP requests it sends.

In original OIDC systems with pop-up UX, the user-i script implements attribute authorization and token receiving, while the user-r script receives token requests from the visited RP and forwards tokens to the target RP. We improve these scripts as below to further support identity transformations. In each login, the user-r script prepares $ID_{RP}$ and $Enpt_{RP}$ through an RP certificate, returned in the token request. The user-r script sends the certificate to the user-i script, which verifies it to extract $ID_{RP}$ and $Enpt_{RP}$. Note that in UPPRESSO a user configures nothing locally for the IdP's public key is set in the user-i script, like in other popular SSO systems [30–32, 47]. Then, within a browser the user-i script calculates $PID_{RP} = [t]ID_{RP}$ based on $ID_{RP}$ extracted from the verified RP certificate, to request a token for $PID_{RP}$. On receiving an identity-token request, the IdP web server checks the included `referer` header to ensure it is sent by the user-i script.

After receiving a token from the IdP, the user-i script needs to ensure the user-r script will forward the token to $Enpt_{RP}$ specified in the verified RP certificate. As the communications between scripts occurs within COTS browsers using the `postMessage` HTML5 API, the `postMessage` targetOrigin mechanism [49] restricts the recipient. When the user-i script sends messages, the recipient's origin is set as a parameter, e.g., `postMessage(tk, 'https://RP.com')`,

including the protocol (i.e., `https`), the domain (i.e., `RP.com`), and a port if applicable. Only the script downloaded from this targetOrigin is a legitimate recipient. This design is commonly used to correctly forward tokens in popular OIDC systems [11, 21, 22, 46, 47].

Besides, when a user attempts to visit an RP in UPPRESSO (and OIDC systems with pop-up UX), the RP window downloads the user-r script, which in turn "pops up" a new browser window to access the IdP and download the user-i script. To prevent referer leakage during this access, we need to ensure the HTTP request does not carry a `referer` header, which reveals the visited RP's domain to the IdP server.[1] Fortunately, in UPPRESSO this pop-up is an HTTP redirection from the new window to the IdP (Steps 1.2-1.3 in Figure 3), but not a direct visit. The HTTP redirection response from the RP server includes a `referrer-policy=no-referrer` header, which ensures that the HTTP request to access the IdP carries no `referer` header. This method is specified by W3C [15] and widely supported. We have tested it in various browsers such as Chrome, Safari, Edge, Opera, and Firefox, and confirmed no referer leakage.

## 4.5 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 sends 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 $Cert_{RP} = [ID_{RP}, Enpt_{RP}, *]_{SK}$, 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 $Cert_{RP}$ if they are valid.

**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 is required that $ID_U$ is known *only* to the IdP. $ID_U$ 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 a token binding $PID_U = [ID_U]PID_{RP}$.

**SSO Login.** A registered user may log into any registered RP, after authenticated herself to the IdP. A login flow involves four steps: script downloading, RP identity transformation, identity-token generation, and *Acct* calculation. 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 `postMessage` invocations between two scripts (or browser windows) within a browser.

1. *Script Downloading*. The browser downloads user-agent scripts from the visited RP and the IdP as below.
   - 1.1 When requesting any protected resources at the RP, the user downloads the user-r script.
   - 1.2 The user-r script opens a window in the browser to visit the RP's login path, which is then redirected to the IdP.
   - 1.3 The redirection to the IdP downloads the user-i script.

2. *RP Identity Transformation.* The user and the RP negotiate $PID_{RP} = [t]ID_{RP}$.
   - 2.1 The user-i script chooses a random number $t \in \mathbb{Z}_n$ and sends it to the user-r script through `postMessage`. The user-r script then forwards $t$ to the RP.
   - 2.2 The RP verifies if the received $t$ is an integer in $\mathbb{Z}_n$, and replies with $Cert_{RP}$ and the scope of requested attributes, forwarded by the user-r script to the user-i script.
   - 2.3 The user-i script verifies $Cert_{RP}$, extracts $ID_{RP}$ and $Enpt_{RP}$ from $Cert_{RP}$, and calculates $PID_{RP} = [t]ID_{RP}$.

3. *Identity-Token Generation.* The IdP calculates $PID_U = [ID_U]PID_{RP}$ and signs an identity token as follows.
   - 3.1 The user-i script sends an identity-token request for $PID_{RP}$ on behalf of the user.
   - 3.2 The IdP authenticates the user, if not authenticated yet.
   - 3.3 The user-i script displays the RP's common name, locally obtains the user's authorization for the requested attributes, and then sends the scope of the authorized attributes. The IdP checks that $PID_{RP}$ is a point on $\mathbb{E}$, calculates $PID_U = [ID_U]PID_{RP}$, and signs $[PID_{RP}, PID_U, Issuer, Validity, Attr]_{SK}$, where $Issuer$ is the IdP, $Validity$ indicates the validity period, and $Attr$ contains the authorized user attributes.
   - 3.4 The IdP replies with the identity token to the user-i script.

4. *Acct Calculation.* The RP receives the identity token and allows the user to log in.
   - 4.1 The user-i script forwards the identity token to the user-r script, which then sends it to the RP through $Enpt_{RP}$.
   - 4.2 The RP verifies the signature and the validity period of the token and calculates $Acct = [t^{-1}]PID_U$.
   - 4.3 The RP allows the user to log in as $Acct$.

If any verification fails, this 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 a token in Step 4.2 if the signature is invalid.

## 4.6 Compatibility with OIDC

An OIDC system works in two alternative modes [11, 25, 40, 51]: pop-up UX with two web scripts, and redirect UX with only the user-r script. UPPRESSO only works in OIDC systems with pop-up UX. When a user attempts to visit an RP in UPPRESSO, a user agent

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

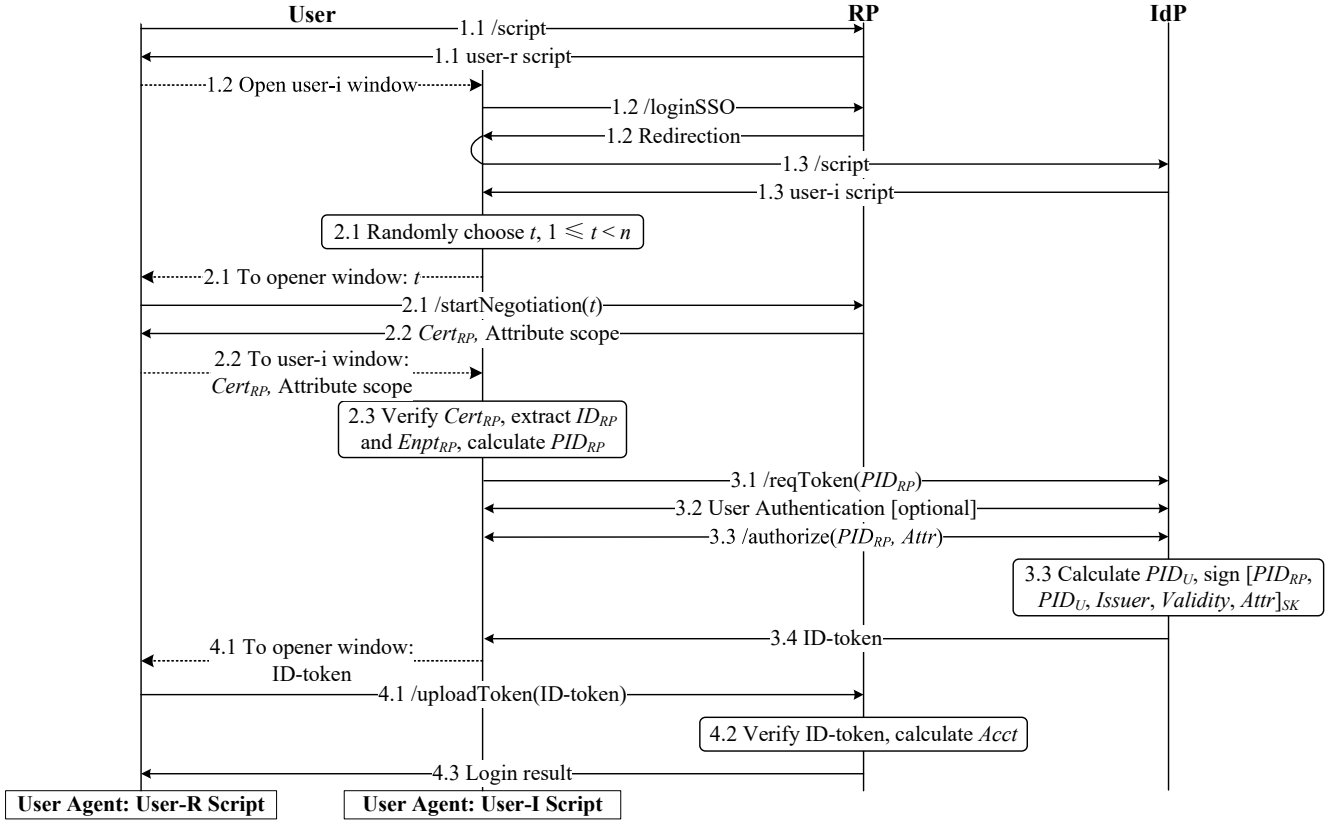



**Figure 3: The SSO login flow of UPPRESSO**

is prepared by two scripts in the *script downloading* step. These scripts are downloaded, like in OIDC systems with pop-up UX.

The *RP identity transformation* step in UPPRESSO take place within a browser, and the RP responds with its certificate after receiving $t$. The RP response is constructed following the standard format of OIDC requests [40], but then processed by the user-i script to remove any information leaking the RP's identity.

The operations of *identity-token generation* in UPPRESSO are compatible with those in OIDC, because (*a*) the calculation of $PID_U$ is actually a special method to generate PPIDs and (*b*) $PID_{RP}$ can be viewed as a "dynamic" RP pseudo-identity so that `redirect_uri` is missing in the responses by the IdP. Finally, the operations of *Acct calculation* in UPPRESSO are identical to those in OIDC, because the calculation of *Acct* is a mapping from the user pseudo-identity enclosed in tokens to a local account at the visited RP.

The compatibility is experimentally confirmed through our prototype implementation. When we built the IdP prototype of UPPRESSO on top of MITREid Connect [46], only 23 lines of Java code are added: three lines to calculate $PID_U$ as a special PPID using Java cryptographic libraries, and 20 lines to support pop-up UX with identity transformations.

## 5 Security and Privacy Analysis

We prove the security and privacy guarantees of UPPRESSO in different adversarial scenarios.

### 5.1 Adversarial Scenarios

Based on our design goals and the potential adversaries discussed in Section 4.1, we consider three adversarial scenarios as below and the guarantees are proved against *different* adversaries.

**Impersonation and identity injection against security.** Malicious RPs and users could collude [20–22], 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 user's account.

**Login tracing by an IdP.** The honest-but-curious IdP tries to infer the identities of the RPs being visited by an honest user [21, 22].

**Identity linkage by colluding RPs.** Malicious RPs could collude with each other and even malicious users, attempting to link logins across these RPs initiated by honest users [43, 48].

### 5.2 Security

In secure SSO systems, an identity token $TK$ 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*, for it makes no sense to discuss the login results at malicious RPs. When authenticity, confidentiality and integrity of $TK$ are ensured by secure communications and digital signatures in UPPRESSO, we summarize the following sufficient conditions of secure SSO services [20–22]:

**RP Designation.** $TK$ designates the target RP, and only the designated honest RP derives a meaningful account which responds to some registered user after accepting $TK$. That is, *at any honest RPs*

*other than the designated one, no meaningful account will be derived from TK.*

**User Identification.** At the designated RP, *TK* identifies only the user who requests this token from the IdP. That is, *from TK the designated honest RP will derive the account exactly corresponding to the identified user or meaningless accounts.*

The above definitions of RP designation and user identification guarantee that even an unmatching token or $t$ does not break security of UPPRESSO. These properties are defined specially for the proposed identity transformations in UPPRESSO, because checking $ID_{RP}$ in identity tokens by an RP is *necessary* in other SSO solutions [21, 22, 26, 27, 39, 47, 57]. Our definitions of RP designation and user identification are different, but well guarantee security of SSO services in UPPRESSO.

Let's assume totally $s$ users and $p$ RPs in UPPRESSO, whose identities are denoted as $\mathbf{u} = \{u_{i;1\le i\le s}\}$ and $\mathbb{ID}_{RP} = \{[r_j;1\le j\le p]G\}$, respectively. There are meaningful accounts $Acct_{i,j} = [u_i]ID_{RP_j} = [u_i r_j]G$ automatically assigned at each RP. Because $[r_j]G$ is also a generator on $\mathbb{E}$ of order $n$, $Acct_{i,j} = [u_i r_j]G$ is *unique* at each RP as $u_i$ is uniquely selected in $\mathbb{Z}_n$.

$Acct_{i,j}$ and $ID_{RP_j}$ are publicly-known, but $u_i$ and $r_j$ are kept unknown to malicious adversaries. We prove that, in UPPRESSO an identity token binding $PID_{RP} = [t]ID_{RP}$ and $PID_U = [u]PID_{RP}$, which is requested by an authenticated user with $ID_U = u$ to visit an RP with $ID_{RP}$, satisfies RP designation and user identification.

Theorem 1 (RP Designation). Given $u \in \mathbf{u}$ and $ID_{RP} \in \mathbb{ID}_{RP}$, from $TK$ binding $PID_{RP} = [t]ID_{RP}$ and $PID_U = [u]PID_{RP}$, an honest RP with $ID_{RP'} \neq ID_{RP}$ cannot derive any $Acct = [u']ID_{RP'}$ where $u' \in \mathbf{u}$ and $ID_{RP'} \in \mathbb{ID}_{RP}$.

**Proof.** A malicious user colluding with malicious RPs, might forward $TK$ to some honest RP other than the designated one, along with a manipulated trapdoor $t'$. Then, this deceived RP derives $Acct = [t'^{-1}]PID_U = [t'^{-1}utr]G$.

We prove that, when $u_{i;1\le i\le s}$ and $r_{j;1\le j\le p}$ are unknown, for any $[r']G \neq [r]G$ but $[r']G \in \mathbb{ID}_{RP}$, the adversaries cannot find $t$ and $t'$ satisfying $[t'^{-1}utr]G = [u'r']G$ where $u' \in \mathbf{u}$. This attack can be described as an account-collision game $\mathcal{G}_A$ between an adversary and a challenger: the adversary receives from the challenger a set of RP identities $\mathbb{ID}_{RP}$, $ID_{RP_a}$ and a set of accounts $\{Acct_{i,j} = [u_i r_j]G\}$, and then outputs $(i_1, i_2, b, t, t')$ where $b \neq a$ and $b \in [1, p]$. If $[t'^{-1}t]Acct_{i_1,a} = [t'^{-1}u_{i_1}tr_a]G = [u_{i_2}r_b]G = Acct_{i_2,b}$, which occurs with a probability $\Pr_s$, the adversary succeeds.

As depicted in Figure 4, based on $\mathcal{G}_A$ we can design a probabilistic polynomial time (PPT) algorithm $\mathcal{D}_A^*$ to solve the ECDLP: find $x \in \mathbb{Z}_n$ satisfying $Q = [x]G$, where $Q$ is a random point on $\mathbb{E}$ and $G$ is a generator on $\mathbb{E}$ of order $n$.

The algorithm $\mathcal{D}_A^*$ works as follows. The input of $\mathcal{D}_A^*$ is in the form of $(G, Q)$. On receiving such an input, the challenger chooses $r_1, \cdots, r_p$ random in $\mathbb{Z}_n$ to calculate $\{[r_{j;1\le j\le p}]G\}$, randomly chooses $a \in [1, p]$ to replace $[r_a]G$ with $Q$, and then constructs $\mathbb{ID}_{RP} = \{[r_1]G, \cdots, [r_{a-1}]G, Q, [r_{a+1}]G, \cdots, [r_p]G\}$. The challenger chooses $u_1, \cdots, u_s$ random in $\mathbb{Z}_n$ to calculate $\{Acct_{i,j} = [u_{i;1\le i\le s}]ID_{RP_{j;1\le j\le p}}\}$. Then, it sends $\mathbb{ID}_{RP}$, $Q$ and $\{Acct_{i,j}\}$ to the adversary, which returns the result $(i_1, i_2, b, t, t')$. Finally, the challenger calculates $e = t'u_{i_1}^{-1}t^{-1}u_{i_2}r_b \bmod n$, and sets $e$ as the output.

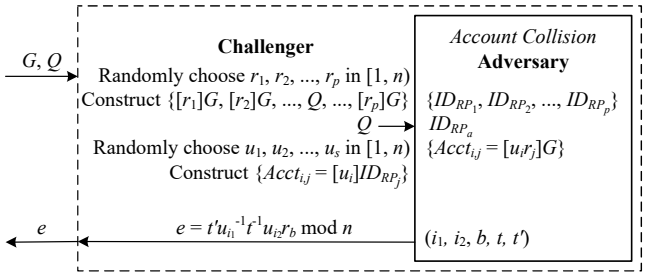



**Figure 4: The PPT algorithm $\mathcal{D}_A^*$ constructed based on the account-collision game to solve the ECDLP.**

If $[t'^{-1}u_{i_1}t]Q = [u_{i_2}r_b]G$ and the adversary succeeds in $\mathcal{G}_A$, $\mathcal{D}_A^*$ outputs $e = x$ because $Q = [t'u_{i_1}^{-1}t^{-1}u_{i_2}r_b]G$. If the adversary would have advantages in $\mathcal{G}_A$ and $\Pr_s$ is non-negligible regardless of the security parameter $\lambda$, we would find that $\Pr\{\mathcal{D}_A^*(G, [x]G) = x\} = \Pr_s$ also becomes non-negligible even when $\lambda$ is sufficiently large. This violates the ECDLP assumption. Thus, the probability that the adversary succeeds in $\mathcal{G}_A$ is negligible, and RP designation is ensured. □

Theorem 2 (User Identification). Given $u \in \mathbf{u}$ and $ID_{RP} \in \mathbb{ID}_{RP}$, from $TK$ binding $PID_{RP} = [t]ID_{RP}$ and $PID_U = [u]PID_{RP}$, the meaningful account derived at the honest RP with $ID_{RP}$ is only $Acct = [u]ID_{RP}$.

**Proof.** $Acct = [t^{-1}]PID_U = [t^{-1}ut]ID_{RP} = [u]ID_{RP}$ is calculated by the designated RP receiving $t$. It exactly corresponds to the authenticated user (i.e., the meaningful account identified by $TK$).

Next, we consider $TK$ forwarded by malicious adversaries to the target RP, but with a manipulated random trapdoor $t' \neq t$. The designated RP will derive another account $[t'^{-1}]PID_U = [t'^{-1}ut]ID_{RP}$. Because $G$ is a generator on $\mathbb{E}$ of order $n$, $ID_{RP}$ and $[ut]ID_{RP}$ are also generators on $\mathbb{E}$. Therefore, when $u_{i;1\le i\le s}$ are unknown, the probability that $[t'^{-1}ut]ID_{RP}$ happens to be another meaningful account $[u']ID_{RP}$ at this RP is $\frac{s-1}{n}$, because $u_i$ is randomly selected in $\mathbb{Z}_n$ by the IdP.

This probability becomes negligible, when $n$ is sufficiently large. Adversaries are unable to solve $t'$ satisfying that $[t'^{-1}]PID_U = Acct'$ due to the ECDLP, where $Acct'$ is any meaningful account at this RP. As a result, $[t'^{-1}]PID_U$ $(t' \neq t)$ results in no meaningful account at the designated RP. □

On receiving a token, in Step 4.2 an RP does *not* check whether $PID_{RP}$ in the received token is equal to $[t]ID_{RP}$ or not. Even if an honest RP accepts a token which is generated for some malicious RP in another login and binds $PID_{U'} = [u't'r']G$ and $PID_{RP'} = [t'r']G$, a colluding user cannot find $t$ satisfying $[t^{-1}]PID_{U'} = Acct$ for any given victim with $Acct = [ur]G$ due to the ECDLP (see Theorem 1). Adversaries cannot entice the target RP to derive another account by a manipulated $t$, either (Theorem 2). That is, an *unmatching* token or $t$ results in a *meaningless* account corresponding to an unregistered (or non-existing) user, and the RP imperceptively treats it as a newly-registered user. Anyway, an RP may check $PID_{RP}$ in received tokens to eliminate the potential waste of storage.

## 5.3 Privacy against IdP-based Login Tracing

The IdP would attempt to infer the RP's identities visited by a user. Since the IdP is considered *honest*, it only collects information from the identity-token requests, but does *not* deviate from the protocol or collude with other entities.

The honest IdP 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 $PID_{RP}$. Next, 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 the same user.

Theorem 3 (IdP Untraceability). In UPPRESSO, 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.** $\mathbb{E}$ is a finite cyclic group, and the number of points on $\mathbb{E}$ is $n$. Because $G$ is a generator of order $n$, $ID_{RP} = [r]G$ is also a generator on $\mathbb{E}$ of order $n$. As $t$ is uniformly-random in $\mathbb{Z}_n$ and unknown to the IdP, $PID_{RP} = [t]ID_{RP}$ is *indistinguishable* from a point that is uniformly-random on $\mathbb{E}$. □

This property of obliviousness has been proved in the HashDH OPRF [34, 44], 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).

## 5.4 Privacy against RP-based Identity Linkage

Malicious RPs, which could collude with some malicious users, aim 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 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, we prove in Theorem 4 that, even if malicious RPs collude with each other and some malicious users by sharing $PID_U$s 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 $L = (ID_{RP}, t, Acct) = ([r]G, t, [ur]G)$.

When $c$ malicious RPs collude with each other, they create a shared view of all their logins, denoted as $\mathbb{L}$. When they collude further with $v$ malicious users, the logins initiated by these malicious 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 $c$ malicious RPs.

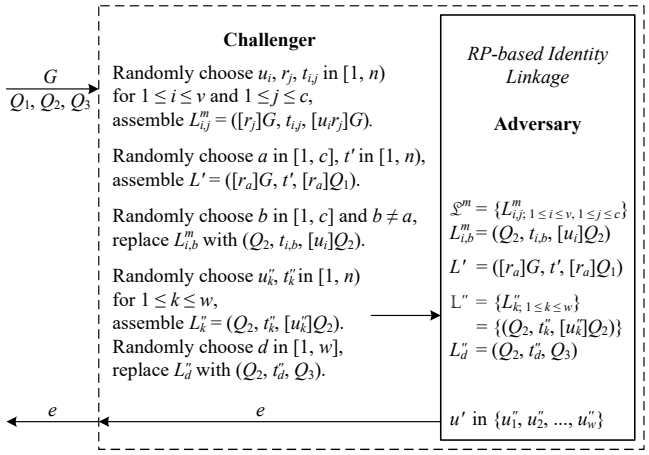



**Figure 5: The PPT algorithm $\mathcal{D}^*_R$ constructed based on the RP-based identity linkage game to solve the ECDDH problem.**

Theorem 4 (RP Unlinkability). In UPPRESSO, given $\mathbb{L}$ and $\mathfrak{L}^m$, $c$ malicious RPs and $v$ malicious users cannot collude to link any login visiting some malicious RP by an honest user to any subset of logins visiting any other malicious 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 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 $\mathbf{u}_w = \{u''_1, u''_2, \cdots, u''_w\} \subset \mathbf{u}$.

Next, we prove that the colluding adversaries cannot distinguish $u' \in \mathbf{u}_w$ from $u'$ randomly selected in the universal set $\mathbb{Z}_n$. 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 $\mathbf{u}_w$ 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 **Adv**:

$$\Pr_1 = \Pr(\mathcal{G}_R(\mathfrak{L}^m, L', \mathbb{L}'') = 1 \mid u' \in \mathbf{u}_w)$$
$$\Pr_2 = \Pr(\mathcal{G}_R(\mathfrak{L}^m, L', \mathbb{L}'') = 1 \mid u' \in \mathbb{Z}_n)$$
$$\mathbf{Adv} = |\Pr_1 - \Pr_2|$$

As depicted in Figure 5, 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 $L' = ([r_a]G, t', [r_a]Q_1) = ([r_a]G, t', [xr_a]G)$. (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 $1 \leq i \leq v$, 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., $\mathbf{u}_w = \{u''_1, u''_2, \cdots, u''_{d-1}, z/y, u''_{d+1}, \cdots, u''_w\}$. (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

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

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 $\lambda$. 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 $\mathbf{u}_w$ or not. Because $RP_b$ is any malicious RP, this proof can be easily extended from $RP_b$ to more colluding malicious RPs. □

## 5.5 Comparison with OPRF-based Applications

PrivacyPass and TrustToken [10, 55] adopted the oblivious pseudo-random function (OPRF) protocol of HashDH [34, 44] to generate tokens, with which a user anonymously accesses resources. 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 $[ke]T$ to the user, who uses $e$ to convert it into $(T, [k]T)$, which is redeemed when the user accesses some protected resource (see [10, 44, 55] for details). In this procedure, the token server obliviously calculates a pseudo-random output as anonymous token, because $(T, [k]T)$ and $([e]T, [ke]T)$ cannot be linked.

UPPRESSO employs a similar cryptographic technique.[2] A user selects $t$ to transform $ID_{RP}$ to $PID_{RP} = [t]ID_{RP}$. 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 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 different from existing OPRF-based systems [2, 4, 10, 23, 24, 35–38, 55] that always use $k$ as a server's secret key. Finally, UPPRESSO leverages randomness of HashDH 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.

Although UPPRESSO mathematically employs the same technique in its identity transformations as the HashDH OPRF [34, 44], we need more properties of these algorithms to ensure security and privacy. 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 [10, 55] depend on only the properties of deterministicness and obliviousness. Moreover, UPPRESSO needs *more* properties for secure SSO services (i.e., RP designation and user identification; see Theorems 1 and 2 in Section 5.2), not discussed in OPRFs [7, 34, 44]. So an OPRF protocol is not always ready to work as the identity transformations in UPPRESSO.

# 6 Implementation and Evaluation

We implemented a prototype of UPPRESSO[3] and conducted experimental comparisons with two open-source SSO systems: (*a*) MITREid Connect [46], a PPID-enhanced OIDC system with redirect UX, preventing RP-based identity linkage, and (*b*) SPRESSO [22], which prevents only IdP-based login tracing. All these solutions work with COTS browsers as user agents.

## 6.1 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 [46], 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

[2] We designed and implemented UPPRESSO in 2020, and in 2023 someone reminded us that the identity transformations are similar to the HashDH OPRF.

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

$PID_{RP}$ negotiation are conducted based on jsrsasign [52], 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.

## 6.2 Performance Evaluation

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

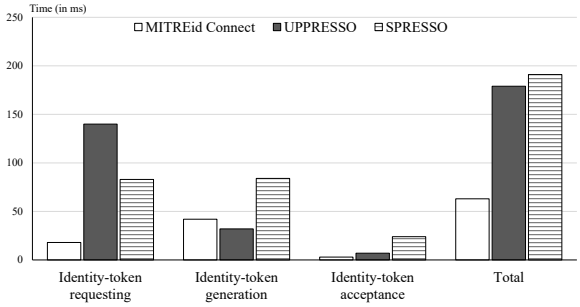
We conducted experiments in two settings: (*a*) a browser, Chrome 104.0.5112.81, ran on a virtual machine on the cloud platform with 8 vCPUs and 32 GB memory, and (*b*) the browser running locally on a PC with Core i7-8700 CPU and 32 GB memory, remotely accessed the servers. All entities except the local 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.

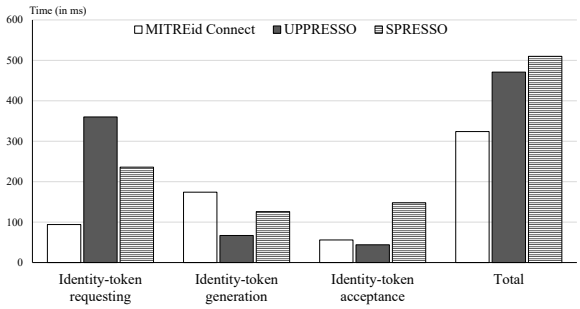
In the token requesting phase of UPPRESSO a browser downloads the two scripts, as described in Section 4.4. As mentioned in Section 2.1, to process the token retrieved from the IdP, in MITREid Connect and SPRESSO a user-r script is downloaded during the phase of token generation. SPRESSO needs another script from the forwarder server in the token acceptance phase [22].

We compared the average time required for an SSO login in three schemes based on 1,000 measurements. As shown in Figure 6, 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




(a) In a virtual private cloud




(b) With a remotely-visiting browser

**Figure 6: The time costs of SSO login in MITREid Connect, UPPRESSO, and SPRESSO**

the scripts.[4] 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 a 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.

## 7 Discussions

**IdP-RP collusive attacks and Malicious IdP.** We do not consider the collusion of the IdP and RPs. Even if the IdP strictly follows the protocols but shares information with RPs, a user would complete

[4]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.

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. Privacy-preserving identity federation [5, 12, 33, 45, 58] install a browser extension or plug-in to handle such a user secret. 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 [33, 58]. Meanwhile, in MISO [57] an *extra fully-trusted server* other than the IdP is introduced to keep such a secret, to calculate user (pseudo-)identities in each login. On the contrary, UPPRESSO is designed to provide privacy-preserving SSO services accessed from COTS browsers, so a trusted server is always needed (e.g., the IdP in UPPRESSO, the mixer server in MISO [57], and the forwarder server in SPRESSO [22]).

So UPPRESSO assumes an honest IdP, and the user-i script downloaded from IdP servers is also honest; otherwise, it might directly disclose the identities of visited RPs to the IdP. Besides, as its functions are predetermined, this script may be checked using subresource integrity (SRI) [56], by the RP window within a browser.

**Support for the authorization code flow.** In the authorization code flow of OIDC [47], 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 to transform (pseudo-)identities in signed tokens. Then, the user-i script will forward an authorization code (but 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 [47], to retrieve tokens.

Meanwhile, to hide RP identities from the IdP, privacy-preserving credentials (e.g., ring signatures [3] or privacy passes [10, 55]) and anonymous communications (e.g., oblivious proxies [50] or Tor [14]) need to be adopted for RPs in the retrieval of identity tokens. Otherwise, if the RP is not anonymously authenticated to the IdP, 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 [39, 47].

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 [18], 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 served by the IdP's services, either authorized or not.

In this case, the user-r script sends only the endpoint but not its RP certificate in Step 2.2, 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 tokens to the user-r script, and the `postMessage` targetOrigin mechanism [49] 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 tokens through different endpoints, e.g., `https://RP.com/honest/tk` and `https://RP.com/malicious/tk`. This mechanism cannot distinguish them. Because a COTS browser controls access to web resources following the same-origin policy (SOP) [53], a user's resources in the honest RP server is always accessible to the malicious one. For example, it could always steal cookies using `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 solutions on COTS browsers [22, 25, 46, 51].

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

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

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

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

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

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

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

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

[9] D. Chaum. 1982. Blind Signatures for Untraceable Payments. In *CRYPTO*. 199–203.

[10] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda. 2018. PrivacyPass: Bypassing Internet challenges anonymously. *Privacy Enhancing Technologies* 2018, 3 (2018), 164–180.

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

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

[13] B. Diamond and J. Posen. 2024. Succinct Arguments over Towers of Binary Fields. https://eprint.iacr.org/2023/1784.

[14] R. Dingledine, N. Mathewson, and P. Syverson. 2004. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*. 303–320.

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

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

[17] Hyperledger Fabric. [n. d.]. MSP implementation with Identity Mixer. https://hyperledger-fabric.readthedocs.io/en/release-2.2/idemix.html. Accessed July 20, 2022.

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

[19] Federated Identity Community Group. [n. d.]. Federated credential management API. https://fedidcg.github.io/FedCM/. Accessed January 22, 2023.

[20] D. Fett, R. Küsters, and G. Schmitz. 2014. 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)*.

[21] D. Fett, R. Küsters, and G. Schmitz. 2015. 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)*.

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

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

[24] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. 2020. 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)*.

[25] Google for Developers. [n. d.]. Google Identity: Integration considerations. https://developers.google.com/identity/gsi/web/guides/integrate/. Accessed January 13, 2025.

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

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

[28] J. Han, L. Chen, S. Schneider, H. Treharne, and S. Wesemeyer. 2018. Anonymous Single-Sign-On for $n$ Designated Services with Traceability. In *23rd European Symposium on Research in Computer Security (ESORICS)*.

[29] J. Han, L. Chen, S. Schneider, H. Treharne, S. Wesemeyer, and N. Wilson. 2020. Anonymous Single Sign-On With Proxy Re-Verification. *IEEE Transactions on Information Forensics and Security* 15 (2020), 223–236.

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

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

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

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

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

[35] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. 2016. 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)*. 276–291.

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

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

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

[39] M. Kroschewski and A. Lehmann. 2023. Save the implicit flow? Enabling privacy-preserving RP authentication in OpenID Connect. *Privacy Enhancing Technologies* 2023, 4 (2023), 96–116.

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

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

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

[43] E. Maler and D. Reed. 2008. The venn of identity: Options and issues in federated identity management. *IEEE Security & Privacy* 6, 2 (2008), 16–23.

[44] M. Naor and O. Reingold. 2004. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM* 51, 2 (2004), 231–262.

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

[46] J. Richer. [n. d.]. MITREid Connect v1.3.3. http://mitreid-connect.github.io/index.html. Accessed August 20, 2021.

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

[48] Firefox Application Services. [n. d.]. About Firefox Accounts. https://mozilla.github.io/application-services/docs/accounts/welcome.html. Accessed August 20, 2019.

[49] The WHATWG Community. [n. d.]. HTML living standard: 9.3 Cross-document messaging. https://html.spec.whatwg.org/multipage/web-messaging.html. Accessed June 7, 2022.

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

[51] Uber Developers. [n. d.]. OIDC Web SDK. https://developer.uber.com/docs/consumer-identity/oidc/web. Accessed April 10, 2025.

[52] K. Urushima. [n. d.]. jsrsasign (RSA-Sign JavaScript Library). https://kjur.github.io/jsrsasign/. Accessed August 20, 2019.

[53] W3C Web Security. [n. d.]. Same Origin Policy. https://www.w3.org/Security/wiki/Same_Origin_Policy. Accessed June 7, 2022.

[54] J. Wang, G. Wang, and W. Susilo. 2013. Anonymous Single Sign-On Schemes Transformed from Group Signatures. In *5th International Conference on Intelligent Networking and Collaborative Systems (INCoS)*.

[55] Web Incubator CG. [n. d.]. TrustToken API. https://github.com/WICG/trust-token-api. Accessed July 20, 2022.

[56] World Wide Web Consortium (W3C). [n. d.]. Subresource Integrity. http://www.w3.org/TR/2016/REC-SRI-20160623/. Accessed April 10, 2025.

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

[58] Z. Zhang, M. Król, A. Sonnino, L. Zhang, and E. Rivière. 2021. EL PASSO: Efficient and lightweight privacy-preserving single sign on. *Privacy Enhancing Technologies* 2021, 2 (2021), 70–87.

[59] Z. Zhang, C. Xu, C. Jiang, and K. Chen. 2024. TSAPP: Threshold single-sign-on authentication preserving privacy. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2024), 1515–1527.