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

Abstract

Single sign-on (SSO) allows a user to maintain only the credential at 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 visits to all 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 protocol, called UPPRESSO, to protect an honest user's online profile against (*a*) an honest-but-curious IdP and (*b*) malicious RPs and users who could collude. UPPRESSO proposes an identity-transformation approach to generate untraceable *ephemeral pseudo-identities* for an RP and a user (denoted as PID_{RP} and PID_U, respectively) from which the target RP derives a *permanent account* for the user (denoted as *Acct*), while the transformations provide unlinkability. This approach protects the unique identities of the user and the RPs that she requests to log into while working compatibly with SSO services and satisfying all the security requirements. We have 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) [52], OAuth 2.0 [36], and SAML [35, 37], 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, in an OIDC system, the user sends a login request to an RP, which constructs an identity-token request with its identity (denoted as ID_{RP}) and redirects it to a trusted IdP. After authenticating the user, the IdP issues an identity token binding the identities of both the user and the 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 authorized to log in. Thus, a user keeps only one credential for the IdP, instead of multiple credentials for different RPs.

SSO services provide a comprehensive solution for identity management and authentication, by enabling an IdP to enclose more user attributes in identity tokens, in addition to the authenticated user's identity. These attributes (e.g., age, hobby, and nationality) are maintained at the IdP and can be enclosed in identity tokens with user authorization [36, 52].

The wide adoption of SSO raises concerns about user privacy because it facilitates the tracking of a user's login activities by interested parties [23, 24, 31, 48]. To issue identity tokens, an IdP should know the RP to be accessed by a user and the login time. Thus, a curious IdP could potentially track a user's login activities over time [23, 24], 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 the same user identity is enclosed in tokens for the visits to different RPs, colluding RPs could link the logins across these RPs to learn the user's online profile [30, 48, 54]. This risk is called *RP-based identity linkage*.

Privacy-preserving SSO schemes aim to comprehensively support identity management and authentication while protecting user privacy [23, 24, 31, 48]. They typically offer the following features: (*a*) *unique user identification at each RP*, which is provided through SSO identity tokens, (*b*) *user authentication only to a trusted IdP*, which eliminates the need for authentication between a user and an RP and requires maintaining only the credential for the IdP, and (*c*) *provision of IdP-confirmed user attributes*, where a user's attributes are maintained at a trusted IdP and provided to RPs with the user's authorization. Meanwhile, the privacy threats posed by different adversaries are considered, including *an honest-but-curious IdP*, *colluding RPs*, and *the honest-but-curious IdP colluding with some RPs*. In Section 2.2, we analyze existing privacy-preserving solutions for SSO and also identity federation in light of these privacy threats.

This paper presents UPPRESSO, an Untraceable and Unlinkable Privacy-PREserving Single Sign-On protocol. It proposes *identity transformations* and integrates them into the SSO login flow compatibly. In UPPRESSO, an RP and a user first transform ID_{RP} into ephemeral PID_{RP}, which is then sent to a trusted IdP to transform ID_U into an ephemeral user pseudo-identity PID_U. The identity token issued by the IdP then binds only PID_U and PID_{RP}, instead of ID_U and ID_{RP}. When the RP receives the identity token, it transforms PID_U into an account that is 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 systems address only one of them [23, 24, 31, 44, 54] or an extra trusted server is needed [60]. Meanwhile, UPPRESSO works compatibly with widely-used SSO protocols [31, 36, 37, 52], providing all the desirable features of SSO services. In contrast, privacy-preserving identity federation [15, 19, 38, 46, 50, 61] supports some but not all of these features.

Our contributions are as follows.

- We proposed a novel identity-transformation approach for privacy-preserving SSO and also designed identity-transformation algorithms with desirable properties.
- We developed the UPPRESSO protocol based on 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 some extended issues in Section 7. Section 8 concludes this work.

2 Background and Related Work

We describe typical SSO services and discuss existing privacy-preserving solutions and other related works.

2.1 OpenID Connect and SSO 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 and receiving identity 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.

In OIDC, users and RPs register at an IdP with their identities and other information such as user credentials and RP endpoints. As shown in Figure 1, when an RP receives a login request, it constructs an identity-token request with its own identity and the scope of requested user attributes. This request is redirected to the IdP. Once the IdP authenticates the user, it issues an identity token that encloses the identities (or pseudo-identities) of the user and the visited RP, the requested user 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 enclosed (pseudo-)identity.

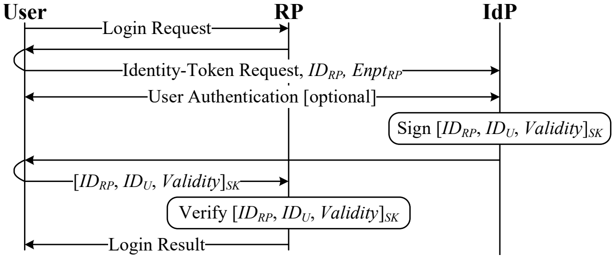

User
RP
IdP
Login Request
Identity-Token Request, ID_{RP}, $Enpt_{RP}$
User Authentication [optional]
Sign [ID_{RP}, ID_U, $Validity$]$_{SK}$
[ID_{RP}, ID_U, $Validity$]$_{SK}$
Verify [ID_{RP}, ID_U, $Validity$]$_{SK}$
Login Result

Figure 1: The implicit SSO login flow of OIDC

The following features are desired in SSO services and supported by popular SSO systems [31, 35–37, 52].

Unique user identification at an RP. An RP recognizes each user by a *unique* identity (or account) at the RP to provide customized services across multiple logins. Such a non-anonymous SSO system is much more desirable in various scenarios than anonymous services.

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 typically conducted *independently* of the steps that deal with identity tokens. This offers advantages. First, the IdP authenticates users by any appropriate means such as passwords, one-time passwords, or multi-factor authentication. Meanwhile, a user only maintains her credential at the IdP; and if it is lost or leaked, the user only needs to renew it at the IdP. However, if a user proves a *non-ephemeral* secret to RPs that is valid across multiple logins, she will have to notify each RP in the events of loss or leakage, or additional revocation checking will be needed [38, 61].

Selective IdP-confirmed attribute provision. An IdP usually includes user attributes in identity tokens [36, 52] along with user (pseudo-)identities. A user maintains these attributes at the trusted IdP, which obtains the user's authorization before enclosing attributes or provides only pre-selected attributes.

2.2 Privacy-Preserving SSO and Identity Federation

Privacy-preserving SSO is expected to offer the desired features listed in Section 2.1 while addressing different privacy threats. In contrast, privacy-preserving identity federation offers more privacy protections but introduces extra complexity in the user authentication process. Identity federation enables a user registered at a trusted IdP to be accepted by other par-

Table 1: Privacy-preserving solutions for SSO and identity federation

Solution	**SSO Feature** - supported ●, unsupported ○, or partially ◐			**Privacy Threat** - prevented ● or not ○		
	User Identification at each RP	User Authentication only to the IdP	IdP-confirmed Selective Attribute Provision	IdP-based Login Tracing	RP-based Identity Linkage	Server Collusion†
OIDC w/ PPID [31]	●	●	●	○	●	○
MISO [60]	●	●	●	●	●	○[1]
BrowserID [23]	●	●[2]	○	●	○	○
SPRESSO [24]	●	●	◐[3]	●	○	○
POIDC [32, 44]	●	●	●	●	○	○
PRIMA [3]	●	○	●	●	○	○
PseudoID [15]	●	○	◐[4]	●	●	●
Opaak [46]	◐[5]	○	○	●	●	●
[38, 50, 61]	●	○	●[6]	●	●	●
Fabric Idemix [19]	◐[7]	○	●	●	●	●
UPPRESSO	●	●	●	●	●	○

†. All servers involved in the processing of identity tokens collude, usually the IdP and target RPs.
1. MISO is immune to collusive attacks by the IdP and RPs, but an extra *fully-trusted* server called mixer is involved in the identity-token generation.
2. A BrowserID user generates an *ephemeral* private key to sign subsidiary "identity assertion" tokens, also verified by the RP.
3. SPRESSO can be extended to provide selective user attributes in the tokens, while the prototype does not implement this feature.
4. Blindly-signed user attributes can be selectively provided but not implemented in the prototype.
5. Opaak supports two exclusive pseudonym options: (*a*) linkable within an RP but unlinkable across multiple RPs and (*b*) unlinkable for any pair of actions.
6. Different from [38, 61], a credential in U-Prove [50] may contain some attributes that are *invisible* to the IdP, in addition to the ones confirmed by the IdP.
7. In the original design of Idemix [8], every user logs into an RP with a unique account, but Fabric Idemix [19] implements completely-anonymous services.

ties, potentially with different accounts, but *additional user operations for the authentication steps between the user and RPs* are involved.[1] As shown in Table 1, none of the existing solutions perfectly satisfies all expectations.

Privacy-preserving SSO. Existing approaches [23, 24, 31] prevent either IdP-based login tracing or RP-based identity linkage, but not both. Pairwise pseudonymous identifiers (PPIDs) are specified [35, 52] and recommended [31] for protecting user privacy against curious RPs. An IdP creates 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.

MISO [60] decouples the calculation of PPIDs from an IdP, and another mixer server 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 linkag, and also IdP-based login tracing because 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 it requires an extra fully-trusted mixer.

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 [23] the IdP issues a "user certificate" token that binds a user identity to an *ephemeral* public key. The user then signs a subsidiary "identity assertion" token that binds the target RP's identity and sends both tokens to the RP. The RP creates a one-time pseudo-identity for each login in SPRESSO [24], or a user sends an identity-token request with a cryptographical-hash commitment on the RP identity in POIDC [32, 44], which are enclosed in identity tokens along with the user's unique identity. Besides, an improved version of POIDC [32] 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 server [18], which is impracticable for a user agent.

Privacy-preserving identity federation. In PRIMA [3], the IdP signs a credential that binds user attributes and a verification key. Using the signing key, the user selectively provides attributes to the RPs. This verification key works as the user's identity but exposes her to RP-based identity linkage.

PseudoID [15] introduces a service in addition to the IdP, to blindly sign [11] an access token that binds a pseudonym and a user secret. The user then unblinds this token and uses the secret to log into an RP. Approaches based on anonymous credentials [8–10] have been proposed to implement privacy-preserving identity federation [19, 38, 46, 50, 61]. For instance, the IdP signs anonymous credentials in Opaak [46], UnlimitID [38], EL PASSO [61], and U-Prove [50], and binds them with non-ephemeral user secrets. Then the users prove ownership of the anonymous credentials using the secrets and disclose IdP-confirmed attributes in the credentials in most schemes except Opaak. Similarly, Hyperledger Fabric [19] integrates Idemix anonymous credentials [8] for completely-unlinkable pseudonyms and IdP-confirmed attribute disclosure.

[1] SSO protocols [35–37, 52] allow a user to log into an RP *without* maintaining an account at the RP by herself or holding a permanent secret to be verified by the RP. Although the same term "single sign-on (SSO)" was used in other schemes [15, 33, 34, 46, 58, 61], they are different from the widely-used SSO protocols because a user needs to maintain the accounts at different RPs and/or hold a permanent secret verified by the RPs. In this paper, we refer to them as *identity federation* to emphasize this difference.

These approaches [15, 19, 38, 46, 50, 61] prevent both IdP-based 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 when visiting different RPs. They even protect user privacy against collusive attacks by the IdP and RPs, as *user-managed* pseudonyms cannot be linked through anonymous credentials [8–10] even when the ownership of these credentials is proved to RPs using one user secret. However, this privacy protection results in additional user operations in identity federation, compared with widely-used SSO. Users are required to maintain not only the authentication credentials for the IdP but also the long-term secrets that are verified by RPs. For example, EL PASSO [61] requires users to keep the secrets securely on their devices and coordinate the credential revocation process [38, 61]. Besides, the users locally manage their accounts at different RPs, and it actually involves authentication steps between the user and RPs, which is referred to as *asynchronous authentication* [61].

Anonymous identity federation. Such approaches offer strong privacy protections. They allow users to access RPs with pseudonyms that cannot be used to link any two actions. Anonymous identity federation was formalized [58] and implemented using cryptographic primitives such as group signature and ZKP [33, 34, 58]. Special features including proxy re-verification [34], designated verification [33] and distributed IdP servers [62], are considered. These completely-anonymous authentication services only work for special scenarios and do not support user identification at an RP, a common requirement in most applications.

2.3 Anonymous Tokens and OPRF-based Applications

PrivacyPass and TrustToken [13, 59] adopted the oblivious pseudo-random function (OPRF) protocol [39, 49] to generate anonymous tokens with which authorized users could access resources anonymously. A user generates a random number e_i to blind an unsigned token T_i into $[e_i]T_i$. After authenticating the user, a token server signs $[e_i]T_i$ with its private key k and returns $[ke_i]T_i$ to the user, who converts it into an anonymous token $(T_i, [k]T_i)$ using e_i (see [13, 49, 59] for details). In this process, the server obliviously calculates a pseudo-random output (i.e., anonymous token). RPs do not identify each user if we use these tokens in an SSO service, because these tokens are completely indistinguishable.

UPPRESSO applies a similar cryptographic technique in identity transformations (details in Section 4). A user selects a random number t_i to transform ID_{RP} to $PID_{RP} = [t_i]ID_{RP}$, protecting ID_{RP} from the IdP. Then, the IdP uses the user's permanent identity u to calculate $PID_U = [u]PID_{RP} = [ut_i]ID_{RP}$, which is similar to token signing by the token server in PrivacyPass/TrustToken. Hence, UPPRESSO provides the *IdP-untraceability* property, i.e., the IdP cannot link ID_{RP} and $[t_i]ID_{RP}$, which roughly corresponds to the *unlinkability of token signing-redemption* in PrivacyPass/TrustToken, i.e., the token server cannot link T_i and $[e_i]T_i$ (or $[k]T_i$ and $[ke_i]T_i$).

To provide privacy-preserving SSO services, UPPRESSO works among three parties, unlike the two-party OPRF and PrivacyPass/TrustToken protocols. So it leverages this cryptographic technique in different ways. First, UPPRESSO securely shares the user-selected random number t_i with the target RP, enabling it to independently derive the user's unique account, i.e., $Acct = [t_i^{-1}]PID_U$. This account is "obliviously" determined by the IdP when it calculates the user's pseudo-identity based on the "blinded" input (i.e., the RP's pseudo-identity). Second, u and ID_{RP}, which roughly correspond to k and T_i in PrivacyPass/TrustToken, are designed as the permanent identities of the user and the RP, respectively, to establish the relationships between identities obliviously. This is very different from existing OPRF-based systems [4, 7, 13, 28, 29, 40–43, 59] that always use k as the server's secret key. The integration of SSO (pseudo-)identities and OPRF variables requires a deep understanding of both SSO and OPRF protocols. Finally, UPPRESSO leverages randomness of OPRF to provide *RP unlinkability*, and this property is not utilized in PrivacyPass/TrustToken. It ensures colluding RPs cannot link any two logins across RPs, even by sharing their knowledge about pseudo-identities and permanent accounts. This property offers desirable indistinguishability of *different users* logging into colluding RPs. It corresponds to the indistinguishability of *different keys* for signing anonymous tokens, not considered in PrivacyPass/TrustToken.

Although UPPRESSO mathematical employs the same technique in its identity transformations as the OPRF protocols [39, 49], we need more properties of these algorithms to ensure security and privacy. UPPRESSO essentially depends on the *obliviousness* property to ensure IdP untraceability, the *deterministicness* property of *pseudo-random* functions to derive a permanent account for any t, and the *randomness* property to provide RP unlinkability. In contrast, PrivacyPass and TrustToken [13, 59] actually depend on only the properties of obliviousness and deterministicness. Besides, UPPRESSO needs an extra property for secure SSO in the case of leaked ID_U, *collision-freeness* of PID_{RP} (see Appendix A.2). So an OPRF protocol is not always ready for identity transformations in UPPRESSO, unless no collision exists in the blinded inputs of the evaluated pseudo-random function [39, 49]. This property is not explicitly required in OPRF-based solutions.

2.4 Extended Related Work

Cryptographic primitives for sign-on privacy. Distributed key generation and ZKPs are used to generate ring-signature private keys as untraceable pseudonyms [47] or prove users' statements about their credentials without revealing the credential contents [53]. One-time key-share tokens [45] are generated in two-party threshold cryptography systems to protect users' key-usage patterns. PESTO [29] combines distributed partially-oblivious pseudo-random functions (dpOPRFs) and distributed signatures in proactively-secure SSO services, and signs a commitment of an identity token (but not the token itself) for better user privacy. Ab-dSSO [27] enforces attribute-based policies, using an MPC scheme between the distributed IdP servers and attribute authorities.

Formal analysis of SSO protocols. A formal analysis on SAML-based SSO [2] found some system does not bind RP identities correctly in the identity tokens. Fett et al. [25, 26] formally analyzed OIDC and OAuth 2.0 using a Dolev-Yao style model [22] and reported the 307 redirection and IdP mix-up attacks.

3 The Identity-Transformation Approach

We discuss the security requirements of privacy-preserving SSO and present the identity-transformation approach.

3.1 Security Requirements for SSO Services

Non-anonymous SSO services [31, 35–37, 52] are designed to allow a *legitimate* 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 trusted IdP issues an identity token that specifies the RP being accessed (i.e., *RP designation*) and the authenticated user (i.e., *user identification*) with identities or pseudo-identities. An honest RP checks the RP's identity designated in the token before accepting it and authorizes the token holder to log in as the specified user. This prevents malicious RPs from replaying any received tokens to gain unauthorized access to other honest RPs as victim users. *Confidentiality* and *integrity* of identity tokens are also necessary to prevent eavesdropping and tampering. Identity tokens are forwarded to the target RPs by the authenticated user, so they are usually signed by the trusted IdP and transmitted over HTTPS [36, 37, 52].

3.2 Identity Transformation

UPPRESSO implements privacy-preserving SSO that ensures security properties while preventing both IdP-based login tracing and RP-based identity linkage. These requirements are satisfied through *transformed identities* in the identity tokens. Table 2 lists the notations, and the subscript j and/or the superscript i may be ignored if it does not cause ambiguity.

Table 2: The (pseudo-)identities in UPPRESSO

Notation	**Description**	**Lifecycle**
ID_U	The user's unique identity at the IdP.	Permanent
ID_{RP_j}	The j-th RP's unique identity at the IdP.	Permanent
$PID^i_{U,j}$	The user's pseudo-identity in her i-th login to the j-th RP.	Ephemeral
$PID^i_{RP_j}$	The j-th RP's pseudo-identity in the user's i-th login to this RP.	Ephemeral
$Acct_j$	The user's identity (or account) at the j-th RP.	Permanent

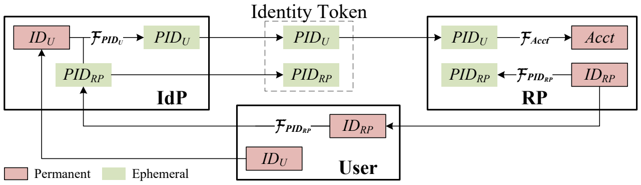


Figure 2: Identity transformations in UPPRESSO

In an SSO login flow, a user initiates the process by negotiating an *ephemeral* pseudo-identity PID_{RP} with the target RP and sending an identity-token request that encloses PID_{RP} to the IdP. After successfully authenticating the user as ID_U, the IdP calculates an *ephemeral* PID_U based on ID_U and PID_{RP} and issues an identity token that binds PID_U and PID_{RP}. Upon receiving a valid token, the RP calculates the user's *permanent* $Acct$ and authorizes the token holder to log in. The relationships among the (pseudo-)identities are depicted in Figure 2, where the labeled arrows denote the transformations of (pseudo-)identities, and the *red* and *green* blocks represent *permanent* and *ephemeral* ones, respectively.

For RP designation, PID_{RP} should be associated *uniquely* with the target RP. For user identification, with *ephemeral* PID^i_U in each login, the designated RP should be able to derive a unique *permanent* account (i.e., $Acct$) of the user at this RP. 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^i_{RP}. Thus, in a user's multiple logins to one RP, she should generate independent PID^i_{RP}s[2] and independent PID^i_Us.[3] 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 different RPs should also be independent of each other.

We propose three identity transformations as below:

- $\mathcal{F}_{PID_{RP}}(ID_{RP}) = PID_{RP}$, calculated by the user and the RP. 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. Given ID_U and ID_{RP}, $Acct$ is *unique* to other accounts at this RP. That is, in a user's two different logins to the RP, $\mathcal{F}_{Acct}(PID^i_U, PID^i_{RP}) = \mathcal{F}_{Acct}(PID^{i'}_U, PID^{i'}_{RP})$.

4 The Designs of UPPRESSO

This section presents the threat model and assumptions, the identity-transformation algorithms, and finally the UPPRESSO protocol with designs specific for web applications.

[2] The IdP should not be able to link multiple logins visiting a given RP, while the RP's identity is unknown to the IdP.

[3] If PID^i_U is not completely independent of each other, it implies that there is a possibility for the IdP to link multiple logins visiting a given RP.

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 [35–37, 52].

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, so a script downloaded from the IdP also strictly follows the protocols (see Section 4.4 for specific designs for web applications). The IdP maintains the private key well for signing identity tokens and RP certificates, which prevents adversaries from forging such messages.

Malicious users. Adversaries could control a set of users by stealing their credentials or registering Sybil users in the system. Their objective [22, 24] 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*) manipulate the generation of PID_U to 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, a malicious RP could collude with malicious users to impersonate a victim at honest RPs or link an honest user's logins visiting colluding RPs.

We do *not* consider the collusion of the IdP and RPs. In this case, a user would complete login flows *entirely* with malicious entities. In principle, it would require a long-term user secret to protect (or transform) permanent accounts across these RPs. This secret should be held *only* by the user; otherwise, colluding adversaries could always link these accounts. However, the accounts derived from this user secret would need to be updated if the secret was lost or leaked (see discussions about IdP-RP collusive attacks in Section 7).

Finally, the security and privacy properties of UPPRESSO are guaranteed against different combinations of the above adversaries. It implements secure SSO services against colluding users and RPs, while preventing (*a*) IdP-based login tracing by an honest-but-curious IdP and (*b*) RP-based identity linkage by colluding RPs and users.

4.2 Assumptions

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

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	$ID_U = u \in [1, n)$ is a 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; $r \in [1, n)$ is known to *nobody*.
t	$t \in [1, n)$ is a user-selected random integer in each login; t is shared with the target RP and kept unknown to the IdP.
$PID^i_{RP_j}$	$PID_{RP} = [t]ID_{RP} = [tr]G$ is the j-th RP's pseudo-identity, in the user's i-th login to this RP.
$PID^i_{U,j}$	$PID_U = [ID_U]PID_{RP} = [utr]G$ is the user's pseudo-identity, in the user's i-th login to the j-th RP.
$Acct_j$	$Acct = [t^{-1} \bmod n]PID_U = [ID_U]ID_{RP} = [ur]G$ is the user's account at the j-th RP.
SK, PK	The IdP's private key and public key, which are 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 $Enpt_{RP_j}$.

The software stack of an honest entity is correctly implemented to transmit messages to receivers as expected.

UPPRESSO is designed for users who value privacy. We assume a user never authorizes the IdP to enclose any *distinctive* attributes, such as telephone number, Email address, etc., in identity tokens or sets such attributes at any RP. Hence, privacy leakages due to re-identification by distinctive attributes across RPs are out of our scope. In addition, we do not consider the tracking of a user's activities by network traffic analysis or crafted web pages, as they can be prevented by existing defenses. For example, FedCM [21] attempts to disable iframe and third-party cookies in SSO, which could be exploited to track users. Our work focuses only on the privacy threats introduced by the flows of SSO protocols.

4.3 Identity-Transformation Algorithms

We design identity-transformation algorithms on an elliptic curve $\mathbb{E}$. Table 3 lists the notations.

The IdP assigns a unique random integer u 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.

***ID_{RP}-PID_{RP}* Transformation.** In each login, the user selects a random number $t \in [1, n)$ as the trapdoor 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 trapdoor t is sent to the target RP. After verifying a token that binds PID_U and PID_{RP}, it calculates *Acct* as follows.

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

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

$$Acct = [t^{-1}utr \bmod n]G = [ur]G = [ID_U]ID_{RP} \qquad (4)$$

With the help of t, the RP derives a *unique* account from different identity tokens for the same user in her different logins. It is the user's *permanent* account at this RP. Meanwhile, the user's accounts at different RPs are inherently different and unlinkable (see Section 5 for detailed proofs).

A user's identity u is kept secret from adversaries. Otherwise, colluding RPs could calculate $[u]ID_{RP_j}$s for any known u and link them. Meanwhile, $ID_{RP} = [r]G$ is publicly-known but r is unknown to RPs. 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 $[r']Acct_j = [r]Acct_{j'}$ holds.

The RP does not need to check if 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 solve and send t satisfying $[t^{-1}]PID_{U'} = Acct$ for a given victim with $Acct = [ur]G$, due to the elliptic curve discrete logarithm problem (ECDLP) when u is kept secret, to manipulate the calculated account (see Theorem 1). Thus, an *unmatched* token results in a meaningless account which corresponds to an unregistered (or non-existing) user, and the RP will treat it as a newly-registered user. We also discuss the impact of leaked user identities in Section 7.

4.4 The Design Specifics for Web Applications

In commonly-used SSO protocols [35–37, 52], an IdP needs to know the visited RP to ensure confidentiality of identity tokens. For instance, in OIDC services for web applications, 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 identity tokens to the RP, by setting this parameter as the target URL in the HTTP responses to a user's identity-token request [52], so the user agent (i.e., browser) forwards it to the designated RP. However, in UPPRESSO, the IdP does not know about the visited RPs, requiring a user agent by itself to calculate PID_{RP} and send identity tokens to the RP's endpoint.

UPPRESSO supports commercial-off-the-shelf (COTS) browsers to work as the user agent and implements the user-agent functions using web scripts in browsers. Two scripts are responsible for communications with the origin web servers, and downloaded from the IdP and the visited RP, respectively. It is worth noting that a script downloaded from an honest entity is considered *honest*.

The IdP script is necessary because the RP script could leak its origin to the IdP web server due to the automatic inclusion of an HTTP `referer` header in all HTTP requests it sends. Besides, the IdP script is trusted to interact with the users for attribute authorization, while the RP (and its script) could be malicious. Thus, on receiving an identity-token request, the IdP web server checks the `referer` header to ensure it is sent by the IdP script.

The RP script is responsible for preparing ID_{RP} and $Enpt_{RP}$ for the IdP script, through an RP certificate issued by the IdP. In each login, the RP script sends the certificate to the IdP script, which then verifies it and extracts ID_{RP} and $Enpt_{RP}$. Like in widely-adopted SSO systems [35–37, 52], in UPPRESSO a user does not need to configure anything locally for the IdP's public key is already set in the IdP script.

After receiving an identity token from the IdP, the IdP script needs to ensure the RP script will forward the token to $Enpt_{RP}$ specified in the RP certificate. As the communication between the scripts occurs within COTS browsers using the `postMessage` HTML5 API, we use the `postMessage` targetOrigin mechanism [55] to restrict the recipient. When the IdP script sends messages, the recipient's origin is set as a parameter, such as `postMessage(tk, 'https://RP.com')`, which includes the protocol (i.e., `https`), the domain (i.e., `RP.com`), and a port if applicable. Only the script downloaded from this targetOrigin is considered as a legitimate recipient.

As discussed in Section 4.3, a user calculates $PID_{RP} = [t]ID_{RP}$ based on ID_{RP} extracted from the RP certificate. Thus, this function should be implemented by an *honest* script that does not leak t to the IdP, from which it could calculate $ID_{RP} = [t^{-1} \bmod n]PID_{RP}$. In UPPRESSO, we consider the IdP script honest; otherwise, it could straightforwardly leak the RP's domain to the IdP. To avoid this assumption, we can implement the user agent with trusted browser extensions, which need to be installed by users before visiting RPs.

When a user is visiting an RP, the browser downloads the RP script, which in turn opens a new window to download the IdP script. To prevent referer leakage during the download of the IdP script, we need to ensure the HTTP request does not automatically carry a `referer` header, which reveals the visited RP's domain to the IdP. In UPPRESSO, this new window is a *redirection* from the RP to the IdP (Steps 1.2-1.3 in Figure 3), but not a direct visit by the browser. The HTTP response from the RP includes a `referrer-policy=no-referrer` header, which ensures that the HTTP request to download the IdP script carries no `referer` header. This approach is specified by W3C [17] and widely supported. We have tested it in various browsers such as Chrome, Safari, Edge, Opera, and Firefox, and confirmed that no referer leakage occurs.

4.5 The UPPRESSO protocol

System Initialization. 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:

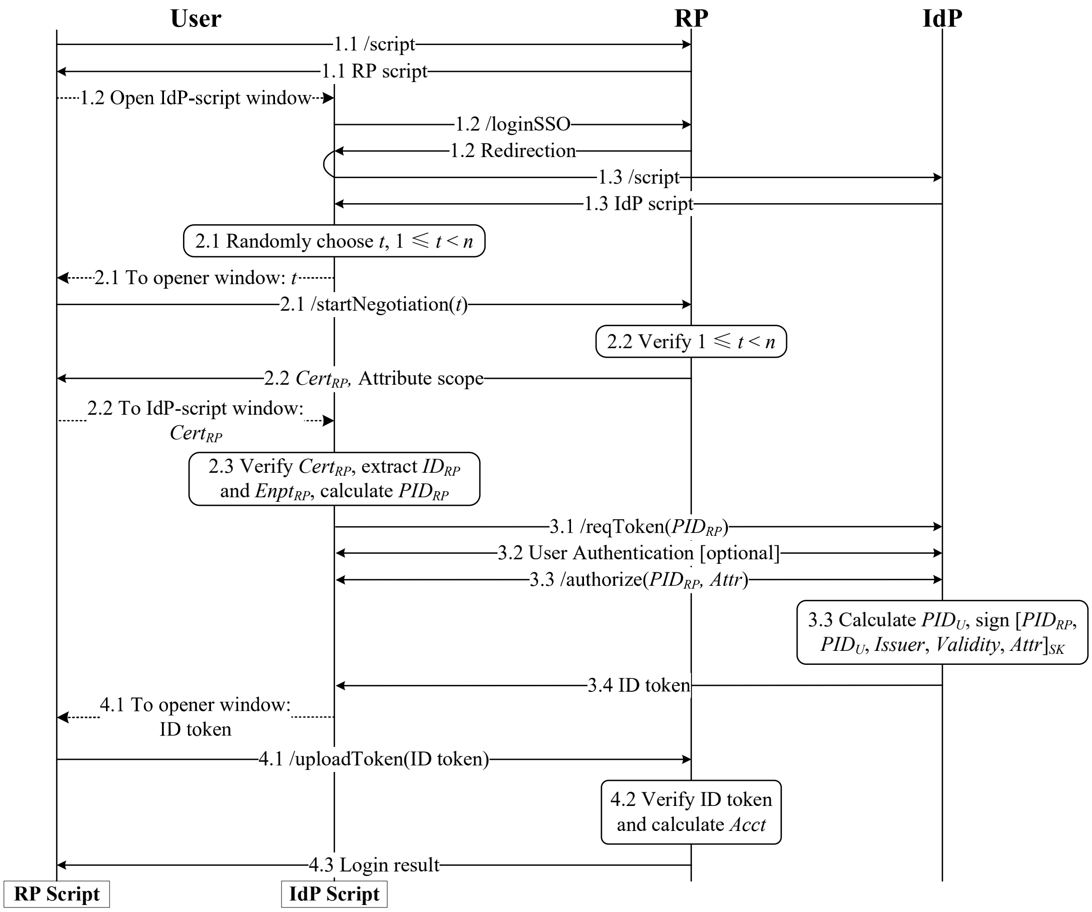

User
RP
IdP
1.1 /script
1.1 RP script
1.2 Open IdP-script window
1.2 /loginSSO
1.2 Redirection
1.3 /script
1.3 IdP script
2.1 Randomly choose t, $1 \leqslant t < n$
2.1 To opener window: t
2.1 /startNegotiation(t)
2.2 Verify $1 \leqslant t < n$
2.2 $Cert_{RP}$, Attribute scope
2.2 To IdP-script window: $Cert_{RP}$
2.3 Verify $Cert_{RP}$, extract ID_{RP} and $Enpt_{RP}$, calculate PID_{RP}
3.1 /reqToken(PID_{RP})
3.2 User Authentication [optional]
3.3 /authorize(PID_{RP}, $Attr$)
3.3 Calculate PID_U, sign $[PID_{RP}, PID_U, Issuer, Validity, Attr]_{SK}$
3.4 ID token
4.1 To opener window: ID token
4.1 /uploadToken(ID token)
4.2 Verify ID token and calculate $Acct$
4.3 Login result
RP Script
IdP Script

Figure 3: The SSO login flow of UPPRESSO

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. The IdP randomly selects a *unique* point on $\mathbb{E}$, and assigns it to ID_{RP}. Thus, $ID_{RP} = [r]G$ but r is known to *nobody* due to the ECDLP. It 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 registers once at the IdP. She sets up her unique username and the corresponding credential. The IdP will then assign a unique random identity $ID_U = u \in [1, n)$ to the user. Note that ID_U is used only by the IdP *internally*, not enclosed in any token or message.

SSO Login. A login 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 two vertical lines, one communicating with the IdP and the other with the RP. Solid horizontal lines indicate messages exchanged between the user and the IdP (or the RP), while dotted lines represent a `postMessage` invocation between two scripts (or browser windows) within the browser.

1. *Script Downloading.* The browser downloads scripts from the visited RP and the IdP as below.

1.1 When requesting any protected resources at the RP, the user downloads the RP script.

1.2 The RP script opens a window in the browser to visit the login path at the RP, which is then redirected to the IdP.

1.3 The redirection to the IdP downloads the IdP script.

2. *RP Identity Transformation.* The user and the RP negotiate $PID_{RP} = [t]ID_{RP}$.

2.1 The IdP script chooses a random number $t \in [1, n)$ and sends it to the RP script through `postMessage`. The RP script then forwards t to the RP.

2.2 The RP verifies if the received t is an integer in $[1, n)$ and replies with $Cert_{RP}$ and the scope of the requested user attributes. The reply is then transmitted through the RP script to the IdP script.

2.3 The IdP 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 IdP 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 IdP script obtains the user's authorization for the requested attributes locally and then sends the scope of the authorized attributes. Then, the IdP checks if the received 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 IdP script.

4. *Acct Calculation.* The RP receives the identity token and authorizes the user to log in.

4.1 The IdP script forwards the identity token to the RP 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 authorizes 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

Both UPPRESSO and OIDC work with COTS browsers. In UPPRESSO, the *script downloading* step prepares the user agent, which assists the communications with the IdP and RP servers. UPPRESSO employs web scripts to hide the RP's endpoint from the IdP, while securely forwarding identity tokens to the RP through $Enpt_{RP}$ extracted from the signed RP certificate. Thus, the IdP does not set `redirect_uri` in the HTTP responses, which is different from OIDC where HTTP redirections are used to implement these communications. Most operations in the *RP identity transformation* step take place within browsers, except that the RP receives t and responds with $Cert_{RP}$, which can be viewed as a supplementary message to users. Consequently, compared to the original OIDC protocol, UPPRESSO simplifies the IdP's operations in these two steps, while allowing "dynamic" RP pseudo-identities.

The operations of *identity-token generation* and *Acct calculation* in UPPRESSO are *identical* to those in OIDC, because (a) the calculation of PID_U in UPPRESSO can be viewed as a method to generate PPIDs in OIDC and (b) the calculation of $Acct$ can be viewed as a mapping from the user identity in tokens to an account at the RP.

The compatibility is experimentally confirmed through our prototype implementation, which modifies only 23 lines of Java code in MITREid Connect [51], an open-source OIDC system, to build an IdP of UPPRESSO (see Section 6.1).

5 Security and Privacy Analysis

We prove the security and privacy guarantees provided by UPPRESSO, against different adversaries.

5.1 Adversarial Scenarios

Based on our design goals (i.e., the desired security and privacy guarantees) and the potential adversaries discussed in Section 4.1, we consider three adversarial scenarios as below.

Adversaries against security. Malicious users could collude with each other and even with malicious RPs [22–24], attempting to (a) impersonate an honest user to log into an honest RP or (b) entice an honest user to log into an honest RP under another user's account.

Adversaries against IdP untraceability. The honest-but-curious IdP tries to infer the identities of the RPs that an honest user requests to access.

Adversaries against RP unlinkability. Malicious RPs could collude with each other and even malicious users, attempting to link logins across these RPs initiated by honest users.

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 exactly as her account at this RP. When confidentiality and integrity of TK are ensured by secure communications and digital signatures in UPPRESSO, we summarize the following necessary and sufficient conditions of secure SSO authentication services [22–24]:

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 other honest RPs, no meaningful account will be derived.

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

Assume totally s users and p RPs in UPPRESSO, whose identities are denoted as $\mathbf{u} = \{u_i, 1 \le i \le s\}$ and $\mathbb{ID}_{RP} = \{[r_j]G, 1 \le j \le p\}$, respectively. Therefor, there are accounts $Acct_{i,j} = [u_i]ID_{RP_j} = [u_i r_j]G$ $(1 \le i \le s)$ automatically registered at each RP. 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 $PID_U = [u]PID_{RP}$, any honest RP with $ID_{RP'} \neq ID_{RP}$ cannot derive $Acct = [u']ID_{RP'}$ where $u' \in \mathbf{u}$ and $ID_{RP'} \in \mathbb{ID}_{RP}$.

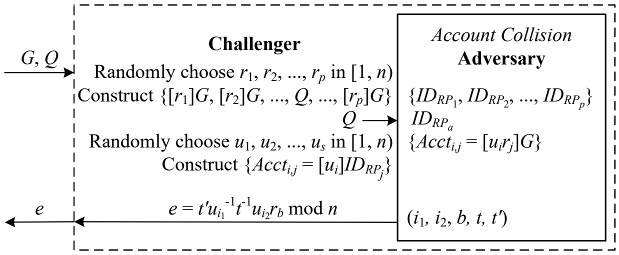

Challenger
Account Collision
Adversary
G, Q
Randomly choose $r_1, r_2, ..., r_p$ in $[1, n)$
Construct $\{[r_1]G, [r_2]G, ..., Q, ..., [r_p]G\}$
$\{ID_{RP_1}, ID_{RP_2}, ..., ID_{RP_p}\}$
Q
ID_{RP_a}
Randomly choose $u_1, u_2, ..., u_s$ in $[1, n)$
$\{Acct_{i,j} = [u_i r_j]G\}$
Construct $\{Acct_{i,j} = [u_i]ID_{RP_j}\}$
e
$e = t'u_{i_1}^{-1}t^{-1}u_{i_2}r_b \bmod n$
(i_1, i_2, b, t, t')

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

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

Next, we prove that, when $u_{i,1\leq i\leq s}$ and $r_{j,1\leq j\leq p}$ are unknown, for any $[r']G \neq [r]G$ where $[r']G \in \mathbb{ID}_{RP}$, 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 $[t'^{-1}u_{i_1}tr_a]G = [u_{i_2}r_b]G$, which occurs with a probability $\Pr_s$, the adversary succeeds.

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

The algorithm $\mathcal{D}_a^*$ works as below. The input of $\mathcal{D}_a^*$ is in the form of (G, Q). On receiving an input, the challenger firstly chooses $r_1, \cdots, r_p$ random in $\mathbb{Z}_n$ to calculate $\{[r_j]G, 1 \leq j \leq p\}$, randomly chooses $a \in [1, p]$ to replace $[r_a]G$ with Q, and constructs $\mathbb{ID}_{RP} = \{[r_1]G, \cdots, Q, [r_{a+1}], \cdots, [r_p]G\}$. The challenger also chooses $u_1, \cdots, u_s$ random in $\mathbb{Z}_n$ to construct $\{Acct_{i,j} = [u_i]ID_{RP_j}\}$. Then, it sends $\mathbb{ID}_{RP}$, Q and $\{Acct_{k,j}\}$ 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 returns e as the output of $\mathcal{D}_a^*$.

If $[t'^{-1}u_{i_1}t]Q = [u_{i_2}r_b]G$ and the adversary succeeds in $\mathcal{G}_a$, then $\mathcal{D}_a^*$ 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 then $\Pr_s$ is non-negligible regardless of the security parameter λ, we would find that $\Pr\{\mathcal{D}_a^*(G, [x]G) = x\}$ also becomes non-negligible even when λ 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 in UPPRESSO. □

Because TK works only at the "implicitly" designated RP and no meaningful user will be derived at any other RPs when u_i and r_j are kept unknown to adversaries, in Step 4.2 an RP does not need to check whether PID_{RP} in the received token equals to $[t]ID_{RP}$ or not.

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. Firstly, on receiving t, the RP designated by TK calculates $Acct = [t^{-1}]PID_U = [t^{-1}ut]ID_{RP} = [u]ID_{RP}$, exactly corresponding to the authenticated user.

Next, we consider TK replayed by malicious users to the target RP, with a manipulated trapdoor $t' \neq t$. The designated RP will derive an account as $[t'^{-1}]PID_U = [t'^{-1}ut]ID_{RP}$. 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 $u_{i,1\leq i\leq s}$ are unknown, the probability that $[t'^{-1}ut]ID_{RP}$ happens to be another meaningful account $[u_i]ID_{RP}$ is $\frac{s-1}{n}$, because u_i is randomly selected by the IdP in $\mathbb{Z}_n$.

This probability becomes negligible, when n is sufficiently large. As a result, $[t'^{-1}]PID_U$ $(t' \neq t)$ does not result in any meaningful account at the designated RP. □

5.3 Privacy against IdP-based Login Tracing

In UPPRESSO the *curious* IdP would attempts to infer the RP's identity visited by a user. Since the IdP is considered *honest*, it only collects information from the login requests, but does not deviate from the protocol or collude with others.

The IdP does not obtain any information related to the target RP in each login (e.g., ID_{RP}, $Enptr_{RP}$ or $Cert_{RP}$) except its *ephemeral* pseudo-identity PID_{RP}. Next, we prove in Theorem 3 that with these PID_{RP}s in the identity-token requests, the IdP cannot (a) associate multiple logins to the same RP, or (b) distinguish a login to one RP from other logins to another RP, because PID_{RP} is *indistinguishable* from any random variables to the IdP. So UPPRESSO prevents the honest-but-curious IdP from tracing a user's login activities.

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. Consider a finite cyclic group $\mathbb{E}$ where 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. When t is randomly chosen in $\mathbb{Z}_n$ and unknown to the IdP, $PID_{RP} = [t]ID_{RP}$ is *indistinguishable* from a point that is randomly chosen on $\mathbb{E}$ [39, 49].[4] □

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.

[4] This property actually has been proved in OPRFs [39, 49], 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.

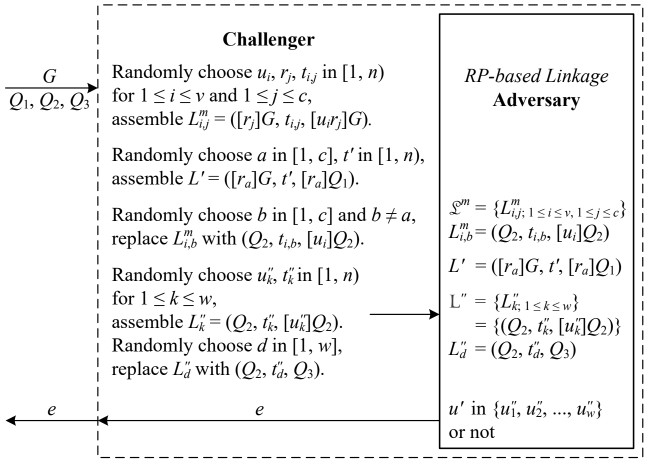

G
Q_1, Q_2, Q_3
Challenger
Randomly choose u_i, r_j, $t_{i,j}$ in $[1, n)$ for $1 \le i \le v$ and $1 \le j \le c$, assemble $L^m_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G)$.
Randomly choose a in $[1, c]$, t' in $[1, n)$, assemble $L' = ([r_a]G, t', [r_a]Q_1)$.
Randomly choose b in $[1, c]$ and $b \neq a$,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

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

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-identifier $PID_U = [ID_U]PID_{RP}$ of the user, from which it derives a permanent pseudo-identifier $Acct = [ID_U]ID_{RP}$. However, it cannot 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_Us and other information observed in all the logins, they still cannot link any login from an honest user to any other logins from any other honest users to malicious RPs.

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

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 and linked together as $\mathscr{L}^m = \left\{\begin{matrix} L^m_{1,1}, & L^m_{1,2}, & \cdots, & L^m_{1,c} \\ L^m_{2,1}, & L^m_{2,2}, & \cdots, & L^m_{2,c} \\ \cdots, & \cdots, & L^m_{i,j}, & \cdots \\ L^m_{v,1}, & L^m_{v,2}, & \cdots, & L^m_{v,c} \end{matrix}\right\}$, where $L^m_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G)$ for $1 \le i \le v$ and $1 \le j \le c$, and $L^m_{i,j} \in \mathbb{L}$. Any login in $\mathbb{L}$ but not linked in $\mathscr{L}^m$ is initiated by an honest user to one of the c malicious RPs.

THEOREM 4 (RP Unlinkability). In UPPRESSO, given $\mathbb{L}$ and $\mathscr{L}^m$, c malicious RPs and v malicious users cannot link any login from an honest user to some malicious RP to any subset of logins from honest users to any other malicious RPs.

PROOF. From the logins in $\mathbb{L}$, we randomly choose one login $L' \neq L^m_{i,j}$, which is from an (unknown) honest user with $ID_{U'} = u'$ to a malicious RP_a and $a \in [1, c]$. 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 logins visiting RP_b by unknown honest users, we denote the identities of the honest users who initiate these logins as $\mathbf{u}_w = \{u''_1, u''_2, \cdots, u''_w\}$. Next, we prove that the colluding adversaries cannot decide if u' is in $\mathbf{u}_w$ or randomly selected from the universal user set. This indicates the colluding adversaries cannot link L' to another login visiting RP_b or even another subset of logins visiting RP_b.

We define an RP-based linkage game $\mathcal{G}_r$ between an adversary and a challenger, which describes this login linkage privacy threat: the adversary receives $\mathscr{L}^m$, L', and $\mathbb{L}''$ from the challenger and outputs e, where $e = 1$ if it decides u' is in $\mathbf{u}_w$ and $e = 0$ if it believes u' is randomly chosen from the universal user set. Thus, the adversary succeeds in $\mathcal{G}_r$ with an advantage **Adv**:

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

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 below. (1) Upon 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 $L^m_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G)$. In this process, it ensures $[r_j]G \neq Q_2$ so that $r_j \neq y$. (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 $b \in [1, c]$ and $b \neq a$, 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 then constructs $\mathscr{L}^m$. (4) The challenger chooses random numbers in $\mathbb{Z}_n$ to construct $\{u''_k\}$ and $\{t''_k\}$ for $1 \le k \le w$, with which it assembles $\mathbb{L}'' = \{L''_{k;1 \le k \le w}\} = \{(Q_2, t''_k, [u''_k]Q_2)\} = \{([y]G, t''_k, [u''_k y]G)\}$. In this process, it ensures that $[u''_k]G \neq Q_1$ (i.e., $u''_k \neq x$) and $u''_k \neq u_i$, for $1 \le i \le v$ and $1 \le k \le w$. 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, $\mathbb{L}'' = \{L''_{k;1 \le k \le w}\}$ 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 $\mathscr{L}^m$, 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 $ID_{U'}$ in the login L' visiting the RP with $ID_{RP_a} = [r_a]G$, and z/y is embedded as $ID_{U''_d}$ in $\mathbb{L}''$ 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 $\mathscr{L}^m$. Because $x \neq u''_{k;1 \le k \le w, k \neq d}$ 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 $\mathbf{Adv}^* = |\mathrm{Pr}_1^* - \mathrm{Pr}_2^*|$, where

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

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 user with an identity in $\mathbf{u}_w$ or by a user in the universal user set. Because RP_b is any malicious RP, this proof can be easily extended from RP_b to more colluding malicious RPs. □

6 Implementation and Evaluation

We implemented a prototype of UPPRESSO[5] and conducted experimental comparisons with two open-source SSO systems: (*a*) MITREid Connect [51], which is a PPID-enhanced OIDC system to prevent RP-based identity linkage, and (*b*) SPRESSO [24], which prevents IdP-based login tracing.

6.1 Prototype Implementation

The UPPRESSO prototype implemented identity transformations on the NIST P256 elliptic curve, with RSA-2048 and SHA-256 serving as the digital signature and hash algorithms, respectively. The IdP and RP scripts consist of approximately 160 and 140 lines of JavaScript code, respectively. The cryptographic computations such as $Cert_{RP}$ verification and PID_{RP} negotiation are performed using jsrsasign [56], an open-source JavaScript cryptographic library.

The IdP was developed on MITREid Connect [51], a Java implementation of OIDC, with minimal code modifications. Only three lines of code were added for calculating PID_U and 20 lines were added to modify the method of forwarding identity tokens. The calculations for ID_{RP} and PID_U were implemented using Java cryptographic libraries.

We developed a Java-based RP SDK with about 500 lines of code on the Spring Boot framework. It includes two functions for encapsulating the UPPRESSO protocol steps: one for requesting identity tokens and the other for deriving accounts. The cryptographic computations are executed using the Spring Security library. An RP can easily integrate UPPRESSO by adding less than 10 lines of Java code to invoke the necessary functions.

6.2 Performance Evaluation

Experiment setting. We compared UPPRESSO comprehensively with MITREid Connect and SPRESSO. MITREid Connect supports the implicit flow of OIDC, while SPRESSO follows a similar approach to forward the identity token from a user to the RP. SPRESSO encrypts the RP's domain in identity tokens and keeps the symmetric key known only to the RP and the user. Both systems employ RSA-2048 and SHA-256 for token generation. 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.

The IdP and RP servers were deployed on Alibaba Cloud Elastic Compute Service, each of which ran Windows 10 with 8 vCPUs and 32 GB RAM. The forwarder of SPRESSO ran Ubuntu 20.04.4 with 16 vCPUs and 16 GB RAM, also on the Alibaba Cloud.

We conducted experiments in two scenarios: (*a*) a browser, Chrome 104.0.5112.81, ran on a virtual machine on Alibaba Cloud with 8 vCPUs and 32 GB memory, and (*b*) a browser running locally on a PC with Core i7-8700 CPU and 32 GB memory, remotely accessed the servers. In the cloud scenario, all entities were deployed in the same virtual private cloud and connected to one vSwitch, which minimized the impact of network delays. In both scenarios, the IdP server never directly communicated with the RPs.

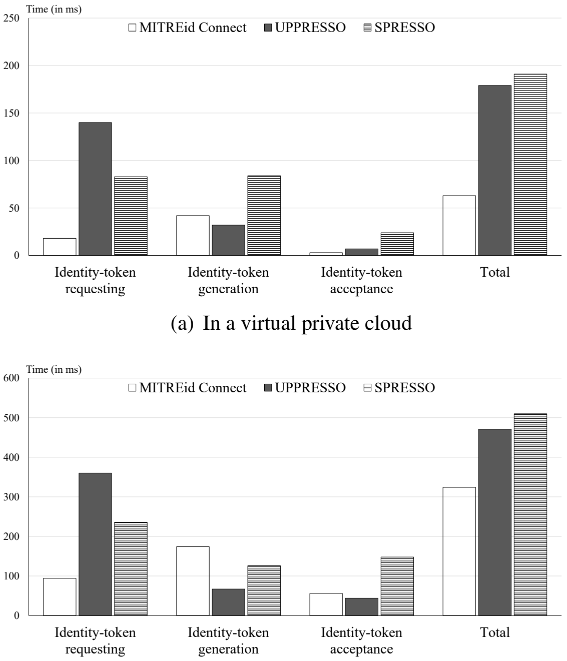
Comparisons. We split the login flow into three phases for detailed comparisons: (1) *identity-token requesting* (Steps 1 and 2 of the UPPRESSO protocol), to construct an identity-token request and send it to the IdP server; (2) *identity-token generation* (Step 3 of UPPRESSO), to generate an identity 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.

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 Alibaba Cloud, 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. In contrast, UPPRESSO incurs extra overheads in opening a new browser window and downloading scripts.[6] In SPRESSO the

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

[6] This overhead may be reduced by implementing a user agent with browser extensions. We implemented such a browser extension while keeping the IdP and RPs unmodified. Our experiments showed a reduction of about 90 ms in the virtual private cloud setting and 260 ms when accessed remotely.



(a) In a virtual private cloud

(b) With a remotely-visiting browser

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

RP needs to obtain information about the IdP and encrypt its domain using an ephemeral key, resulting in extra overheads.

Compared with the two schemes, UPPRESSO requires less time for identity token generation since it retrieves the token from the IdP without any additional processing. In contrast, MITREid Connect requires more time as the user is required to download a script from the RP to process the token retrieved from the IdP, which is carried with a URL following the fragment identifier # instead of ? due to security considerations [14]. 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 less efficient than the Java library used by the other two schemes.

In the identity-token acceptance phase, MITREid Connect and UPPRESSO take similar amounts of time to send an identity token to the RP and verify it. In contrast, SPRESSO takes the longest time due to its complex process at the user agent. After receiving the identity tokens from the IdP, the browser needs to download JavaScript code from the trusted forwarder server, decrypt the RP endpoint, and then send the identity tokens to this endpoint.

7 Discussions

IdP-RP collusive attacks. Privacy-preserving identity federation approaches [8, 15, 38, 46, 50, 61] can prevent collusive attacks by a malicious IdP and malicious RPs. However, they require (*a*) a long-term secret that is held by each user and verified by the RPs and (*b*) user-managed accounts for each user at different RPs. While these accounts can be derived from an RP's domain and the user's secret [8, 38, 46, 50, 61], they still cause inconvenience to users. For example, in web applications, the users need to install a browser extension to handle the long-term secret. If it is lost or leaked, the user must notify all the RPs to update her accounts derived from the secret. If the relationship of these accounts is not masked by the user secret or another trusted server [60], the colluding IdP and RPs can eventually find a way to link them; Unlike these approaches, UPPRESSO is not designed to prevent such collusive attacks, because a user is authenticated only *once* in the login flow. In UPPRESSO the user's identity at the IdP is obliviously transformed into accounts at RPs, which are not related to any user credentials such as passwords, one-time passwords, or FIDO devices.

Scalability. ID_{RP} is generated uniquely during the initial registration of an RP with a capacity of n, which is the order of G. For the NIST P256 elliptic curve, n is approximately 2^{256}. PID_{RP} is ensured to be unique in unexpired tokens. The probability of at least two identical PID_{RP}s among σ unexpired tokens is $1-\prod_{i=0}^{\sigma-1}(1-i/n)$. If the system serves 10^8 requests per second and each token has a validity period of 10 minutes, σ is less than 2^{36}. So the PID_{RP}-collision probability is negligible, i.e., less than 2^{-183} for the NIST P256 curve.

The capacity of accounts at any RP is the same as the capacity of user identities at the IdP. Since $\mathbb{E}$ is a finite cyclic group, $ID_{RP}=[r]G$ is also a generator of order n. Therefore, at any RP, a unique account is automatically assigned to each user because $Acct=[ID_U]ID_{RP}=[u]ID_{RP}$. In addition, stronger elliptic curves accommodate more RPs and users, e.g., n is about 2^{384} for the NIST P384 curve.

Risk of leaked user identities. UPPRESSO requires that ID_U is known only to the honest IdP, so a user inputs her username in the authentication to the IdP, and ID_U is processed only by the IdP internally. The IdP should random user identities in storage and decrypt them only in memory.

If user identities were leaked, RP unlinkability is broken; however, in this case, UPPRESSO still provides secure SSO services, providing that in Step 4.2 an RP accepts only identity tokens with *matching* PID_{RP} and then adversaries cannot deceive an honest RP other than the designated one to accept a token due to the *collision-freeness* property of PID_{RP}. We discuss this risk in Appendix A with more details.

Alternative methods for generating ID_{RP} and binding $Enpt_{RP}$. In UPPRESSO, the IdP generates random ID_{RP} and uses an RP certificate to bind ID_{RP} and $Enpt_{RP}$, which is verified by the IdP script. This ensures that the target RP has already registered itself at the IdP and prevents unauthorized RPs from accessing the IdP's services [44].

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 [20], 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} to ensure the *uniqueness* of $ID_{RP} = [r]G$ while keeping r unknown.

In this case, the RP script sends only the endpoint but not its RP certificate in Step 2.2, and the IdP script calculates ID_{RP} by itself. However, if the RP changes its domain, such as from `https://theRP.com` to `https://RP.com`, the account (i.e., $Acct = [ID_U]ID_{RP}$) will inevitably change. As a result, the 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 from an RP to the updated one; otherwise, it implies two colluding RPs could link a user's accounts across these RPs.

Restriction of the RP script's origin. When the IdP script forwards identity tokens to the RP script, the `postMessage` targetOrigin mechanism [55] is used to restrict the recipient of the 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 RP script's origin must accurately match the targetOrigin to receive the tokens.

The targetOrigin mechanism does not check the whole URL path in $Enpt_{RP}$, but it introduces *no additional* risk. Consider two RPs in the same domain but receiving tokens through two 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 with the same-origin policy (SOP) [57], a user's resources in the honest RP server is always accessible to the malicious one. For example, it could 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 but not our designs.

Support for the authorization code flow. In OIDC authorization code flow [52], the IdP does not directly return identity tokens. Instead, it sends an authorization code to the RP, which uses this code to request identity tokens. The identity-transformation algorithms, namely $\mathcal{F}_{PID_U}$, $\mathcal{F}_{PID_{RP}}$, and $\mathcal{F}_{Acct}$, can be integrated into this flow as below.

The IdP script can forward an authorization code to the RP script, and then to the RP. An authorization code only serves as an index to retrieve identity tokens from the IdP and does not reveal any more information about the authenticated user. After receiving an authorization code, an RP uses it along with a secret credential issued by the IdP during the initial registration [52] to retrieve identity tokens from the IdP. However, to protect the RP identities from the IdP, anonymous tokens (e.g., ring or group signatures [5, 12] and TrustToken [59]) and anonymous networks (e.g., Tor [16]) need to be adopted for RPs in the retrieval of identity tokens.

Quantum resistance. The current designs of UPPRESSO do not achieve quantum resistance. In the future, we will investigate alternative algorithms for quantum-secure services. We plan to adopt a quantum-resistant public-key algorithm to sign identity tokens and RP certificates, and study existing quantum-resistant ORPF protocols [1, 6] to investigate if they support RP designation as discussed in Section 5.2 (i.e., no collision exists in the blinded inputs of evaluated pseudo-random functions).

8 Conclusion

This paper presents UPPRESSO, an untraceable and unlinkable privacy-preserving SSO system for protecting a user's online profile across different RPs against both a curious IdP and colluding RPs. We propose an identity-transformation approach for privacy-preserving SSO and design algorithms that satisfy the requirements, where *(a)* $\mathcal{F}_{PID_{RP}}$ protects an RP's identity from the curious IdP, *(b)* $\mathcal{F}_{PID_U}$ prevents colluding RPs from linking a user across different RPs, and *(c)* $\mathcal{F}_{Acct}$ enables an RP to derive an identical account for a user in multiple logins. These 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 users are deployed in a virtual private cloud, or 421 ms when a user visits remotely.

References

[1] M. Albrecht, A. Davidson, A. Deo, and N. Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In *24th IACR International Conference on Practice and Theory of Public Key Cryptography (PKC)*, pages 261–289, 2021.

[2] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: Breaking the SAML-based single sign-on for Google Apps. In *6th ACM Workshop on Formal Methods in Security Engineering (FMSE)*, 2008.

[3] 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.

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

[5] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random

oracles. In *3rd Theory of Cryptography Conference (TCC)*, pages 60–79, 2006.

[6] D. Boneh, D. Kogan, and K. Woo. Oblivious pseudorandom functions from isogenies. In *AsiaCrypt*, 2020.

[7] 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)*, pages 337–356, 2019.

[8] 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.

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

[10] 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.

[11] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[12] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.

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

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

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

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

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

[18] 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.

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

[20] 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.

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

[22] 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.

[23] D. Fett, R. Küsters, and G. Schmitz. Analyzing the BrowserID SSO system with primary identity providers using an expressive model of the Web. In *20th European Symposium on Research in Computer Security (ESORICS)*, 2015.

[24] 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)*, pages 1358–1369, 2015.

[25] D. Fett, R. Küsters, and G. Schmitz. A comprehensive formal security analysis of OAuth 2.0. In *23rd ACM Conference on Computer and Communications Security (CCS)*, pages 1204–1215, 2016.

[26] D. Fett, R. Küsters, and G. Schmitz. The Web SSO standard OpenID Connect: In-depth formal security analysis and security guidelines. In *30th IEEE Computer Security Foundations Symposium (CSF)*, 2017.

[27] T. K. Frederiksen, J. Hesse, B. Poettering, and P. Towa. Attribute-based single sign-on: Secure, private, and efficient. *Privacy Enhancing Technologies*, 2023(4):35–65, 2023.

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

[29] 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.

[30] Google Developers Blog. Google Identity Platform. https://developers.google.com/identity/. Accessed August 20, 2019.

[31] 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.

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

[33] 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.

[34] 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*, 15:223–236, 2020.

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

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

[37] 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.

[38] 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)*, pages 139–142, 2016.

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

[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)*, pages 276–291, 2016.

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

[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)*, pages 577–594, 2009.

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

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

[45] W. Lueks, B. Hampiholi, G. Alpar, and C. Troncoso. Tandem: Securing keys by using a central server while preserving privacy. *Privacy Enhancing Technologies*, 2020(3):327–355, 2020.

[46] 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.

[47] J. Maheswaran, D. Wolinsky, and B. Ford. Crypto-book: An architecture for privacy preserving online identities. In *12th ACM Workshop on Hot Topics in Networks (HotNets)*, 2013.

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

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

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

[51] J. Richer. MITREid Connect. http://mitreid-connect.github.io/index.html. Accessed August 20, 2021.

[52] 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.

[53] M. Schanzenbach, T. Kilian, J. Schutte, and C. Banse. ZKlaims: Privacy-preserving attribute-based credentials using non-interactive zero-knowledge techniques. In *16th International Joint Conference on e-Business and Telecommunications (ICETE), Vol 2: SECRYPT*, 2019.

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

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

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

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

[58] 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.

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

[60] 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.

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

[62] Z. Zhang, C. Xu, C. Jiang, and K. Chen. TSAPP: Threshold single-sign-on authentication preserving privacy. *IEEE Transactions on Dependable and Secure Computing*, Early Access.

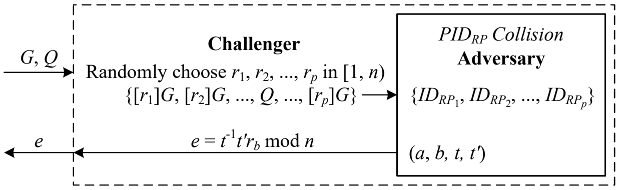
A Security and Privacy with Leaked ID_U

If some users' identities were leaked, colluding RPs could calculate $[ID_U]ID_{RP_j}$ for each known ID_U and link them. So RP unlinkability is broken in this case. Meanwhile, IdP-based login tracing is prevented, because the IdP always knows user identities and the proof of privacy against IdP-based login tracing (i.e., Theorem 3) still holds in this case.

A.1 Patched Step for Leaked ID_U

In the case of leaked user identities, RP designation and user identification are broken as below. For example, in order to log into an honest RP as a victim user, whose account is $Acct' = [u']ID_{RP'}$, a malicious user with $ID_U = u$ chooses x random in $\mathbb{Z}_n$, and colludes with some malicious RP to request an identity token by directly sending $PID_{RP} = [x^{-1}u^{-1}]Acct'$ to the IdP. A token binding $PID_{RP} = [x^{-1}u^{-1}]Acct'$ and $PID_U = [u]PID_{RP} = [x^{-1}]Acct'$ will be signed. Then, the malicious user visits the honest RP with this token and the trapdoor x, and the derived account will be $[xx^{-1}]Acct' = Acct'$. Or, after obtaining a token binding $PID_{RP} = [t]ID_{RP}$ and $PID_U = [u]PID_{RP}$, this malicious user could impersonate the victim at the designated honest RP, with this token and the trapdoor $tu^{-1}u'$, and the derived account will be $[u']ID_{RP}$.

Therefore, in order to provide secure SSO services proactively against this threat, in Step 4.2 an RP calculates $PID_{RP} = [t]ID_{RP}$ by itself and accepts identity tokens only with matched PID_{RP}.



G, Q
Challenger
Randomly choose $r_1, r_2, ..., r_p$ in $[1, n)$
$\{[r_1]G, [r_2]G, ..., Q, ..., [r_p]G\}$
PID_{RP} Collision
Adversary
$\{ID_{RP_1}, ID_{RP_2}, ..., ID_{RP_p}\}$
(a, b, t, t')
$e = t^{-1}t'r_b \bmod n$
e

Figure 7: The PPT algorithm $\mathcal{D}_c^*$ constructed based on the PID_{RP} collision game to solve the ECDLP.

A.2 Security Analysis

Assume p RPs totally in UPPRESSO, whose identities are denoted as $\mathbb{ID}_{RP} = \{[r_j]G, 1 \leq j \leq p\}$. We analyze that, in the case of leaked user identities, RP designation and user identification are ensured by the patch in Step 4.2.

Lemma 1 (PID_{RP} Collision-Freeness)**.** Given $\mathbb{ID}_{RP}$, an adversary cannot find t and t' satisfying $[t]ID_{RP_j} = [t']ID_{RP_{j'}}$ where $j \neq j'$.

Proof. Finding t and t' that satisfy $[t]ID_{RP_j} = [t']ID_{RP_{j'}}$, can be described as a PID_{RP}-collision game $\mathcal{G}_c$ between an adversary and a challenger: the adversary receives from the challenger a finite set of RP identities, i.e., ID_{RP_1}, ..., ID_{RP_p}, and outputs (a, b, t, t') where $a \neq b$. If $[t]ID_{RP_a} = [t']ID_{RP_b}$, which occurs with a probability $\Pr_s$, the adversary succeeds in this game.

As depicted in Figure 7, we design a probabilistic polynomial time (PPT) algorithm $\mathcal{D}_c^*$ based on $\mathcal{G}_c$, to solve the ECDLP: find a number $x \in \mathbb{Z}_n$ satisfying $Q = [x]G$, where Q is a point on $\mathbb{E}$ and G is a generator on $\mathbb{E}$ of order n.

The algorithm $\mathcal{D}_c^*$ works as below. The input of $\mathcal{D}_c^*$ is in the form of (G, Q). On receiving an input (G, Q), the challenger first randomly chooses $r_1, \cdots, r_p$ in $\mathbb{Z}_n$ to calculate $[r_1]G, \cdots, [r_p]G$. Then, it randomly chooses $j \in [1, p]$, replaces $[r_j]G$ with Q, and sends these p RP identities to the adversary, which returns the result (a, b, t, t'). Finally, the challenger calculates $e = t^{-1}t'r_b \bmod n$ and returns e as the output of $\mathcal{D}_c^*$.

If the adversary succeeds in $\mathcal{G}_c$ and $[r_a]G$ happens to be replaced with Q, then $\mathcal{D}_c^*$ outputs $e = t^{-1}t'r_b = x$ because $[tr_a]G = [t]Q = [t'r_b]G$. For the adversary, Q is indistinguishable from any other RP identities in the input set, as $[r_j]G$ is randomly replaced by the challenger. Hence, the probability of solving the ECDLP using $\mathcal{D}_c^*$ is formulated as:

$$\Pr\{\mathcal{D}_c^*(G, [x]G) = x\} = \Pr\{e = x\} = \Pr\{a = j\}\Pr_s = \frac{1}{m}\Pr_s$$

If the probability of finding t and t' satisfying $[t]ID_{RP_j} = [t']ID_{RP_{j'}}$ is non-negligible, the adversary would have advantages in $\mathcal{G}_c$ and $\Pr_s$ is non-negligible regardless of the security parameter λ. Thus, we would find that $\Pr\{\mathcal{D}_c^*(G,[x]G)=x\}$ also becomes non-negligible even when λ is sufficiently large, because m is a finite integer and $m \ll 2^{\lambda}$. This violates the ECDLP assumption. Therefore, the probability of finding t and t' that satisfy $[t]ID_{RP_j} = [t']ID_{RP_{j'}}$ is negligible. □

Due to the collision-freeness property of PID_{RP}, an identity token designates only the target RP and other (honest) RP will reject this token. So RP designation is ensured by patched Step 4.2, in the case of leaked user identities.

Because t is verified with PID_{RP} and ID_{RP}, the only meaningful account derived from TK binding $PID_{RP} = [t]ID_{RP}$ and $PID_U = [u]PID_{RP}$, is $Acct = [u]ID_{RP}$. Thus, user identification is also ensured.