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 login to numerous RPs. However, SSO introduces extra privacy threats, compared with traditional authentication mechanisms, as (*a*) the IdP could track all RPs which a user is visiting, and (*b*) colluding RPs could learn a user's online profile by linking his identities across these RPs. This paper proposes a privacy-preserving SSO system, called *UPPRESSO*, to protect a user's online profile against both a curious IdP and colluding RPs. We analyze the identity dilemma between the security requirements and these privacy concerns, and convert the SSO privacy problems into an identity transformation challenge. In each login instance, an *ephemeral pseudo-identity* of the RP (denoted as PID_{RP}), is firstly negotiated between the user and the RP. PID_{RP} is then sent to the IdP and designated in the identity token, so the IdP is not aware of the visited RP. Meanwhile, PID_{RP} is used by the IdP to transform the *permanent user identity* ID_U into an *ephemeral user pseudo-identity* (denoted as PID_U) in the identity token. On receiving the identity token, the RP transforms PID_U into a *permanent account* of the user (denoted as *Acct*), by a trapdoor in the negotiation. Given a user, the account at each RP is unique and different from ID_U, so colluding RPs cannot link his identities across these RPs. We build the UPPRESSO prototype on top of MITREid Connect, an open-source OIDC implementation. The extensive evaluation shows that UPPRESSO fulfills the requirements of both security and privacy and introduces reasonable overheads.

1 Introduction

Single sign-on (SSO) protocols such as OpenID Connect (OIDC) [1], OAuth 2.0 [2] and SAML [3,4], are widely deployed for identity management and authentication. With the help of SSO, a user logins to a website, referred to as the *relying party* (RP), using his account registered at a trusted web service, known as the *identity provider* (IdP). An RP delegates user identification and authentication to an IdP, which issues an *identity token* (e.g., id token in OIDC or identity assertion in SAML) for a user to visit the RP. For example, in a popular OIDC system, a user sends a login request to an RP, and this RP constructs an identity-token request with its identity (denoted as ID_{RP}) and redirects this request to the IdP of this system. After authenticating the user, the IdP issues an identity token explicitly binding the identities of both the user and the RP (i.e., ID_U and ID_{RP}), which is returned to the user and forwarded to the RP. Finally, the RP verifies the identity token to decide whether the token holder is allowed to login or not. So a user keeps only one credential for the IdP, instead of several credentials for different RPs.

As the comprehensive solution of identity management and authentication, SSO services allow an IdP to provide more user attributes in identity tokens, along with an authenticated user's identity. The attributes (e.g., age, hobby, education, and nationality) are maintained at the IdP, and enclosed in identity tokens after the user's authorization [1,2].

The wide adoption of SSO raises concerns on user privacy [5–8], because SSO facilitates curious parties to track a user's login activities. To issue identity tokens, in each login instance an IdP is aware of when and to which RP a user attempts to login. As a result, an honest-but-curious IdP could track all the RPs that each user has visited over time [6,7], called the *IdP-based login tracing* in this paper. Meanwhile, RPs learn users identities from the received identity tokens. If the IdP encloses an identical user identity in the tokens for a user to visit different RPs [8–10], colluding RPs could link these login instances across the RPs to learn his online profile [8]. We denote this privacy risk as the *RP-based identity linkage*.

Privacy-preserving SSO schemes try to provide comprehensive identity management and authentication, while protecting user privacy [5–8]. The following features of SSO are commonly desired: (*a*) *User identity at an RP*, i.e., an identity token enables an RP to identify every user uniquely, (*b*) *User authentication to only a trusted IdP*, i.e., the steps of authentication between a user and the RP are eliminated, and a user only needs to hold the secret credential to authenticate himself to an IdP, and (*c*) *Provision of IdP-confirmed user* *attributes*, i.e., a user maintains his attributes at the trusted IdP, and RP-requested attributes are provided after authorized by the user. Meanwhile, the privacy threats from different types of adversaries are considered: (*a*) *an honest-but-curious IdP*, (*b*) *colluding RPs*, and (*c*) *an honest-but-curious IdP colluding with some RPs*. We analyze existing privacy-preserving solutions of SSO and also identity federation in Section 2.2.

We conceptualize the privacy requirements of SSO into an *identity transformation* problem, and propose an Untraceable and Unlinkable Privacy-PREserving Single Sign-On (UPPRESSO) protocol. We design three identity-transformation functions in the SSO login flow. In each login instance, ID_{RP} is transformed to an ephemeral PID_{RP} by the RP and the user. Then, PID_{RP} is sent to the IdP to transform ID_U into ephemeral PID_U, so the identity token binds PID_U and PID_{RP}, instead of permanent ID_U and ID_{RP}. Finally, after receiving an identity token with matching PID_{RP}, the RP transforms PID_U into an account. Given a user, this account is identical across multiple login instances for an RP and unique at each RP.

UPPRESSO prevents the IdP-based login tracing because only PID_{RP} is in the identity-token request, and the RP-based identity linkage because every user account is unique (see Section 5 for details). On the contrary, existing privacy-preserving SSO solutions [5–7, 10] prevent only one of these two privacy threats. The identity transformations work compatibly with the widely-used SSO protocols [1–3,5], so the above desirable features of SSO services are kept in UPPRESSO, while not all these features are supported in privacy-preserving identity federation [11–16]. Our contributions are as follows.

- We formalize the SSO privacy problems as an identity-transformation challenge, and propose a solution to protect the users' online profiles; that is, solve this challenge by designing identity-transformation functions.
- The UPPRESSO protocol is proposed based on the identity transformations, with several designs specific for web applications. We prove that UPPRESSO satisfies the security and privacy requirements of SSO services.
- We build the UPPRESSO prototype for web applications, on top of an open-source OIDC implementation. The experimental performance evaluations show that UPPRESSO introduces reasonable overheads.

The remainder is organized as below. Section 2 presents the background and related works. The identity dilemma of privacy-preserving SSO is analyzed in Section 3, and Section 4 presents the designs of UPPRESSO. Security and privacy are analyzed in Section 5. We explain the prototype implementation and experimental evaluations in Section 6, and discuss extended issues in Section 7. Section 8 concludes this work.

2 Background and Related Works

We introduce typical SSO login flows, and discuss existing privacy-preserving solutions and other related works.

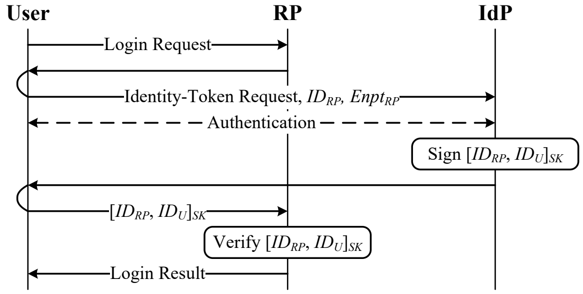

User
RP
IdP
Login Request
Identity-Token Request, ID_{RP}, $Enpt_{RP}$
Authentication
Sign $[ID_{RP}, ID_U]_{SK}$
$[ID_{RP}, ID_U]_{SK}$
Verify $[ID_{RP}, ID_U]_{SK}$
Login Result

Figure 1: The implicit SSO login flow of OIDC.

2.1 OpenID Connect and SSO Services

OIDC is one of the most popular SSO protocols. Users and RPs initially register at an IdP with their identities and other information such as user credentials (e.g., passwords) and RP endpoints (i.e., the URLs to receive tokens). It supports three types of login flows, i.e. implicit flow, authorization code flow, and hybrid flow (a mix-up of the other two). They work with different steps to request/receive identity tokens, but share the common security requirements of identity tokens. Next, we focus on the implicit flow to present our designs. In Section 7 we discuss the supports for the authorization code flow.

As shown in Figure 1, when a user initiates a login request to an RP, the RP constructs an identity-token request with its own identity and the scope of the requested user attributes. This request is redirected to an IdP. After authenticating the user, the IdP issues an identity token that will be forwarded by the user to the RP's endpoint. The token contains the identities (or pseudo-identities) of user and RP, a validity period, the requested user attributes, etc. Finally, the RP verifies the received identity token and allows the user to login as the enclosed (pseudo-)identity. The user's operations including redirection, authorization, and forwarding are implemented in the user agent (e.g., a browser for web applications).

Three features are desired in SSO, supported by popular SSO systems [1–5].

User identification at an RP. The RP recognizes each user (by an identity or account *unique* at the RP) and then provides customized services across multiple logins.

User authentication to *only* an IdP. The authentication between a user and an IdP is usually conducted *independently* of the specifications of widely-adopted SSO protocols [1–3], where the RPs verify only the tokens issued by the IdP. This brings advantages. First, the IdP authenticates users by any appropriate means (e.g., password, one-time password, or multi-factor authentication). Meanwhile, a user only maintains his credential at the IdP. If it is lost or leaked, the user only renews it at the IdP. However, if a user proves some non-ephemeral secret to RPs, which is valid across multiple login instances (i.e., authentication steps are actually involved), he has to notify each RP when lost or leaked, or additional revocation checking is needed [12, 13].

Table 1: Privacy-Preserving Solutions of SSO and Identity Federation.

Solution	**SSO Feature** - supported ●, unsupported ○, or partially ◐			**Privacy Threat** - prevented ●, or not ○		
	User Identity at an RP	User Authentication to Only an IdP	IdP-Confirmed Selective Attribute Provision	IdP-based Login Tracing	RP-based Identity Linkage	Collusive Attack by the IdP and RPs
OIDC w/ PPID [5]	●	●	●	○	●	-
BrowserID [7]	●	●[1]	○	●	○	-
SPRESSO [6]	●	●	◐[2]	●	○	-
PRIMA [17]	●	○	●	●	○	-
PseudoID [11]	●	○	◐[3]	●	●	●
EL PASSO [12]	●	○	●	●	●	●
UnlimitID [13]	●	○	●	●	●	●
Opaak [14]	◐[4]	○	○	●	●	●
Fabric Idemix [16]	○[5]	○	●	●	●	●
U-Prove [15]	●	○	◐[6]	●	●	●
UPPRESSO	●	●	●	●	●	○

1. A BrowserID user generates an *ephemeral* private key to sign the "subsidiary" token, which is verified by the RP.
2. SPRESSO can be extended to provide user attributes in the tokens, while the prototype does not implement this feature.
3. Blindly-signed user attributes can be selectively provided using zero-knowledge proofs, but not implemented in the prototype.
4. Opaak supports exclusive pseudonym options: (*a*) linkable within an RP but unlinkable across multiple RPs and (*b*) unlinkability for any two actions.
5. In the original design of Idemix [18], every user logins to an RP with a unique account.
6. A U-Prove token may contain some attributes *invisible* to the IdP, in addition the ones confirmed by the IdP.

IdP-confirmed selective attribute provision. An IdP usually provides user attributes in the tokens [1, 2], in addition to user (pseudo-)identities. These attributes are maintained by users at the IdP. Before enclosing any attributes in a token, the IdP obtains the user's authorization or provides only the attributes pre-selected by the user.

2.2 Privacy-Preserving SSO and Identity Federation

We summarize existing privacy-preserving solutions for SSO and identity federation in Table 1. Widely-adopted SSO protocols [1–4] allow a user to login to an RP *without holding any permanent secret verified by the RP or by himself maintaining an account at the RP*. While preserving these features, existing privacy-preserving SSO approaches [5–7] prevent either the IdP-based login tracing or the RP-based identity linkage, and UPPRESSO prevents both of them.

Identity federation enables a user registered at a trusted IdP to be accepted by other parties, with different accounts sometimes, but more user operations are involved than those of SSO. Privacy-preserving identity federation protects user privacy against even collusive attacks by the IdP and RPs, but requires a user [11–16] to (*a*) *hold long-term secrets verified by RPs*, in addition to the authentication credentials for the IdP, and (*b*) *locally manage the accounts at different RPs*. That is, there are actually some authentication steps between the user and RPs (called asynchronous authentication [12]).

Pairwise pseudonymous identifiers (PPIDs) are specified in SSO protocols [1, 4] and recommended [5] to protect user privacy against curious RPs. When issuing an identity token, an IdP encloses a user PPID (but not the identity at the IdP). Given a user, the IdP assigns a unique PPID based on the target RP. So colluding RPs cannot link the users. PPIDs cannot prevent the IdP-based login tracing because the IdP needs the RP's identity to issue tokens.

Several solutions prevent the IdP-based login tracing but are vulnerable to the RP-based identity linkage. In BrowserID [7] (formerly known as Firefox Accounts [10] and Mozilla Persona [19]), an IdP issues a special token (called user certificate) to bind a user identity to an ephemeral public key. With the corresponding private key, the user signs a "subsidiary" token (called identity assertion) to bind the target RP's identity and sends both tokens to the RP. In SPRESSO [6] an RP creates a verifiable one-time pseudo-identity for itself in each login instance, which is enclosed in the identity token issued by an IdP. A PRIMA IdP signs a credential binding a verification key and user attributes [17], where the key is considered the user identity. The user selectively provides IdP-confirmed attributes to an RP using his signing key [20]. In these schemes [6, 7, 17], colluding RPs could link a user based on his unique identity in the tokens (or credentials).

PseudoID [11] introduces an independent service in addition to the IdP to *blindly* sign [21] an access token binding a pseudonym and a user secret. The user unblinds this token and an IdP will assert it, which allows the user to login to an RP using his secret. Two kinds of privacy threats are prevented, because (*a*) the RP's identity is not enclosed in the token and (*b*) the user selects different pseudonyms when visiting RPs. Collusive attacks by the IdP and RPs are also prevented, for they cannot link two blindly-signed tokens.

In EL PASSO [12], after authenticating a user, an IdP signs an *anonymous credential* [22] binding a secret, both of which are securely kept on the user's device. When attempting to login to an RP, the user proves that he is the owner of this credential using the secret, and discloses selective attributes in the credential. Although one credential is proved to multiple

RPs, *user-managed pseudonyms* and *anonymous credentials* prevent the RPs, even when collusive with the IdP, from linking the users across the RPs. UnlimitID [13] presents similar designs based on anonymous credentials [22], to prevent collusive attacks by the IdP and RPs. NEXTLEAP [23] builds anonymous secure messaging on top of UnlimitID.

Anonymous credentials [22, 24] are used in more flexible ways. Opaak [14] keeps IdP-signed anonymous credentials as pseudonym tokens, which bind a user's secret key. Idemix anonymous credentials [18] are integrated in Hyperledger Fabric [16] to implement completely-unlinkable pseudonyms and IdP-confirmed selective attribute disclosure. With a U-Prove token [15, 25] from an IdP, a user is enabled to authenticate himself to an RP and selectively disclose attributes.

2.3 Extended Related Works

Privacy-Preserving Token or Credential. In addition to sign-on, privacy enhancing technologies have been applied in different scenarios, but not adopted to comprehensively transform the five (pseudo-)identities in SSO services. PrivacyPass and TrustToken [26, 27] allow a user to receive a great number of *anonymous* tokens. These tokens are used to access resources on content delivery networks, so the user does not interact with challenges such as CAPTCHAs. ZKlaims [28] allow users to prove statements on the credentials issued by a trusted party using zero-knowledge proofs, but the credential contents are not revealed. Crypto-Book [29] coordinates servers to generate a ring-signature private key, and a user picks up his private key through a list of Email addresses (i.e., an anonymity set). Then the key pair works as an untraceable pseudonym to sign messages. Two-party threshold schemes are implemented with a central server, to protect user private keys [30, 31]: to sign or decrypt a message, the user needs a token from the server. Tandem [32] decouples the obtaining and using of such tokens, for the privacy of key usage.

Anonymous SSO. Such schemes allow authenticated users to access a service protected by an IdP, without revealing their identities. Anonymous SSO was proposed for GSM communications [33], and formalized [34]. Privacy-preserving primitives, such as group signature, zero-knowledge proof, Chebyshev Chaotic Maps and proxy re-verification, were adopted to design anonymous SSO [34–37]. Anonymous SSO schemes work for some applications, but are unapplicable to most systems that require user identification for customized services.

Formal Analysis on SSO Protocols. Fett et al. [38, 39] formally analyzed OAuth 2.0 and OIDC using an expressive Dolev-Yao style model [40], and presented the attacks of 307 redirection and IdP mix-up. SAML-based SSO is also analyzed [41], and the RP identity is found not to be correctly bound in the identity tokens of a variant designed by Google.

SSO Implementation Vulnerabilities. Vulnerabilities were found in SSO implementations for web applications, resulting in effective attacks by breaking confidentiality [42–46], integrity [42, 46–50] or RP designation [46, 48–51] of identity tokens. Integrity of identity tokens was violated in SSO systems due to software flaws such as defective verification by RPs [42, 48, 50], XML signature wrapping [47], and IdP spoofing [49, 50]. RP designation is broken for incorrect binding by the IdP [48, 51] or insufficient verification by RPs [49–51].

Automatic tools such as SSOScan [52], OAuthTester [53] and S3KVetter [51], detect the violations of confidentiality, integrity, or RP designation of SSO identity tokens. Wang et al. [54] detect the vulnerable applications built with authentication/authorization SDKs, due to the implicit assumptions of these SDKs. Navas et al. [55] discussed the possible attack patterns against OIDC services.

In a mobile system, browsers, IdP Apps, or IdP-provided SDKs are responsible for forwarding identity tokens, but none of them ensures the identity tokens are sent to the designated RP only [56, 57]. Vulnerabilities were found in Android Apps, to break confidentiality [56–59], integrity [56, 58], and RP designation [56, 59] of identity tokens. A flaw was found in Google Apps [44], allowing a malicious RP to hijack a user's authentication attempt and inject a payload to steal the cookie (or identity token) for another RP.

If a user is compromised, the attackers will login to RPs on behalf of him. Single sign-off [60] helps the victim to revoke all his tokens accepted and logout from the RPs.

3 The Identity-Transformation Framework

This section investigates the security requirements of privacy-preserving SSO, and explains the identity dilemma. Then, we present the identity-transformation framework.

3.1 Security Requirements of SSO

The primary goal of non-anonymous SSO services is to ensure that a *legitimate* user is able to login to an *honest* RP as his permanent account at this RP, by presenting the *identity tokens* issued by an *honest* IdP.

To achieve this goal, an identity token issued by an IdP [1–7] specifies (*a*) the RP to which the user requests to login (i.e., *RP designation*) and (*b*) the user who is authenticated by the IdP (i.e., *user identification*). Therefore, an honest RP compares the designated RP identity (or pseudo-identity) in identity tokens with its own before accepting the tokens; otherwise, a malicious RP could replay a received identity token to the honest RP and login as the victim user. The RP allows the token holder to login as the user (pseudo-)identity specified in the accepted tokens.

The SSO login flow also requires *confidentiality* and *integrity* of identity tokens. An identity token should be forwarded by the authenticated user to the target RP only, not leaked to any other parties; otherwise, an adversary who presents the token, would successfully login to the RP. Integrity is necessary to prevent adversaries from tampering with a token. So

Table 2: The (pseudo-)identities in privacy-preserving SSO.

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 the user's i-th login instance to the j-th RP.	Ephemeral
$PID^i_{RP_j}$	The j-th RP's pseudo-identity, in the user's i-th login instance to this RP.	Ephemeral
$Acct_j$	The user's identity (or account) at the j-th RP.	Permanent

identity tokens are signed by the IdP and usually transmitted over HTTPS [1–3].

These security requirements (i.e., RP designation, user identification, confidentiality, and integrity) of SSO identity tokens are well analyzed [38, 39, 41], and vulnerabilities breaking any of the properties result in attacks [42–53, 56–59, 61, 62].

3.2 The Identity Dilemma of Privacy-Preserving SSO

We aim to design a privacy-preserving SSO system with the four security properties as above, while preventing the privacy threats due to both the IdP-based login tracing and the RP-based identity linkage. Table 2 lists the notations in the following explanations, and the subscript j and/or the superscript i may be omitted when there is no ambiguity. We explicitly distinguish a user's identity (or *account*) at the target RP, from the user's *identity* at the IdP and the user's *pseudo-identity* enclosed in identity tokens.

An identity token contains the (pseudo-)identities of the authenticated user and the target RP. Since an IdP authenticates users and always knows the user's identity (i.e., ID_U), to prevent the IdP-based login tracing, we should not reveal the target RP's permanent identity (i.e., ID_{RP}) to the IdP. So an *ephemeral* pseudo-identity for the RP (i.e., PID_{RP}) should be used in the identity-token request: (a) to ensure RP designation, PID_{RP} should be *uniquely* associated with the target RP; and (b) the IdP cannot derive any information about ID_{RP} from any PID^i_{RP}, which implies PID^i_{RP} in multiple login instances should be independent of each other.[1]

To prevent the RP-based identity linkage, the IdP does not enclose ID_U in identity tokens. A user pseudo-identity (i.e., PID_U) is bound instead: (a) in multiple login instances to an RP, PID^i_U should be independent of each other and generated ephemerally, to prevent the IdP-based login tracing;[2] (b) the RP cannot derive any information about ID_U from any $PID_{U,j}$, which implies $PID_{U,j}$ for different RPs should be independent of each other; and (c) to ensure user identification, an *ephemeral* PID^i_U in each login instance should enable the RP to recover the *permanent* account (i.e., $Acct$) at this RP.

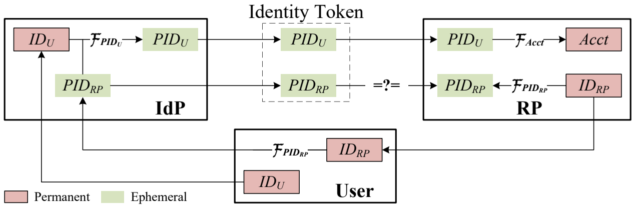


Figure 2: Identity transformations in privacy-preserving SSO.

Given a user, (a) an identity token contains only pseudo-identities, i.e., $PID^i_{U,j}$ and $PID^i_{RP_j}$, which are independent of each other for different RPs and in multiple login instances, respectively, and (b) the two *ephemeral* pseudo-identities enable the target RP to derive a *permanent* account, i.e., $Acct_j$. The relationship among the (pseudo-)identities in identity tokens is illustrated in Figure 2. The *red* and *green* blocks represent *permanent* and *ephemeral* (pseudo-)identities, respectively. The arrows denote the transformations of (pseudo-)identities.

To satisfy the requirements of both security and privacy, we pose the following *identity dilemma* of SSO identity tokens: *Given an authenticated user and an unknown RP (i.e., permanent ID_U and ephemeral PID_{RP}), an IdP is expected to generate an ephemeral pseudo-identity (i.e., PID_U) which will be correlated with the user's permanent identity at this RP (i.e., $Acct$), while knowing nothing about the RP's identity or the user's account at this RP (i.e., ID_{RP} or $Acct$).*

Existing privacy-preserving SSO solutions (i.e., SPRESSO [6], BrowserID [7] and PPID [5]) do not explicitly and comprehensively consider all five (pseudo-)identities in the SSO login flow, and then ID_U or ID_{RP} is still enclosed in identity tokens. So only one type of privacy threat is prevented.

3.3 Identity Transformation

The privacy protection of SSO is converted into a challenge to design *identity-transformation functions* as below.

- $\mathcal{F}_{PID_{RP}}(ID_{RP}) = PID_{RP}$, calculated by the user and the RP. From 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. From the 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 RP. Given ID_U and ID_{RP}, $Acct$ keeps permanent and unique to other accounts at this RP. In the user's any i-th and i'-th ($i \neq i'$) login instances to the RP, $\mathcal{F}_{Acct}(PID^i_U, PID^i_{RP}) = \mathcal{F}_{Acct}(PID^{i'}_U, PID^{i'}_{RP})$.

In an SSO login flow with identity transformations, a user firstly negotiates an ephemeral PID_{RP} with the target RP. An identity-token request with PID_{RP} is sent by the user to an IdP. After 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 binding PID_U and PID_{RP}. After matching the

[1] Even when the target RP is kept unknown to the IdP, the IdP should not link multiple login instances visiting this RP.

[2] If PID^i_U is not completely independent of each other, it implies the IdP could link multiple login instances visiting this RP.

designated RP pseudo-identity in the token, the RP calculates $Acct$ and allows the token holder to login as $Acct$.

4 The Designs of UPPRESSO

This section presents the threat model and assumptions. We then introduce identity-transformation functions satisfying the requirements and the detailed UPPRESSO protocols.

4.1 Threat Model

The system consists of an honest-but-curious IdP, a number of RPs and users which could be compromised. This model is consistent with the widely-used SSO services [1–5].

Honest-but-curious IdP. The IdP strictly follows the protocol, while being interested in learning user profiles. For example, it might store all received messages to infer the relationship among ID_U, ID_{RP}, PID_U, and PID_{RP} to track a user's login activities. The IdP never actively violates the protocols, so a script downloaded from the IdP also strictly follows the designs (see Section 4.4). The IdP is trusted to maintain the private key for signing identity tokens and RP certificates. So adversaries cannot forge such tokens or certificates.

We do *not* consider the collusion of the IdP and RPs. If the IdP could collude with some RPs, a user will finish login flows completely with colluding entities and then in principle we need a long-term user secret to protect (or transform) the permanent accounts at these RPs. This secret is held *only* by the user; otherwise, the colluding IdP and RPs can always link these accounts. However, such accounts derived from this secret have to be updated, when it is lost or leaked (see discussions in Section 7).

Malicious Users. We assume the adversary could control a set of users, by stealing users' credentials or registering Sybil users in the system. They want to impersonate a victim user at honest RPs, or allure the benign user to login to an honest RP under the adversary's account. A malicious user might modify, insert, drop or replay a message, or behave arbitrarily in SSO login flows.

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

Colluding Users and RPs. Malicious users and RPs might collude, attempting to break the security and privacy guarantees for benign users. For example, a malicious RP might collude with malicious users to allure victim users to leak an identity token, to impersonate the victim user and login to some honest RP.

Table 3: The notations in the UPPRESSO protocols.

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) of $\mathbb{E}$, and the order of G is a prime number n.
ID_U	$ID_U = u$, $1 \leq u < n$; the user's unique identity at the IdP.
ID_{RP_j}	$ID_{RP} = [r]G$, $1 \leq r < n$; the j-th RP's unique identity.
t	A user-generated random integer in a login instance, $1 \leq t < n$.
$PID^i_{RP_j}$	$PID_{RP} = [t]ID_{RP} = [tr]G$; the j-th RP's pseudo-identity, in the user's i-th login instance to this RP.
$PID^i_{U,j}$	$PID_U = [ID_U]PID_{RP} = [utr]G$; the user's pseudo-identity, in the user's i-th login instance to the j-th RP.
$Acct_j$	$Acct = [t^{-1} \bmod n]PID_U = [ID_U]ID_{RP} = [ur]G$; the user's account at the j-th RP.
SK, PK	The IdP's key pair, a private key and a public key, to sign and verify identity tokens and RP certificates.
$Enpt_{RP_j}$	The j-th RP's endpoint, to receive the identity tokens.
$Cert_{RP_j}$	A signed RP certificate, binding ID_{RP_j} and $Enpt_{RP_j}$.

4.2 Assumptions

HTTPS is adopted to secure the communications between honest entities, and the adopted cryptographic primitives are secure. The software stack of a honest entity is correctly implemented, to transmits messages to the receivers as expected.

UPPRESSO is designed for users who care about privacy, and a user never authorizes the IdP to enclose any *distinctive attributes* in identity tokens, such as telephone number, Email address, etc. A user does not configure distinctive attributes at any RP, either. So the privacy leakage due to re-identification by distinctive attributes across RPs, is out of the scope.

We focus on the privacy threats introduced by SSO services. The network traffic analysis that tracks a user's activities from packets, and the active account linkage through web documents, exist in SSO scenarios but also in all web applications. When a user visits multiple RPs concurrently from one browser, an RP might actively redirect his account to another RP server by crafted web documents. Such attacks will be prevented by other defenses, and are not considered in our work.

4.3 Identity-Transformation Functions

We design identity-transformation functions, on an elliptic curve $\mathbb{E}$. Table 3 lists the notations, and the subscript j and/or the superscript i may be omitted in the case of no ambiguity.

For each user, a unique integer u is assigned by the IdP and $ID_U = u$. When an RP is registering, the IdP generates a random number r, and $ID_{RP} = [r]G$, a unique point on $\mathbb{E}$, is assigned. Here, $[r]G$ is the addition of G on the curve r times.

ID

ID_{RP}-PID_{RP} Transformation. The user selects a random number t ($1 \leq t < n$) as the trapdoor and calculates 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 with ID_U and PID_{RP}, the IdP calculates PID_U.

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

***PID_U-Acct* Transformation.** The trapdoor t is sent to the target RP, which also calculates PID_{RP} to match the RP pseudo-identity in identity tokens. On verifying a token binding PID_U and PID_{RP}, it calculates $Acct$ as below.

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

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

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

So the RP derives an *identical permanent account* from the identity tokens in different login instances, with the help of t. Given a user, the accounts at different RPs are inherently unique; while, given an RP, the accounts of different users are also unique. Moreover, due to the elliptic curve discrete logarithm problem (ECDLP), it is impossible for the RP to derive ID_U from either PID_U or $Acct$, and for the IdP to derive ID_{RP} from PID_{RP}. Section 5 presents the detailed proofs.

Note that r is kept secret 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 $[r']Acct_j = [r]Acct_{j'}$ or not.

4.4 The Designs Specific for Web Applications

UPPRESSO works with commercial-off-the-shelf (COTS) browsers as the user agent. First of all, in UPPRESSO the IdP is not aware of the visited RP, so the user agents (or browsers) have to deal with the forwarding of identity tokens to the target RP, as well as the calculation of PID_{RP}. On the contrary, in commonly-used SSO protocols [1–4] the IdP needs this information to ensure *confidentiality* of identity tokens. In the OIDC services, when an RP registers itself at the IdP, the `redirect_uri` parameter is set as the endpoint URL to receive tokens [1]. Then, when the IdP wants to transmit identity tokens to an RP, it utilizes HTTP 302 redirection with this endpoint as the target URL in the HTTP response, so the user browser forwards it to the designated RP.

In UPPRESSO such user-agent functions are implemented by web scripts within COTS browsers, and the scripts downloaded from an honest entity are also *honest*. Two scripts are from the visited RP and the IdP, and each is responsible for the communications with the origin web server. Only the RP script is not enough to implement a user agent; otherwise, the script will leak its origin to the IdP web server (e.g., an identity-token request sent by the RP script will automatically carry an HTTP `referer` header that discloses the RP domain). Moreover, a trusted script from the honest IdP ensures confidentiality of identity tokens (i.e., sent to only the designated RP) and interacts with the user for the authorization of user attributes, for the RP (and also the RP script) might be malicious. Thus, on receiving a request with the `referer` header, the IdP checks that it is from the IdP script.

The RP script prepares ID_{RP} and $Enpt_{RP}$ for the IdP script, through RP certificates. An RP certificate is signed by the IdP during the RP registration, binding the RP's identity and its endpoint. In a login instance the RP will provide its certificate through the RP script, to the IdP script. The IdP script verifies the RP certificate to extract ID_{RP} and $Enpt_{RP}$. The IdP's public key is set in the IdP script, so a user does not configure anything *locally*, as it does in popular SSO systems [1–4].

After extracting ID_{RP} to calculate PID_{RP} and receiving an identity token from the IdP, the IdP script needs to ensure the RP script will forward this token to $Enpt_{RP}$ which is bound with ID_{RP} in the RP certificate. The scripts communicate with each other within the user browser through the `postMessage` HTML5 API, and the receiver (i.e., the RP script) is restricted by the `postMessage` targetOrigin mechanism [63]. When the IdP script sends messages, the receiver's origin is set as a parameter, e.g., `postMessage(tkn, 'https://RP.com')`. It consists of the protocol (i.e., `https://`), the domain (i.e., `RP.com`) and a port which may be implicit. So only the script downloaded from this targetOrigin is a legal receiver.

Finally, the browser downloads the RP script when visiting an RP, and this RP script opens a new window that downloads the IdP script. We should prevent the referer leakage when the IdP script is downloaded. Generally, when a browser window visits another website not belonging to its opener's origin, the HTTP request to this website automatically carries the `referer` header (i.e., the opener's origin). Such an HTTP header leaks the visited RP's domain to the IdP. Fortunately, in UPPRESSO this newly-opened window is a *redirection* from the RP to the IdP, but not a direct visit by the browser (Figure 3, Steps 1.2-1.3). This leakage is prevented by setting the `referrer-policy=no-referrer` header in the HTTP response from the RP, when it is redirected to the IdP. So the HTTP request to download the IdP script carries no `referer` header. This method is specified by W3C [64] and widely supported. We tested it in browsers including Chrome, Safari, Edge, Opera and Firefox, and confirmed no referer leakage.

4.5 The UPPRESSO Protocols

System Initialization. The IdP generates a key pair (SK, PK) to sign/verify identity tokens and RP certificates. The IdP keeps SK secret, and PK is publicly known.

RP Initial Registration. Each RP registers itself at the IdP to obtain ID_{RP} and its RP certificate $Cert_{RP}$ as follows:

1. An RP sends a registration request, including the endpoint to receive identity tokens and other information.
2. The IdP randomly generates $r \in [1,n)$, until $ID_{RP} = [r]G$ is unique. It 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 to set up a unique identity ID_U and the corresponding credential.

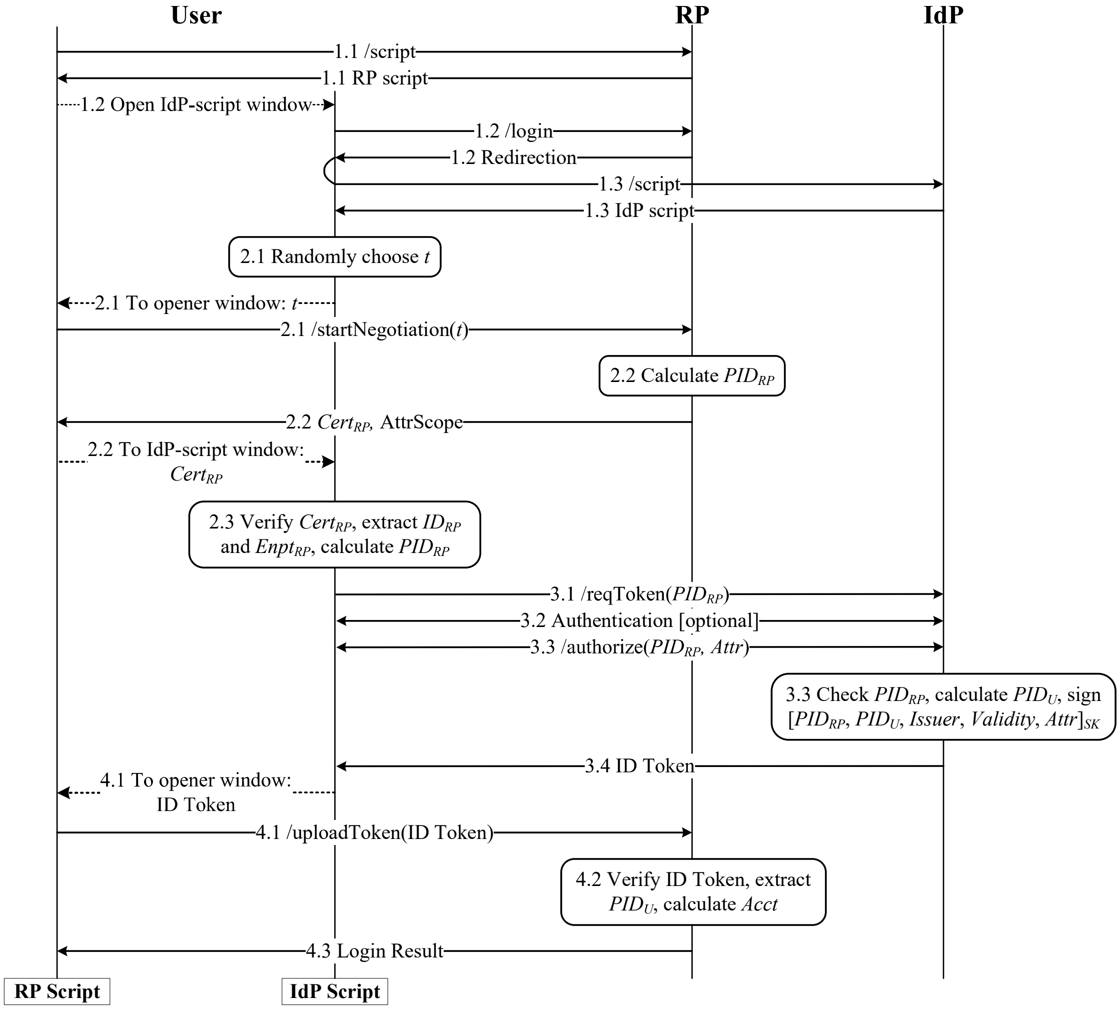

User
RP
IdP
1.1 /script
1.1 RP script
1.2 Open IdP-script window
1.2 /login
1.2 Redirection
1.3 /script
1.3 IdP script
2.1 Randomly choose t
2.1 To opener window: t
2.1 /startNegotiation(t)
2.2 Calculate PID_{RP}
2.2 $Cert_{RP}$, AttrScope
2.2 To IdP-script window: $Cert_{RP}$
2.3 Verify $Cert_{RP}$, extract ID_{RP} and $Enpt_{RP}$, calculate PID_{RP}
3.1 /reqToken(PID_{RP})
3.2 Authentication [optional]
3.3 /authorize(PID_{RP}, $Attr$)
3.3 Check PID_{RP}, 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, extract PID_U, calculate $Acct$
4.3 Login Result
RP Script
IdP Script

Figure 3: The SSO login flow of UPPRESSO.

SSO Login. A login instance consists of four steps, namely script downloading, RP identity transformation, identity-token generation, and *Acct* calculation, as shown in Figure 3. In this figure, the IdP's operations are linked by a vertical line, so are the RP's. Two vertical lines split the user operations into two groups (i.e., in two browser windows), one of which is to communicate with the IdP, and the other is with the target RP. Each solid horizontal line means some messages between the user and the IdP (or the RP), and each dotted line means a `postMessage` invocation between two scripts (or browser windows) within the browser.

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

1.1 When attempting to visit 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 $\mathbb{Z}_n$ and sends it to the RP script through `postMessage`. The RP script sends t to the RP.

2.2 On receiving t, the RP verifies $1 \leq t < n$ and calculates PID_{RP}. The RP replies with $Cert_{RP}$, which is then transmitted from the RP script to the IdP script, as well as the scope of requested user attributes.

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 the identity token.

3.1 The IdP script requests an identity token for PID_{RP} and the user attributes.

3.2 The IdP authenticates the user, if not authenticated yet.

3.3 The IdP script locally obtains the user's authorization to enclose the requested attributes, and send the identity-token request. The IdP checks that the received PID_{RP} is valid, calculates $PID_U = [ID_U]PID_{RP}$ for the user, and signs $[PID_{RP}, PID_U, Issuer, Validity, Attr]_{SK}$, where $Issuer$ is the IdP's identity, $Validity$ indicates the validity period, and $Attr$ contains the requested attributes.

3.4 The IdP replies with the identity token, to the IdP script.

4. *Acct Calculation.* The RP receives the identity token and allows the user to login.

4.1 The IdP script forwards the identity token to the RP script, which sends it to the RP through $Enpt_{RP}$.

4.2 The RP verifies the identity token, including the IdP's signature and its validity period. It also verifies PID_{RP} in the token matches the one negotiated in Step 2.2. Then, the RP extracts PID_U and calculates $Acct = [t^{-1}]PID_U$.

4.3 The RP allows the user to login as $Acct$.

If any verification or check fails, this login flow will be halted immediately. For example, the user halts it on an invalid $Cert_{RP}$. The IdP rejects an identity-token request, if the received PID_{RP} is not on the elliptic curve $\mathbb{E}$. Or, the RP rejects an identity token when PID_{RP} in it does not match the negotiated one.

4.6 Compatibility with OIDC

First of all, UPPRESSO and OIDC work with COTS browsers. Among the four steps of the login flow in UPPRESSO, *script downloading* prepares the user agent. The user agent is responsible for the communications between the IdP and the RP, which are implemented by HTTP redirections in OIDC. On the contrary, in UPPRESSO the scripts hide $Enpt_{RP}$ from the IdP, and forward the identity token to $Enpt_{RP}$ extracted from the RP certificate. So the IdP does not set `redirect_uri` in the HTTP response. Most operations of *RP identity transformation* are conducted within browsers, while the RP only receives t to calculate PID_{RP} and responds with $Cert_{RP}$. The calculation of PID_{RP} is viewed as an operation to prepare the RP identity in OIDC, and the *static* $Cert_{RP}$ is a supplementary message to users. Thus, compared with the original OIDC protocol, in these two steps, the IdP's operations are simplified and an RP customizes its *dynamic* pseudo-identity.

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

Finally, this compatibility is experimentally confirmed by our prototype implementation: only 23 lines of Java code in MITREid Connect [65], an open-source OIDC system, are modified to build the IdP of UPPRESSO (see Section 6.1).

5 The Analysis of Security and Privacy

5.1 Security

UPPRESSO satisfies the security requirements of SSO identity tokens [38, 39, 41], explained in Section 3.1.

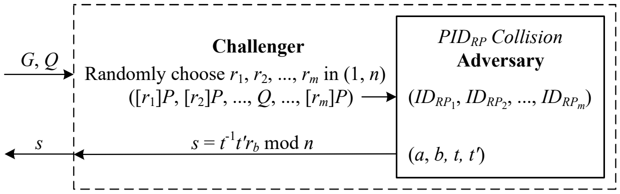

G, Q
Challenger
Randomly choose $r_1, r_2, ..., r_m$ in (1, n)
$([r_1]P, [r_2]P, ..., Q, ..., [r_m]P)$
PID_{RP} Collision
Adversary
$(ID_{RP_1}, ID_{RP_2}, ..., ID_{RP_m})$
s
$s = t^{-1}t'r_b \bmod n$
(a, b, t, t')

Figure 4: The algorithm based on the PID_{RP} collision, to solve the ECDLP.

- **RP Designation** The RP (pseudo-)identity bound in the identity token identifies the target RP, and only this RP.
- **User Identification** The user (pseudo-)identity bound in the identity token identifies the authenticated user, and only this user.
- **Confidentiality** An identity token is accessible to only the authenticated user, the target RP, and the IdP.
- **Integrity** An honest RP accepts only identity tokens binding its (pseudo-)identity and the authenticated user's (pseudo-)identity.

RP Designation. The identity token binds PID_{RP} identifying the target RP, because t is sent to the target RP with ID_{RP} and $PID_{RP} = [t]ID_{RP}$.

Next, based on the ECDLP we prove that, for an adversary, the probability of finding t and t' satisfying $[t]ID_{RP_j} = [t']ID_{RP_{j'}}$ is negligible, where RP_j and $RP_{j'}$ are any two RPs in the finite set of RPs (i.e., $ID_{RP_j} = [r_j]G$ and $ID_{RP_{j'}} = [r_{j'}]G$, while r_j and $r_{j'}$ are kept secret to adversaries). This negligible probability means the token designates *only* the target RP.

Let $\mathbb{E}$ be an elliptic curve, G be a point on $\mathbb{E}$ of order n, and $Q = [x]G$ where x is a random integer in $\mathbb{Z}_n$. Given G and Q, the probability that a probabilistic polynomial time (PPT) algorithm calculates x (i.e., solve the ECDLP) is negligible. For any PPT algorithm $\mathcal{D}$ to calculate x, we define

$$\Pr\{\mathcal{D}(G, [x]G) = x\} = \varepsilon_c(k)$$

Here, $\Pr\{\}$ denotes the probability. So $\varepsilon_c(k)$ becomes negligible with the increasing security parameter k.

Assume a game $\mathcal{G}_c$ between an adversary and a challenger, to describe this PID_{RP}-collision attack: the adversary receives a finite set of RP identities from the challenger, denoted as $(ID_{RP_1}, ID_{RP_2}, ..., ID_{RP_m})$ where m is the amount of RPs in the system, and then outputs (a, b, t, t'). If $[t]ID_{RP_a} = [t']ID_{RP_b}$, the adversary succeeds in this game. Note that m is a finite integer, and $m \ll 2^k$ as k increases. We define the probability that the adversary succeeds in this game as $\Pr_s$.

Figure 4 shows a PPT algorithm $\mathcal{D}_c^*$ based on this game, to solve the ECDLP. The input of $\mathcal{D}_c^*$ is in the form of (G, Q). On receiving an input, the challenger of $\mathcal{G}_c$ randomly chooses $r_1, r_2, \cdots, r_m$ in $\mathbb{Z}_n$, calculates $[r_1]G, [r_2]G, \cdots, [r_m]G$, and randomly replaces some $[r_j]G$ with Q. Then, these m RP identities are sent to the adversary, which returns the result (a,

b, t, t'). Finally, the challenger calculates $s = t^{-1}t'r_b \bmod n$ and returns s as the output of $\mathcal{D}_c^*$.

If $[r_a]G$ happens to be replaced with Q and the adversary succeeds, we find $Q = [s]G$ and then $s = x$ because $[tr_a]G = [t]Q = [t'r_b]G$. As $[r_j]G$ is randomly replaced by the challenger, Q and other RP identities in the input set are indistinguishable to the adversary. Thus,

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

If the adversary is able to find t and t' satisfying that $[t]ID_{RP_j} = [t']ID_{RP_{j'}}$, it will have advantages in $\mathcal{G}_c$ and then $\Pr_s$ becomes non-negligible as k increases. Because $m \ll 2^k$, $\Pr\{\mathcal{D}_c^*(G,[x]G) = x\} = \frac{1}{m}\Pr_s$ also becomes non-negligible with the increasing k. This violates the ECDLP assumption. Thus, the probability of finding t and t' satisfying that $[t]ID_{RP_j} = [t']ID_{RP_{j'}}$ in UPPRESSO is negligible, and then adversaries cannot break RP designation.

User Identification. Given a user, an honest RP with ID_{RP} always deterministically derives an identical account from different identity tokens binding PID_U and PID_{RP}. That is, in the user's any i-th and i'-th ($i \neq i'$) login instances to the RP, $\mathcal{F}_{Acct}(PID_U^i, PID_{RP}^i) = \mathcal{F}_{Acct}(PID_U^{i'}, PID_{RP}^{i'}) = [ID_U]ID_{RP}$.

In the calculation of $Acct = [t^{-1}]PID_U = [t^{-1}][u]PID_{RP}$, PID_U is calculated by the honest IdP based on (a) the authenticated user, i.e., $ID_U = u$, and (b) the received PID_{RP}, while this PID_{RP} is generated by the target RP based on ID_{RP} and t. Thus, the calculated account is always exactly the authenticated user's account at the RP (i.e., $[ID_U]ID_{RP}$).

Confidentiality. No event leaks an identity token to any malicious entity other than the authenticated user and the designated RP. First of all, the communications among the IdP, RPs and users, are protected by HTTPS, and the `postMessage` HTML5 API ensures the dedicated channels between two scripts within the browser, so adversaries cannot eavesdrop the identity tokens. Further, the IdP sends the identity token only to the authenticated user (i.e., the IdP script). The IdP script forwards the token to the RP script only if it is downloaded from the same origin as $Enpt_{RP}$, and the binding of $Enpt_{RP}$ and ID_{RP} is ensured by the signed RP certificate. So only the RP that holds $Enpt_{RP}$ and ID_{RP}, receives this token.

Integrity. The identity token binds $Acct$ and ID_{RP} implicitly, and any breaking results in some failed check or verification in the login flow as below. The identity token binding PID_U and PID_{RP} is signed by the IdP. According to the proof of RP designation, there is no $t' \neq t$ but satisfying that $PID_{RP} = [t]ID_{RP_j} = [t']ID_{RP_{j'}}$. That is, the identity token explicitly binding PID_U and PID_{RP}, matches *only* one ID_{RP} and then also *only* one $Acct = [t^{-1}]PID_U$. Thus, $Acct$ and ID_{RP} are actually bound by the IdP's signatures, due to the one-to-one mapping between (a) the pair of $Acct$ and ID_{RP} and (b) the triad of PID_U, PID_{RP}, and t.

Finally, we analyzed the security properties of UPPRESSO, based on a Dolev-Yao style model [6]. The model abstracts the entities in a web system, such as web servers and browsers, as *atomic processes*. It defines *script processes* to formulate client-side scripts. UPPRESSO contains atomic processes as follows: an IdP process, a finite set of web servers for honest RPs, a finite set of honest browsers, and a finite set of attacker processes. These processes communicate with each other through events such as HTTPS request and response. We consider all RP and browser processes are honest, while model an RP or a browser controlled by an adversary as attacker processes. Within a browser, honest IdP scripts, honest RP scripts and also attacker scripts are invoked. Script processes communicate with each other through `postMessage`, modelled as transmitted-to-itself events of the browser process.

After formulating the system by the Dolev-Yao style model, we trace the lifecycle of an identity token, starting when it is generated and ending when accepted by the RP, to ensure it is not leaked to attackers or tampered with by any adversary without checks. We locate the generation of a token, and trace to all places where PID_U, PID_{RP} and other parameters in the token are calculated and transmitted, to ensure no adversary able to retrieve or manipulate them. That is, confidentiality and integrity of identity tokens are proved formally.

5.2 Privacy

UPPRESSO effectively prevents the threats of IdP-based login tracing and RP-based identity linkage.

IdP-based Login Tracing. The information accessible to the IdP and derived from the RP's identity, is only PID_{RP}, where $PID_{RP} = [t]ID_{RP}$ is calculated by the user. Because (a) t is a random number from $\mathbb{Z}_n$ and kept secret to the IdP and (b) $ID_{RP} = [r]G$ and G is the base point (or generator) of $\mathbb{E}$, from the IdP's view PID_{RP} is indistinguishable from a variables randomly chosen from $\mathbb{E}$, and it cannot distinguish $[t]ID_{RP_j} = [tr]G$ from any $[t']ID_{RP_{j'}} = [t'r']G$. So, the IdP cannot infer the RP's identity or link any pair of PID_{RP}^i and $PID_{RP}^{i'}$, and the IdP-based login tracing is impossible.

RP-based Identity Linkage. We prove UPPRESSO prevents the RP-based identity linkage, based on the elliptic curve decision Diffie-Hellman (ECDDH) assumption. Let $\mathbb{E}$ be an elliptic curve, and G be a point on $\mathbb{E}$ of order n. For any PPT algorithm $\mathcal{D}$, the probability of distinguishing ($[x]G$, $[y]G$, $[xy]G$) and ($[x]G$, $[y]G$, $[z]G$) is negligible, where x, y and z are integers randomly and independently chosen from $\mathbb{Z}_n$. Let $\Pr\{\}$ denote the probability and we define

$$\Pr_1 = \Pr\{\mathcal{D}(G,[x]G,[y]G,[xy]G) = 1\}$$
$$\Pr_2 = \Pr\{\mathcal{D}(G,[x]G,[y]G,[z]G) = 1\}$$

So $\varepsilon_r(k) = |\Pr_1 - \Pr_2|$ becomes negligible as k increases.

In every login instance, the RP holds ID_{RP} and $Acct$, receives t, calculates PID_{RP}, and verifies PID_{RP} and PID_U in the identity token. After filtering out the redundant information (i.e., $PID_{RP} = [t]ID_{RP}$ and $Acct = [t^{-1}]PID_U$), the RP actually receives $(ID_{RP}, t, Acct) = ([r]G, t, [ur]G)$.

The prevention against the RP-based identity linkage is proved by this proposition: when c colluding RPs collect the information of login instances by v users, they still cannot determine whether a login instance to another RP belongs to one of these v users or not. The collected login instances are expressed as $\mathfrak{L} = \begin{Bmatrix} L_{1,1}, & L_{1,2}, & \cdots, & L_{1,c} \\ L_{2,1}, & L_{2,2}, & \cdots, & L_{2,c} \\ \cdots, & \cdots, & \cdots, & \cdots \\ L_{v,1}, & L_{v,2}, & \cdots, & L_{v,c} \end{Bmatrix}$, where $L_{i,j} = (ID_{RP_j}, t_{i,j}, [ID_{U_i}]ID_{RP_j}) = ([r_j]G, t_{i,j}, [u_i r_j]G)$. Given a login instance to another RP $L' = (ID_{RP_{c+1}}, t', [ID_{U'}]ID_{RP_{c+1}}) = ([r_{c+1}]G, t', [u' r_{c+1}]G)$, we define the RP-based identity linkage game $\mathcal{G}_r$: after receiving $\mathfrak{L}$ and L' from a challenger, the adversary outputs the result $s = 1$ if it determines that $u' \in \{u_1, u_2, \cdots, u_v\}$, or $s = 0$ otherwise.

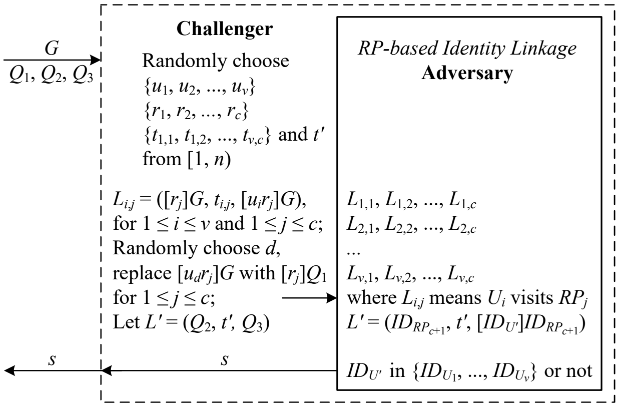
We define the adversary's advantage in $\mathcal{G}_r$ as $\mathbf{Adv}_A$. Then,

$$\mathrm{Pr}'_1 = \mathrm{Pr}\{\mathcal{G}_r(\mathfrak{L}, L' | ID_{U'} \in \{ID_{U_1}, ID_{U_2}, \cdots, ID_{U_v}\}) = 1\}$$
$$\mathrm{Pr}'_2 = \mathrm{Pr}\{\mathcal{G}_r(\mathfrak{L}, L' | ID_{U'} \in \mathbb{Z}_n) = 1\}$$
$$\mathbf{Adv}_A = |\mathrm{Pr}'_1 - \mathrm{Pr}'_2|$$

We design a PPT algorithm $\mathcal{D}_r^*$ based on $\mathcal{G}_r$, shown in Figure 5, to solve the ECDDH problem. The input is in the form of (G, $Q_1 = [x]G$, $Q_2 = [y]G$, $Q_3 = [z]G$). On receiving the input, the challenger of $\mathcal{G}_r$ randomly chooses $\{u_1, u_2, \cdots, u_v\}$, $\{r_1, r_2, \cdots, r_c\}$, $\{t_{1,1}, t_{1,2}, \cdots, t_{v,c}\}$, and t' from $\mathbb{Z}_n$. Then the challenger constructs $\mathfrak{L}$ and L' as below. It first assigns $L_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G)$, and randomly chooses $d \in [1, v]$ to replace $[u_d r_j]G$ with $[r_j]Q_1 = [x r_j]G$ for $1 \leq j \leq c$. So $\mathfrak{L} = \begin{Bmatrix} L_{1,1}, & L_{1,2}, & \cdots, & L_{1,c} \\ L_{2,1}, & L_{2,2}, & \cdots, & L_{2,c} \\ \cdots, & \cdots, & \cdots, & \cdots \\ ([r_1]G, t_{d,1}, [r_1]Q_1), & \cdots, & \cdots, & ([r_c]G, t_{d,c}, [r_c]Q_1) \\ \cdots, & \cdots, & \cdots, & \cdots \\ L_{v,1}, & L_{v,2}, & \cdots, & L_{v,c} \end{Bmatrix}$. Next, it constructs $L' = (Q_2, t', Q_3) = ([y]G, t', [z/y][y]G)$. Finally, $\mathfrak{L}$ and L' are sent to the adversary, and the output s of $\mathcal{G}_r$ is output by the challenger. According to the above construction of $\mathfrak{L}$ and L', x is actually inserted into $\mathfrak{L}$ as u_d and z/y is assigned to u'. So, if $z = xy$, then $z/y = x$ and $ID_{U'} \in \{ID_{U_1}, ID_{U_2}, \cdots, ID_{U_v}\}$; otherwise, $ID_{U'} \in \mathbb{Z}_n$. Thus,

$$\mathrm{Pr}_1 = \mathrm{Pr}\{\mathcal{D}_r^*(G, [x]G, [y]G, [xy]G) = 1\} = \mathrm{Pr}'_1$$
$$= \mathrm{Pr}\{\mathcal{G}_r(\mathfrak{L}, L' | ID_{U'} \in \{ID_{U_1}, ID_{U_2}, \cdots, ID_{U_v}\}) = 1\}$$
$$\mathrm{Pr}_2 = \mathrm{Pr}\{\mathcal{D}_r^*(G, [x]G, [y]G, [z]G) = 1\} = \mathrm{Pr}'_2$$
$$= \mathrm{Pr}\{\mathcal{G}_r(\mathfrak{L}, L' | ID_{U'} \in \mathbb{Z}_n) = 1\}$$
$$\mathbf{Adv}_A = |\mathrm{Pr}'_1 - \mathrm{Pr}'_2| = |\mathrm{Pr}_1 - \mathrm{Pr}_2| = \varepsilon_r(k)$$

The ECDDH assumption means that in $\mathcal{G}_r$ the adversary does not have advantages, i.e., cannot distinguish a user U' chosen from $\{U_1, U_2, \cdots, U_v\}$ or randomly from the user set of UPPRESSO. So the RP-based identity linkage is impossible.



Challenger
G
Q_1, Q_2, Q_3
Randomly choose
$\{u_1, u_2, ..., u_v\}$
$\{r_1, r_2, ..., r_c\}$
$\{t_{1,1}, t_{1,2}, ..., t_{v,c}\}$ and t'
from $[1, n)$
$L_{i,j} = ([r_j]G, t_{i,j}, [u_i r_j]G)$,
for $1 \leq i \leq v$ and $1 \leq j \leq c$;
Randomly choose d,
replace $[u_d r_j]G$ with $[r_j]Q_1$
for $1 \leq j \leq c$;
Let $L' = (Q_2, t', Q_3)$
RP-based Identity Linkage
Adversary
$L_{1,1}, L_{1,2}, ..., L_{1,c}$
$L_{2,1}, L_{2,2}, ..., L_{2,c}$
...
$L_{v,1}, L_{v,2}, ..., L_{v,c}$
where $L_{i,j}$ means U_i visits RP_j
$L' = (ID_{RP_{c+1}}, t', [ID_{U'}]ID_{RP_{c+1}})$
$ID_{U'}$ in $\{ID_{U_1}, ..., ID_{U_v}\}$ or not
s
s

Figure 5: The algorithm based on the RP-based identity linkage, to solve the ECDDH problem.

6 Implementation and Evaluation

We implemented the UPPRESSO prototype,[3] and experimentally compared it with two open-source SSO systems: (*a*) MITREid Connect [65] which supports the PPID-enhanced OIDC protocol to prevent the RP-based identity linkage, and (*b*) SPRESSO [6] preventing the IdP-based login tracing.

6.1 Prototype Implementation

The identity-transformation functions are defined on the NIST P256 elliptic curve. RSA-2048 and SHA-256 are adopted as the signature algorithm and the hash function, respectively.

An IdP is built on top of MITREid Connect [65], an open-source OIDC Java implementation, and only small modifications are needed. We add only 3 lines of Java code to calculate PID_U, and 20 lines to modify the way to send identity tokens. The calculations of ID_{RP} and PID_U are implemented based on Java cryptographic libraries.

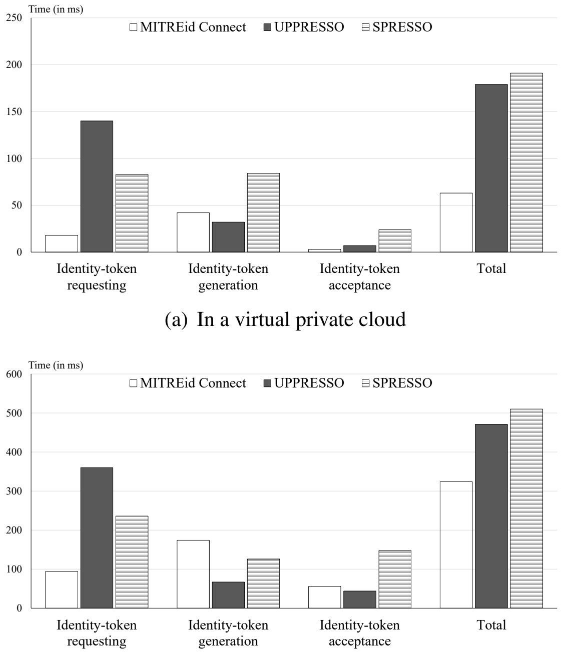
The IdP and RP scripts include about 160 and 140 lines of JavaScript code, respectively. The cryptographic computations such as $Cert_{RP}$ verification and PID_{RP} negotiation, are finished based on jsrsasign [66], an efficient JavaScript cryptographic library.

We provide a Java RP SDK. The SDK provides two functions to encapsulate the protocol steps: one to request identity tokens, and the other to derive the accounts. It is implemented based on the Spring Boot framework with about 500 lines of Java code and cryptographic computations are finished based on the Spring Security library. An RP invokes these functions for the integration, by less than 10 lines of Java code.

6.2 Performance Evaluation

MITREid Connect runs with the implicit flow of OIDC, and the identity tokens in SPRESSO are forwarded by a user to

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



(a) In a virtual private cloud

(b) With a remotely-visiting browser

Figure 6: The time cost of SSO login.

the RP, similarly to the implicit flow. In the identity tokens of SPRESSO, PID_{RP} is the encrypted RP domain, while the symmetric key only known to the RP and the user. They adopt RSA-2048 and SHA-256 in the generation of tokens.

MITREid Connect provides Java implementations of IdP and RP SDK, while SPRESSO implements all entities by JavaScript based on node.js. We implemented the RPs based on Spring Boot for UPPRESSO and MITREid Connect, by integrating the corresponding SDKs. The RPs in three schemes provide the same function, i.e., simply obtain the user's account from verified identity tokens.

The IdP and RP servers are deployed on Alibaba Cloud Elastic Compute Service, each of which runs Window 10 with 8 vCPUs and 32 GB memory. The forwarder of SPRESSO runs Ubuntu 20.04.4 with 16 vCPUs and 16 GB memory, also on Alibaba Cloud. We compare the schemes in two scenarios: (*a*) a user browser, Chrome 104.0.5112.81, runs on another virtual machine with 8 vCPUs and 32 GB memory on Alibaba Cloud, and (*b*) a browser runs locally on a PC with Core i7-8700 CPU and 32 GB memory, to remotely visit the servers. In the cloud scenario, all entities are deployed in the same virtual private cloud and connected to one vSwitch, which minimizes the influence of network delays. In any scenario, the IdP never directly communicates with the RP.

We divide a login flow into three parts: *identity-token requesting* (for UPPRESSO, it includes Steps 1-2 in Figure 3), to construct an identity-token request transmitted to the IdP; *identity-token generation* (Step 3 in Figure 3), for the IdP to generate an identity token, while the user authentication and the authorization of user attributes are excluded; and *identity-token acceptance* (Step 4 in Figure 3), as the RP receives, verifies and parses the identity token.

Figure 6 shows the average time cost of $1,000$ measurements. The overall times of an SSO login instance for MITREid Connect, UPPRESSO, and SPRESSO are (*a*) 63 ms, 179 ms, and 190 ms, respectively, when all entities are deployed on Alibaba Cloud, or (*b*) 312 ms, 471 ms, and 510 ms, respectively, when the user browser runs locally to remotely visit the servers. In the part of identity-token requesting, the RP of MITREid Connect constructs the identity-token request immediately. Compared with MITREid Connect, the main overhead of UPPRESSO is to open a new browser window and download the scripts.[4] The RP in SPRESSO needs to obtain some information on the IdP and encrypt its domain using an ephemeral key, resulting in the additional overhead.

In the identity-token generation, UPPRESSO simply retrieves a token from the IdP. On the contrary, in MITREid Connect when a user retrieves the identity token from the IdP, the token must be carried with a URL following the fragment identifier # instead of ?, due to some security considerations [67]. So the user needs to first download a script from the RP to process this token, which takes the most time. SPRESSO takes a little more time to generate an identity token, as it implements the IdP based on node.js and adopts a JavaScript cryptographic library, while a more efficient Java library is used in the others.

In the identity-token acceptance, MITREid Connect and UPPRESSO spend the comparable amounts of time for sending identity token to the RP and verifying this token. SPRESSO needs the longest time due to the complicated process at the user browser: after receiving identity tokens from the IdP, the browser downloads the JavaScript code from the trusted forwarder server, decrypts the RP endpoint, and finally sends identity tokens to this endpoint.

7 Discussions

Collusive Attack by an IdP and RPs. Privacy-preserving identity federation solutions [11–15, 18] prevent the collusive attacks by the IdP and RPs, at the cost of (*a*) a long-term secret held by a user but verified by RPs and (*b*) user-managed accounts [11] for different RPs. The account may be derived from an RP domain and the secret [12–15, 18], but this design still brings inconveniences. For web applications, a user needs to install a browser extension to handle this long-term secret; and if it is lost or leaked, a user has to notify all RPs to update his accounts which are derived from the secret (if the accounts are not masked by the user's secret, the colluding IdP and RPs

[4] This overhead may be mitigated by implementing a user agent with browser extensions, but users need to install the extension before visiting RPs. We tested such a browser extension while the IdP and RPs are unmodified, and experiments show about 90 ms and 260 ms are saved in a login instance, in a virtual private cloud and by a remotely-visiting user, respectively.

can eventually link them). Meanwhile, UPPRESSO does not protect user privacy against such collusive attacks, but a user is authenticated to *only* an IdP: the identity at the IdP are transformed to the accounts at RPs, unrelated to any user credential (e.g., password, one-time password or smart card).

Scalability. ID_{RP} is generated uniquely in an RP's initial registration, and the capacity is n (i.e., the order of G). For the NIST P256 elliptic curve, n is approximately 2^{256}. Adversaries cannot exhaust PID_{RP}, either. We only ensure PID_{RP} is unique in *unexpired* tokens, the number of which is denoted as σ. The probability that at least two PID_{RP}s are identical among the σ ones, is $1-\prod_{i=0}^{\sigma-1}(1-i/n)$. If the system serves 10^8 requests per second and the validity period is 10 minutes, σ is less than 2^{36} and the PID_{RP}-collision probability is negligible (i.e., less than 2^{-183}) for the P256 curve.

The capacity of accounts at any RP is also n. $\mathbb{E}$ is a finite cyclic group, so $ID_{RP}=[r]G$ is also a generator of order n. Thus, given any RP, a *unique* account is assigned to every user, because $Acct=[ID_U]ID_{RP}=[u]ID_{RP}$.

Besides, stronger elliptic curves accommodate more RPs and users; e.g., n is nearly 2^{384} for the NIST P384 curves.

Alternative Way to Generate ID_{RP} and Bind $Enpt_{RP}$. In the prototype, an RP certificate binds ID_{RP} and $Enpt_{RP}$, verified by the honest IdP script. The RP certificates ensure the target RP has already registered itself at the IdP, which prevents unauthorized RPs from accessing the IdP's services.

This binding may be finished in another way: ID_{RP} is *deterministically* calculated based on the RP's *unambiguous name*. For example, $Hs()$ encodes an RP's domain (or the RP script's origin, e.g., `https://RP.com`) to a point on the elliptic curve $\mathbb{E}$ as ID_{RP}, where hashing to elliptic curves $Hs()$ [68] provides collision resistance and does not reveal the discrete logarithm of the output (i.e., $ID_{RP}=[r]G$ is unique and r is unknown). Then, in Step 2.2 the RP script will send the endpoint but not its RP certificate, and ID_{RP} is calculated by the IdP script. However, if the RP updates its domain, for instance, from `https://RP.com` to `https://theRP.com`, $Acct=[ID_U]ID_{RP}$ will change *inevitably*. In such cases, it needs special operations by each user to migrate his account to the updated RP system. This account migration requires extra operations *explicitly by each user*; otherwise, colluding RPs could actually link a user's accounts across RPs.

Restriction of the RP Script's Origin. When identity tokens are forwarded by the IdP script to the RP script, the receiver is restricted by the `postMessage` targetOrigin mechanism [63], to ensure it will forward the tokens to $Enpt_{RP}$ that is bound in the RP certificate. A targetOrigin is specified as a domain (e.g., `RP.com`) and the parts of protocol and port (if not presented, implicitly 80 for `http` and 443 for `https`), and it requires the RP script's origin *accurately* matches the targetOrigin.

Although the URL path part of $Enpt_{RP}$ is not checked by the targetOrigin mechanism, which assumes *only one* RP runs on a domain, it brings no *extra* risk as below. If two RPs run on one domain but with different endpoints to receive identity tokens (e.g., `https://RP.com/honest/tkn` and `https://RP.com/malicious/tkn`), they cannot be distinguished by `postMessage`. Meanwhile, browsers enforce the same-origin policy in the access control of web resources [69]. So a (malicious) RP could always access the other RP's resources in the browser, e.g., steal cookies by the scripts `vwin = window.open('https://RP.com/honest')` and `vwin.document.cookie`, even if the honest RP restricts that only HTTP requests to specific paths carry its cookies.

Support for the Authorization Code Flow. In the authorization code flow of OIDC [1], an IdP does not directly issue the identity token; instead, an authorization code is sent to the RP, and then the RP uses this code to ask for identity tokens. The identity-transformation functions $\mathcal{F}_{PID_U}$, $\mathcal{F}_{PID_{RP}}$ and $\mathcal{F}_{Acct}$ can be integrated into the authorization code flow: an authorization code is forwarded to the RP script by the IdP script and this code is used to ask for an identity token binding PID_U and PID_{RP}. An authorization code is usually the index to retrieve the identity token from the IdP, and does not disclose any information on the authenticated user.

After receiving the authorization code, the RP uses it and another secret credential which is issued by the IdP during the initial registration [1], to retrieve the identity token from the IdP. In order to protect RP identities from the IdP, privacy-preserving credentials (e.g., ring or group signatures [70, 71]) and anonymous networks (e.g., Tor [72]) need to be adopted for RPs in the final retrieval of identity tokens.

Applicability of Identity Transformations. These identity-transformation functions, i.e., $\mathcal{F}_{PID_{RP}}()$, $\mathcal{F}_{PID_U}()$, and $\mathcal{F}_{Acct}()$, are applicable to various SSO scenarios (e.g., web application, mobile App, and native software), because these functions follow the common model of popular SSO protocols and do not depend on any special implementation or runtime.

8 Conclusion

This paper proposes UPPRESSO, an untraceable and unlinkable privacy-preserving SSO system, to protect a user's online profile at different RPs against a curious IdP and colluding RPs. We convert the identity dilemma of privacy-preserving SSO into an identity-transformation challenge and design three functions satisfying the requirements, where (a) $\mathcal{F}_{PID_{RP}}$ protects the RP's identity from the curious IdP, (b) $\mathcal{F}_{PID_U}$ prevents colluding RPs from linking a user across these RPs, and (c) $\mathcal{F}_{Acct}$ enables the RP to derive an identical account for a user in his multiple logins. These functions are integrated with the widely-adopted OIDC protocol to protect user privacy, without breaking the security guarantees of SSO services. The experimental evaluation of the UPPRESSO prototype shows that it provides efficient SSO services: on average a login instance takes 174 ms when the IdP, the visited RP and a user are deployed in a virtual private cloud, or 421 ms when the user visits remotely.

References

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

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

[3] J. Hughes, S. Cantor, J. Hodges, F. Hirsch, P. Mishra, R. Philpott, and E. Maler, *Profiles for the OASIS security assertion markup language (SAML) V2.0*, OASIS, 2005.

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

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

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

[7] 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, pp. 43–65.

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

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

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

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

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

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

[14] 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.

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

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

[17] M. R. 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.

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

[19] Mozilla Developer Network (MDN), "Mozilla Persona," https://developer.mozilla.org/en-US/docs/Archive/Mozilla/Persona, Accessed August 20, 2019.

[20] M. Simeonovski, F. Bendun, M. R. Asghar, M. Backes, N. Marnau, and P. Druschel, "Oblivion: Mitigating privacy leaks by controlling the discoverability of online information," in *13th International Conference on Applied Cryptography and Network Security (ACNS)*, 2015.

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

[22] 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.

[23] H. Halpin, "NEXTLEAP: Decentralizing identity with privacy for secure messaging," in *12th International Conference on Availability, Reliability and Security (ARES)*, 2017.

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

[25] W. Mostowski and P. Vullers, "Efficient U-Prove implementation for anonymous credentials on smart cards," in *7th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2011.

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

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

[28] 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), Volume 2: SECRYPT*, 2019.

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

[30] D. Boneh, X. Ding, G. Tsudik, and C.-M. Wong, "A method for fast revocation of public key certificates and security capabilities," in *10th USENIX Security Symposium*, 2001.

[31] A. Buldas, A. Kalu, P. Laud, and M. Oruaas, "Server-supported RSA signatures for mobile devices," in *22nd European Symposium on Research in Computer Security (ESORICS)*, 2017.

[32] W. Lueks, B. Hampiholi, G. Alpar, and C. Troncoso, "Tandem: Securing keys by using a central server while preserving privacy," *Privacy Enhancing Technologies*, vol. 2020, no. 3, pp. 327–355, 2020.

[33] K. Elmufti, D. Weerasinghe, M. Rajarajan, and V. Rakocevic, "Anonymous authentication for mobile single sign-on to protect user privacy," *International Journal of Mobile Communications*, vol. 6, no. 6, pp. 760–769, 2008.

[34] 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, pp. 560–567.

[35] 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, pp. 470–490.

[36] T.-F. Lee, "Provably secure anonymous single-sign-on authentication mechanisms using extended Chebyshev Chaotic Maps for distributed computer networks," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1499–1505, 2018.

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

[38] 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)*, 2016, pp. 1204–1215.

[39] 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, pp. 189–202.

[40] 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, pp. 673–688.

[41] 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.

[42] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed single-sign-on web services," in *33rd IEEE Symposium on Security and Privacy (S&P)*, 2012, pp. 365–379.

[43] S.-T. Sun and K. Beznosov, "The devil is in the (implementation) details: An empirical analysis of OAuth SSO systems," in *19th ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 378–390.

[44] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, G. Pellegrino, and A. Sorniotti, "An authentication flaw in browser-based single sign-on protocols: Impact and remediations," *Computers & Security*, vol. 33, pp. 41–58, 2013.

[45] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis," *Journal of Computer Security*, vol. 22, no. 4, pp. 601–657, 2014.

[46] W. Li and C. Mitchell, "Analysing the security of Google's implementation of OpenID Connect," in *13th International Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016, pp. 357–376.

[47] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann, and M. Jensen, "On breaking SAML: Be whoever you want to be," in *21th USENIX Security Symposium*, 2012.

[48] H. Wang, Y. Zhang, J. Li, and D. Gu, "The achilles heel of OAuth: A multi-platform study of OAuth-based authentication," in *32nd Annual Conference on Computer Security Applications (ACSAC)*, 2016, pp. 167–176.

[49] C. Mainka, V. Mladenov, and J. Schwenk, "Do not trust me: Using malicious IdPs for analyzing and attacking single sign-on," in *1st IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, pp. 321–336.

[50] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich, "SoK: Single sign-on security - An evaluation of OpenID Connect," in *2nd IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 251–266.

[51] R. Yang, W. C. Lau, J. Chen, and K. Zhang, "Vetting single sign-on SDK implementations via symbolic reasoning," in *27th USENIX Security Symposium*, 2018.

[52] Y. Zhou and D. Evans, "SSOScan: Automated testing of web applications for single sign-on vulnerabilities," in *23rd USENIX Security Symposium*, 2014, pp. 495–510.

[53] R. Yang, G. Li, W. C. Lau, K. Zhang, and P. Hu, "Model-based security testing: An empirical study on OAuth 2.0 implementations," in *11th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2016, pp. 651–662.

[54] R. Wang, Y. Zhou, S. Chen, S. Qadeer, D. Evans, and Y. Gurevich, "Explicating SDKs: Uncovering assumptions underlying secure authentication and authorization," in *22th USENIX Security Symposium*, 2013.

[55] J. Navas and M. Beltrán, "Understanding and mitigating OpenID Connect threats," *Computer & Security*, vol. 84, pp. 1–16, 2019.

[56] E. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "OAuth demystified for mobile application developers," in *21st ACM Conference on Computer and Communications Security (CCS)*, 2014, pp. 892–903.

[57] H. Wang, Y. Zhang, J. Li, H. Liu, W. Yang, B. Li, and D. Gu, "Vulnerability assessment of OAuth implementations in Android applications," in *31st Annual Computer Security Applications Conference (ACSAC)*, 2015.

[58] R. Yang, W. C. Lau, and S. Shi, "Breaking and fixing mobile App authentication with OAuth2.0-based protocols," in *15th International Conference on Applied Cryptography and Network Security (ACNS)*, 2017.

[59] S. Shi, X. Wang, and W. C. Lau, "MoSSOT: An automated blackbox tester for single sign-on vulnerabilities in mobile applications," in *14th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2019, pp. 269–282.

[60] M. Ghasemisharif, A. Ramesh, S. Checkoway, C. Kanich, and J. Polakis, "O single sign-off, where art thou? An empirical analysis of single sign-on account hijacking and session management on the web," in *27th USENIX Security Symposium*, 2018, pp. 1475–1492.

[61] Y. Cao, Y. Shoshitaishvili, K. Borgolte, C. Krügel, G. Vigna, and Y. Chen, "Protecting web-based single sign-on protocols against relying party impersonation attacks through a dedicated bi-directional authenticated secure channel," in *17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2014.

[62] M. Shehab and F. Mohsen, "Towards enhancing the security of OAuth implementations in smart phones," in *3rd IEEE International Conference on Mobile Services (MS)*, 2014, pp. 39–46.

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

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

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

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

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

[68] 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.

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

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

[71] D. Chaum and E. van Heyst, "Group signatures," in *Advances in Cryptology - EUROCRYPT*, 1991, pp. 257–265.

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