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

Abstract

Single sign-on (SSO) allows a user to maintain only the credential at the identity provider (IdP), instead of one credential for each relying party (RP), to login to numerous RPs. However, SSO introduces extra privacy leakage threats, compared with traditional authentication mechanisms, as (*a*) the IdP could track all RPs which a user is visiting, and (*b*) collusive 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 login activities against both the curious IdP and collusive RPs. We analyze the identity dilemma between the SSO security requirements and these privacy concerns, and convert the SSO privacy problems into an identity transformation challenge. In each login instance of UPPRESSO, an *ephemeral pseudo-identity* (denoted as PID_{RP}) of the RP which the user is attempting to visit, is firstly negotiated between the RP and the user. Then, PID_{RP} is sent to the IdP and designated in the identity token, so the IdP is not aware of the visited RP. Meanwhile, PID_{RP} is also 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* (denoted as *Acct*) of the user, by an ephemeral trapdoor in the negotiation. Given a user, the account at each RP is unique and different from ID_U, so collusive RPs cannot link his identities across multiple RPs. These identity transformations are independent of the authentication between a user and the IdP, so UPPRESSO works compatibly with various types of user credentials (e.g., password, one-time password, multi-factor authentication, and smart card) as widely-used OpenID Connect (OIDC), OAuth 2.0 and SAML.

We build the UPPRESSO prototype system for web applications on top of MITREid Connect, an open-source implementation of OIDC. The extensive evaluation shows that UPPRESSO introduces reasonable overheads and fulfills the requirements of both security and privacy.

1 Introduction

Single sign-on (SSO) systems such as OpenID Connect (OIDC) [1], OAuth 2.0 [2] and SAML [3], are widely deployed in the Internet 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 identity registered at another trusted web service, known as the *identity provider* (IdP). An RP delegates user identification and authentication to the trusted IdP, which issues an *identity token* (e.g., id token in OIDC or identity assertion in SAML) for a user to visit the RP after authenticating this user. Thus, the user keeps only one credential for the IdP, instead of maintaining several credentials for different RPs.

The SSO login flow for web applications works as below. For example, in the widely-used OIDC systems, a user sends a login request to the target RP, and the RP constructs an identity-token request with its identity (denoted as ID_{RP} in this paper) and redirects the request to the IdP. 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 user is allowed to login or not.

The wide adoption of SSO raises concerns on user privacy [4–7], because SSO facilitates curious parties to track which RPs a user visits. In order to issue identity tokens, in each login instance the IdP is always aware of when and to which RP a user attempts to login. As a result, a curious-but-honest 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, the RPs learn users identities from the identity tokens. If the IdP encloses a unique user identity in the tokens for a user to visit different RPs [8, 9], collusive RPs could link these login instances across the RPs, to learn his online profile [4]. We denote this privacy risk as the *RP-based identity linkage*. Existing SSO protocols leak user privacy in different ways, and it is worth noting that *these two kinds of privacy threats result from the designs of SSO protocols* [5], but not

any specific implementations of SSO services.

Solutions have been proposed to protect user privacy in SSO services [4–7]. ++++++++++

In this paper, 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 to comprehensively protect user privacy. In particular, we design three identity-transformation functions in the SSO login flow. In each login instance of UPPRESSO, ID_{RP} is firstly transformed to an ephemeral PID_{RP} cooperatively by the RP and the user. Then, PID_{RP} is sent to the IdP to transform ID_U to ephemeral PID_U, so that 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 (*a*) is unique at each RP and (*b*) keeps permanent across multiple login instances. While providing non-anonymous SSO services, UPPRESSO prevents the IdP from tracking a user's login activities because it receives only PID_{RP} in the identity-token request, and collusive RPs from linking a user's accounts across these RPs because every account is unique.

We summarize our contributions as follows.

- We formalize the SSO privacy problems as an identity-transformation challenge, and then propose a comprehensive solution to protect the users' login activities against the curious IdP and collusive RPs; that is, solve this challenge effectively by designing identity-transformation functions. The UPPRESSO protocol is then designed based on these identity-transformation functions.
- We build the UPPRESSO prototype system 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 follows. 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 detailed designs of UPPRESSO. The properties of security and privacy are analyzed in Section 5. We explain the prototype implementation and experimental evaluations in Section 6, and discuss some extended issues in Section 7. Section 8 concludes this work.

2 Background and Related Works

This section introduces OIDC [1], to describe typical SSO login flows. Then, we discuss existing privacy-preserving SSO solutions and other related works.

2.1 OpenID Connect

OIDC is one of the most popular SSO protocols for web applications. Users and RPs initially register at the IdP with their identities and other necessary information such as user credentials (i.e., passwords or public keys) and RP endpoints (i.e., the URLs to receive identity tokens).

OIDC supports three types of login flows: implicit flow, authorization code flow, and hybrid flow (i.e., a mix-up of the previous two). These flows mainly result from the different steps to request and receive identity tokens, but work with the common security requirements of identity tokens. We introduce the implicit flow and present our design and implementation based on this flow, and Section 7 presents the extensions to support the authorization code flow.

As shown in Figure 1, a user firstly initiates a login request to an RP. Then, the RP constructs an identity-token request with its own identity and the scope of requested user attributes. This identity-token request is redirected to the IdP. After authenticating the user, the IdP issues an identity token which is forwarded by the user to the RP endpoint. The token contains a user identity (or pseudo-identity), the RP identity, a validity period, the requested user attributes, etc. Finally, the RP verifies the received identity token and allows the user to login as the (pseudo-)identity enclosed in this token.

Before issuing the identity token, the IdP obtains the user's authorization to enclose the requested user attributes. The user's operations including authentication, redirection, authorization, and forwarding, are implemented in a software called *user agent* (i.e., a browser for web applications).

As described above, a "pure" SSO protocol does not include any authentication step: the authentication between a user and the IdP is conducted independently of the SSO protocol. There is no authentication step between users and an RP, and an RP only verifies tokens issued by the IdP. This feature of commonly-used SSO protocols [1–3] brings advantages as below. It enables the IdP to authenticate a user by any appropriate means (e.g., password, one-time password, multi-factor authentication, and smart card). Moreover, if a credential was lost or compromised, the user only needs renew it at the IdP. On the other hand, if a user needs to prove some secret to the RP and this secret is valid across multiple login instance (i.e., some authentication steps are actually involved), the user has to notify each RP if this secret was leaked or lost.

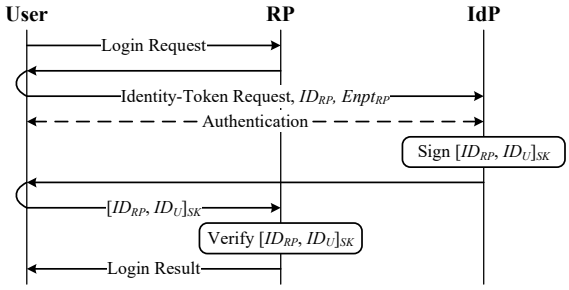

Figure 1: The implicit SSO login flow of OIDC.

2.2 Existing Privacy-Preserving Solutions for SSO

Pairwise pseudonymous identifiers (PPIDs) are recommended by NIST [5] and specified in several SSO protocols [1, 10] to protect user privacy against curious RPs. When issuing an identity token, the IdP encloses a user pseudo-identity (but not the user identity at the IdP) in the token. Given a user, the IdP assigns a unique PPID based on the target RP, so that collusive RPs cannot link the user's different PPIDs across these RPs. PPID-based approaches cannot prevent the IdP-based login tracing, since in the generation of identity tokens the IdP needs to know which RP the user is visiting.

BrowserID [6] and SPRESSO [7] are proposed to defend SSO services against the IdP-based login tracing, but both solutions are instead vulnerable to the threat of RP-based identity linkage. In BrowserID (and its prototypes known as Mozilla Persona [11] and Firefox Accounts [9]), the IdP (called the primary identity authority in BrowserID [6]) issues a special "token" to bind the user identity to an ephemeral public key, so that the user uses the corresponding private key to sign a "subsidiary" identity token to bind his identity with the target RP's identity and then sends both tokens to the RP. When a user logins to different RPs, the RPs could still extract the identical user identities from different pairs of tokens and link these login instances. Meanwhile, in SPRESSO an RP creates a one-time but verifiable pseudo-identity for itself in each login instance. Then, the IdP generates an identity token binding this RP pseudo-identity and the user identity. Similarly, collusive RPs could link a user based on his permanent user identity in these tokens.

PPIDs cannot be directly integrated in either BrowserID or SPRESSO. PPIDs are assigned in identity tokens based on the visited RP [1,5,10], but the IdP receives (*a*) nothing about the RP in BrowserID or (*b*) only an ephemeral pseudo-identity of the RP in SPRESSO.

2.3 Extended Related Works

Certified RP-Verified Credential. A user of EL PASSO [12] keeps a secret on his device. After authenticating the user, the IdP certifies a credential binding the secret, also kept on the user's device. When attempting to login to any RP, the user proves that he is the owner of this credential to the RP without exposing the secret; when such a credential is verified by multiple RPs, user-maintained pseudonyms and anonymous credentials [13] prevent collusive RPs from linking the login instances. UnlimitID [14] presents similar designs, also based on anonymous credentials [13]. PseudoID [15] introduces an independent token service in addition to the IdP, to blindly sign (or certify) a pseudonym credential binding a user's secret. Then, the user proves to the RP that he owns this secret to login. These RP-verified credentials protect user privacy well and two kinds of privacy threats are prevented [12,14,15], but a user has to by himself manage pseudonyms for different RPs. For example, the domain of an RP is used as a factor to locally generate the user's account (or pseudonym) at this RP. It brings some burdens to users, while PPIDs are maintained by the IdP and in UPPRESSO the accounts are determined automatically.

Although EL PASSO calls itself an SSO scheme and the service signing tokens or credentials in EL PASSO, UnlimitID and PseudoID is called the IdP [12,14,15], there are some authentication steps between the user and RPs to verify the user secrets. Therefore, in these systems a user needs to notify each RP if a credential was lost or compromised, for the user is authenticated by the RP with his credentials to prove that he owns the long-term secret. Moreover, a user has to change its account at each RP if its long-term secret was lost, because the account is calculated based on this secret [12, 14]. On the contrary, in the commonly-used SSO systems [1–3] and privacy-preserving schemes such as BrowserID [6], SPRESSO [7] and also UPPRESSO, a user only renews his credential at the IdP if it was compromised, because (*a*) authentication happens only between the user and the IdP and (*b*) an RP verifies only tokens generated by the IdP.

Formal Analysis on SSO Protocols. Fett et al. [16, 17] formally analyzed OAuth 2.0 and OIDC using an expressive Dolev-Yao style model [18], and presented the attacks of 307 redirection and IdP mix-up. SAML-based SSO is also analyzed [19], and the RP identity is found not to be correctly bound in the identity tokens of a variant designed by Google.

Vulnerable SSO Implementation. Vulnerabilities were found in SSO implementations for web applications, exploited to launch attacks of impersonation and identity injection by breaking confidentiality [20–24], integrity [20, 24–28] or RP designation [24, 26–29] of identity tokens. In the SSO services of Google and Facebook, logic flaws of the IdPs and RPs were detected [20]. Integrity of identity tokens was not ensured in SSO systems [20, 25–28] due to software flaws, such as XML signature wrapping [25], incomplete verification by RPs [20,26,28], and IdP spoofing [27,28]. RP designation of identity tokens is broken, due to incorrect binding at the IdP [26, 29] and insufficient verification by RPs [27–29].

Automatic tools, such as SSOScan [30], OAuthTester [31] and S3KVetter [29], are designed to detect the violations of confidentiality, integrity, or RP designation of SSO identity tokens. Wang et al. [32] present a tool to detect the vulnerable applications built on top of authentication/authorization SDKs, due to the implicit assumptions in these SDKs. Besides, Navas et al. [33] discussed the possible attack patterns of the specification and implementations of OIDC.

In mobile SSO scenarios, the IdP App, IdP-provided SDKs (e.g., an encapsulated WebView) or system browsers are responsible for forwarding identity tokens from the IdP App to RP Apps. However, none of them ensures that the identity tokens are sent to the designated RP only [34, 35], because a WebView or the system browser cannot authenticate the RP Apps and the IdP App may be repackaged. The SSO protocols

are modified to work for mobile Apps, but these modifications are not well understood by RP developers [34, 36]. Vulnerabilities were disclosed in lots of Android applications, to break confidentiality [34–37], integrity [34, 36], and RP designation [34, 37] of SSO identity tokens. A software flaw was found in Google Apps [22], 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.

Once a user account at the IdP is compromised, the adversaries will control all his accounts at all RPs. An extension to OIDC, named single sign-off [38], is proposed and then the IdP helps the victim user to revoke all his identity tokens accepted and logout from these RPs.

Anonymous SSO System. Anonymous SSO schemes allow authenticated users to access a service (i.e. RP) protected by the IdP, without revealing their identities. Anonymous SSO was proposed for the global system for mobile (GSM) communications [39], and the notion of anonymous SSO was formalized [40]. Privacy-preserving primitives, such as group signature, zero-knowledge proof, Chebyshev Chaotic Maps and proxy re-verification, etc., were adopted to design anonymous SSO [40–43]. Anonymous SSO schemes work for special applications, but are unapplicable to lots of systems that require user identification for customized services.

3 The Identity-Transformation Framework

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

3.1 Security Requirements of SSO

The primary goal of non-anonymous SSO services is secure user authentication and identification [7], to ensure that a *legitimate* user can always login to an *honest* RP as his permanent identity at this RP, by presenting the *identity tokens* issued by the *honest* IdP.

To achieve this goal, an identity token generated by the IdP is required to specify (*a*) the RP to which the user requests to login (i.e., *RP designation*) and (*b*) exactly the user who is authenticated by the IdP (i.e., *user identification*). Accordingly, an honest RP verifies 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 identity (or pseudo-identity) specified in the accepted tokens; otherwise, the IdP provides only anonymous services.

The SSO login flow implies *confidentiality* and *integrity* of identity tokens. An identity token shall 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 this honest RP. Integrity is necessary, to prevent adversaries from tampering with an identity token. So identity tokens are signed by the IdP and usually transmitted over HTTPS [1–3].

These four security requirements (i.e., RP designation, user identification, confidentiality, and integrity) of SSO identity tokens have been discussed [16, 17, 19], and vulnerabilities breaking any of these properties in SSO systems result in effective attacks [20–31, 34–37, 44, 45]. An adversary might attempt to login to an honest RP as a victim user (i.e., *impersonation*), or allure a victim user to login to an honest RP as the attacker (i.e., *identity injection*). For example, Friendcaster used to accept any received identity token, which violates RP designation [34], so a malicious RP could replay a received identity token to Friendcaster and login as the victim user. The defective IdP of Google ID SSO signs identity tokens where the Email element was not enclosed, while some RPs use a user's Email as his unique username [20]. So this violation of user identity resulted in vulnerable services. Because identity tokens were leaked in different ways [20–24], the eavesdroppers could impersonate the victim users. Some RPs even accept user attributes that are not bound in identity tokens (i.e., a violation of integrity) [20], so that adversaries could insert arbitrary attributes into the identity tokens to impersonate another user at these RPs.

Table 1: The (pseudo-)identities in privacy-preserving SSO.

Notation	Description	Attribute
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

3.2 The Identity Dilemma of Privacy-Preserving SSO

A *completely* privacy-preserving SSO system shall offer the four security properties as mentioned above, while prevent the privacy threats due to the IdP-based login tracing and the RP-based identity linkage. However, to satisfy the requirements of security and privacy at the same time, poses a dilemma in the generation of identity tokens as follows. Table 1 lists the notations used in the following explanation, and the subscript j and/or the superscript i may be omitted sometimes, when there is no ambiguity.

A valid identity token contains the identities (or pseudo-identities) of the authenticated user and the target RP. Since the IdP authenticates users and always knows the user's identity (i.e., ID_U), in order to prevent the IdP-based login tracing, we shall not reveal the target RP's permanent identity (i.e., ID_{RP}) to the IdP in the login flow. So an *ephemeral* pseudo-identity for the RP (i.e., PID_{RP}) shall be used in the

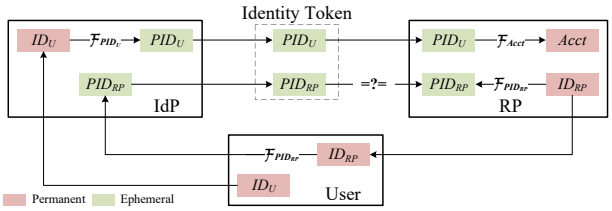

Figure 2: Identity transformations in privacy-preserving SSO.

identity-token request to the IdP: (*a*) to ensure RP designation, PID_{RP} shall 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 shall be independent of each other.[1]

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

Give a user, (*a*) any 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*) these two *ephemeral* pseudo-identities enable the target RP to derive a *permanent* account, i.e., $Acct_j$.

Figure 2 illustrates the relationships among the identities and pseudo-identities in identity tokens. The red and green blocks represent permanent identities and ephemeral pseudo-identities, respectively. The arrows denote the transformations of (pseudo-)identities. The figure describes the *identity dilemma* of privacy-preserving SSO:

> Given an authenticated user and an unknown RP (i.e., permanent ID_U and ephemeral PID_{RP}), the 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*).

We explicitly distinguish a user's identity at the RP, i.e., the account, from (*a*) the user's identity at the IdP and (*b*) the user's pseudo-identity in identity tokens. This conceptualization essentially helps to effectively propose the identity-transformation framework.

[1] While the target RP is kept unknown to the IdP, the IdP shall not link multiple login instances which attempt to visit this RP.

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

3.3 Identity Transformation

The privacy-protection problem of SSO is converted into an identity-transformation challenge, to design three *identity-transformation functions* as follows.

- $\mathcal{F}_{PID_{RP}}(ID_{RP}) = PID_{RP}$, calculated by the user and/or the RP. From the IdP's view, $\mathcal{F}_{PID_{RP}}()$ is a one-way function and the calculated PID_{RP} appears a random variable.
- $\mathcal{F}_{PID_U}(ID_U, PID_{RP}) = PID_U$, calculated by the IdP. From the RP's view, $\mathcal{F}_{PID_U}()$ is a one-way function and the calculated PID_U appears a random variable.
- $\mathcal{F}_{Acct}(PID_U, PID_{RP}) = Acct$, calculated by the RP. Given ID_U and ID_{RP}, *Acct* keeps permanent and unique to other accounts at this RP; i.e., 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 the identity-transformation functions, a user firstly negotiates an ephemeral PID_{RP} with the target RP. Then, an identity-token request with PID_{RP} is sent by the user to the IdP. After authenticating the user as ID_U, the IdP calculates an ephemeral PID_U based on ID_U and the received PID_{RP}, and issues an identity token binding PID_U and PID_{RP}. This token is forwarded by the user to the RP. Finally, after verifying the designated RP pseudo-identity in the token, the RP calculates *Acct* and allows the token holder to login as *Acct*.

The identity-transformation functions satisfy the privacy requirements, while the authentication between users and the IdP is still kept independent of the steps involving identity tokens. After the identity-transformation functions are integrated in an SSO system, the steps dealing with identity tokens work compatibly with any type of user credentials (e.g., password, one-time password, multi-factor authentication, and smart card), as those in the commonly-used SSO protocols [1–3].

4 The Designs of UPPRESSO

This section presents the threat model and assumptions of UPPRESSO. Then, we design three identity-transformation functions satisfying the requirements, and present the detailed protocols for web applications. The compatibility with OIDC is also discussed, which will help the adoption and deployment of UPPRESSO.

4.1 Threat Model

The IdP is curious-but-honest, while some users and RPs could be compromised. Malicious users and RPs behave arbitrarily and might collude with each other, attempting to break the security and privacy guarantees for benign users.

Curious-but-honest IdP. The IdP strictly follows the protocol, while being interested in learning user privacy. 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. We assume the IdP is well-protected. For example, 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 in the protocols. If the IdP could collude with some RPs, a user would finish login instances completely with collusive entities and it is rather impossible to prevent the IdP-based login tracing across these RPs.

Malicious Users. We assume the adversary could control a set of users, by stealing users' credentials or registering Sybil accounts 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 any SSO login instances.

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

Collusive Users and RPs. Malicious users and RPs might collude with each other. For example, an adversary might try to pretend to be an RP and allure victim users to forward an identity token to him, to impersonate the victim user and login to some honest RP.

4.2 Assumptions

We assume that a user never authorizes the IdP to enclose any *distinctive attributes* in identity tokens, where distinctive attributes are identifiable information such as telephone number, driver license, Email, etc. The user does not configure distinctive attributes at any RP, either. Therefore, the privacy leakage due to user re-identification by distinctive attributes is out of the scope of our work.

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

We focus on the privacy attacks introduced by the design of SSO protocols, but not network attacks such as the traffic analysis that trace a user's login activities from network packets. Such attacks shall be prevented by other exiting defenses.

Table 2: The notations in the UPPRESSO protocols.

Notation	Description
$\mathbb{E}$	An elliptic curve over a finite field $\mathbb{F}_q$, where the ECDLP is computationally impossible.
G, n	A base point (or generator) of $\mathbb{E}$, where the order of G is a prime number n.
ID_U	$ID_U = u$, $1 < u < n$; the user's unique identity at the IdP.
ID_{RP_j}	$ID_{RP} = [r]G$, $1 < r < n$; the j-th RP's unique identity.
t	The user-generated random integer number in a login instance, $1 < t < n$.
$PID^i_{RP_j}$	$PID_{RP} = [t]ID_{RP} = [tr]G$; 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.
$Cert_{RP_j}$	The RP certificate signed by the IdP, binding ID_{RP_j} and $Enpt_{RP_j}$.

4.3 Identity-Transformation Functions

We design three identity-transformation functions, $\mathcal{F}_{PID_{RP}}$, $\mathcal{F}_{PID_U}$ and $\mathcal{F}_{Acct}$, over an elliptic curve $\mathbb{E}$, where G is a base point (or generator) of this elliptic curve and the order of G is a big prime number denoted as n. Table 2 lists the notations, and the subscript j and/or the superscript i may be omitted in the case of no ambiguity.

ID_U is a unique integer satisfying $1 < ID_U < n$, and ID_{RP} is a unique point on $\mathbb{E}$. When a user is registering, a unique random number u $(1 < u < n)$ is generated and $ID_U = u$ is assigned to this user; when an RP is initially registering, a unique random number r $(1 < r < n)$ is generated by the IdP and $ID_{RP} = [r]G$ is assigned to this RP. Here, $[r]G$ is the addition of G on the curve r times.

***ID_{RP}-PID_{RP}* Transformation.** In each login instance, the user selects a random number t $(1 < t < n)$ as the trapdoor and calculates PID_{RP} as below.

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

***ID_U-PID_U* Transformation.** On receiving an identity-token request with ID_U and PID_{RP}, the IdP calculates PID_U.

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

***PID_U-Acct* Transformation.** In the negotiation of PID_{RP}, the user sends the trapdoor t to the target RP. So the RP also calculates PID_{RP} to verify the RP pseudo-identity designated in identity tokens. After verifying an identity token binding PID_U and PID_{RP}, the RP calculates $Acct$ as below.

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

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

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

The RP derives the identical permanent account from different identity tokens in multiple login instances, with the help of t from the user. Given a user, the accounts at different RPs are inherently unique. Moreover, (a) due to the elliptic curve discrete logarithm problem (ECDLP), it is computationally infeasible for the RP to derive ID_U from either PID_U or $Acct$; and (b) because t is a random number kept secret to the IdP, it is impossible for the IdP to derive ID_{RP} from PID_{RP}.

4.4 The Designs for Web Applications

We further propose the designs specific for web applications, and these designs enable UPPRESSO to provide SSO services for users with commercial-off-the-shelf (COTS) browsers. More efficient but less portable implementations with browser extensions are discussed in Section 7.

First of all, in the SSO login flow, the user has to deal with RP endpoints (i.e., the URLs to receive identity tokens) by himself. In existing SSO protocols, an RP initially registers its endpoint at the IdP, and then in each login instance, the IdP will set this endpoint in the identity-token response. This instructs the user browser to forward it correctly; otherwise, confidentiality of identity tokens might be broken.

In UPPRESSO the IdP is not aware of the visited RP and cannot set the endpoints, so *RP certificates* are designed to instruct the user agents (or browsers) about RP endpoints. An RP certificate is a document signed by the IdP during the RP initial registration, binding the RP's identity and its endpoint. This attribute certificate is sent by the RP in the login flow, so a user forwards identity tokens to the verified endpoint.

Secondly, browser scripts are needed to implement the user functions, including the generation of t and PID_{RP} and the dealing with RP certificates and endpoints, for they are not standard functions of a browser. Two scripts, one downloaded from the IdP and the other from the target RP, work together (with the standard browser functions) as the user agent in UPPRESSO. The RP script maintains the communications with the RP, and it does not communicates directly with the IdP because an HTTP request launched by the RP script will automatically carry an HTTP `Referer` header, which discloses the RP's domain. The IdP script downloaded from the IdP, is responsible for the communications with the IdP, and two scripts communicates with each other within the user browser through the standard `postMessage` HTML5 API.

The IdP's public key is downloaded in the IdP script to verify RP certificates. So the user agent does not configure anything locally, as it does in widely-adopted SSO systems.

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, while PK is publicly known.

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

1. The RP sends a registration request to the IdP, including the endpoint to receive identity tokens, and other optional information.
2. The IdP generates a unique random number r, calculates $ID_{RP} = [r]G$, and assigns ID_{RP} to this RP. The IdP then signs $Cert_{RP} = [ID_{RP}, Enpt_{RP}, *]_{SK}$, where $[\cdot]_{SK}$ means a message signed using SK and $*$ denotes supplementary information such as the RP's common name and Email, and returns $Cert_{RP}$ to the RP.
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 user identity ID_U and the corresponding credential. This is similar to the ones in existing SSO systems.

SSO Login. A login instance is typically launched through a browser, when a user attempts to login to an RP. It 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 operations by the IdP are linked by a vertical line, so are the RP's operations. Two vertical lines split the user's operations into two groups (i.e., in two browser windows), one of which is to communicate with the IdP, and the other 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 windows) within the user browser.

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

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 in the browser chooses a random number t ($1 < t < n$) and sends it to the RP script through `postMessage`. Then, the RP script sends t to the RP.

2.2 On receiving t, the RP verifies $1 < t < n$ and calculates PID_{RP}. The RP replies with $Cert_{RP}$, which is then transmitted from 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 $PID_{RP} = [t]ID_{RP}$.

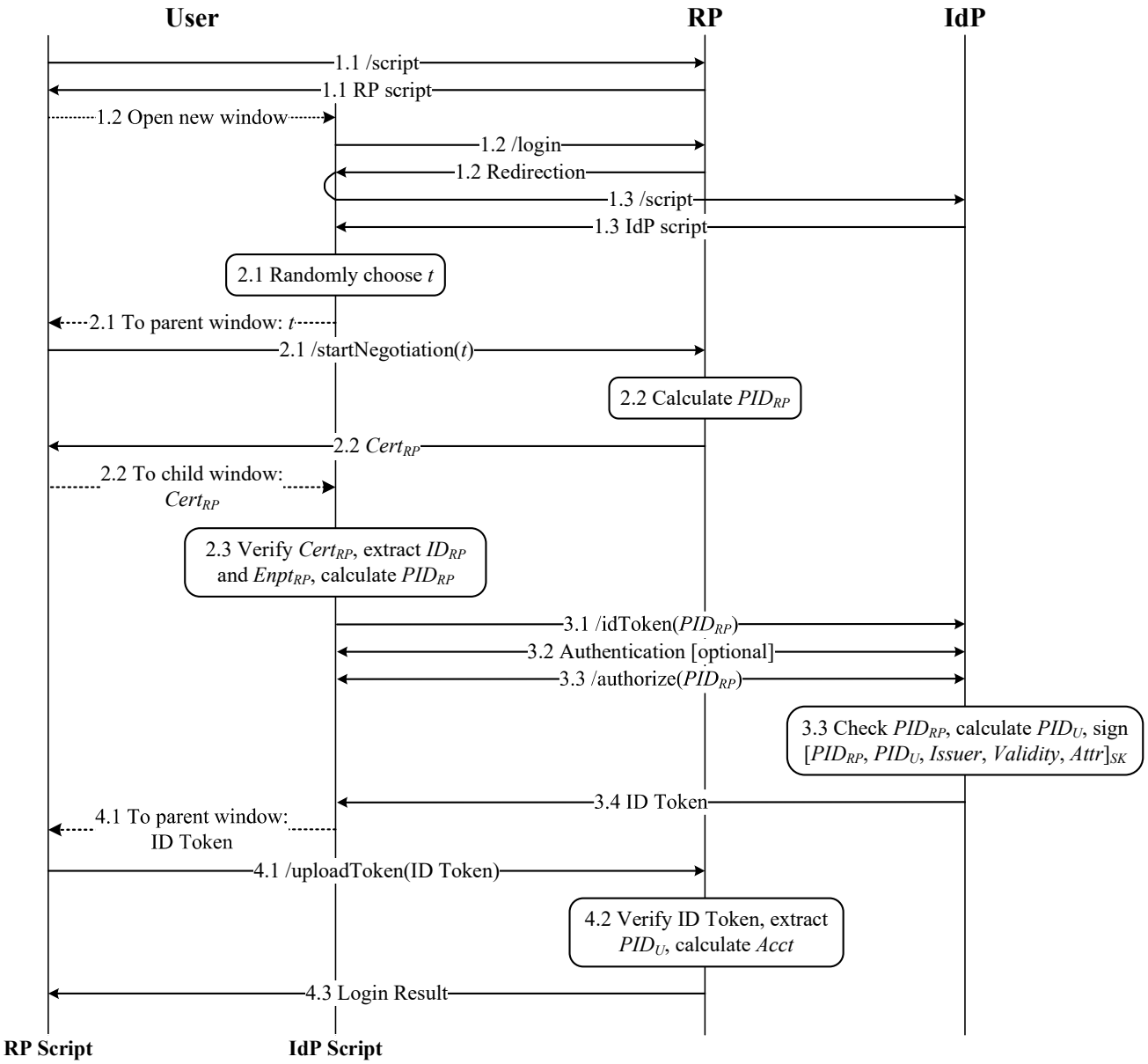

Figure 3: The SSO login flow of UPPRESSO.

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}.

3.2 On receiving the request, the IdP authenticates the user if he has not been authenticated yet.

3.3 After obtaining the user's authorization to enclose the requested attributes, the IdP checks whether the received PID_{RP} is valid, and calculates $PID_U = [ID_U]PID_{RP}$ for the authenticated user. The IdP then signs an identity token $[PID_{RP}, PID_U, Issuer, Validity, Attr]_{SK}$, 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.

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

4.1 The IdP script forwards this token to the RP script, which then 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 is consistent with the one negotiated in

Step 2.2. Then, the RP extracts PID_U, and calculates $Acct = [t^{-1}]PID_U$.

4.3 The RP returns the login result, and allows the user to login as $Acct$.

If any verification or check fails in some step, the flow will be halted immediately. For example, the user halts the flow when receiving 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 the token does not match the negotiated one.

4.6 Compatibility with OIDC

Among the four steps of the SSO login flow in UPPRESSO, the script downloading prepares the user agent. The user agent of SSO is responsible for the communications between the IdP and the RP, which are implemented by browser redirections in OIDC. On the other hand, in UPPRESSO the script hides the $Enpt_{RP}$ from IdP, and then it forwards the identity token to $Enpt_{RP}$ which is extracted from the RP certificate, that to protect the identity token from being sent to adversaries.

In the step of RP identity transformation, most operations are conducted by the user agent, while the RP only receives t to calculate PID_{RP} and sends $Cert_{RP}$.

The operations of identity-token generation and $Acct$ calculation, are actually identical to those in the implicit login flow of OIDC [1], because (a) the calculation of PID_U is viewed as a method to generate PPIDs by the IdP and (b) the calculation of $Acct$ is viewed as a mapping from the user identity in tokens to a local account at the RP.

5 The Analysis of Security and Privacy

In this section, we presents the analysis that UPPRESSO achieves the required properties of security and privacy.

5.1 Security

UPPRESSO satisfies the four security requirements of identity tokens in SSO services, as listed in Section 3.1.

User Identification. An honest RP always derives an identical permanent 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 honest 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. There is no event leaking the identity tokens 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 that adversaries cannot eavesdrop the identity tokens. Meanwhile, the honest IdP sends the identity token only to the authenticated user, and this user forwards it to the RP through $Enpt_{RP}$. The binding of $Enpt_{RP}$ and ID_{RP} is ensured by the signed RP certificate, so only the designated target RP receives this token.

Integrity. The identity token binds ID_U and ID_{RP} implicitly, and any breaking will result in some failed check or verification in the login flow. The integrity is ensured by the IdP's signatures: (a) the identity token binding PID_U and PID_{RP}, is signed by the IdP, and (b) the relationship between PID_{RP} and t is guaranteed by the proof of RP designation. According to RP designation, there would not be the t and t' happening to satisfy that $PID_{RP} = [t]ID_{RP_j} = [tr]G = [t'r']G = [t']ID_{RP_{j'}}$. Thus, ID_U and ID_{RP} are actually bound by the IdP's signatures, due to the one-to-one mapping between (a) the pair of ID_U and ID_{RP} and (b) the triad of PID_U, PID_{RP}, and t.

We also formally analyze the three security properties of UPPRESSO, based on an Dolev-Yao style model of the web infrastructure [7], which has been used in the formal analysis of SSO protocols such as OAuth 2.0 [16] and OIDC [17]. The Dolev-Yao style model abstracts the entities in a web system, such as browsers and web servers, as *atomic processes*, which communicate with each other through *events*. It also defines *script processes* to formulate client-side scripts, i.e., JavaScript code.

The UPPRESSO system contains 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. Here, we consider all RP processes and browser processes are honest, while model an RP or a browser controlled by an adversary as atomic attacker processes. It also contains script_rp and script_idp, where script_rp and script_idp are honest scripts downloaded from an RP process and the IdP process, respectively.

After formulating UPPRESSO by the Dolev-Yao style model, we trace the identity token, starting when it is generated and ending when accepted by the RP, to ensure that an identity token is not leaked or tampered with. We locate the generation of an identity token in UPPRESSO, and trace back to the places where PID_U, PID_{RP} and other parameters enclosed in this token are generated and transmitted, to ensure that no adversary is able to retrieve or manipulate them. The tracing of identity tokens also confirm no adversary retrieves the token.

RP Designation.

5.2 Privacy

UPPRESSO effectively prevents the attacks 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 number randomly chosen from $(1,n)$ by the user and kept secret to the IdP and (*b*) $ID_{RP} = [r]G$ and G is the base point (or generator) of $\mathbb{E}$, the IdP has to view PID_{RP} as randomly and independently chosen from $\mathbb{E}$, and cannot distinguish $[t]ID_{RP_j} = [tr]G$ from any $[t']ID_{RP_{j'}} = [t'r']G$. So, the IdP cannot derive the RP's identity or link any pair of PID^{i}_{RP} and $PID^{i'}_{RP}$, and then the IdP-based login tracing is impossible.

RP-based Identity Linkage. We prove that UPPRESSO prevents the RP-based identity linkage, based on the elliptic curve decision Diffie-Hellman (ECDDH) assumption [46].

Let $\mathbb{E}$ be an elliptic curve over a finite field $\mathbb{F}_q$, and P be a point on $\mathbb{E}$ of order n. For any probabilistic polynomial time (PPT) algorithm $\mathcal{D}$, ($[x]P$, $[y]P$, $[xy]P$) and ($[x]P$, $[y]P$, $[z]P$) are computationally indistinguishable, where x, y and z are integer numbers randomly and independently chosen from $(1,n)$. Let $Pr\{\}$ denote the probability and we define

$$\begin{aligned} Pr_1 =& Pr\{\mathcal{D}(P,[x]P,[y]P,[xy]P) = 1\} \\ Pr_2 =& Pr\{\mathcal{D}(P,[x]P,[y]P,[z]P) = 1\} \\ \varepsilon(k) =& Pr_1 - Pr_2 \end{aligned}$$

Then, $\varepsilon(k)$ becomes negligible with the security parameter k.

In the login flow, an RP holds ID_{RP} and $Acct$, receives t, calculates PID_{RP}, and verifies the signed message (i.e., 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 $\{ID_{RP}, t, PID_U\}$ in each SSO login instance, where $PID_U = [ID_U][t]ID_{RP}$.

In the RP-based identity linkage, two RPs bring two triads received in SSO login instances, $\{ID_{RP_j}, t_j, [ID_U][t_j]ID_{RP_j}\}$ and $\{ID_{RP_{j'}}, t_{j'}, [ID_{U'}][t_{j'}]ID_{RP_{j'}}\}$. We describe the attack as the following game $\mathcal{G}$ between an adversary and a challenger: the adversary receives $\{ID_{RP_j}, t_j, [ID_U][t_j]ID_{RP_j}, ID_{RP_{j'}}, t_{j'}, [ID_{U'}][t_{j'}]ID_{RP_{j'}}\}$ from the challenger, and outputs the result s. The result is 1, when the adversary guesses that $ID_U = ID_{U'}$; otherwise, the adversary thinks they are different users (i.e., $ID_U \neq ID_{U'}$) and $s = 0$.

We define Pr_c as the probability that the adversary outputs $s = 1$ when $ID_U = ID_{U'}$ (i.e., a *correct* identity linkage), and $Pr_{\bar{c}}$ as the probability that $s = 1$ but $ID_U \neq ID_{U'}$ (i.e., an *incorrect* result). The successful RP-based identity linkage means the adversary has non-negligible advantages in $\mathcal{G}$.

We design a PPT algorithm $\mathcal{D}^*$ based on $\mathcal{G}$, shown in Figure 4. The input of $\mathcal{D}^*$ is in the form of $\{Q_1, Q_2, Q_3, Q_4\}$, and each Q_i is a point on $\mathbb{E}$. On receiving the input, the challenger of $\mathcal{G}$ randomly chooses t_j and $t_{j'}$ in $(1,n)$, and sends $\{Q_1, t_j, [t_j]Q_3, Q_2, t_{j'}, [t_{j'}]Q_4\}$ to the adversary in $\mathcal{G}$. Finally, it directly outputs s from $\mathcal{G}$ as the result of $\mathcal{D}^*$.

Let (P, $[x]P$, $[y]P$, $[xy]P$) and (P, $[x]P$, $[y]P$, $[z]P$) be two

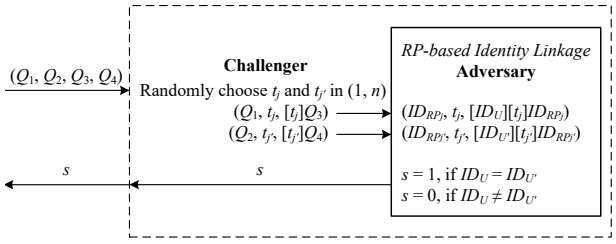

Figure 4: The algorithm based on the RP-based identity linkage, to solve the ECDDH problem.

inputs of $\mathcal{D}^*$. Thus, we obtain

$$\begin{aligned} & Pr\{\mathcal{D}^*(P,[x]P,[y]P,[xy]P) = 1\} \\ =& Pr\{\mathcal{G}(P,t_j,[t_j][y]P,[x]P,t_{j'},[t_{j'}][xy]P) = 1\} \\ =& Pr\{\mathcal{G}(P,t_j,[y][t_j]P,[x]P,t_{j'},[y][t_{j'}][x]P) = 1\} = Pr_c \end{aligned} \tag{4}$$

$$\begin{aligned} & Pr\{\mathcal{D}^*(P,[x]P,[y]P,[z]P) = 1\} \\ =& Pr\{\mathcal{G}(P,t_j,[t_j][y]P,[x]P,t_{j'},[t_{j'}][z/x][x]P) = 1\} \\ =& Pr\{\mathcal{G}(P,t_j,[y][t_j]P,[x]P,t_{j'},[z/x][t_{j'}][x]P) = 1\} = Pr_{\bar{c}} \end{aligned} \tag{5}$$

Equation 4 is equal to Pr_c because it represents the correct case of $ID_U = ID_{U'} = y$, while Equation 5 is $Pr_{\bar{c}}$ for it represents the incorrect case of $ID_U = y$ but $ID_{U'} = z/x \bmod n$.

The adversary has non-negligible advantages in $\mathcal{G}$ means $Pr_c - Pr_{\bar{c}}$ is non-negligible, and then $\mathcal{D}^*$ significantly distinguishes (P, $[x]P$, $[y]P$, $[xy]P$) from (P, $[x]P$, $[y]P$, $[z]P$), which violates the ECDDH assumption. So the adversary has no advantages in the game, and the RP-based identity linkage is computationally impossible in UPPRESSO.

6 Implementation and Evaluation

We have implemented the UPPRESSO prototype system, and experimentally evaluated its performance by comparing it with two open-source systems: (*a*) MITREid Connect [47] which implements the PPID-enhanced OIDC protocol and prevents the RP-based identity linkage only, and (*b*) SPRESSO [7] which only prevents the IdP-based login tracing.

6.1 Prototype

First of all, three identity-transformation functions are defined over the NIST P256 elliptic curve. RSA-2048 and SHA-256 are adopted as the signature algorithm and the hash function, respectively.

The IdP is built on top of MITREid Connect [47], an open-source OIDC Java implementation, and only small modifications are needed as follows. We add only 3 lines of Java code to calculate PID_U, and about 20 lines to modify the way to send identity tokens. The calculations of ID_{RP}, PID_U,

and RSA signature are implemented based on built-in Java cryptographic libraries.

We implemented the user-side functions by scripts downloaded from the IdP and RPs, containing about 200 and 150 lines of JavaScript code, respectively. The cryptographic computations, e.g., $Cert_{RP}$ verification and PID_{RP} negotiation, are implemented based on jsrsasign [48], an efficient JavaScript cryptographic library.

We provide a Java SDK for RPs in UPPRESSO. The SDK provides two functions to encapsulate the protocol steps: one to request identity tokens, and the other to derive the accounts from identity tokens. The SDK is implemented based on the Spring Boot framework with about 1,000 lines of Java code and cryptographic computations are implemented based on the Spring Security library. An RP invokes these two functions for the integration, by less than 10 lines of Java code.

6.2 Performance Evaluation

Three machines connected in an isolated 1Gbps LAN, build the experimental SSO environment. The CPUs are Intel Core i7-4770 3.4 GHz for the IdP, Intel Core i7-4770S 3.1 GHz for the RP, and Intel Core i5-4210H 2.9 GHz for users. Each machine is configured with 8 GB RAM and installs Windows 10 as the operating system. The user agent is Chrome v75.0.3770.100.

We compared UPPRESSO with MITREid Connect and SPRESSO. MITREid Connect runs with the standard implicit login flow of OIDC, while the identity tokens in SPRESSO are also forwarded by a user to the RP, similarly to the implicit login flow to some extent. In the identity tokens of SPRESSO, PID_{RP} is the encrypted RP domain, while the one-time symmetric key only known by the RP and the user. They also configure RSA-2048 and SHA-256 in the generation of identity tokens.

MITREid Connect provides open-source 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 extract the user's account from verified identity tokens.

We measured the time for a user to login to an RP and calculated the average of 1,000 measurements. We divide a login flow into three parts: *Preparation and identity-token requesting* (for UPPRESSO, it includes Steps 1-2 in Figure 3), as the RP constructs an identity-token request transmitted to the IdP, cooperatively with the user sometimes; *Identity-token generation* (Step 3 in Figure 3), when the IdP generates an identity token (and the user authentication is not included); and *Identity-token acceptance* (Step 4 in Figure 3), as the RP receives, verifies and parses the identity token.

The results are shown in Figure 5. The overall times of SSO login are 113 ms, 310 ms, and 308 ms for MITREid Connect, UPPRESSO, and SPRESSO, respectively.

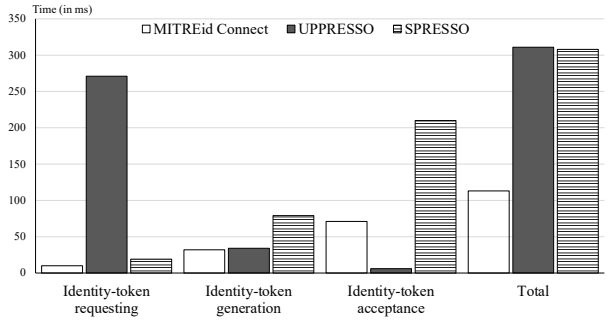

Figure 5: The time cost of SSO login.

In the preparation and identity-token requesting, MITREid Connect only needs 10 ms but UPPRESSO requires 271 ms. The main overhead in UPPRESSO is to open the new browser window and download the scripts, which needs about 104 ms. This overhead can be mitigated by silently conducting these operations when the user visits the RP website, or by the implementation with browser extensions. SPRESSO needs 19 ms in the preparation and identity-token requesting, a little more than MITREid Connect, for an RP to obtain some information on the IdP and encrypt its domain using an ephemeral symmetric key.

In the identity-token generation, UPPRESSO needs 34 ms. Compared with MITREid Connect, it needs 2 more ms to calculate PID_U. SPRESSO requires 71 ms to generate an identity token, as it implements the IdP based on node.js and therefore adopts a JavaScript cryptographic library, while a more efficient Java library is used in others.

In the identity-token acceptance, UPPRESSO only needs about 6 ms for the scripts to send an identity token to the RP, which verifies it and calculates $Acct$. It takes 71 ms for MITREid Connect to accept this identity token: when the token is redirected to the RP, it must be carried within an URL following the fragment identifier `#` instead of `?`, due to some security considerations [49], so the identity token has to be sent to the RP by JavaScript functions (but not HTTP requests) and most time is spent to download the script from the RP. SPRESSO needs the longest time (210 ms) due to the complicated process at the user's browser: after receiving identity tokens from the IdP, the browser downloads the JavaScript program from a trusted forwarder, decrypts the RP endpoint, and finally sends identity tokens to this endpoint.

7 Discussions

Applicability. Three 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 even native software), because these designs do not depend on any special implementation or runtime environment. Although the proto-

type system runs for web applications, it is feasible to apply the identity-transformation functions to other SSO scenarios to protect user privacy. Moreover, because the authentication steps are independent of the UPPRESSO protocol, the IdP can choose any appropriate mechanism to authenticate users.

Compatibility with the Authorization Code Flow. In the authorization code flow of OIDC [1], the IdP does not directly issue the identity token; instead, an access token is forwarded to the RP, and then the RP uses this access token to ask for identity tokens from the IdP. The identity-transformation functions $\mathcal{F}_{PID_U}$, $\mathcal{F}_{PID_{RP}}$ and $\mathcal{F}_{Account}$ can be integrated into the authorization code flow to generate and verify two tokens, so that the privacy threats are still prevented in the generation and verification of tokens. Accordingly, only PID_{RP} but not ID_{RP} is enclosed in the access tokens (as well as the identity tokens). However, as the RP receives the identity token directly from the IdP in this flow, it allows the IdP to obtain the RP's network information (e.g., IP address). Therefore, to prevent this leakage in the authorization code flow, UPPRESSO needs to integrate anonymous networks (e.g., Tor) for the RP to ask for identity tokens.

Implementation with Browser Extensions. To improve the portability of user agents, the user functions of UPPRESSO are implemented by browser scripts in the prototype system. However, these functions can be implemented with browser extensions, which will result in better performance. In this case, a user downloads and installs the browser extension, before he visits the RPs. Experiments show that at least 102 ms will be saved for each login instance (i.e., about 208 ms in total), compared with the version implemented with only browser scripts.

Collusive Attacks by the IdP and RPs. If the IdP is kept curious-but-honest and shares messages in the login flow (i.e., ID_U, PID_{RP}, and PID_U) with some collusive RPs, UPPRESSO still provides secure SSO services, provided that the signed identity tokens are sent to the authenticated users only. Moreover, in this case, a user's login activities at the other RPs, are still protected from the IdP and these collusive RPs, because a triad of t, PID_U and PID_{RP} is ephemeral and independent of each other.

8 Conclusion

This paper proposes UPPRESSO, an untraceable and unlinkable privacy-preserving single sign-on system, to protect a user's login activities at different RPs against both the curious IdP and collusive RPs. We convert the identity dilemma in privacy-preserving SSO services 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 collusive RPs from linking a user based on his identities at these RPs, and (*c*) $\mathcal{F}_{Acct}$ allows the RP to derive an identical account for a user in his multiple login instances. These functions can be integrated with existing SSO protocols, such as OIDC, to enhance the protections of user privacy, without breaking the security guarantees of SSO services. The experimental evaluation of the UPPRESSO prototype demonstrates that it provides efficient SSO services, where a login instance takes only 310 ms on average.

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] 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.

[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, 2017.

[6] 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.

[7] 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.

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

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

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

[11] Mozilla Developer Network (MDN), "Persona," https://developer.mozilla.org/en-US/docs/Archive/Mozilla/Persona.

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

[14] 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.

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

[16] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of OAuth 2.0," in *2016 ACM Conference on Computer and Communications Security (CCS)*, 2016, pp. 1204–1215.

[17] 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.

[18] 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.

[19] 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, pp. 1–10.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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, pp. 397–412.

[26] 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.

[27] 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.

[28] 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.

[29] 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, pp. 1459–1474.

[30] 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.

[31] 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.

[32] 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, pp. 399–314.

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

[34] E. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "OAuth demystified for mobile application

developers," in *2014 ACM Conference on Computer and Communications Security (CCS)*, 2014, pp. 892–903.

[35] 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, pp. 61–70.

[36] 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, pp. 313–335.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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, pp. 276–298.

[45] 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.

[46] S. Goldwasser and Y. Tauman Kalai, "Cryptographic assumptions: A position paper," in *13th International Conference on Theory of Cryptography (TCC)*, 2016, vol. 9562, pp. 505–522.

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

[48] "jsrsasign," https://kjur.github.io/jsrsasign/, Accessed August 20, 2019.

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