ARPSSO: An OIDC-Compatible Privacy-Preserving SSO Scheme based on RP Anonymization

Junlin He[1], Lingguang Lei[2,3], Yuewu Wang[4(✉)], Pingjian Wang[2,3], and Jiwu Jing[4]

[1] School of Computer Science and Technology,
University of Chinese Academy of Sciences, Beijing, 100043, China
[2] Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, 100089, China
[3] School of Cyber Security,
University of Chinese Academy of Sciences, Beijing, 100043, China
[4] School of Cryptology,
University of Chinese Academy of Sciences, Beijing, 100043, China
{wangyuewu}@ucas.ac.cn

Abstract. OpenID Connect (OIDC) is one of the most widely used single sign-on (SSO) protocols today. However, like all other popular SSO protocols, it does not take user privacy into account, which means that Identity Providers (IdPs) and colluding Relying Parties (RPs) may carve a user's behavior through protocol interaction easily. Many studies have attempted to address these privacy issues, but none of them support the most commonly adopted OIDC code flow very well because of their insufficient consideration of the IdP-to-RP authentication, which is the key mechanism to enhance the security of OIDC. In this paper, we propose a privacy-preserving SSO scheme named ARPSSO that may be adapted to OIDC code flow very well. We first realize RP anonymous authentication by issuing anonymous credentials to the RP, thus preventing the user's login behavior from being tracked by the IdP. We also design a two-party secure computation scheme based on anonymous credentials, which implements user identity mix-up and thus prevents the user identity from being linked by colluding RPs. In addition, we introduce a user-imperceptible trusted in-browser data forwarding mechanism to ensure that the entire SSO process is transparent to the user, allowing users to seamlessly use conventional authentication mechanisms and standard browsers for a great user experience. The security analysis shows that ARPSSO achieves the privacy goals without compromising the intrinsic security properties of the OIDC protocol. Performance evaluation on the prototype implementation shows that ARPSSO may work with acceptable overhead compared to original SSO systems.

Keywords: Single Sign-On · OpenID Connect · Code Flow · Privacy.

1 Introduction

OpenID Connect (OIDC) [41] is the most widely adopted single sign-on (SSO) protocol for public online services. It is currently used by over 450,000 websites, representing 61% of the total login provider market [4]. OIDC provides a seamless user experience by enabling users to access multiple services, known as relying party (RP), using a single identity maintained by the identity provider (IdP). Specifically, the IdP authenticates the user and issues an ID token, which the RP uses to extract and verify the user's identity. There are two types of ID token forwarding mechanisms in OIDC: a risky implicit flow and a more secure code flow (as shown in Fig. 1). The former forwards the token via the risky user agent (e.g., browser), with risks such as token leakage [35]. The latter only forwards an authorization code through the user agent, which the token is transferred directly between the RP's and the IdP's back-end servers, and is only sent to the designated RP after its identity has been authenticated. OIDC code flow is a widely used protocol today and is recommended by industry (e.g., Yahoo [3] and Microsoft [42]) and Technical Standards Organizations [36].

Recent studies [25–27, 34, 47, 48] have paid much attention to two serious privacy issues with OIDC, namely traceability and linkability. The former arises from the need of IdP to generate an ID token containing both the user identifier (*uid*) and the RP identifier (*cid*) representing a specific user on a specific RP, which allows the IdP (usually assumed as honest-but-curious) to track the user's login events. The latter is because the IdP typically uses the user's unique identifier to generate their login credentials on different RPs to ensure a consistent identity for multiple login sessions on the same RP. This allows colluding RPs to link a user's login events across different RP services.

A number of privacy-preserving SSO protocols [14,18,19,24–27,32,34,47,48] have been proposed to address the traceability and linkability issues, but none of them may be adopted directly for the OIDC code flow. These solutions can be divided into two categories, namely, user anonymization and RP identifier randomization. The user anonymization schemes [14,32] anonymize the user identity at the user side, which requires the user to maintain/generate multiple unlinkable identities for different RPs and authenticated by the IdP through anonymization mechanisms such as anonymous credentials [32], blind signature [14], etc. Therefore, a customized client is usually required, leading to poor usability and little adoption in practice. The RP identifier randomization solutions [19,25–27,34] allow users to be authenticated by the IdP using conventional mechanisms such as username-password, two-factor authentication (2FA) and FIDO2, etc., thus providing a better user experience. However, these solutions are tailored for OIDC implicit flow that does not involve RP authentication, enabling RPs to provide only randomized RP identifiers to IdPs for untraceability.

In this paper, we provide an RP-anonymization-based privacy-preserving SSO scheme called ARPSSO, which can be adapted to OIDC code flow well. This scheme is transparent to the users (i.e., conventional authentication mechanisms and standard browsers can be used) and solves both traceability and linkability problems while retaining the intrinsic security properties of the OIDC

protocol [17]. We first design an RP anonymous credential system to implement IdP-to-RP anonymous authentication, which requires the RP to generate an RP anonymous identifier in each SSO session randomly. Because of the introduced IdP-to-RP anonymous authentication, generating a consistent identity for the same user across multiple login sessions on the same RP becomes difficult. To address this problem, we introduce a two-party secure computation scheme that supports IdP-to-RP anonymous authentication, where the RP provides the RP anonymous identifier (i.e., the anonymous *cid*), the IdP provides the user identifier (i.e., *uid*), and they cooperate in generating the ID token. The RP then derives a unique, unlinkable, and consistent identifier for each user on this RP through its own identifier (i.e., *cid*) and the ID token.

Moreover, we also introduce a user-transparent forwarding mechanism to securely forward the confidential data via the user agent, i.e., the ID token in implicit flow or the authorization code in code flow. Existing solutions achieve secure forwarding by introducing a trusted component (e.g., a trusted plug-in or a trusted execution environment) in the user agent, which is less user-friendly due to the need for each user to participate. In ARPSSO, we enable secure in-browser data forwarding via a trusted script. The trusted script is publicly available with fixed content, so RPs, IdPs, and users can verify its integrity and security easily.

We prove the security of ARPSSO through protocol analysis and formal methods, including whether it guarantees two privacy properties (i.e., user sign-on untraceability and user identity unlinkability) as well as the intrinsic security properties of the OIDC protocol. We also evaluate the performance of ARPSSO through a prototype system. The results show that the overhead introduced is acceptable compared to conventional SSO systems.

The main contributions of this paper are:

- We propose an OIDC-compatible privacy-preserving SSO scheme named ARPSSO that enables user sign-on untraceability and user identity unlinkability. To the best of our knowledge, it is the first scheme to support the code flow of the OIDC protocol.
- We introduce a user-imperceptible in-browser data forwarding mechanism. The entire SSO process is transparent to the user and works seamlessly with conventional authentication mechanisms and standard browsers, making it user-friendly.
- We analyze the security of ARPSSO and show that it balances both privacy goals and the intrinsic security properties of the OIDC protocol. We also evaluate the performance of the protocol through a prototype implementation and show that the overhead is acceptable compared to conventional SSO systems.

2 Background

2.1 OpenID Connect

OpenID Connect provides an identity layer on top of the OAuth 2.0 framework [30], tailored for SSO. There are two primary OIDC protocol flows, the code flow and the implicit flow, as illustrated in Fig. 1. The Code flow may be viewed as a full version of the implicit flow, with additional RP authentication (i.e., steps ⑨) added to ensure secure ID token forwarding. It consists of 13 steps: ① The user accesses the RP service and initiates a login request. ②-③ The RP redirects the login request to the IdP. ④-⑤ The IdP prompts the user to perform user authentication. ⑥-⑧ Upon successful authentication, the IdP generates an authorization *code* and redirects it to the RP via the user agent. ⑨-⑪ The RP backend authenticates to the IdP and exchanges the *code* for the `id_token`. ⑫-⑬ Finally, the RP verifies the `id_token` and notifies the user of the login result.

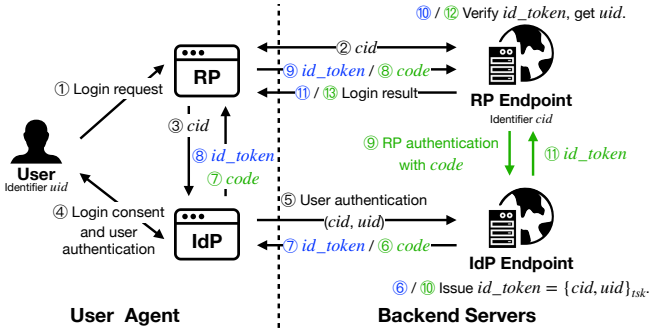


Fig. 1: OIDC Protocol Flow. It includes two working flows: a risky implicit flow and a more secure code flow, marked in blue and green.

2.2 Proof of Knowledge

Proof of Knowledge (PoK) is realized by using zero-knowledge proofs. It allows the prover to prove to the verifier that they know a piece of information without revealing it. The PoK scheme in this paper is non-interactive, which is based on the Sigma protocol [13] and the Fiat-Shamir heuristic [21]. For instance, the notation $PoK\left\{(\alpha,\beta) : y = g^{\alpha}h^{\beta} \wedge \tilde{y} = \tilde{g}^{\alpha} \wedge \phi(\alpha,\beta) = true\right\}$ represents a Non-Interactive Zero-Knowledge (NIZK) proof of knowledge of integers α, β such that $y = g^{\alpha}h^{\beta}$ and $\tilde{y} = \tilde{g}^{\alpha}$ where α, β satisfy relation ϕ.

2.3 Anonymous Credential Based on PS Signature

An anonymous credential system typically involves three roles: the issuer, the prover, and the verifier. The issuer issues anonymous credentials containing a

set of identity attributes for the prover. These credentials enable the prover to authenticate itself to the verifier without revealing personal information (e.g., age, gender), a process called anonymous authentication. Our scheme uses the Pointcheval-Sanders (PS) signature algorithm [44] to build the anonymous credential system, described as follows.

Setup$(1^\ell) \rightarrow (pp)$: Choose a type-3 bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T, \mathbb{G}_1 \neq \mathbb{G}_2$ [22] with generators g_1 and g_2 for groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to define system parameters $pp \leftarrow (e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$.

KeyGen$(pp, n) \rightarrow (sk, vk)$:For n attributes, randomly select $x, y_1, ..., y_n \xleftarrow{\$} \mathbb{Z}_p$. Construct $X = g_1^x, Y_i = g_1^{y_i}$ and $\hat{X} = g_2^x, \hat{Y}_i = g_2^{y_i}$ for $i = 1 \ldots n$. The keypair is (sk, vk) where $sk = (x, y_1, \ldots, y_n, X)$ and $vk = (Y_1, \ldots, Y_n, \hat{X}, \hat{Y}_1, \ldots, \hat{Y}_n)$.

PSSign$(A, h, sk) \rightarrow \sigma$: With a random generator $h \xleftarrow{\$} \mathbb{G}_1$, produce signature $\sigma = (\sigma_1, \sigma_2) = (h, h^{x+A_1 \cdot y_1 + ... + A_n \cdot y_n})$ on attributes $A = (A_1, ..., A_n)$.

PSVerify$(\sigma, A, vk) \rightarrow b$: Verify the signature $\sigma \leftarrow (\sigma_1, \sigma_2)$ on attributes A using $e(\sigma_2, g_2) = e(\sigma_1, \hat{X}) \prod_{i=1}^{n} e(\sigma_1, \hat{Y}_i)^{A_i}$.

It is essential to highlight that the PS signature possesses a randomizable feature. Specifically, **PSVerify**$((\sigma_1^k, \sigma_2^k), A, vk) = true, k \xleftarrow{\$} \mathbb{Z}_p$.

Leveraging the PS signature, we can build an anonymous credential system containing two distinct phases. In the initial phase, the prover obtains the validation of a set of attributes $A = (A_1, ..., A_n)$ from the issuer, i.e., it obtains a PS signature on attributes A. One approach is to sign directly using **PSSign**, requiring the prover to disclose all attributes. Another approach is to use a blind issue protocol **CredIssue**, where the prover separates attributes into hidden attributes $A_h = (A_1, ..., A_j)$ and public attributes $A_p = (A_{j+1}, ..., A_n)$. When the issuer signs these attributes A, the hidden attributes A_h remain unknown.

CredIssue$(pp, sk, A_h, A_p, \phi) \rightarrow (\sigma)$: The issuer and the prover run the following interactive protocol for anonymous credential issuance.

- **PrepareBlindSign**$(vk, A_h, \phi) \rightarrow (d, \lambda, \phi)$: The prover chooses $d \xleftarrow{\$} \mathbb{Z}_p$ and computes commitment $C = {g_1}^d \prod_{i=1}^{j} {Y_i}^{A_i}$, then generates a NIZK proof $\lambda = PoK\left\{(d, A_h) : C = {g_1}^d \prod_{i=1}^{j} {Y_i}^{A_i} \wedge \phi(A_h) = true\right\}$, which validates C and a relation ϕ. This λ is then sent to the issuer.

- **BlindSign**$(sk, A_p, \lambda, \phi) \rightarrow (A_p, \hat{\sigma})$: The issuer verifies the NIZK proof λ, and extracts C from it. Then the issuer chooses $r \xleftarrow{\$} \mathbb{Z}_p$, and computes blind signature $\hat{\sigma} = ({g_1}^r, (XC \prod_{i=j+1}^{n} {Y_i}^{A_i})^r)$ with public attributes A_p. Finally, the issuer sends $A_p, \hat{\sigma}$ to the prover.
- **Unblind**$(d, A_h, A_p, \hat{\sigma}) \rightarrow (A, \sigma)$: The prover receives $(\hat{\sigma}_1, \hat{\sigma}_2) \leftarrow \hat{\sigma}$, then unblinds it by computing $\sigma = (\hat{\sigma}_1, \hat{\sigma}_2 / \hat{\sigma}_1^d)$ to obtain the signature $\sigma = (\sigma_1, \sigma_2) = (h, h^{x+A_1 \cdot y_1 + ... + A_n \cdot y_n}), h = {g_1}^r$ on the attributes $A = A_h \cup A_p$.

In the second phase, the prover demonstrates to the verifier the possession of certain attributes ${A_p}' = (A_{j'+1}, ..., A_n)$, previously authenticated by the issuer, and hides other attributes ${A_h}' = (A_1, ..., A_{j'})$. These hidden attributes can be

verified to satisfy a specific relation ϕ' (e.g., age over 18) without the verifier being explicitly aware of them. This involves the following two algorithms.

CredPresent$(vk, A, \sigma, \phi') \rightarrow ({A_h}', {A_p}', \pi, \phi')$: The prover first selects $k, t \xleftarrow{\$} \mathbb{Z}_p$ and randomizes the signature by computing $\sigma' = (\sigma_1', \sigma_2') = ({\sigma_1}^k, ({\sigma_2 {\sigma_1}^t})^k)$. Then the prover chooses a set of hidden attributes ${A_h}' \in A$ and public attributes ${A_p}' = A \setminus {A_h}'$. Finally, the prover generates the credential presentation containing a NIZK proof $\pi = PoK\{(t, {A_h}') : \text{PSVerify}((\sigma_1', \sigma_2'/{\sigma_1'}^t), A, vk) = true \wedge \phi'({A_h}') = true\}$ to prove the correctness of the signature and sends it to the verifier.

CredPresentVerify$(vk, A_p', \pi, \phi') \rightarrow (true/false)$: The verifier verifies the credential presentation with the NIZK proof π to check whether the hidden attributes satisfy the statement ϕ' or not.

3 Adversary Model, Assumptions and Target Properties

In this section, we describe the adversary model, the security assumptions, and the target properties that we aim to achieve.

3.1 Adversary Model

Honest-but-curious IdP: The IdP in ARPSSO is non-malicious and follows the protocol honestly. However, it is interested in tracking the user's login events across multiple protocol sessions.
Colluding RPs: A set of RPs that are controlled by an adversary. Each RP has limited knowledge of the user identity, while the adversary attempts to link the user based on its knowledge.
Malicious users and/or malicious RPs: An adversary can control users, RPs, or a combination of both to participate in the SSO process. These malicious entities aim to impersonate a victim user while logging into honest RPs.

3.2 Assumptions

We assume that appropriate protection measures are implemented at the network level to prevent IdPs from tracking RP identities (e.g., the IP address) across the network, such as Tor [15] or the more efficient Oblivious HTTP (OHTTP) [45]. In addition, we assume that users do not grant their personally identifiable information (PII), such as phone numbers, e-mail addresses, etc., to RPs via ID tokens.

3.3 Target Properties

We aim to achieve the following properties in the ARPSSO.

The first is the privacy properties, including user login untraceability and user identity unlinkability. The former requires RPs to remain anonymous to

IdPs so that curious IdPs cannot determine which RP the user is requesting to log into. The latter requires that user identifiers in different RPs are unique and unlinkable, preventing colluding RPs from identifying the target user.

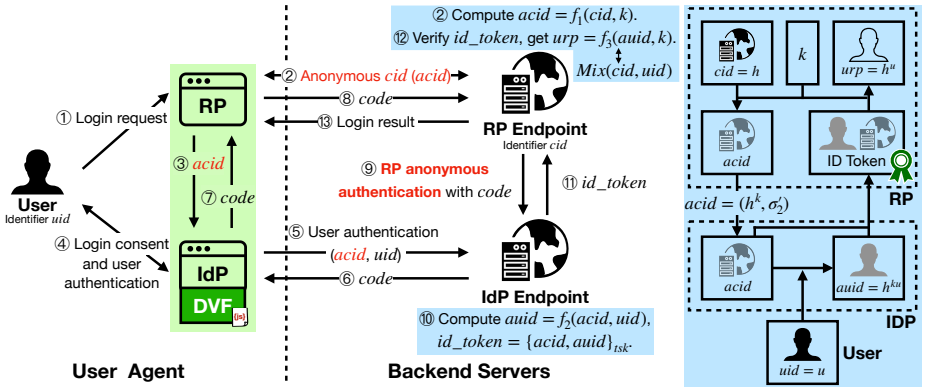
The second is the security properties, including RP anonymous authentication unforgeability, i.e., the attackers can not impersonate an honest RP to pass the IdP-to-RP authentication and obtain the ID token; user identity uniqueness and consistency, i.e., each user has a unique identity in the RP which remains consistent across multiple logins to the same RP; in-browser data forwarding trustworthiness, i.e., the *code* can be securely forwarded to the specified RP via the user agent.

The last is a transparent user experience. Users can complete the privacy-preserving SSO process using conventional authentication mechanisms in standard browsers, without installing additional plug-ins or software. They can also provide consent based on the verified RP's identity, ensuring a consistent experience similar to their daily activities.

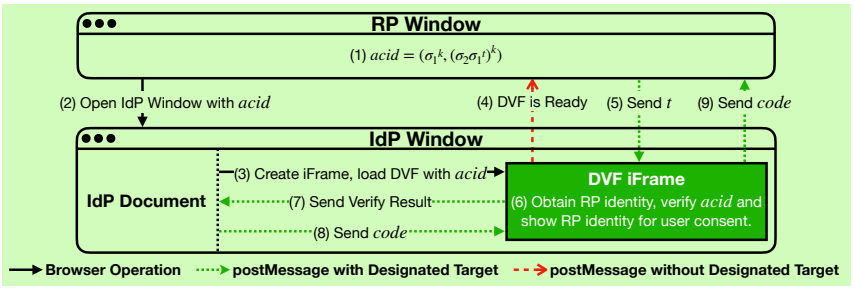
4 Design

ARPSSO is a privacy-preserving SSO scheme based on OIDC code flow, which implements the properties described in Section 3.3. Fig. 2(a) shows the overview of ARPSSO. It accomplishes privacy preservation through three mechanisms, namely the RP anonymous authentication, the user identity mix-up, and the trusted in-browser data forwarding, highlighted in red, blue, and green, respectively. First, to achieve user login untraceability, the IdP-to-RP authentication in step ⑨ needs to be anonymous. Therefore, we propose an RP anonymous authentication mechanism, which generates an anonymous identifier (*acid*) for the RP in each SSO session and achieves RP authentication based on *acid*. Second, in order to achieve untraceability and unlinkability, it needs to ensure that the IdP does not know the RP's real identity for this login (i.e., *cid*) and the RP does not know the user's real identity (i.e., *uid*) throughout the SSO process. This poses a challenge to generate a unique, consistent and unlinkable identity for each user in the RP. To address this problem, we propose a user identity mix-up mechanism based on two-party secure computation. Finally, we provide a user-transparent trusted in-browser data forwarding mechanism. It completes the user agent operations in steps ③, ④ and ⑦ (i.e., presenting the RP's real identity for user consent after verifying the *acid* and forwarding the IdP-generated *code* securely to the specified RP) without disclosing the RP's real identity to the IdP via a trusted third-party script called data verify forwarder (DVF). Details are as follows.

RP Anonymous Authentication. We achieve RP anonymous authentication based on the anonymous credential scheme described in Section 2.3. In ARPSSO, the IdP acts as both the issuer and the verifier, while the RP acts as the prover. Before the SSO process shown in Fig. 2(a), the RP requests an anonymous credential from the IdP by submitting a secret $\mathcal{S}$ (similar to a private key) and the RP domain $\mathcal{D}$ for issuance. This process is implemented through the



(a) Overview of ARPSSO. (b) User Identity Mix-up.



(c) Trusted In-Browser Data Forwarding.

Fig. 2: The workflow of ARPSSO. (a) shows the overview of ARPSSO, where the user identity mix-up marked in blue is detailed in (b), and the trusted in-browser data forwarding marked in green is detailed in (c).

blind issue protocol **CredIssue** to keep the $\mathcal{S}$ known only to the RP. In the **BlindSign** algorithm, the IdP chooses a unique random number $r \xleftarrow{\$} \mathbb{G}_1$ to compute $h = g^r$, which serves as the unique RP identifier (i.e., $cid = h$) and is encoded in the first part of the RP anonymous credential signature $\sigma = (\sigma_1, \sigma_2)$, i.e., $\sigma_1 = h = cid$. During the SSO process, the RP generates random number $k, t \xleftarrow{\$} \mathbb{Z}_p$ and computes a randomized signature $\sigma' = (\sigma_1^k, (\sigma_2 \sigma_1^t)^k)$ based on the signature σ. It then uses this randomized signature as the RP anonymous identifier $acid = \sigma'$ for this SSO session and sends it to the IdP in step ②. After receiving the $code$, the RP generates a credential presentation using the **CredPresent** algorithm and sends it to the IdP in step ⑨. This credential presentation contains a NIZK proof to prove the correctness of the randomized signature σ' without revealing the $\mathcal{S}$ which is only known to the RP, and $\mathcal{D}$ which is supposed to be unknown to the IdP. Then, the IdP anonymously authenticates the RP using the **CredPresentVerify** algorithm.

User Identity Mix-up. There is a unique RP identifier in the IdP (i.e., $cid = h$). Suppose the unique user identifier in the IdP is $uid = u, u \leftarrow \mathbb{Z}_p$, we can

construct the user identifier in the RP as $urp = cid^{uid} = h^u$, which is unique for each user in each different RP and unlinkable according to the Decision Diffie-Hellman (DDH) [11] assumption. To allow the RP to compute the identifier *urp* without disclosing to the IdP the RP's real identity for this login (i.e., *cid*) and without knowing the user identifier obtained by the IdP (i.e., *uid*), we introduce a user identity mix-up mechanism as shown in Fig. 2(b). During the SSO process, the RP provides an anonymous identifier to the IdP, i.e., $acid = (\sigma_1', \sigma_2') = (h^k, (\sigma_2 h^t)^k)$ in step ②. In step ⑤, the IdP authenticates the user and obtains the user identifier $uid = u$. In step ⑩, the IdP computes the anonymous user identifier $auid = acid^{uid} = h^{ku}$ and encloses it in the ID token. It then sends the token to the RP in step ⑪. When the RP receives the ID token, it obtains *auid* and calculates $urp = auid^{1/k} = h^u$ in step ⑫. Note that the RP chooses the random factor k, making it easy to compute $1/k$ in $\mathbb{Z}p$.

Trusted In-Browser Data Forwarding. We introduce the DVF to perform the user agent operations in steps ③, ④ and ⑦ of Fig. 2(a) without revealing the RP identity to the IdP. The DVF is a trusted third-party script with the workflow shown in Fig. 2(c), which consists of 9 steps: (1) When a user accesses the RP service, the RP front-end window obtains the RP anonymous identifier (i.e., *acid*) for this session from the RP back-end server. (2) The RP window opens a new IdP window to perform user authentication and passes *acid* to the IdP window. (3) In the IdP window, an `iframe` is created to load the DVF script, and *acid* is further passed to DVF. We use an `iframe` instead of loading the script directly in the IdP document to ensure that the DVF and IdP documents work in two different origins. (4)-(5) The DVF interacts with the RP window via `postMessage` to get the random number t in *acid*. The interaction is initiated by the DVF. Since the RP origin (i.e., domain $\mathcal{D}$) is not known to the DVF, the "ready" `postMessage` in step (4) does not specify the designated target (i.e., broadcast). Upon receipt of the "ready" message from the DVF, the RP verifies the origin of the message sender and responds t only to the trusted DVF script domain (specifying the DVF origin as the designated target in the `postMessage` function). This message event will also inform DVF of the RP domain $\mathcal{D}$. (6) DVF verifies the correspondence between *acid* and $\mathcal{D}$, and presents the RP identity to the user for consent if the correspondence holds. (7) The DVF notifies the IdP of the RP verification result. The IdP performs user authentication and generates the *code*. (8)-(9) The DVF forwards the IdP-generated *code* to the verified RP. During this process, the DVF acts as an intermediate agent to verify the RP's identity and securely forward the data between the IdP and the RP.

In step (6), the DVF needs to verify the correspondence between *acid* and the RP domain $\mathcal{D}$. Directly using the anonymous credential will involve creating and verifying NIZK proofs because the credential contains the secret $\mathcal{S}$, known only to the RP. This process will increase system complexity and performance overhead. However, we can avoid verifying NIZK proofs in the user agent via a simple optimization. Specifically, we split the RP anonymous credential into a pair of credentials, i.e., an identity credential $\mathcal{IC}$ containing the RP domain $\mathcal{D}$ and the RP identifier *cid*, but excluding the secret $\mathcal{S}$; and an authentication

credential $\mathcal{ATC}$ containing the RP domain $\mathcal{D}$ and the secret $\mathcal{S}$. $\mathcal{ATC}$ and $\mathcal{IC}$ are associated through the RP domain $\mathcal{D}$. The random number of the signature in $\mathcal{IC}$ is used as the RP identifier *cid*. The RP authentication in step ⑨ can be done cooperatively by $\mathcal{IC}$ and $\mathcal{ATC}$, while the RP verification in the user agent can be done independently through $\mathcal{IC}$. Since $\mathcal{IC}$ does not contain the secret $\mathcal{S}$, this design allows DVF to verify the authenticity of *acid* without any NIZK proof. Specifically, the RP's anonymous identifier $acid = ({\sigma_1}^k, ({\sigma_2{\sigma_1}^t})^k)$ is the randomization of the signature $\sigma = (\sigma_1, \sigma_2)$ in $\mathcal{IC}$. The RP obtains *acid* in step (3), and the domain $\mathcal{D}$ and the random number t in step (5). It can then extract ${\sigma_1}^k$ from *acid* and compute $({\sigma_1}^k)^{-t}$, and verify *acid* using equation $\textbf{PSVerify}((\sigma_1{}^k, (\sigma_2{\sigma_1}^t)^k/({\sigma_1}^k)^t), \mathcal{D}, vk) \rightarrow b$.

5 ARPSSO

The main workflow of ARPSSO consists of three stages.

5.1 Initialization

In this stage, the IdP generates the global parameters *pp* and the signing key pairs (tsk, tvk, csk, cvk), and publishes *pp* and public keys (cvk, tvk). (tsk, tvk) is the token key pair used to issue and verify ID tokens. It is generated based on JSON Web Key (JWK) [33] since the standard ID token is used in ARPSSO. (csk, cvk) is the credential key pair used to issue and verify RP anonymous credentials $\mathcal{IC}$ and $\mathcal{ATC}$, as shown below.

The IdP chooses a secure parameter ℓ and an type-3 bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, where g_1, g_2 are the generators of group $\mathbb{G}_1$, $\mathbb{G}_2$ respectively. The public parameters are denoted as $pp \leftarrow (e, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2)$. With the *pp*, the IdP chooses $x, y, \overline{x}, \overline{y}_1, \overline{y}_2 \xleftarrow{\$} \mathbb{Z}_p$, and computes $X = {g_1}^x$, $Y = {g_1}^y$, $\hat{X} = {g_2}^x$, $\hat{Y} = {g_2}^y$, $\overline{X} = {g_1}^{\overline{x}}$, $\overline{Y}_1 = {g_1}^{\overline{y}_1}$, $\overline{Y}_2 = {g_1}^{\overline{y}_2}$, $\hat{\overline{X}} = {g_2}^{\overline{x}}$, $\hat{\overline{Y}}_1 = {g_2}^{\overline{y}_1}$ and $\hat{\overline{Y}}_2 = {g_2}^{\overline{y}_2}$. The credential key pair (csk, cvk) is set as $csk \leftarrow (x, y, X, \overline{x}, \overline{y}_1, \overline{y}_2, \overline{X})$, $cvk \leftarrow (Y, \hat{X}, \hat{Y}, \overline{Y}_1, \overline{Y}_2, \hat{\overline{X}}, \hat{\overline{Y}}_1, \hat{\overline{Y}}_2)$. The values of $x, y, X, Y, \hat{X}, \hat{Y}$ are used for $\mathcal{IC}$ and others for $\mathcal{ATC}$.

5.2 Registration

In this stage, users and RPs register with the IdP, which assigns unique identifiers to the users and RPs and issues a pair of anonymous credentials to each RP. The IdP verifies the user's identity and provides a unique identifier $uid = u, u \leftarrow \mathbb{Z}_p$ to the user. The RP registration is accomplished via the interactive protocol shown in Fig. 3, through which the IdP issues two credentials to the RP, namely $\mathcal{IC}$ and $\mathcal{ATC}$. Briefly, each RP has two attributes, i.e., the RP secret $\mathcal{S}$ and the RP domain $\mathcal{D}$. The secret $\mathcal{S}$ is generated randomly and stored by the RP secretly. The domain $\mathcal{D}$ has two states: a public state used for implementing RP verification in the user agent through **PSVerify**, and a hidden state used for achieving RP

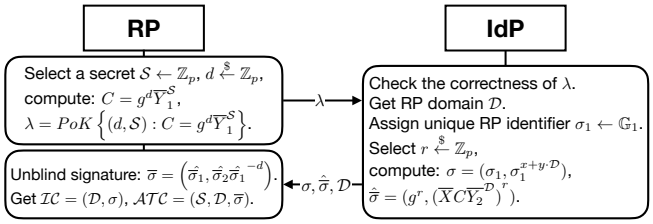


Fig. 3: RP Registration Protocol.

anonymous authentication in the IdP backend through **CredPresentVerify**. The $\mathcal{IC}$ contains $\mathcal{D}$ and a signature σ generated via **PSSign**. The signature σ contains a unique RP identifier assigned by the IdP, i.e., $\sigma_1 = cid$. $\mathcal{ATC}$ contains $\mathcal{S}$, $\mathcal{D}$ and a signature $\overline{\sigma}$ generated via **CredIssue**.

5.3 Single Sign-on

In this stage, users perform untraceable and unlinkable SSO based on OIDC code flow. The SSO flow is illustrated in Fig. 4, which contains three parts. Steps ① to ⑦, the RP backend randomizes the signature of $\mathcal{IC}$ to generate an anonymous identifier *acid* for the SSO session, and sends it to the IdP backend. Steps ⑧ to ⑱, the IdP document creates and loads the DVF document to perform the trust in-browser data forwarding, which displays the RP identity corresponding to *acid* to the user, obtains the user's consent, and forward the *code* generate by the IdP backend after the user authentication. Steps ⑲ to ㉑, the RP backend exchanges the ID token from the IdP backend with *code* and a NIZK proof π, which is used for RP anonymous authentication to prove the RP's identity authenticity corresponding to the *acid*. Then, the RP extracts the unlinkable user identifier from the ID token through the user-identity mixup mechanism.

Referer Header Privacy. Instead of opening IdPdoc directly in RPdoc, we introduce an intermediate document (i.e., RPRedirdoc) after the step ②. This is because the browsers add a referer header [5] to tell the backend server where the request came from, when opening a new window or redirecting to a new document. Therefore, opening IdPdoc directly in RPdoc would expose the RP's domain to the IdP. We can solve this problem by setting the "noreferrer" attribute. However, directly setting "noreferrer" in step ② will cause the popup window to lose the RP window handle, making it impossible to communicate between the two windows (e.g., step ⑩ and ⑪). For this reason, we introduce an intermediate document RPRedirdoc and set "noreferrer" for redirecting to IdPdoc in the popup window (i.e., step ⑤).

DVF Document Trustworthiness. Since DVFdoc is a fixed piece of code that implements a well-defined function (i.e., verifying the RP's identity and securely forwarding the IdP-generated *code* to the verified RP), its trustworthiness is critical to both the RP and the IdP. For example, the RP needs to ensure that

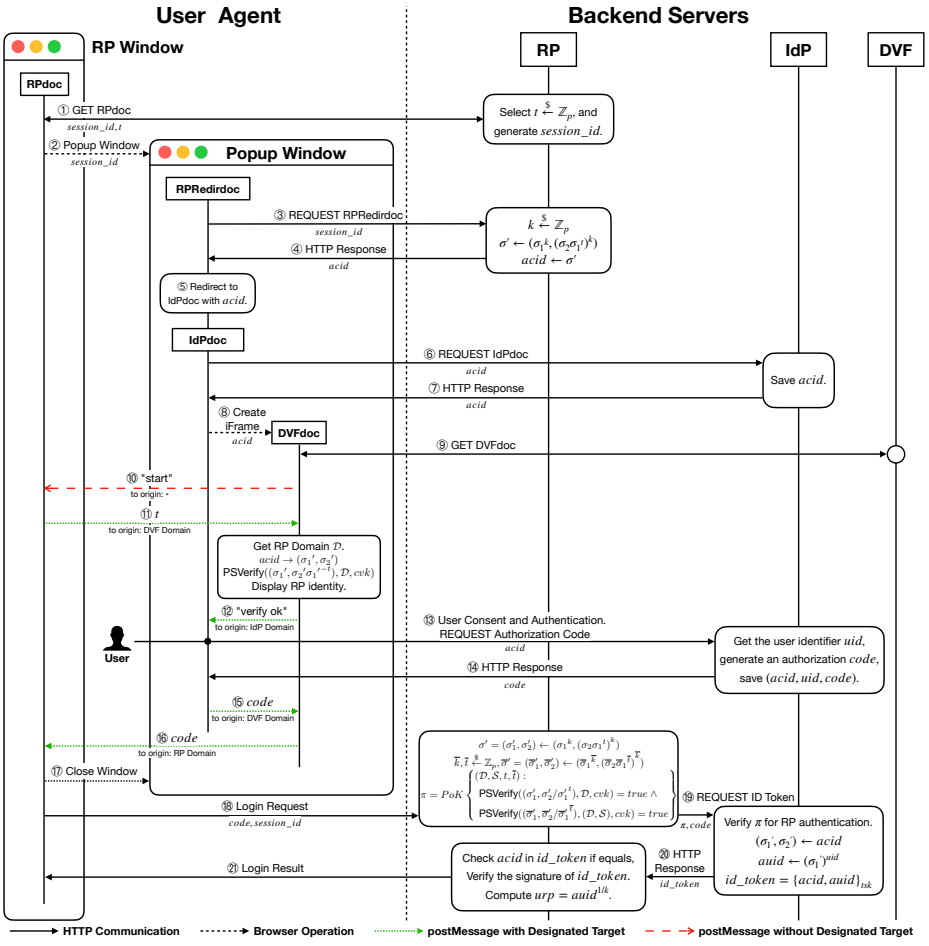


Fig. 4: SSO Flow of ARPSSO.

it does not expose the RP's identity to the IdP, and the IdP needs to ensure that it only sends *code* to the verified RP. We provide multiple DVF deployment methods to ensure the DVF is trustworthy, as discussed in Section 9.

6 Analysis

6.1 Security of ARPSSO

The security properties of the standard OIDC protocol have been defined and proven in the Dolev-Yao (DY) model [17, 20, 27], and they fall into two main categories, i.e., authentication property and session integrity [17]. The ARPSSO aims to introduce privacy preservation to the standard OIDC protocol while maintaining these security properties. Overall, we introduce three mechanisms,

i.e., RP anonymous authentication, user identity mix-up, and trusted in-browser data forwarding, whose main impact lies in the authentication property, while the other security properties are consistent with the standard OIDC framework. Specifically, ARPSSO is able to employ the same countermeasures against attacks such as relay attack and CSRF as OIDC [41], ensuring authentication session integrity. In addition, ARPSSO uses the standard ID token, which provides integrity and authenticity. Next, we specifically analyze the potential impact of the introduced mechanisms on the authentication property.

To begin, we examine the implementation of the authentication property by the standard OIDC code flow protocol. This property ensures that each user can log in to the intended RP and operate under their unique identity within that RP. The following mechanisms have been provided to fulfill this requirement.

- **RP Verification in User Agent:** In OIDC, the `id_token` in the implicit flow and the *code* in the code flow should only be forwarded to the corresponding RP through the user agent (usually the browser). To achieve this, the standard protocols enforce real-name RP verification before forwarding. Specifically, the IdP maps RP identifiers to RP domains and requires user agents to redirect the ID token or *code* to the RP endpoint. The RP endpoint URL is provided in HTTPS form to enable the user agent to verify the correspondence between the RP domain and the RP via a digital certificate.
- **RP Authentication:** To address token leakage in the implicit flow, the code flow passes the ID token directly and only between the IdP and the RP backend server, and the IdP authenticates the RP before sending the token. This ensures that confidential ID tokens are only passed to the RP with which the user intends to log in.
- **Designated ID Token:** The ID token in the OIDC is an attestation of an authenticated user identity in a specific RP. It should contain the user identity and the RP identity and be signed by the IdP. This ensures that the ID token can only be used to log in as the specified user to the specified RP. In addition, the user identity at the RP should be unique and consistent, ensuring that each user has a unique identity and remains consistent across multiple logins to the same RP.

Definition 1 formalized the authentication property for ARPSSO, similar to [17]. In short, it means the adversary should be unable to log in under a victim user's account at an honest RP using an honest IdP.

Definition 1 (Authentication Property of ARPSSO). *The authentication property in ARPSSO is satisfied when the following condition is met: For every honest RP rp, honest IdP idp, honest DVF dvf, honest user u, and their browser b. If u initiates the protocol on b to log in to rp via idp, and gives consent to do so with both idp and dvf, then at the end of the SSO process, only rp receives an ID token from idp (with attestations for u and rp) and returns the login status to u via b.*

We introduce three mechanisms in ARPSSO to address privacy issues in SSO. To achieve the above authentication property, they need to fulfill the following

three essential requirements. First, the RP anonymous authentication should be unforgeable such that no adversary can successfully pass the authentication, ensuring that the ID token only reaches its designated RP (RP anonymous authentication unforgeability). Second, through the user identity mix-up algorithm, the designated RP can extract from the ID token a specific user identity that is the user previously authenticated to the IdP (user identity uniqueness and consistency). Third, since the *code* generated by the IdP is intrinsically tied to the authenticated user, it must be kept confidential from all entities outside this SSO session. Consequently, the *code* must be securely forwarded to the target RP within the browser through the DVF (in-browser data forwarding trustworthiness).

Theorem 1 (RP Anonymous Authentication Unforgeability). *For a given honest RP's anonymous identifier acid, it is infeasible for any adversary to create a valid anonymous credential presentation to pass the authentication.*

Proof. (sketch) During the SSO process, the RP first randomizes the signature of $\mathcal{IC}$ to an RP anonymous identifier *acid* based on the random numbers k, t and sends it to the IdP. Then, in the RP anonymous authentication, the RP generates a NIZK proof π to the IdP. The π proves the correctness of *acid* and $\mathcal{ATC}$ without revealing the random number t used to generate the *acid* and the secret $\mathcal{S}$ in $\mathcal{ATC}$. Based on the soundness property of NIZK, no adversary can construct a NIZK proof to pass the RP anonymous authentication, since t and $\mathcal{S}$ are only known to the RP. However, if an adversary can use the victim RP's *acid* bound to his $\mathcal{ATC}$, the RP anonymous authentication unforgeability can be compromised. Specifically, for a given victim RP $RP_{\mathcal{V}}$ and its randomly generated $acid_{\mathcal{V}}$, if an adversary can efficiently find a random number t' corresponding to its domain $\mathcal{D}_{\mathcal{A}}$ to pass the RP verification in the DVF, i.e. the Game 1, it means that *acid* cannot effectively identify the RP's identity. Since the RP's identity is identified by the domain $\mathcal{D}$, which is used to bind $\mathcal{IC}$ and $\mathcal{ATC}$. If the attacker can find this t', he can use his own $\mathcal{ATC}$ and corresponding secret $\mathcal{S}_{\mathcal{A}}$ to pass the RP anonymous authentication.

Game 1 $G^{A}_{ARPSSO}(1^{\ell}, \mathcal{A})$

initialize pp, csk, cvk
simulate $RP_{\mathcal{V}}$ and $RP_{\mathcal{A}}$ registered at IdP
$\mathcal{IC}_{\mathcal{V}} \leftarrow \mathbf{PSSign}(\mathcal{D}_{\mathcal{V}}, cid_{\mathcal{V}}, csk) = (cid_{\mathcal{V}}, {cid_{\mathcal{V}}}^{x+y\cdot\mathcal{D}_{\mathcal{V}}})$
$\mathcal{IC}_{\mathcal{A}} \leftarrow \mathbf{PSSign}(\mathcal{D}_{\mathcal{A}}, cid_{\mathcal{A}}, csk) = (cid_{\mathcal{A}}, {cid_{\mathcal{A}}}^{x+y\cdot\mathcal{D}_{\mathcal{A}}})$
$k, t \xleftarrow{\$} \mathbb{Z}_p$
$acid_{\mathcal{V}} \leftarrow (\sigma'_1, \sigma'_2) = ({cid_{\mathcal{V}}}^{k}, ({cid_{\mathcal{V}}}^{x+y\cdot\mathcal{D}_{\mathcal{V}}} \cdot {cid_{\mathcal{V}}}^{t})^{k})$
$t' \leftarrow \mathcal{A}()$
if $\mathbf{PSVerify}((\sigma'_1, \sigma'_2/(\sigma'_1)^{t'}), \mathcal{D}_{\mathcal{A}}, cvk) = true$ **then**
  return t'
end if

The t' can be expressed in the form of the equation (1). Based on the Discrete Logarithm problem (DLP) [39] assumption, there does not exist such a probabilistic polynomial time (PPT) Algorithm 1. Consequently, in ARPSSO, a given *acid* can identify its corresponding RP, ensuring that RP anonymous authentication is unforgeable.

$$\begin{aligned} &\text{Let } (\sigma_1, \sigma_2) \leftarrow (cid_{\mathcal{A}}, {cid_{\mathcal{A}}}^{x+y\cdot\mathcal{D}_{\mathcal{A}}}) \\ &\quad \mathbf{PSVerify}((\sigma_1', \sigma_2'/(\sigma_1')^{t'}), \mathcal{D}_{\mathcal{A}}, pk) = true \\ \Longleftrightarrow\ & e({\sigma_1'}^{x+y\cdot\mathcal{D}_{\mathcal{V}}} \cdot {\sigma_1'}^{t}/{\sigma_1'}^{t'}, g_2) = e(\sigma_1', \hat{X}\hat{Y}^{\mathcal{D}_{\mathcal{A}}}) \\ \Longleftrightarrow\ & t' = \log_{\sigma_1'} \sigma_2' - \log_{\sigma_1} \sigma_2 \end{aligned} \tag{1}$$

Theorem 2 (User Identity Uniqueness and Consistency). *For any user who initiated the SSO flow to log in to the RP multiple times, the RP can extract a unique and consistent user identifier urp from the ID token, and the urp represents the user in the RP.*

Proof. (sketch) For an RP and a user with identifiers *cid* and *uid* at the IdP, respectively, after the user identity mix-up, the RP will eventually compute the user identifier at the RP as $urp = cid^{uid}$. Since the user identifier *uid* is fixed and unique in the IdP, the pair (idp, urp) in the RP can uniquely identify the user in the RP. On the other hand, for a given ID token $\{acid, auid\}_{tsk}$ and according to the Theorem 1, the *acid* can effectively identify the RP. In addition, the RP backend keeps the random number k that generates *acid*. Therefore, the *auid* in the token can be uniquely converted to *urp* using this random number k.

Theorem 3 (In-Browser Data Forwarding Trustworthiness). *For every honest RP rp, honest IdP idp, honest DVF dvf, honest user u, and user-agent b. Whenever u logs into rp with idp and dvf on b, as a result of running the protocol, dvf guarantees that the data is sent only to rp, i.e., any malicious entity other than rp, idp, dvf, or u cannot know the transmitted data.*

Proof. (sketch) We can prove this theorem intuitively. In the trusted in-browser data forwarding mechanism, the DVF first broadcasts a handshake message to the RP window. Although any document (possibly not from the originating RP) that receives this message can respond to the DVF, only the originating RP that provided *acid* to the IdP can pass the RP verification according to Theorem 1. This ensures that the verified RP is the one to which the user initiates the login. Once the DVF has verified the identity of the RP, subsequent communications are made using the `postMessage` with the designated target (i.e., RP domain). The browser ensures that all subsequent data transfers are secure.

Next, we use the Tamarin Prover [38] to formally verify this property under the DY model. The Tamarin Prover is an automated tool for verifying security protocols within the symbolic cryptographic framework. Within this framework, protocols are outlined using multiset rewriting rules, while trace properties, such as trustworthiness, are defined in first-order logic. With respect to the Dolev-Yao adversary, the Tamarin Prover can be used to simulate the attacker's behavior and provide potential attack scenarios if it fails to find a proof. Based on the

Tamarin Prover, we have modeled the rules for the `postMessage` mechanism used in the in-browser channel communication. The rest of the work involves modeling the behavior of the network, browser, and user, following the formalized modeling of the standard OIDC in [27]. The source code for the proof model is provided on the GitHub [1]. The result shows that the *code* is securely forwarded to the RP where the user initiated the login, proving that in-browser data forwarding is trustworthy.

6.2 Privacy of ARPSSO

Theorem 4 (User Sign-On Untraceability). *For a PPT adversary $\mathcal{A}$ controlling an IdP, it is infeasible to distinguish whether a user who has initiated two different SSO sessions is trying to sign on to the same RP or not.*

Proof. (sketch) During the SSO process, the IdP can obtain the RP anonymous identifier *acid* and the NIZK proof π from the RP. The *acid* is generated by randomizing the RP identifier *cid* with the random numbers s, t. Based on the random oracle model, *acid* is a uniformly distributed random value and perfectly hides the RP identifier *cid*. Furthermore, due to the zero-knowledge property, the identity of the RP (i.e., the RP domain) is hidden in the NIZK proof and will not be revealed to the IdP. Therefore, the adversary cannot trace the user's login to the RP based on its knowledge.

Theorem 5 (User Identity Unlinkability). *For a PPT adversary $\mathcal{A}$ controlling two colluding RPs, it is infeasible to distinguish whether two users in two different RPs correspond to the same user within the IdP.*

Proof. (sketch) For two users with identifiers $urp_1 = {cid_1}^{u_1}$ and $urp_2 = {cid_2}^{u_2}$ at two different RP (with identifiers cid_1 and cid_2). If $\mathcal{A}$ can distinguish whether urp_1 and urp_2 correspond to the same identifier u_1, this implies that $\mathcal{A}$ has the ability to compare urp_2 with ${urp_2}' = {cid_2}^{u_1}$. However, due to the Decision Diffie-Hellman (DDH) [11] assumption, it is computationally infeasible for $\mathcal{A}$, knowing $urp_1, urp_2, cid_1, cid_2$, to determine whether or not $urp_2 = {cid_2}^{u_1}$. In addition, based on the Symmetric External Diffie-Hellman (SXDH) [23] assumption, the DDH problem is still hard in groups G_1 for type-3 pairings.

7 Evaluation

We implement a prototype system [1] for ARPSSO and evaluate the performance. The BLS12-381 curve [12] is used for the PS signature due to its strong security and computational efficiency [9]. We develop RP and IdP backend in Go, and implement the PS signature and NIZK protocol for RP anonymous credential using Cloudflare's CIRCL library [16]. The front-end DVF script is implemented in Javascript using the Nobel-bls12-381 package [40]. We deploy the RP and IdP backend on a virtual machine (with a 2.7GHz single-core virtual CPU, 50GB SSD, and 8GB RAM), and use Chrome on a Google Pixel 8 and

a MacBook Pro (with an Intel Core i7-9750H CPU, 512GB SSD, and 16GB of RAM) to execute client-side codes including the DVF script. Table 1 shows the overhead of ARPSSO compared to standard OIDC, and all values in the table are averaged over 1000 repetitions. The overall overhead is about 200ms. Client-side operations introduce the most overhead due to the inefficiency of executing JavaScript code in the browser, which could be optimized with WebAssembly (WASM) [48].

Table 1: Overhead of ARPSSO Compared to Standard OIDC

Protocol Stage	**Operation**	**Time (in ms)**
Initialization	Key generation	6.6
RP Registration	RP generates commitment	2.6
	IdP generates blind signature	2.6
	RP unblinds signature	7.9
SSO	RP generates anonymous id *acid*	1.3
	DVF verifies *acid* in browser	107.1 / 123.1 *
	RP generates a proof for *acid*	68.8
	IdP verifies the proof	24.3

* Overhead on two client devices, Google Pixel 8 and MacBook Pro.

8 Related Work

In this section, we describe existing privacy-preserving SSO schemes, which can be categorized into two types, i.e., asynchronous authentication privacy-preserving SSO and synchronous authentication privacy-preserving SSO.

Asynchronous Authentication Privacy-Preserving SSO. Based on anonymous credentials, many works [7, 8, 28, 29, 37, 43, 46, 48] achieve untraceability by constructing an asynchronous authentication model [48]. In this model, users obtain anonymous credentials from the IdP and use these anonymous credentials to authenticate directly to the RP during the SSO process. Since the IdP is not required to participate in the SSO process, there is no traceability issue. In addition, users are allowed to generate unlinkable identifiers for different RPs based on the anonymous credential to achieve unlinkability. However, managing the credentials issued by the IdP can be a burden on users, resulting in a poor user experience. In addition, this model increases the burden on the RP to authenticate the user, e.g., to verify whether the anonymous credentials have been revoked.

Synchronous Authentication Privacy-Preserving SSO. In most public SSO systems, the IdP performs user authentication centrally and synchronously when users access RP services. This is the standard SSO model and can reduce the cost of identity management for RP services. There are also a range of

Table 2: Comparison of Synchronous Authentication Privacy-preserving SSO solutions.

Solution	SSO Feature			Privacy-Preserving Feature		User-Friendly Feature	
	Unique and Consistent User ID at RP	User Authentication Only to IdP	Compatible with OIDC Code Flow	User Sign-on Untraceability	User Identity Unlinkability	No Special Demands for User Agent[1]	No Special Demands for User Authn[2]
BrowserID [18]	✔	◐[3]	✗	✔	✗	✔	✔
SPRESSO [19]	✔	✔	✗	✔	✗	✔	✔
PRIMA [10]	✔	✗	✗	✔	✗	✗	✗
PPID [24]	✔	✔	✔	✗	✔	✔	✔
Apple [31]	✔	✔	✔	✗	✔	✔	✔
PseudoID [14]	✔	✗	✗	◐[5]	✔	✔	✗
UnlimitID [32]	✔	✗	✔	◐[5]	✔	✗	✗
POIDC [27]	✔	✔	✗	✔	✔	◐[6]	✔
UPPRESSO [26]	✔	✔	✗	✔	✔	◐[6]	✔
AIF [34]	✔	✔	✗	✔	✗	◐[6]	✔
UP-SSO [25]	✔	✔	✗	✔	✔	✗	✔
MISO [47]	✔	✔	◐[4]	✔	✔	✔	✔
ARPSSO	✔	✔	✔	✔	✔	✔	✔

◐ denotes the feature is partially supported.

1 The user can use standard browsers as the user agent without installing a specialized client or browser plug-ins.

2 The user can authenticate to the IdP through conventional authentication methods (e.g., username-password) or enhanced authentication methods (e.g., 2FA, FIDO2).

3 In BrowserID, the user generates a private key to sign an identity attestation, and the RP must verify the signature.

4 MISO is not a standard SSO model. It introduces the Mixer as a bridge between the IdP and the RP. When exchanging the ID token using the *code*, the IdP authenticates the Mixer instead of the RP to achieve untraceability. Therefore, it is not a strict OIDC code flow mode.

5 In PseudoID and UnlimitID, the IdP can obtain the user's identifier in the RP, so it can track the user's login behaviors in the same RP.

6 AIF and PIODC do not provide implementations. According to our analysis, they both require a browser plug-in or a dedicated user agent software to perform data forwarding between the RP and the IdP. UPPRESSO provides two implementations, one requiring a browser plug-in and the other not. However, in the non-plug-in version, the script responsible for data forwarding is provided and controlled by the IdP, which would violate the "IdP is curious" model.

SSO solutions [10, 14, 18, 19, 24–27, 31, 32, 34, 47] that achieve privacy-preserving while retaining centralized synchronous authentication. ARPSSO belongs to this category and is the first privacy-preserving SSO solution compatible with the OIDC code flow. Table 2 provides a detailed comparison of these solutions.

BrowserID is the first privacy-preserving SSO solution to implement user sign-on untraceability, and a privacy vulnerability has been discovered in its implementation [18]. SPRESSO [19] and PRIMA [10] achieve untraceability by randomizing RP identifiers to IdP. They use self-designed SSO processes that do not include RP authentication and are, therefore, incompatible with the OIDC

code flow. In addition, none of these solutions provide user identity unlinkability. The main idea to achieve unlinkability is user identifier anonymization, i.e., ensuring that a user's identifier is distinct in different RPs. NIST's Pairwise Pseudonymous Identifiers (PPIDs) [24] and Sign in with Apple [31] anonymize user identifiers through the IdP, and therefore suffer from the traceability issue. PseudoID [14] and UnlimitID [32] anonymize user identifiers at the user side, enabling users to authenticate to the IdP anonymously using blind signature [14] or anonymous credentials [32]. However, these schemes are restricted to protocol-specific user authentication credentials, limiting their practical usability.

POIDC [27] and UPPRESSO [26] achieve untraceability and unlinkability through RP identifier randomization. However, both POIDC and UPPRESSO are designed based on the OIDC implicit flow and lack a method for RP authentication based on the randomized RP identifier. A recent paper [34] points out the importance of RP authentication and constructs an Authenticated Implicit Flow (AIF) that supports untraceability, but it does not support unlinkability and is not compatible with OIDC code flow.

UP-SSO [25] and MISO [47] are privacy-preserving SSO schemes based on Trusted Execution Environment (TEE). They use a TEE component to mediate communication between the RP and the IdP, including randomizing the RP identifier to the IdP and generating a consistent, unlinkable user identifier in the RP. UP-SSO deploys the TEE component at the user side, imposing specific requirements on the user's device. MISO deploys the TEE component (called Mixer) in the cloud as an intermediary between the RP and the IdP. This solution deviates from the standard SSO model, doubling the protocol latency, introducing a single point of failure risk, and increasing deployment costs.

9 Discussion

Compatibility and Extendability. Since the implicit flow is a simplified version of the code flow, ARPSSO naturally supports untraceability and unlinkability for the OIDC implicit flow, except that the DVF shall securely forward the ID token instead of the *code* in the user agent. In addition, the RP anonymous credential and user identity mix-up we introduce can be used as building blocks to achieve untraceability and unlinkability simultaneously. Further research could build on this foundation to create customized IdP-to-RP Identity and Access Management (IAM), e.g., to support revocation, tracking, and de-anonymization of RP anonymous credential to meet regulatory requirements.

DVF Deployment. Since the DVF script is static, it can be easily deployed by hosting it on a file server. To establish a trustworthy DVF, as mentioned in Section 5.3, deploying it on a trusted third-party server is an option, although it would increase the deployment costs. Another option is to deploy it through a static site hosting service and verify its integrity during the SSO process. GitHub Pages [2] offers this service at no cost and supports custom domain configuration (providing a unique domain name for the `postMessage` mechanism). It also allows for public inspection of the files stored on it. We can verify the integrity

of the DVF script based on the Subresource Integrity (SRI) Web technology currently being developed by the W3C [6]. Specifically, SRI allows a document to create an `iframe` or `link` with an attribute integrity that takes a hash value. Initially, the RP and the IdP review the script file and record its hash value. During SSO, the RP loads the DVF script as a `link` and uses SRI to verify the script's integrity before redirecting the user to the IdP. Similarly, the IdP verifies the script's integrity before loading the DVF iframe.

10 Conclusion

In this paper, we introduce a privacy-preserving SSO scheme based on OIDC code flow. It achieves user sign-on untraceability by designing an RP anonymous credential that allows RP to generate anonymous identifiers for each SSO session. These RP anonymous identifiers are used in a two-party secure computation to ensure user identity unlinkability and enable RP anonymous authentication to comply with OIDC code flow. We also introduce the DVF for secure data forwarding between the IdP and the RP, accompanied by user login consent. Our security and privacy analysis shows that our system preserves the security of the conventional OIDC system while satisfying two essential privacy goals. Also, the prototype of ARPSSO shows that the overhead it introduces is reasonable.

Acknowledgments. This work is supported by the National Key R&D Program of China (Grant No.2022YFB3103303).

References

1. ARPSSO/ARPSSO. https://github.com/ARPSSO/ARPSSO
2. GitHub Pages. https://pages.github.com/
3. Implicit Grant Flow for Client-Side Apps - Yahoo Developer Network. https://developer.yahoo.com/oauth2/guide/flows_implicitgrant/
4. Login Provider Technologies Market Share and Web Usage Statistics. https://www.similartech.com/categories/login-provider
5. Referrer Policy. https://www.w3.org/TR/referrer-policy/
6. Subresource Integrity. https://www.w3.org/TR/SRI//#verification-of-html-document-subresources (2016)
7. Alpár, G., Van Den Broek, F., Hampiholi, B., Jacobs, B., Lueks, W., Ringers, S.: IRMA: Practical, decentralized and privacy-friendly identity management using smartphones. In: 10th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2017). pp. 1–2 (2017)
8. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y.: Hyperledger fabric: A distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference. p. 30. ACM (2018)
9. Aranha, D.F., Fuentes-Castañeda, L., Knapp, E., Menezes, A., Rodríguez-Henríquez, F.: Implementing Pairings at the 192-Bit Security Level. In: Abdalla, M., Lange, T. (eds.) Pairing-Based Cryptography – Pairing 2012. pp. 177–195. Lecture Notes in Computer Science, Springer (2013)

10. Asghar, M.R., Backes, M., Simeonovski, M.: PRIMA: Privacy-Preserving Identity and Access Management at Internet-Scale. In: 2018 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2018)
11. Boneh, D.: The Decision Diffie-Hellman problem. In: Buhler, J.P. (ed.) Algorithmic Number Theory. pp. 48–63. Lecture Notes in Computer Science, Springer (1998)
12. Bowe, S.: BLS12-381: New zk-SNARK Elliptic Curve Construction. https://electriccoin.co/blog/new-snark-curve/
13. Damgård, I.: On Σ-protocols. Lecture Notes, University of Aarhus, Department for Computer Science p. 84 (2002)
14. Dey, A., Weis, S.: PseudoID: Enhancing privacy in federated login (2010)
15. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. Tech. rep., Naval Research Lab Washington DC (2004)
16. Faz-Hernández, A., Kwiatkowski, K.: Introducing CIRCL: An advanced cryptographic library. https://github.com/cloudflare/circl/
17. Fett, D., Küsters, R., Schmitz, G.: The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF). pp. 189–202. IEEE (2017)
18. Fett, D., Küsters, R., Schmitz, G.: Analyzing the BrowserID SSO System with Primary Identity Providers Using an Expressive Model of the Web. In: Pernul, G., Y A Ryan, P., Weippl, E. (eds.) 20th European Symposium on Research in Computer Security. pp. 43–65. Computer Security – ESORICS 2015, Springer International Publishing (2015)
19. Fett, D., Küsters, R., Schmitz, G.: SPRESSO: A Secure, Privacy-Respecting Single Sign-On System for the Web. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1358–1369. CCS '15, Association for Computing Machinery (2015)
20. Fett, D., Küsters, R., Schmitz, G.: A Comprehensive Formal Security Analysis of OAuth 2.0. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1204–1215. CCS '16, Association for Computing Machinery (2016)
21. Fiat, A., Shamir, A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 186–194. Lecture Notes in Computer Science, Springer (1987)
22. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008)
23. Ghadafi, E., Smart, Nigel.P., Warinschi, B.: Groth–Sahai Proofs Revisited. In: Nguyen, P.Q., Pointcheval, D. (eds.) Public Key Cryptography – PKC 2010. pp. 177–192. Lecture Notes in Computer Science, Springer (2010)
24. Grassi, P., Nadeau, E., Richer, J., Squire, S., Fenton, J., Lefkovitz, N., Danker, J., Choong, Y.Y., Greene, K., Theofanos, M.: Digital Identity Guidelines: Federation and Assertions. Tech. Rep. NIST Special Publication (SP) 800-63C, National Institute of Standards and Technology (2020)
25. Guo, C., Lang, F., Wang, Q., Lin, J.: UP-SSO: Enhancing the User Privacy of SSO by Integrating PPID and SGX. In: 2021 International Conference on Advanced Computing and Endogenous Security. pp. 01–05 (2022)
26. Guo, C., Lin, J., Cai, Q., Li, F., Wang, Q., Jing, J., Zhao, B., Wang, W.: UP-PRESSO: Untraceable and Unlinkable Privacy-PREserving Single Sign-On Services. arXiv preprint arXiv:2110.10396 (2021)
27. Hammann, S., Sasse, R., Basin, D.: Privacy-Preserving OpenID Connect. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security. pp. 277–289. ASIA CCS '20, Association for Computing Machinery (2020)

28. Han, J., Chen, L., Schneider, S., Treharne, H., Wesemeyer, S.: Anonymous Single-Sign-On for n Designated Services with Traceability. In: Lopez, J., Zhou, J., Soriano, M. (eds.) European Symposium on Research in Computer Security. pp. 470–490. Computer Security, Springer International Publishing (2018)
29. Han, J., Chen, L., Schneider, S., Treharne, H., Wesemeyer, S., Wilson, N.: Anonymous Single Sign-On With Proxy Re-Verification. IEEE Transactions on Information Forensics and Security **15**, 223–236 (2020)
30. Hardt, D.: The OAuth 2.0 authorization framework. RFC 6749 (2012)
31. Inc, A.: Sign in with Apple. https://developer.apple.com/sign-in-with-apple/
32. Isaakidis, M., Halpin, H., Danezis, G.: UnlimitID: Privacy-Preserving Federated Identity Management using Algebraic MACs. In: Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society. pp. 139–142. WPES '16, Association for Computing Machinery (2016)
33. Jones, M.B.: JSON web key (JWK). RFC 7517 (2015)
34. Kroschewski, M., Lehmann, A.: Save The Implicit Flow? Enabling Privacy-Preserving RP Authentication in OpenID Connect. Proceedings on Privacy Enhancing Technologies (2023)
35. Lodderstedt, T., McGloin, M., Hunt, P.: RFC 6819: OAuth 2.0 threat model and security considerations. Internet Engineering Tast Force (IETF) pp. 1–71 (2013)
36. Lodderstedt, T., Bradley, J., Labunets, A., Fett, D.: OAuth 2.0 security best current practice. Internet-Draft draft-ietf-oauth-security-topics-23, Internet Engineering Task Force / Internet Engineering Task Force (2023)
37. Maganis, G., Shi, E., Chen, H., Song, D.: Opaak: Using mobile phones to limit anonymous identities online. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services. pp. 295–308. MobiSys '12, Association for Computing Machinery (2012)
38. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 696–701. Lecture Notes in Computer Science, Springer (2013)
39. Menezes, A.: Evaluation of security level of cryptography: The elliptic curve discrete logarithm problem (ECDLP). University of Waterloo **14** (2001)
40. Miller, P.: Noble-bls12-381. https://github.com/paulmillr/noble-bls12-381
41. Nat, S., John, B., Mike, J., de Medeiros, B., Mortimore, C.: Openid connect core 1.0 incorporating errata set 1. The OpenID Foundation, specification (2014)
42. nickludwig: OAuth 2.0 implicit grant flow - The Microsoft identity platform - Microsoft Entra. https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-implicit-grant-flow (2022)
43. Paquin, C.: U-prove technology overview v1. 1. Microsoft Corporation Draft Revision **1** (2011)
44. Pointcheval, D., Sanders, O.: Short randomizable signatures. In: Sako, K. (ed.) Topics in Cryptology - CT-RSA 2016. pp. 111–126. Springer International Publishing (2016)
45. Thomson, M., Wood, C.A.: Oblivious HTTP. Internet Draft draft-thomson-http-oblivious-01, Internet Engineering Task Force (2021)
46. Wang, J., Wang, G., Susilo, W.: Anonymous Single Sign-On Schemes Transformed from Group Signatures. In: 2013 5th International Conference on Intelligent Networking and Collaborative Systems. pp. 560–567. IEEE (2013)
47. Xu, R., Yang, S., Zhang, F., Fang, Z.: MISO: Legacy-Compatible Privacy-Preserving Single Sign-on using Trusted Execution Environments. In: 2023 IEEE

8th European Symposium on Security and Privacy (EuroS&P). pp. 352–372. IEEE Computer Society (2023)
48. Zhang, Z., Król, M., Sonnino, A., Zhang, L., Rivière, E.: EL PASSO: Privacy-preserving, Asynchronous Single Sign-On (2021)