UPPRESSO vs. PrivacyPass/TrustToken(R1&R3)

PrivacyPass and other BLS-signature-based anonymous protocols hide/anonymize user identities from both the IdP (i.e., the signing server in PrivacyPass) and RPs (i.e., the redemption servers), and therefore sacrifice an important function that allows an RP to recognize the same user across multiple logins, which is required in commonly-used SSO (the dilemma discussed in Section-3.2).

PrivacyPass actually provides token services, (*a*) anonymous from the view of RP and (*b*) privacy-preserving from the view of IdP; while UPPRESSO provides token services, (*a*) non-anonymous from the view of RP and (*b*) privacy-preserving from the view of both IdP and RP. We will finish the detailed comparison with PrivacyPass.

UPPRESSO vs. EL-PASSO/UnlimitID/Tandem(R3)

EL-PASSO, UnlimitID and Tandem rely on a user-maintained secret (e.g., s in EL-PASSO/UnlimitID, or x_U and sk_U in Tandem) to derive the anonymous user identities at RPs (i.e., Accounts in UPPRESSO). This results in two disadvantages: (1) As the user-maintained secret is necessary for generating a user's accounts (in EL-PASSO and UnlimitID) or restoring a user's long-term key (in Tandem), if the secret is lost or leaked, the user will permanently lose access to his accounts at the RPs. This risk is also noticed by the designers of EL-PASSO, and they recommended a back-up device for keeping/recovering the secret. In UPPRESSO t used in identity transformations is ephemeral, and a user only keeps the secret/credential to authenticate himself to the IdP which has nothing to do with the (pseudo-)identities. (2) User-secret-based designs require users to maintain pseudo-credentials by themselves, deviating from the initial SSO objective. The secret is too complex to remember and must be stored in devices. Logging in from a new device requires interactions with the old one. UPRESSO uses the same credential format as commonly-used SSO protocols and doesn't require any additional hardware for credentials.

Referer leakage(R3)

We thank the reviewer for pointing out this attack! The HTTP referer header may leak the information on the visited RP, but this will be prevented by setting the header *referrer-policy=no-referrer* in the HTTP response at Step-1.2. Then, the HTTP request to the IdP at Step-1.3 carries no *referer* header. This setting is specified by W3C, and widely supported. We have tested it in Chrome and Firefox, and confirmed no referer leakage.

Proofs of the prevention against RP-based linkage(R3)

The game in Section 5.2 can be extended as new game $\mathcal{G}'$, when there are login instances (called background instances), the adversary tries to correlates the user of new login instance with background instances. The output is 1, when adversary guesses the user is owner of some background instances; otherwise 0. Algorithm $\mathcal{D}'^{*}$ is designed based on $\mathcal{G}'$. Let (P, $[x]P$, $[y]P$, $[xy]P$) and (P, $[x]P$, $[y]P$, $[z]P$) be two inputs of $\mathcal{D}^{*}$.

The background instances of user U_0 are Ins_{u_0,r_i}=$\{([r_i]P, t_i, [r_i t_i][y]P)\}$=$\{([r_i]P, t_i, [r_i t_i][u_0]P)\}, i = 1, 2, 3...$, while r_i is random value, $[r_i]P$ stands for RP, and $u_0 = y$ is unknown to adversary.

The background instances from other users are Ins_{u_j,r_i}=$\{([r_i]P, t_{i,j}, [r_i t_{i,j} u_j]P)\}, i, j = 1, 2, 3...$, while u_j stands for user identity.

The new instance is $([x]P, t', [y][t'][x]P)$ or $([x]P, t', [z/x][t'][x]P)$.

Thus the sets of login instances received by adversary are $\{\mathrm{Ins}_{u_0,r_i}$,Ins_{u_j,r_i},$([x]P, t', [y][t'][x]P)\}$ and $\{\mathrm{Ins}_{u_0,r_i}$,Ins_{u_j,r_i},$([x]P, t', [z/x][t'][x]P)\}$.

As the probability of u_j equals to y or z/x is negligible (otherwise, it breaks DL assumption), if $z \not\equiv xy$, there must the non-negligible difference between

$Pr\{\mathcal{D}^*(P, [x]P, [y]P, [xy]P) = 1\} = Pr\{\mathcal{G}'(Ins_{u_0,r_i}, Ins_{u_j,r_i}, ([x]P, t', [y][t'][x]P)) = 1\} = Pr_c$ and

$Pr\{\mathcal{D}^*(P, [x]P, [y]P, [z]P) = 1\} = Pr\{\mathcal{G}'(Ins_{u_0,r_i}, Ins_{u_j,r_i}, ([x]P, t', [z/x][t'][x]P)) = 1\} = Pr_{\bar{c}}$

Thus, any adversary's advantage on this game equals to advantage on breaking DDH assumption.

We will improve the definitions and proofs.

Combination of PID_{RP} and $H(t)$(R3)

$H()$ is a common hash function such as SHA-256. So the result of $H(t)$ brings no advantage for the IdP to break the ECDLP between ID_{RP} with PID_{RP} (i.e., to solve t).

Evaluation (R2&R3&R4)

We are deploying the prototype in the Internet, to evaluate the performance on WAN and will give more evaluation details, such as network latency.

Relaxing the uniqueness requirement of PID_{RP} & Using H(RP-identifier) as base point (R3)

We thank the reviewer for these constructive suggestions. We'll carefully analyze the feasibility and security. Relaxing the uniqueness requirement of PID_{RP} will improve the performance.

Uniqueness of IDs and Scalability(R2)

The IdP generates ID_U and ID_{RP}, and ensures their uniqueness, as in other SSO schemes. PID_{RP} is generated randomly and its uniqueness is checked in PID_{RP} registration (Step-3.2 in Section-4.5). PID_U and *Account* are not required to be unique. The scale of IDs is up to n, the order of base point G, approximately 2^{256} in the prototype. A strong elliptic curve accommodates more users and RPs.

Real-world impact(R4)

We consider SSO login tracing as one of the web tracking techniques. Some privacy threat examples include receiving targeted advertisement after visiting a particular website, while there is no direct evidence showing any IdP or RP is actively tracing the users.

Source code(R3&R5)

We'll open-source the prototype through GitHub.

Writing(R3&R4&R5)

We will explicitly add the DDH assumption in Section-4.3, improve the security analysis, explain implementation details and polish the writing.