Reviews from the Submission to USENIX Security 2021 and Our Improvements

This manuscript was submitted to USENIX Security 2021 under the title "UPPRESSO: An Unlinkable Privacy-PREserving Single Sign-On System", and the decision was Reject and Resubmit.

The following are the *weaknesses* extracted from the reviews. We do not cite the strengths. We have improved this work greatly, to solve these weaknesses.

Review 1

1.1 Related work

A number of proposals for such privacy-friendly SSO exist already, but most target only one of the two goals. One recent exception is the EL PASSO protocol [1], which is not mentioned in the paper. The idea of using $ID_{RP}^{ID_U}$ to create domain-specific pseudonyms has been widely used in several pseudonym schemes (anonymous credentials, pseudonym systems, group signatures, DAA ..), but also in the context of SSO (e.g. [2,1]).
[1] EL PASSO: Efficient and lightweight privacy-preserving single sign on. PETS 2021.
[2] UnlimitID: Privacy-preserving federated identity management using algebraic MACs. WPES 2016.
[3] PseudoID: Enhancing privacy for federated login. PETS 2010.

Our improvement While there are a number of proposals claiming to provide privacy-preserving single sign-on services, they follow a login flow very different from the commonly-used SSO systems such as OIDC and OAuth 2.0. For example, in EL PASSO, UnlimitID or PseudoID, a user is authenticated by the RP with his credentials to prove that he owns the long-term secret; on the contrary, the authentication steps between the user and the RP do not exist in the commonly-used SSO protocols. In the commonly-used SSO systems and "pure" privacy-preserving SSO solutions such as BrowserID, SPRESSO and also UPPRESSO, authentication happens only between a user and the IdP, and an RP verifies only tokens generated by the IdP. Therefore, a user needs to notify each RP when a credential is lost or compromised in EL PASSO, UnlimitID or PseudoID, because the user is authenticated by the target RP (in addition to the authentication by the IdP). This feature is noticed by the authors of EL PASSO, so they update the credentials periodically. However, in the commonly-used SSO protocols, the user only needs to renew his only credential at the IdP if it is compromised, because no user credential is verified by the RPs.

This is a very important reason that the existing SSO protocols are widely deployed in the real world. This feature enables an SSO system such as OIDC and OAuth 2.0 to adopt any authentication mechanism (password, one-time password, 2-factor authentication, FIDO, etc.) for the IdP to authenticate users, which is actually independent of the SSO protocol. A "pure" SSO protocol deals with only tokens. No long-term secret is introduced by UPPRESSO as well as BrowserID and SPRESSO, while such a secret is needed in EL PASSO, UnlimitID and PseudoID and explicitly bound in the credentials verified by RPs. Such secrets are usually kept in browsers by extensions, so a user needs to install extra software.

We clarify this difference in Section 2.3 Extended Related Works, and name these solutions as Certified RP-Verified Credential because the user credential is certified by the IdP and used in the authentication steps with the RPs.

1.2 Prototype implementation and evaluation

While the exact application of the DDH idea to this particular type of SSO seems new, the cryptographic novelty is rather limited. This is not an issue, if the paper properly integrates this simple idea into SSO and demonstrate its viability for the considered use case. Unfortunately, this is not done in a satisfactory manner.

There is also a lot of room for improvement regarding the implementation and evaluation. In particular, discussing the user and RP integration of the new functionality, as the construction now shifts significant parts of the protocol to the user.

Our improvement Although the cryptographic novelty is not so complicated, our designs of PID_U, PID_{RP} and *Acct* work compatibly with the commonly-used SSO protocols, to finish the first practical SSO solution which solves the privacy problems caused by both the curious IdP and collusive RPs. We conceptualize the privacy requirements of SSO into an identity transformation problem, and this problem is formalized for the first time. It is worth noting that this problem is different from the one solved by EL PASSO, UnlimitID or PseudoID. Our identity-transformation design is independent of the authentication steps in the SSO protocols, satisfying the requirements of commonly-used SSO services.

We re-write the sections of designs and implementations. Firstly, the designs of identity-transformation functions are presented in Section 4.3, and then Section 4.4 explains the design specific for web applications. Our identity-transformation functions 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.

We finish the prototype system on top of open-source MITREid Connect, with about 75 lines of Java code. The user functions are implemented by about 350 lines of JavaScript code, which is portable for COTS browsers.

We compare UPPRESSO with (a) the PPID-enhanced OIDC protocol which only prevents the RP-based identity linkage and (b) SPRESSO which only prevents the IdP-based login tracing. It takes 310 ms for UPPRESSO to finish a login instance.

1.3 Proofs of security and privacy

It claims provable security, but there are no proper security models or proofs. There are already a number of troubling high-level claims on the security of UPPRESSO. Abstract and Sec 5.3 claim that UPPRESSO achieves the same security guarantees as OIDC and "UPPRESSO does not introduce any new role nor change the security assumptions for each role". This is not true: in standard OIDC, the user does not have to perform any crucial computations or elaborate certificate checks (beyond standard TLS authentication). However, UPPRESSO crucially relies on the user to verify certificates that the IdP issued to the RP, implicitly comparing it with the RP's TLS certificate to check it is the right RP, and extracting ID_{RP} from it. This is a critical part of the protocol in order to avoid phishing attacks. In standard OIDC, the user relies on the IdP to "authenticate" the RP and bind the id token to it. The security analysis focuses on the unlinkability of pseudonyms which is expressed in three games that I found hardly intelligible. There is no proper description of the games and the figure is barely readable. From what I could extract, it looks rather static though, i.e., with only limited possibilities for the adversary to interact with honest parties.

The brief privacy analysis is then complemented by a security analysis, that discusses cookies and seems misplaced. What is missing is a clear description of the desired security and privacy properties and the respective corruption setting and sound justifications and assumptions for the underlying building blocks. In particular, the binding of id tokens to a designated RP now requires much more care and should be analyzed.

There is a worrisome mismatch with related DDH-based OPRF/pseudonym constructions: All simple DDH-based OPRF constructions require a second layer of hashing and a Gap One-More type of assumption for the security proof – and a security model that takes the one-more-type of "forgeries" into account. OPRFs or blind pseudonym systems that achieve stronger security require zero-knowledge proofs of well-formed inputs. Neither of these approaches is used by UPPRESSO, but I would expect that a proper security analysis would reveal similar challenges here too.

Our improvement In UPPRESSO the security assumptions of a user is the same as that in OIDC, while a user in UPPRESSO conducts more cryptographic computations. Either in UPPRESSO or OIDC, the correct RP endpoint is ensured by the honest IdP, indirectly or directly.

We firstly list the four security requirements in Section 3.1, and prove that UPPRESSO achieves these requirements one by one in Section 5.1, especially RP designation. Meanwhile, in the appendix, the formal definition of secure SSO services is presented, and Lemmas and Theorems about the security of UPPRESSO are proved using an expressive Dolev-Yao style model.

We greatly improve the proofs of privacy in Section 5.2. The protections against IdP-based login tracing and RP-based identity linkage are proved formally and carefully.

In Section 3.3 Identity Transformation, we list the required properties of the identity-transformation functions. Note that these identity-transformation functions work compatibly for the commonly-used SSO protocols, but very different from the principles of certified RP-verified credential and anonymous credentials. Although in this manuscript we present the construction based on the elliptic curve discrete logarithm problem (and a construction based on the discrete logarithm problem is easy), we think more constructions are possible.

UPPRESSO is a system work rather than a theoretical one, so we do not discuss more possibilities. Anyway, we hope more constructions in the future, for example, a quantum-resilient one.

1.4 Usability

It contains several unsubstantiated claims regarding usability of the proposed solution.

- Compatibility with OIDC: "we require minimal modifications to the IdP and RP servers". OIDC outputs standard signatures to the RP, whereas UPPRESSO requires the RP to participate in an interactive protocol to blind and unblind the pseudonym, and check the related signatures. Overall, it is still a relatively lightweight protocol, but it is substantially different from OIDC and not compatible with any standard.

- UPPPRESSO & authorization code flow: "UPPRESSO can also support the authorization code flow of OIDC with small modifications" / "can be integrated into OIDC authorization code flow directly". In the authorization code flow, the IdP server and the RP communicate directly, which renders the authorization code flow unsuitable for any protocol where the IdP is not supposed to learn the RP's identity. The authors suggest to use TOR, but this would not be helpful as the IdP still need to know to whom it is talking.

Our improvement We have revised these statements. We list the modifications when we implemented the prototype system in Section 6.1: We add only 3 lines of Java code to calculate PID_U, about 20 lines to modify the way to send identity tokens, and about 50 lines to the function of RP Dynamic Registration to support the step of PID_{RP} registration. We also 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.

Although the computations in UPPRESSO are heavier than those in OIDC, the most compatibility issues are (a) a user keeps no long-term secret, except the credential in the authentication with the IdP, and the authentication steps are independent of the UPPRESSO protocol, as well as other commonly-used SSO protocols, and (b) an RP only needs to verify the identity tokens and also the PID_{RP} registration results, which are signed by the IdP, and no long-term user secret is bound or verified in these steps. These compatible designs will help the adoption and deployment of UPPRESSO.

In the authorization code flow of OIDC, the IdP server and the RP communicate directly, so that the IdP will know to whom it is talking. However, in UPPRESSO there is no RP identity in the access token which is used to return the identity token, and there is no RP identity in this identity token, either. So, if the access token is not verified using a symmetric key between the IdP and an RP (but by signatures instead), the authorization code flow will be supported well in UPPRESSO and no privacy information is leaked through these tokens, when anonymous networks (e.g., Tor) for the RP to ask for identity tokens.

Review 2

2.1 Motivation

Lack of motivation or justification on the hardness of combining existing solutions that separately solves IdP-based login tracing and RP-based identifier linkage problems.

How hard is combining existing solutions, SPRESSO and PPID, to prevent both privacy problems? Or are these two protocols not compatible with each other?

Our improvement Section 2.2 explains why PPIDs cannot be directly integrated in either BrowserID or SPRESSO. PPIDs are assigned in identity tokens based on the visited RP, but the IdP receives (a) nothing about the visited RP in BrowserID or (b) an ephemeral pseudo-identity of the RP in SPRESSO.

Then, in Section 3.2, we introduce the identify dilemma of privacy-preserving SSO. This identify dilemma is different from existing protocols involving authentication steps, as we explain in 1.1 Related work (for Review 1). This identity problem is really essential for a "pure" SSO protocol to protect user privacy. To the best of our knowledge, all existing solutions claiming to provide SSO services and protect user privacy against the two kinds of threats, are not "pure" SSO protocols. The authentication steps are involved in the protocols in fact, implicitly or explicitly. Other "pure" privacy-preserving SSO protocols such as BrowserID and SPRESSO, protect user privacy against only one kind of threats.

In fact, these privacy requirements are noticed by several researchers, for example, PseudoID proposed 11 years ago, but no perfect solution is proposed before UPPRESSO. This implies the difficulty of our work. PeseudoID provides similar services to EL PASSO, and a PeseudoID also needs to keep a long-term secret which is verified by the RPs.

2.2 Contribution and related work

Contributions are not clear for the individual building blocks. This may not be the first protocol that solves both privacy issues. An existing work is not compared with. The authors claim that this is the first system that solves both privacy problems at the same time. However, the following work also claims the same. I would like to see a comparison with this system "EL PASSO: Privacy-preserving, Asynchronous Single Sign-On".

Even though the privacy problems, which are being targeted to solve, are separately addressed by the prior art, it seems achieving privacy against both problems is not trivial because of the privacy dilemma behind the existing solutions. However, this point is not well-justified in the paper. Furthermore, it is not clear that if the individual solutions are also novel compared to the existing work.

Our improvement As also mentioned by Review 1, EL PASSO claims that it is an SSO protocol protecting user privacy against both the curious IdP and collusive RPs. However, it is not a "pure" SSO protocol. We call it a scheme of Certified RP-Verified Credential, in Section 2.3 Extended Related Works.

In EL PASSO and some other solutions, a user is authenticated by the RP with his credentials to prove that he owns the long-term secret; on the contrary, in the commonly-used SSO protocols the authentication steps between the user and the RP do not exist. In the commonly-used SSO systems and privacy-preserving SSO solutions such as BrowserID, SPRESSO and also UPPRESSO, authentication happens only between a user and the IdP, and an RP verifies only tokens generated by the IdP. Therefore, in EL PASSO a user needs to notify each RP when a credential is lost or compromised, because the user is authenticated by the target RP (in addition to the authentication by the IdP); however, in the commonly-used SSO protocols, the user only needs to renew his only credential at the IdP if it is compromised.

This is a very important reason that the existing SSO protocols are widely deployed in the real world. This feature enables an SSO system such as OIDC and OAuth 2.0 to adopt any authentication mechanism (password, one-time password, 2-factor authentication, FIDO, etc.) for the IdP to authenticate users, which is actually independent of the SSO protocol. This explanation is also presented in Section 3.3.

2.3 Usability

The paper also is not discussing about the user acceptance of this new design. Okay, it does not require major updates to build the whole system, but still requires modifications on IdP and RP sides. It would be very useful to include such an analysis about how the new protocol affects the usability of existing OIDC. Or, how easy it will be to deploy/convince RPs and IdPs for the new design. What is the user acceptance and usability of your system compared to existing OIDC protocols?

Our improvement This issue is explained by the prototype implementation in Section 6.1. We finish the prototype system on top of open-source MITREid Connect, with about 75 lines of Java code. The user functions are implemented by about 350 lines of JavaScript code, which is portable for COTS browsers.

We also 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 only needs to invoke these two SDK functions for the integration, by less than 10 lines of code.

2.4 Writing

Some sections are hard to follow due to excessive use of notations, and figures are not self-explanatory.

Some clarification on the text could improve the readability. For instance, it is not clear that $PID_{rp} = NuID_{rp}$ at page 7, bullet 2.3, is a regular multiplication or an operation on elliptic curve cryptography.

Our improvement We have polished this manuscript for several times. All notations are carefully explained in Tables 1 and 2. Figure 3 is re-drawn to explain the steps of UPPRESSO.