PseudoID: Enhancing Privacy for Federated Login

Arkajit Dey[1] and Stephen Weis[2]

[1] Massachusetts Institute of Technology, Cambridge, MA, USA 02139
[2] Google Inc., Mountain View, CA, USA 94043

Abstract. PseudoID is a federated login system that protects users from disclosure of private login data held by identity providers. We offer a proof of concept implementation of PseudoID based on blind digital signatures that is backward-compatible with a popular federated login system named OpenID. We also propose several extensions and discuss some of the practical challenges that must be overcome to further protect user privacy in federated login systems. [3]

1 Introduction

Internet users often manage login credentials for many accounts across multiple web sites. This is both an inconvenience and a potential security risk, as users often resort to reusing passwords. Users also become accustomed to typing user names and passwords in many different interfaces. This can leave users more susceptible to phishing, that is, having their credentials stolen by imposter websites.

Issues with managing web login credentials helped motivate the creation of web single sign-on (SSO) systems. One SSO model is for users to have a single *identity provider* (IDP) for all logins. Arbitrary web sites may then become *relying parties* (RPs), who delegate logins to the identity provider. The IDP handles authenticating the user and attesting an identity back to the RP.

Some proposals, such as Windows Live ID or Facebook Connect, rely on a centralized identity provider. Other systems, such as OpenID, allow users to have identities from among a federation of identity providers. Federated login systems like OpenID offer more flexibility to end users, since they are able to choose among many identity providers. Large web mail providers like Yahoo, Google, and MSN have all adopted OpenID [22, 19, 21] and are already capable of serving as identity providers for hundreds of millions of users.

While federated login systems like OpenID may streamline logins, they may create risks to user privacy. The core problem in both centralized and federated login systems is that all user logins to relying party web sites must flow through an identity provider. A user's identity provider can easily link together the various websites that the user visits. An identity provider could, for example, release data about which sites users visited without user consent.

In a federated system with many providers to choose from, users could avoid identity providers that abused privacy and use reputable firms. Unfortunately, honest identity providers may still be compromised and leak logs, or otherwise be compelled to reveal logs. Besides simply revealing which sites a user visits, identity providers often reveal personal information about users through extensions like OpenID Attribute Exchange (AX) [16] or Simple Registration (SREG) [17]. The goal of this exchange is typically to pass information like an email address, real name, or birth date from an identity provider to a web site. Automatically obtaining these data can greatly streamline the user sign-up process for relying parties.

Although most identity providers will prompt users whether they want to reveal this information, identity providers could reveal whichever data they want to a relying party. Thus, there is no way for a user to *selectively disclose* certain properties (e.g. age, gender, etc.) about themselves to a relying party. Much work has gone into developing cryptographic schemes for selective disclosure [4, 5, 3, 2], but these have yet to be adopted in practice.

In this paper, we outline a privacy-preserving federated login system called PseudoID and offer a proof of concept implementation as a pseudonymous OpenID provider located at `http://pseudoid.net`. The system utilizes blind signatures [7] as part of a blind signature service. This service allows users to generate a pseudonym that can be used to login to relying parties, but cannot be linked to their true identity. We also propose extensions based on zero-knowledge proofs [11] to support selective disclosure of user properties.

[3] Draft in submission, Feb. 2010

2 Federated Login Overview

Web users who want to use a particular website most often authenticate themselves directly to the site by entering a user name and password as in Figure 1. Maintaining many sets of user credentials across different sites carries a burden for the user and can lead to password reuse. Websites must carry the burden of creating accounts, storing credentials, and authenticating users. Account creation, or "on-boarding", is often a large barrier to signing up new users. It is not uncommon for the over half of sign-up attempts to be abandoned.

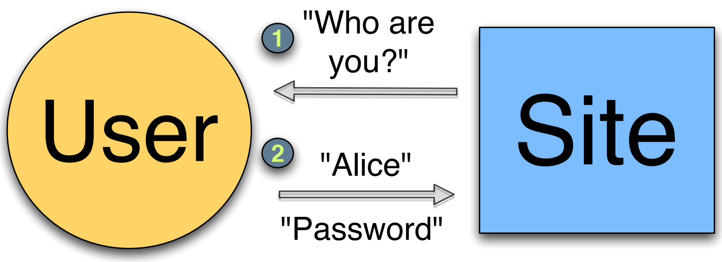

Fig. 1. A typical web login system where users log into websites by entering site-specific credentials.

Federated login systems, on the other hand, extract authentication as a service in its own right. Just as websites rely on third-party services for traffic analysis, CAPTCHA verification, or file hosting, they can also rely on separate services for authentication.

Federated login adds a third party to the interaction between the user and the website: the *identity provider* (IDP). Instead of authenticating herself to the website directly, the user authenticates herself to the IDP. The IDP then returns a user identifier to the website. Thus, the website is often referred to as the *relying party* (RP) since it relies on the identity provider for authentication.

Federated login alleviates the need for websites to store user credentials, making them less desirable targets for attackers who want to hijack user accounts. The user benefits from federated login too. Instead of managing separate login credentials for every website he wants to use, the user can just log into a single identity provider.

Systems like Facebook Connect and OpenID 2.0 are able to offer one-click logins for relying parties, which greatly simplifies the login process. For example, Plaxo, a social networking and address book site, performed a two-click OpenID login experiment where 92% of users successfully completed registration after starting the sign-up process [13]. In contrast, on-boarding abandonment rates of 50% are common for many websites.

Accordingly, federated login systems are being adopted by a growing number of Internet sites, particularly by large web mail providers and social networks. Several different federated login technologies have arisen over the years, such as Microsoft Passport (now Windows Live ID), OpenID, Facebook Connect, and SAML.

However, popular federated login systems have generally been designed without privacy as a primary concern. Subsequently, current widely-used federated login systems could put sensitive user data at risk. The problem of user privacy is indeed magnified in federated login systems since identity providers act as stewards of user data for multiple websites. This not only makes identity providers more appealing targets to attackers, but also more likely to be subpoenaed for user records.

2.1 OpenID: A web-based federated login system

OpenID is a popular federated login system that we focus on for a proof of concept implementation. In OpenID, users can claim identifiers in the form of URIs. To login to a website that supports OpenID, the user enters his OpenID URI and the relying party redirects him to his identity provider's page. The identity provider authenticates the user through its choice of authentication system (e.g. passwords, smart cards, etc.) and then returns the user to the relying party with

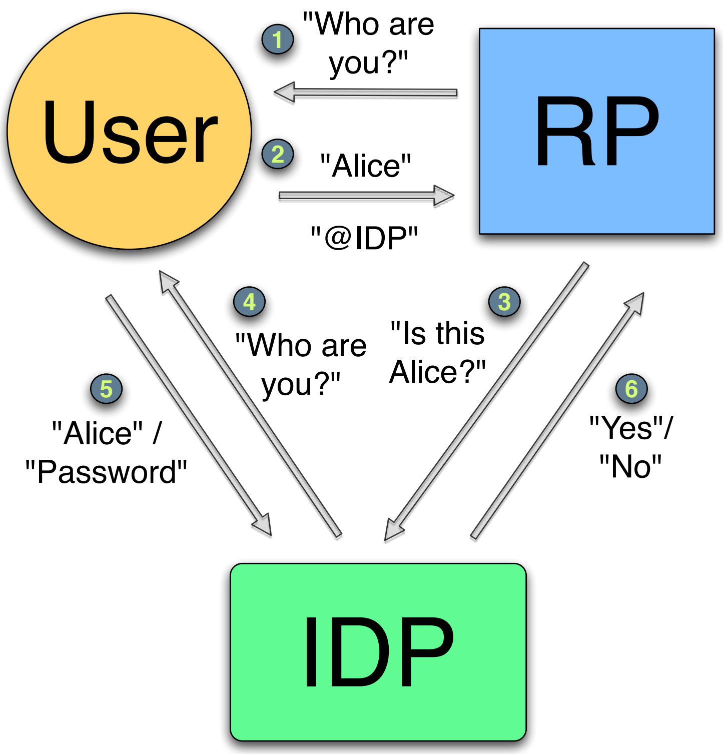

Fig. 2. A federated login system: (1) The RP prompts the user for an identity. (2) The user enters an identifier that specifies an IDP. (3) The RP requests that the IDP authorize the user. (4) The IDP prompts the user for her credentials. (5) The user presents her credentials to the IDP. (6) The IDP returns the result of the authorization to the RP.

either a positive or negative assertion that the user owns the claimed identifier. If the relying party receives a positive assertion from the identity provider, it may allow the user to enter the site under the name of the claimed identifier.

With the advent of OpenID 2.0 [15], the protocol also began to support the concept of *directed identity* [6] or *private digital addresses* through a new feature called *identifier select* [18]. This allows the user to just specify the URI of his identity provider instead of claiming a personal identifier when logging into a website. The site then redirects the user to the identity provider as before, but the identity provider now has the opportunity to select an identifier for the user. Upon successfully authenticating the user, the identity provider returns the selected identifier to the site.

This allows the identity provider more flexibility in selecting identifiers for its users. For example, the identity provider may decide to return a different identifier for the same user for different relying parties in order to implement true directed identity as defined in Kim Cameron's Laws of Identity [6]. Indeed, some OpenID providers like Google do return a per site unique identifier rather than a globally unique identifier for its users.

2.2 Privacy Concerns in Federated Login

In federated login systems, users entrust identity providers to manage their identity, so privacy concerns may seem relatively minor. After all, in OpenID and most other major single sign-on systems, a malicious identity provider could easily impersonate users to relying parties. However, even if an identity provider is not corrupt, there are privacy concerns for *honest, but retentive* providers who may reveal user data due to a security breach or a legal subpoena.

The core privacy issue with widely-deployed federated login systems is that a user's identity can be correlated with the sites she logs into. For example, OpenID identity providers will authenticate users, then redirect them back to a relying party. This makes it trivial for an identity provider to know all the web sites a user logs into. The same is true for Live ID or Facebook Connect.

One might develop a different federated login flow where a user acted as an intermediary between relying parties and identity providers. The user could avoid passing any information about the specific relying party to the identity provider. In this case, the identity provider might return an anonymized identifier via the user to the relying party. However, if the provider colluded with the relying party, they could link the user's real identity with the account on the relying party.

Alternatively, an identity provider could abstain from logging, could try to anonymize or delete identifying information, or could simply destroy logs completely. Logs are retained for many valid reasons including analytics, diagnostics, and security auditing. In practice, abstaining from logging is often not a viable option.

Removing or anonymizing identifying information ex post facto is one option, but has proved difficult in practice. Supposedly anonymized logs released by AOL [1] and Netflix [14] were both de-anonymized to some extent. An identity provider would need to be vigilant and thoroughly scrub logs to remove identifying data. They would also need to ensure that identifying data was not being unintentionally logged by an unrelated service or a different layer of the stack.

Another issue is that both logs anonymization and destruction may be subject to data retention laws that specify minimum retention periods [10]. Identity providers may be legally compelled to collect identifying data and retain it for a minimum period. In some jurisdictions, authorities have broad powers to rapidly seize these data, often without user knowledge.

Given these considerations, we will focus on privacy in a setting where: (1) identity providers are unable to assure that logs are not retained, and (2) identity providers may be compelled to reveal logs at some time.

There are several real-world risks where *honest, but retentive* identity providers may threaten user privacy. One risk is simply if the provider were compromised and logs were leaked by an attacker. Another risk is for providers operating in jurisdictions where logs may be seized without due legal process.

Identity providers may have an interest in *not* being able to link a particular user's identity to logins on a relying party. An identity provider may want to provably show that they cannot link logins on a relying party to a particular user. With this in mind, we informally define what it means for an identity provider to be *private* in Section 3. Section 4 propose a practical system that meets this definition.

3 Properties of Private Federated Login

For the scope of this paper, we are going to focus on privacy in federated login systems with an OpenID-like login flow illustrated in Figure 2. The relevancy to other types of federated login systems may vary.

We assume that identity providers have a set of users that can be thought of as “real” identities. Users may possess credentials which are presented to identity providers for authentication. If the credentials are valid, identity providers will return some identifier to a relying party. This identifier may be of any form, i.e. a “real” user name, a pseudonym, a value derived from the credentials, or even a random value. In order to contrast these different behaviors, we will first state several informal properties.

Definition 1 (One-wayness). *An identity provider is one-way if given a specific identifier, attackers have no significant ability to cause identity providers to return that value.*

One-wayness means that users actually “own” their identities and people cannot imitate them on relying parties. For example, one cannot typically login with a specific user name without knowing that user’s password. A trivial example of an identity provider that is *not* one-way is one where an identity provider will assert any identity without authorizing the user. We’ll refer to this as the “Yes IDP”. Users of a Yes IDP could log into relying parties with arbitrary identities, but would not be able to prevent other people from using the same identities.

Definition 2 (Consistency). *An identity provider is consistent if users may present credentials that will return the same identifier over multiple sessions.*

The consistency property means that users can have long-lived identities on relying parties. That is, users can log in as the same identity to a relying party any number of times. A trivial inconsistent identity provider would be one which returned a random value for each login, or a “Random IDP”. Users of a Random IDP would be anonymous on relying parties and could not be linked to their real identities, but would not be able to establish long-lived accounts on relying parties.

Definition 3 (Unlinkability). *An identity provider is unlinkable if given a transcript of an authentication event and a set of users, an attacker has no significant advantage in distinguishing the user being authenticated.*

Unlinkability is intended to capture the notion that an attacker who obtains access logs from an identity provider and relying party should not be able to tell which “real” user was logging in. OpenID identity providers are generally linkable in practice, although there are exceptions. An attacker obtaining a user’s credentials or identity provider access logs would be able to trivially see which user was associated with a particular identity on a relying party.

Note that this property is not specific to OpenID or a flaw in the OpenID protocol. Instead, it’s an artifact of how real world identity providers typically authenticate users: with user names and passwords. For example, an attacker may observe “Alice” authenticate herself to an identity provider and the identity “Bob” returned to a relying party; linking her real identity to her pseudonym.

Thus, an unlinkable relying party must not require any identifying information about their real users during the federated login protocol. The Yes and Random IDPs mentioned before are in fact unlinkable, but are not practical in many use cases since they are respectively not one-way or consistent. A practical unlinkable system must be both one-way and consistent. We will present such a system in Section 4.

4 PseudoID: A privacy-preserving federated login system

PseudoID is designed to be a one-way, consistent, and unlinkable federated login system. It consists of a token service used during setup, and a private identity provider used for sign-ons. The user has an account with the token service, which may be a persistent, “real” identity like an email address. During setup, the user logs on to the token service using a familiar authentication scheme, such as entering a user name and password.

The user then requests an access token from the token service that is bound to a desired pseudonym. When logging into a relying party, the user presents this token to an identity provider. The identity provider will verify the authenticity of the token and return the user’s pseudonym to the relying party.

To be unlinkable, the access tokens must be generated such that even if both the token service and identity provider are compromised, the user's "real" identity with the token service cannot be linked to their pseudonyms on different relying parties. PseudoID achieves this property by employing *blind signatures*.

4.1 Blind Signatures

Traditional public key digital signature schemes [9] consist of a private signing function S known only to the signer and a public verifying predicate V. Then for any message m that is provided to the signer to be signed, a verifier can check that $V(m, S(m))$ is true. It is infeasible to produce the signature $S(m)$ without knowledge of the signing function S.

Blind signature systems [7] augment this traditional scheme with a blinding function B and its inverse unblinding function B^{-1}, such that $B^{-1}(S(B(m)) = S(m)$ and both functions are known only to a user getting a message signed.

In a blind signature scheme, the user wishes to obtain a signature $S(m)$ on some message m without revealing the contents of m to the signer. To do so, the user sends the blinded message, $B(m)$, to the signer that leaks no information about m. The signer then signs the blinded message and returns $S(B(m))$ to the user. Finally, the user unblinds this signed message to obtain

$$B^{-1}(S(B(m))) = S(m),$$

a valid signature on m that can be publicly verified.

One example of a blind signature system is Chaum's RSA blind signatures. In a standard RSA digital signature system, the public parameters are a modulus n and an exponent e. Only the signer knows the private exponent d. To blind a message m prior to sending it to the signer, the user multiplies it by a random blinding factor r to produce $B(m) = mr^e$. The signer signs $B(m)$ to produce

$$m^d r^{ed} \equiv m^d r \pmod{n}$$

by Euler's theorem. Since the user can compute r^{-1}, he can unblind the returned signature to obtain

$$m^d \pmod{n},$$

a valid signature on the original message m.

4.2 Blind Token Service

PseudoID employs a blind signature service (BSS) or *blind signer* that generates blinded access tokens. These tokens are redeemed with an identity provider and used to derive identifiers that are returned to relying parties. This setup phase is outlined in Figure 3.

During a setup phase, the user will visit the blind signer and login to an existing account. The user then selects a pseudonym that they want to use on a relying party and a secret value. This pseudonym and random secret value are bundled into an access token that the blind signer will sign. That is, the user will prepare a token $T = (pseudonym, secret)$.

To prevent the signer from being able to link a user with her pseudonym, the user first blinds the token $B(T)$ before sending it to the blind signer. The blind signer will sign this token without knowing its contents and return it to the user as $S(B(T))$. Upon receiving the singed token back from the service, the user unblinds it to obtain a signed token $S(T)$ that contains the user's chosen pseudonym and secret value. Note that the blind signer will not see the user's pseudonym or secret value in the clear; it will only see the blinded token.

4.3 Private Identity Provider

The identity provider relies on the blindly signed tokens to be able to authenticate users without forcing them to reveal their identity. When a user is redirected to her identity provider by a relying party, the provider checks whether the user has an access token that has been signed by the blind signer.

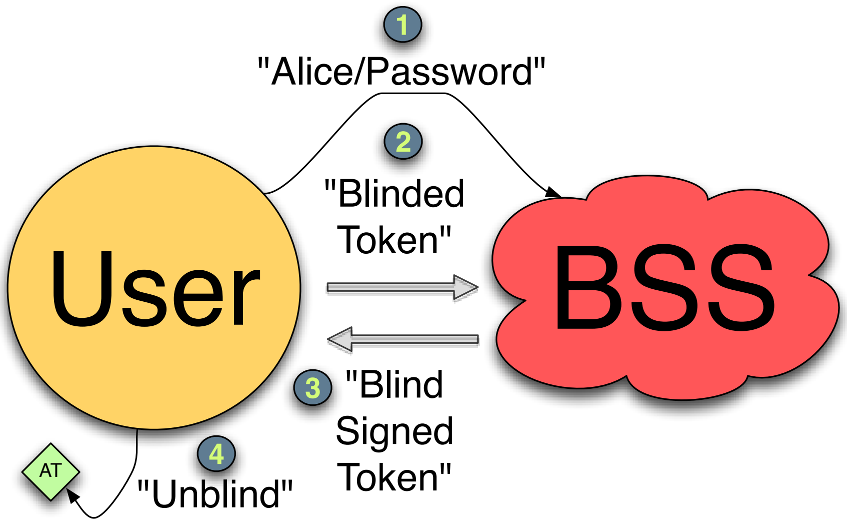

Fig. 3. Blind Signer Setup: (1) User first authenticates herself to the BSS normally. (2) Then the user sends the BSS a blind token to sign. (3) The BSS signs the token and returns it. (4) The user unblinds the blind signed token to obtain a valid, untraceable access token (AT).

The signature on the token may be either publicly verifiable or privately verifiable. In the former case, the identity provider can verify the signature on the access token using the blind signer's public key. In the latter case, the identity provider could send the token to the blind signer and ask them whether they signed it. The sign-on process using an access token in the publicly verifiable case is illustrated in Figure 4. If the access token is valid, the provider is only assured that the user has been authenticated by the blind signer. Thus the provider knows that the user is a valid user of the blind signer, but does not know which user.

Given a valid access token from the user, the identity provider will compute a one-way, collision-resistant function F on its value and return that as part of an identifier to the relying party. For instance, if the access token contains $T = (pseudonym, secret)$ the identity provider could assert $(pseudonym, F(T))$ as the user's identifier to the relying party. Given the properties of F, attackers will not be able to invert the value $F(T)$ to obtain T or find a value T' such that $F(T) = F(T')$.

4.4 Properties of PseudoID

We informally argue that PseudoID can meet the properties described in Section 3.

Claim (1). **PseudoID is one-way** The access token T that is presented by the user to the IDP will contain a random secret value. The IDP will then compute a one-way, collision-resistant function of that token $F(T)$ that is presented as part of the user's identity. An attacker will only be able to cause an IDP to return that same identity if they are either able to learn the secret, invert $F(T)$, or find a colliding token value T' such that $F(T) = F(T')$. Thus, attackers will have no significant ability to cause the IDP to return a given identity.

Claim (2). **PseudoID is consistent** The private identity provider will always return the same identifier for a given access token. It's output is completely deterministic, and thus consistent.

Claim (2). **PseudoID is unlinkable** Suppose an attacker has all the logs of the blind signing service, all blinded access tokens, and all unblinded access tokens used to authenticate to the IDP. The attacker will win if they are able to link a specific blinded access token to a specific unblinded access token, i.e. revealing a user's identity. However, this is precisely what blinding prevents. PsuedoID uses blinding in exactly the same fashion as Chaum's untraceable payment scheme [7]. Thus, PsuedoID is unlinkable based on the properties of the blind signature scheme.

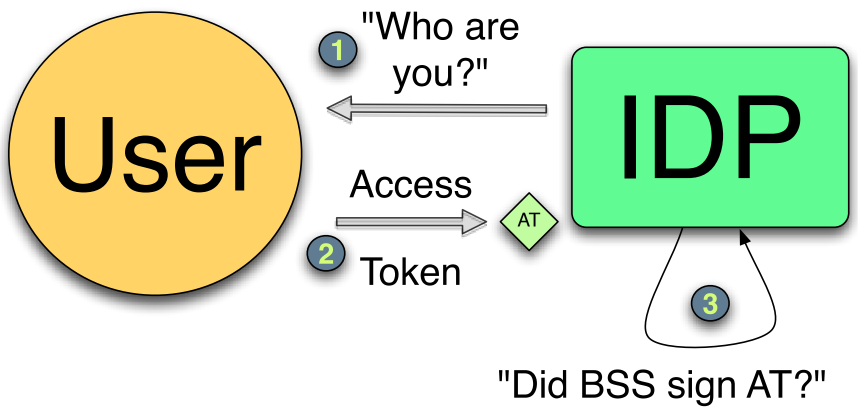

Fig. 4. Identity Provider Sign-on with Blind Signed Access Token: (1) IDP asks users to authenticate (2) User supplies access token rather than true identity or credentials (3) IDP verifies whether BSS signed the token using BSS's public key.

4.5 OpenID with PseudoID

PseudoID is practical to implement in a web setting, such as for OpenID. We have implemented a proof of concept blind signing service and identity provider available at `http://pseudoid.net`. The proof of concept blind signer is implemented as a web service. Users visit the blind signer and prepare a blinded token signed with JavaScript (see Section 4.6 for a discussion of some security caveats).

The blind signer will blindly sign this value and return it to the user. This value is unblinded and stored as a cookie in the user's browser. This cookie will be set on the identity provider's domain. The identity provider itself may be slightly modified from existing OpenID providers. It simply needs to read an access token from a cookie on the user's browser, verify a signature on it, and return the pseudonym it contains to a relying party. From the user's perspective, this eliminates the need to retype a user name and password on an identity provider.

PseudoID identity providers are fully compatible with existing OpenID relying parties. Existing relying parties do not have to change anything about their current OpenID flow in order to be able to accept users from private identity providers. From the perspective of the relying party, a private identity provider is indistinguishable from a regular provider. A private provider simply uses a different authentication mechanism than most other identity providers, but it still participates in the same federated login flow outlined in Figure 2.

4.6 Caveats of the proof of concept implementation

The proof of concept PseudoID implementation stores a user's access tokens as cookies in the browser. This is not the ideal solution. Users who clear cookies frequently or use a private browsing mode would lose their blinded access token. Cookies are also set in a single browser, and thus it would be complicated to extend a user's pseudonym across different machines.

There is also a security issue with setting a cookie on the identity provider's domain. JavaScript executing on one domain cannot set a cookie on another. The proof of concept PseudoID implementation must make a call to the identity provider to set an unblinded cookie during the setup phase. This means that access logs on the blind signer and the identity provider could be joined to correlate the user's login with the value that was set as a cookie.

Even if logins are private, user requests can still be tied to a particular IP address. Anonymization on the network level is an independent risk and may be mitigated by the use of web proxies or anonymous browsing technology like Tor [20].

In practice, users are often tracked by cookies as well as by IP addresses. One risk of the proof of concept system is that a malicious blind signer or identity provider may try to set tracking cookies on the user's browser while they are logged in with their real identity. Thus, a user would need to scrub all cookies except their access token from their browser. This is not practical from a usability standpoint. Another issue is that blinding and unblinding is being done with JavaScript in the user's browser. This code must be trusted, otherwise it could leak their identity.

5 Extensions and Future Work

The caveats of the proof of concept PseudoID implementation make it impractical for general use. There are both unresolved usability and security issues which would need to be addressed in it and private federated login systems in general.

5.1 Simplified Cryptography in the Browser

Modern browsers are equipped with support for a broad range of cryptographic functionality to support SSL/TLS. Yet, it is difficult for a typical web application to make use of it. In the case of PseudoID, server-side JavaScript was used to blind and unblind tokens. Using JavaScript is both inefficient and insecure. Basic cryptographic functionality has to be reimplemented in JavaScript and interpreted, rather than using the native cryptographic libraries already available in the browser.

There is also a question of where the JavaScript code comes from. If it is hosted on a server, it may later be substituted with malicious code without the user's knowledge. For example, if the host of the JavaScript code were compromised, an attacker could inject code to leak the user's identity.

Browser cryptographic support could be made available through a browser plug-in or extension, but this is a barrier to adoption and difficult to support on multiple platforms. PseudoID and many other applications could benefit from a simple, standardized, and cross-platform API to client-side cryptographic services.

5.2 Browser Storage and Cross-Domain Communication

Another practical challenge in the proof of concept implementation was communicating cryptographic tokens across domains subject to the constraints of the *same-origin policy*. In this case, a page on the blind signer's domain cannot set a cookie on a different domain.

This was a difficulty because we wanted to set an *unblinded* token that was readable from the identity provider's domain, but not visible to the blind signer. The workaround was to load a hidden cross-domain iframe. This technique was used to allow the blind signer to set a cookie on the identity provider's domain. That represents a privacy risk, since it would be easy to correlate logs between the blind signer and the identity provider.

PseudoID could benefit from a more flexible browser storage model than cookies, and the means to pass messages from one domain to another using the browser as an intermediary. Several features proposed in HTML 5 may help facilitate this [12].

5.3 Selective Disclosure

In the current version, PseudoID access tokens contain a user-selected pseudonym and a random nonce. Tokens do not contain any meaningful semantics nor any properties of the user's real identity. By using zero-knowledge proofs, one may extend the blind signer to support selective disclosure. There is a broad range of literature on this topic [8, 4, 5, 3, 2].

The basic idea is that users will engage in a zero-knowledge proof with the blind signer. They will prove that the contents of blindly signed messages convey some meaningful data or have a proper semantic form. For example, the user may prove that a blindly signed message contains a bit value representing "Is this user over 18 years of age?" that is true for their real identity, without revealing any other information about the message. Another example use is obtaining a token with an expiration time in it. The user would prove to the blind signer that a blinded message contains a valid expiration time, without revealing any other knowledge of the message.

By allowing tokens to have these types of semantics, identity providers will be able to offer more fine-grained access policies. In the simple PseudoID system, identity providers can only verify a signature on a token – all they learn is that a blind signer signed it at some point in time. If tokens had semantics, they could, for instance, only allow access to users with tokens that were issued within some time period.

From a web-based implementation standpoint, performing zero-knowledge proofs in the browser requires better support for both cryptography and storing persistent values. While it is possible to implement zero-knowledge proof systems with a JavaScript and cookie approach, this would have the same security issues that the proof of concept PseudoID implementation has.

References

1. Barbaro, M., Zeller Jr., T.: A face is exposed for AOL searcher no. 4417749. New York Times (August 9, 2006), http://www.nytimes.com/2006/08/09/technology/09aol.html
2. Camenisch, J., Groß, T.: Efficient attributes for anonymous credentials. In: Proceedings of the 15th ACM conference on Computer and communications security. pp. 345–356 (2008)
3. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: EUROCRYPT. pp. 302–321 (2005)
4. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: EUROCRYPT. vol. 2045, pp. 93–118 (2001)
5. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: CRYPTO. pp. 56–72 (2004)
6. Cameron, K.: Laws of identity, http://www.identityblog.com/?p=354
7. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology: Proceedings of CRYPTO. pp. 23–25 (1982)
8. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Communications of the ACM 28(10), 1030–1044 (1985)
9. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory IT-22(6), 644–654 (November 1976)
10. EU data directive (1995), http://www.cdt.org/privacy/eudirective/EU_Directive_.html
11. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal of Computing 18(1), 186–208 (February 1989)
12. Hickson, I., Hyatt, D.: HTML5 (December 21, 2009), http://dev.w3.org/html5/spec/Overview.html
13. Kirkpatrick, M.: Comcast property sees 92% success rate with new OpenID method. ReadWriteWeb Article (February 10, 2009), http://www.readwriteweb.com/archives/comcast_property_sees_92_success_rate_openid.php
14. Narayanan, A., Shmatikov, V.: Robust de-anonymization of large sparse datasets. In: Proceedings of the 2008 IEEE Symposium on Security and Privacy. pp. 111–125 (2008)
15. OpenID authentication 2.0, http://openid.net/specs/openid-authentication-2_0.html
16. OpenID attribute exchange 1.0, http://openid.net/specs/openid-attribute-exchange-1_0.html
17. OpenID simple registration extension 1.0, http://openid.net/specs/openid-simple-registration-extension-1_0.html
18. Recordon, D., Reed, D.: OpenID 2.0: A platform for user-centric identity management. In: Proceeedings of the second ACM workshop on Digital identity management. pp. 11–16 (2006)
19. Sachs, E.: Google moves toward single sign-on with OpenID. Blog (October 29, 2008), http://googlecode.blogspot.com/2008/10/google-moves-towards-single-sign-on.html
20. TOR: The onion router project, http://www.torproject.org
21. Windows live ID becomes an OpenID provider. Blog (October 27, 2008), http://winliveid.spaces.live.com/Blog/cns!AEE1BB0D86E23AAC!1745.entry
22. Yahoo! announces support for OpenID. Press Release (January 17, 2008), http://yhoo.client.shareholder.com/press/releasedetail.cfm?releaseid=287698