Federated Identity as Capabilities

Harry Halpin and Blaine Cook

[1] W3C/MIT, 32 Vassar Street Room 32-G515 Cambridge, MA 02139, United States
hhalpin@w3.org,
WWW home page: http://www.ibiblio.org/hhalpin
[2] 30 Ballater Road, London SW2 5QR, United Kingdom
romeda@gmail.com,
WWW home page: http://blog.romeda.org/

Abstract. Keywords: identity, capabilities, federation, BrowserID, OpenID

1 Introduction

As the Web leads to a proliferation of services that each require some form of authentication, the problem of sign-in has returned with a vengeance: Current approaches that federate a single identity across multiple services ("federated identity") solve this problem at the cost of sacrificing user privacy. Due to an inability to solve sign-in and data-sharing in a user-centric manner that is usable by developers, currently the most popular identity system is the non-federated Facebook Connect [13]. Also, most federated identity solutions are not standardized via a standards body like the IETF or W3C, so there is not a single alternative federated model capable of being easily deployed across browsers. As we can not determine the security or privacy characteristics of proprietary solutions or iterate through the large list of federated identity systems produced in both the academic literature and in enterprise computing, we will confine our privacy analysis to two open-source efforts that have attracted attention and user-bases: OpenID Connect [2], as currently implemented by Google and Microsoft, and BrowserID [1], also called Mozilla Personae. After discussing the design of both of these federated identity solutions, we outline a "capabilities-based" approach that considers identity simply to be a set of capabilities that a user can prove to a service that they possess without revealing their identity. Then we show how we can extend existing federated identity approaches such as OpenID Connect [9] and BrowserID [1] with capabilities based on using key material, and outline the future work necessary to integrate such work with anonymous credentials and zero-knowledge proof systems.

1.1 Terminology and Assumptions

In this work, we assume there is a user that is sending some kind of information to a *relying party*, a services that wish to access verified identity claims. The source of the identity claims is called an *identity provider*, a service that stores and can

possibly verify identity claims on behalf of a user. The common example would be having a user send their username and password combination to Facebook via Facebook Connect, the identity provider, to sign-on to a third party service such as music-sharing service Spotify, the relying party [13]. Spotify also may require some information from Facebook, such as the full name of the users and the songs they like in their Facebook profile, in order to customize their service. This information required by the relying party from the identity provider are considered *identity claims*.

We assume that users don't care about sign-in, they just want to access a service as quickly as possible. It is well-known from anecdotes from companies that forcing users to register a new account and sign-in leads to almost half of users being so discouraged that they do not even bother to register a new account. Developers shouldn't care either: For most developers, all they want is the ability to authenticate a user in order to provide them with some sort of access to a system, possibly with a persistent state tracked by their system. Unless their particular service happens to be in the business of buying and selling user's personal data - as many "social" services are - the developer simply wants to authenticate and store just enough state about the user in order to have the service fulfill its functionality. This state is usually thought of as a set of verified identity claims, such as that a user's first name is "Bob" and that has the email address *bob@example.org*. These claims are verified in a number of different ways, such as having the user entered them into a form when registering a new account (as is the most common practice today) ranging or having the claimed stored locally encrypted in a device and transferring them with some time-limited scope that includes a digital signature. In general, *identity* can be broadly construed as a set of claims about a user or a particular persona of the user, where a *persona* is a set of claims that exposes only (a possibly "fictional") facet of the user.

We also assume that users would like to minimize the amount of disclosure of personal information to be verified to the relying party. For example, minimizing the collection of personal data is sometimes required by regulatory and government authorities, as exemplified by NIST's "Guide to Protecting the Confidentiality of Personally Identifiable Information" [12]. However, there are also everyday examples that do not involve regulation. For example, a user may want to sign-into a site such as Youtube that features videos that have content that require the user to identify themselves as over a certain minimum age, such as 18. However, we assume the user would prefer not disclose their entire name and other personal information in this case to the relying party. Not only will we assume that the relying party is potentially untrusted, but we would also like to assume that the trust in the identity provider should be minimized. So, we would like to minimize the amount of information the identity provider observes about the actions of the user at various relying parties, which can be considered the linkability of the transactions with relying parties by the identity provider. We will also note that this can be generalized to any passive adversary that is observing the identity provider.

1.2 Related Literature

Indeed, these problems are not new, but have already to a large extent theoretically been solved by the advent of anonymous credentials [4] and accompanying work related to zero-knowledge proofs [8]. The European Commission has funded a number of large-scale projects in this area that have produced open-source implementations of anonymous credentials with zero-knowledge proof systems, such as *PrimeLife*[3] and *ABC4Trust*[4] as well as commercial systems such as Microsoft's UProve. [5] However, the concepts behind cryptographically anonymous credentials and zero-knowledge proofs so far have not yet been introduced into common federated identity schemes, much less incorporated into the code of popular browsers. One reason is likely that the conceptual complexity of anonymous credentials and zero-knowledge proofs seems not to match the desire by companies or even regulatory authorities to define the flow of information of personal data. Another reason is that the ability to create anonymous credentials with efficient zero-knowledge protocol signature schemes has still been an active area of research, although efficiency gains are very promising [3]. Perhaps the most likely reason is that these products do not easily seem to graft on top of either the existing Web server or client (browser) architecture, including increasingly high-profile federated identity schemes such as OpenID Connect [2] and BrowserID [1]. So in this paper, we focus on existing deployed Web-based identity systems and their common (if simple) attributes such as e-mail verification, although we present e-mail verification and identity claims within a new framework of generalized capabilities verified by digital signatures. Second, our proposed simple extensions of existing federated identity scheme should be capable of using *any* digital signature, including both conventional digital signatures from identity providers like Facebook and Google as well as the possibility of using the kinds of signature schemes needed by anonymous credentials [5].

1.3 Capabilities

There has long been a debate between access-control lists and capabilities as a way of providing authorization restrictions [6]. As put by Laurie, "a capability can be thought of as a 'handle' representing some operation on some object. Possession of the capability is all that is required to perform that operation." [11]. While it may seem counter-intuitive to remove the explicit identity of the user from an identity-based system, the framework of capabilities is extremely useful as it can also provide a much more fine-grained level of control than per-user access control. Furthermore, this lack of an explicit user identifier is nonetheless possibly a requirement for data minimization principles and anonymous credentials. In the next section, we show that current federated identity-based systems can be considered as proving the possession of a capability by a user (rather

[3] http://primelife.ercim.eu/
[4] https://abc4trust.eu/
[5] https://research.microsoft.com/en-us/projects/u-prove/

than a program), with the possession of an email address and a password to access being the 'handle' that defines the operation of a user authenticating their identity.

2 Identity as Capability to Check Email

2.1 Problems and Example

Currently, despite years of effort around solutions like OpenID [9], the primary form of identity on the Web is still a HTML form for filling out claims, which in return delivers a cookie that tracks a user's session state. Between sessions, users have to enter a password, which is an easily phishable symmetric shared secret. Without proper security considerations, such as enforcing TLS, the cookie that proclaims the session state is also easily swiped by a man-in-the-middle attack. Not only is the security poor, but so is the user-experience: For users, sign-in is not only just a means to an end, it's often a confusing annoyance. For developers, the situation is unsatisfactory: Instead of having to re-implement sign-in code for their services, most developers would prefer to use a commonly available library.

2.2 Using Email Addresses as Identifiers

In practice identity for Web-based services can be reduced to a number of rather simple primitives. Let us walk through a simple example. Assume "you" are the service provider, providing your service via a web-site. The user who is visiting a site knows who they are. You do not, yet. The first element of a sign-in system allows the user to tell you who they are, i.e. to allow the user to identify themselves. This is thought of both in federated systems and non-federated systems as requiring the user to present an identifier. What the identifier hopes to identify is a single individual human (or perhaps also a personae thereof). However, natural language names will not work as these can be ambiguous. Assigning numbers (as done with credit cards, telephones, and the like) is a possibility, but are difficult to remember and require a centralized database lest they also suffer from ambiguity. Currently, email addresses of the form *name@domain* seems to be the best distributed system for identity, as they are generally associated with individual humans and tend to be globally ambiguous. Email addresses tend to be rather natural identifiers for individual humans, as they seem to roughly approximate other naming schemes that attach some local context in order to disambiguate a name. For example, "Jesus of Nazareth" is approximately equal to *jesus@nazareth.israel*, or to use a more modern example, "Blaine Cook, 30 Ballater Road, London" can be thought of as *blaine.cook@30.ballater-road.london*. In the global system of the Internet, the domain name is the necessary context needed to disambiguate a string identifier that names an individual person. Although the domain name system is as centralized under as phone numbers or credit card numbers, it offers a number of additional capabilities and a well-understood extension mechanism, i.e. the purchase of a new domain name from a domain registrar and the setting up of an SMTP server.

There are numerous other options for what identifier can be used beyond email addresses. Often a simple string for a user name is all that is required, assuming the user has previously registered with the service. Yet users often forget the particular string they may use as their identifier for a service (or be forced to used, due to some other user registering the same string before them). More importantly, usernames are usually "disguised" email addresses, as they are directly tied to email addresses in the registration process. Certain systems such as earlier versions of OpenID [9] and WebID [14] have claimed that a URI by itself (such as *http://www.example.org/bob*) is a good identifier, however, in practice these URIs are in general tied to e-mail addresses as well (the owner of the domain of the URI as determined by the email of whoever registered the domain name or controls on behalf of the user, as easily determined by protocols such as WHOIS). Also, as users do not remember URIs and do not imagine that they themselves are URIs, so federated identity systems such as OpenID 2.0 based on URIs confused users (and thus, were not used) despite considerable deployment by companies such as Google and Yahoo! [9]. In comparison to using self-signed certificates like WebID [14], email addresses tend to be better than cryptographic credentials stored in the browser as the browser is not the thing that is being identified in a sign-on process: The human is the thing that is being identified, and the human may use multiple browsers across different devices. If a system is based entirely on using cryptographic credentials in the browser but does not include a prompt for an email address, then the user is unable to sign in if they lose their device. This is one of the major problems with losing private keys in general, unless keys are synchronized amongst multiple devices, but in this case the user is at the behest of whoever runs the key hosting service that manages the keys of a user in absence of their client device. This is not to say the approaches are completely incompatible: When a client device goes away (seized by police, lost in a river, or borrowed from a friend) and with it your cryptographic tokens, a service can still use e-mail addresses to identify a user and authenticate claims, and even revoke and re-provision key material for client devices.

2.3 Checking E-mail as a Capability

The last step is to authenticate the user, in which the service needs to prove that the user is who they say they are. This is done by proving that the user controls the e-mail address and can respond to an email, usually a temporary URI embedded in an email with some secret code. This also addresses concerns that certain email addresses can be checked by more than one person, such as a couple (such as "Sheri" and "Stan" in *sherinstan@gmail.com*) or that e-mail addresses can be recycled by different users over time (as done by Yahoo!), but these are cases can be dealt with by determining if someone can check the email. Thus, the vast majority of the time, all sign-in systems do from a the perspective of security is validate that a user controls an email address. Traditional username-password combinations are an often unnecessary convenience as if a site has a link to recover a forgotten password via checking for an email from the service, then the

password saved on the site for a user is irrelevant from a security perspective, since anyone with access to their email account can reset the password. Whenever a site uses a *username-password*, this almost always becomes a *email-password-reminder* token. The majority of existing identity systems in the wild are at their foundation simply systems that prove the user has the capability to check email.

3 Privacy Issues in Federated Identity Systems

Indeed, existing federated identity systems simply hope to re-use the same identity, an *email-password-reminder* token, across different sites. There are two different systems currently under development that have attracted attention: OpenID Connect [2] and BrowserID [1]. These two systems are not designed (or at least thought to be) capability-based systems, and do not vary in terms of authentication, with BrowserID directly using email addresses and OpenID Connect (in its usual deployment) simply redirecting the user to an identity provider for signing-in with a *username-password* token that can be reduced to an email address. Instead, OpenID Connect and BrowserID vary primarily in how they conceive of the flow of information from the identity provider to relying parties and these differences in information flow have important ramifications for privacy. Note that federated identity by nature imposes a security risk due to single point of losing control over identity claims at the identity provider, but this risk already present in *username-password* re-use, and putting the user "in-the-loop" via the information flow of identity claims in BrowserID mitigates this risk.

3.1 OpenID Connect

OpenID Connect is a popular federated identity system meant to provide much of the same functionality as Facebook Connect, and is deployed by large identity providers (email providers) such as Google and Microsoft [2]. OpenID Connect builds upon the well deployed base of OAuth 2.0 standard for server-side claim exchange [10], but optimizes certain elements of OAuth for server-side exchange of identity claims and requires no changes to current browsers. OpenID Connect uses OAuth 2.0 for the authorization flow while adding a small number of non-opaque identifiers in the response between an identity provider and relying party as well as adding more detailed hooks for using cryptographic signing. Once the user authenticates to an identity provider (usually via a HTTP redirection and a username-password) and so provides the relying party an access token, the identity claims are passed directly from the identity provider to the relying party, and the user is out of the loop. The flow of OpenID Connect is illustrated in Diagram 1 (the transfer of identity claims is given in *green* in this and subsequent diagrams) and outlined below:

3.2 Flow

1. A user visits a relying party that needs identity claims.

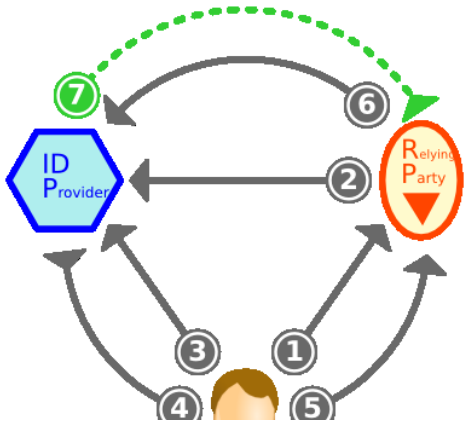

Fig. 1. Flow of OpenID Connect (OAuth 2.0)

2. The relying party makes a request for identity claims to the identity provider.
3. The user is redirected to the identity provider from the relying party.
4. The user authenticates to the identity provider (typically using a username-password combination) and is granted a bearer token.
5. User is redirected back to relying party and grants authorization token to relying party.
6. The relying party sends the authorization token to the identity provider and receives an access token (a bearer token with a scope and limited lifespan).
7. While the access token is valid, the identity provider sends identity claims to the relying party.

OpenID Connect gives the identity provider ability to observe all requests for identity claims by relying parties, which is the primary flaw from a privacy standpoint as identities cannot be delinked from the identity provider. As the traffic of identity claims flows directly between identity provider and relying party, the interaction between the user and a relying party can be logged by the identity provider, as well as traced by third-parties via traffic analysis between the relying party and identity provider. Although this particular flow has the advantage of possibly authorizing requests for personal data when the user is not online and thus unable to directly intervene at the time of the request, which we call the *offline server flow*, as the user can be off-line at the time of the interaction. Although this offline server flow is a distinct advantage for some use-cases (such as when the user authorizes the requests ahead of time or on a regularly occurring basis), it is also a danger, as the identity provider may exchange user data with relying parties without the consent of the user, leading to the possibility of identity interactions being unknown to the user. As regarding anonymity, although the architecture of OpenID Connect does not require that identifiers be persistent when sent to the relying parties (and thus allows anonymity to relying parties), the authentication mechanism to the iden-

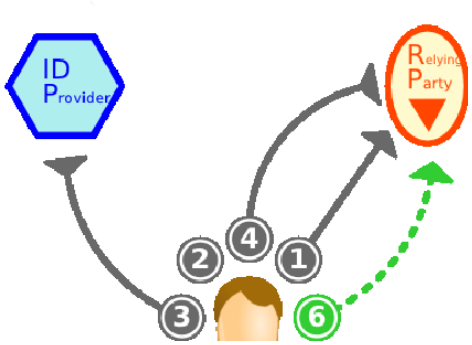

Fig. 2. Flow of BrowserID

tity provider does not authenticate particular capabilities but instead identifies the entire user or personae on a coarse-grained manner to the identity provider, and so the identity provider is aware of all relying party requests even if the user is anonymous to the relying party. Thus, OpenID Connect can be thought of as absolutely trusting the identity provider, which may be a reasonable assumption in some circumstances but this seems to be a poor choice use-cases that require a higher degree of privacy.

3.3 BrowserID

BrowserID, also called Mozilla Persona, is backed by Mozilla as its primary identity system [1]. In particular, BrowserID breaks the direct connection between the identity provider and the relying party given by OpenID Connect [2]. Instead, the identity provider can instead mediate the transfer of identity claims via the browser, which provides better unlinkability compared to OpenID Connect. Separately, BrowserID allow users to authenticate their identity via a "verified" email as their primary authentication scheme. However, beyond traditional email, the authentication of both the user and the identity provider (which in the case of BrowserID is always the email provider) is done via the use of key material. The authorization flow of BrowserID is outlined below and illustrated in Diagram 2.

3.4 Flow

1. The user attempts to identify themselves by giving an e-mail address, and wants to bind that address to a particular set of key material (for which the user then provide a proof of possession) by having the identity provider attest to that binding.
2. The browser checks to see if a private key is present in the browser associated with that email address.

3. If no key exists for the email address locally in browser, the browser generates key material and registers the public key with the identity provider.
4. The browser sends a signed authentication credential to the relying party.
5. The relying party verifies the authentication credentials with their locally stored database of identity provider public keys and authenticates the user if verification succeeds.
6. The user sends signed identity claims to the relying party from browser (for example, possibly using HTTP POST or PUT).

From a privacy standpoint, the flow of BrowserID is superior as it avoids the observability of relying party transactions to identity providers, although it is a requirement that the user be "in the loop" via the browser, so the flow only works for use-cases where the user is actually online. Thus, we call this particular authorization flow the *online browser flow*. Although the relying party knows the identity provider's key and will have to check at least once with the identity provider to determine the public key of user's email address, it does not have to check more than once per e-mail (given some reasonable caching time limits). Currently, there is only one identity provider (*browserid.org*), but regardless any identity provider would have the ability to transfer of identity claims to the browser and then from the browser to the relying party. So for observability and linkability, both the identity provider-browser and browser-relying party connections would have to be observed by a third-party in order to make the entire transaction observable, and thus the transaction is not easily logged by the identity provider (although it could be by the browser). Lastly, BrowserID is still ultimately using a coarse-grained version of identity rather than capabilities, for using the email address as identifier enables an observer the ability to possibly link all observed transactions across different relying parties if they have access to these parties. Email addresses can be multiple (as would be required for personae), pseudonymous, or throw-away, but most email addresses have value for users insofar as they are also long-term valuable identifiers. So while BrowserID has a superior privacy compared to OpenID Connect as regards observability of transactions and assumes the identity provider is not to be fully trusted, BrowserID assumes the browser (and all plug-ins) are trusted and reveals the email address to the relying party for every identity claim-based transaction. While the capability to check an email address is the mainstay of authentication today, this may not be enough: the reduction of the security in federated identity to checking email can itself be problematic, for example with many email clients not producing an error message when STARTLS fails to upgrade to TLS.

4 A Privacy-Preserving Capabilities-Based Approach

Earlier, we demonstrated that identity on the Web can be considered as a capability to check email and showed how BrowserID's information flow breaks the observability of the identity provider to relying party requests. In this section, we combine these two insights by outlining how an identity provider could allow

the verification of any signed claim to have a capability. The crucial component of BrowserID is not the use of an email address per se, but that it implements a flow for transmitting personal data about a user as verified by signature. However, BrowserID only exposes the "user can check this email" capability, but it hints at a much bigger opportunity: the ability for an identity flow to handle general-purpose capabilities. In essence, the "signature" from the identity provider is the proof of the possession of the capability.

The key is that the identity provider can verify other capabilities about the user by virtue of their access to the personal data of the user. For example, a user may have the capability to use a smart-card to verify their identity or complete two-factor authentication via a mobile device. In fact, one can also generalize user-centric capabilities to a more standard intuitive understanding of capabilities, with the identity provider having information about what programs ('apps') a user possesses and can run. Thus, by virtue of handing a signed token over to the relying party, the identity provider is granting the relying party that particular capability, in a similar manner as is done with bearer tokens in OAuth [10]. Capabilities provides important privacy advantages as a capability-based approach does not have to disclose an email address (and so reduce the security of the system that of SMTP), as all that is revealed is the presence or absence of a capability. The use of digital signatures in the verification step is crucial, as otherwise the client could simply state any capability was possessed. In this particular variation on federated identity, there is still an assumption of trust between the relying party and identity provider, as essentially the identity provider is responsible for the veracity of the claim and any capabilities associated thereof.[6]

In addition to BrowserID's "online-browser" flow, we can combine the idea of signed tokens as capabilities to deal with the "offline-server" flow of OpenID Connect by simply having OAuth 2.0 deliver a token that contain signed assertions of capabilities instead of the usual OAuth bearer token [10]. Thus, the capability approach allows for the user not to be online, although the "online-browser" approach would be superior for privacy-respecting use-cases. The flow is illustrated in Diagram 3:

1. A user attempts to authenticate to a relying party, and the relying party requests a capability.
2. The browser checks to see if a capability, i.e. a signed token, exists in the browser.

[6] The creation of such an API is also straightforward; in Javascript code comparable to existing BrowserID implementations, the generic framework would then be thought of as *navigator.capabilities.get(domain,capability_type[required_capability_value])*. So the generic capability to check email could then be applied using *navigator.capabilities.get('gmail.com', 'email-ownership', 'romeda@gmail.com')*. Asserting a particular user is over the legal drinking age by citing the appropriate domain name but without necessarily giving away the precise age or the identity (e-mail) of the user is done by asserting the value: *navigator.capability.get('gov.uk', 'above-legal-drinking-age', 'yes')*.

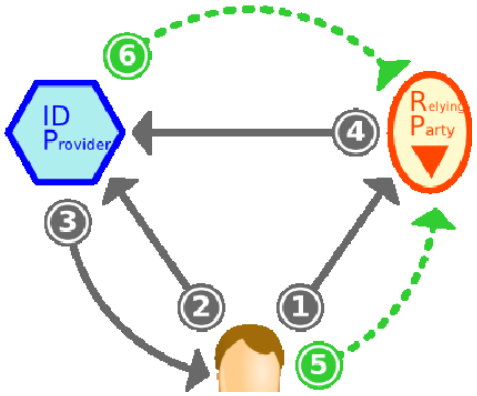

Fig. 3. Flow of Capabilities-Based System

3. If no capability is present, the browser requests a capability from the identity provider, to be sent using either with the "offline-server" flow or using the "online-browser" flow depending on whether or not the user is online and their level of privacy requirements.
4. With conventional signatures, the relying party can check to the signature with the public key of the identity provider, which they can request if the key of the identity provider is not locally cached.
5. For the "offline-server" flow, the relying party requests the capability from the identity provider. The identity provider sends capability to the relying party, which then the relying provider can verify using the public key material from the identity provider it has already received in the previous step.
6. For the "online-browser" flow, the browser sends the capability to the relying party directly, which again the relying provider can verify using the public key material from the identity provider.

There are also a number of variations on the flow. For example, we may want the user to grant the identity provider permission themselves by giving them a signed token for a capability (signed by the client device itself using a local private key) to act on their behalf before the identity provider sends another signed token (signed by the identity provider) to a relying party. In this case, the role of an identity provider is shared between a browser and a service provider for the user, which can then contact other services on behalf of the user if authorized by the correct capability by a user via their client device. We can imagine this being extremely useful in some socially-oriented use-cases, as we illustrate in the following example that demonstrates the "offline-server" flow with a step of user-verification that ensures the the ability to grant that capability to the relying party is explicitly given by the user:

1. User A to User A's service provider: "I want to follow User B."

2. User A's identity provider to User A's browser: "Give me a signed token that says I have the capability to act on User A's behalf."
3. User A's browser to User A: "Is your service provider allowed to act on your behalf?"
4. User A to User A's browser: "Yes."
5. User A's browser to User A's service provider: "Here's the signed token that says you're allowed to act on User A's behalf."
6. User A's service provider to User B's service provider: "Here's a signed token that says I'm allowed to act on User A's behalf, and they'd like to follow User B. Here's a URI you can contact me at to let me know if User B consents or declines consent and publishes content."
7. User B's service provider to User B: "Can User A follow you?"
8. User B to User B's service provider: "Yes"
9. User B's service provider to User A's service provider: "User B says you can follow her, and here is some recent content published by User B."
10. User A's service provider to User A's: "You're now following User B, and here's some recent content published by User B."

5 Conclusions

We have so far demonstrated that the signed tokens used in identity systems such as OpenID Connect and BrowserID can be considered capabilities. In fact, even the bearer tokens used by implementations of OpenID Connect and OAuth implementations that avoid key material can be considered as capabilities, as the token giving the capability would just be the symmetric shared secret given by the "string" of the bearer token. However, digital signatures present a much more secure manner to use key material to guarantee the possession of the capability. From an implementation stand-point, the work is already done and so such as system is merely a manner of tweaking current deployments: One can use Javascript Web Tokens in either BrowserID or OpenID Connect flows [2]. However, one problem is then tying the capabilities to the token in a uniform manner. Given the immense number of possible capabilities, a central registry is probably not feasible. Commonly used services could just publish their capabilities in their documentation, and using a protocol such as WebFinger[7] would also allow relying parties to inquire about the capabilities in an automatic and decentralized manner. Capabilities actually simplifies both OpenID Connect and BrowserID, as it gets rid of making the relying party having to do requests for the precise personal data specified by OpenID Connect and does not require using BrowserID's verified email in use-cases where it is unnecessary.

Although the proposed system presents a simple methodology to create a kind of anonymous credentials for users, the proposed system does this by rooting the capabilities in key material of the identity provider, as possibly authorized by key material given by the user on their client device. However, the

[7] https://code.google.com/p/webfinger/

signature schemes and interactive proof systems needed by anonymous credentials and zero-knowledge proof systems can also at some point be added to the capabilities-based federated identity system outlined here, in particular as the system proposed here presents an easy-to-implement path to implementation. The next step would be using blind signatures such as those suggested by the PseudoID system, with a blinding service being part of the information flow and committed either by the identity provider as a service or a third-party service [7]. Then one would need to implement a zero-knowledge proof that selectively disclosed only some part of their capability. The system sketched here is only the beginning as regards a holistic approach for preserving privacy. For example, in order to avoid IP-address based tracking, one could use either the "browser-online" and "server-offline" flows over a proxy system such as Tor.[8] One could forgo the concept of persistent sessions at all by ensuring that no logs of transactions are kept and carefully decoupling the client and server. Future research needs to determine not only what is possible, but how these options can be realistically and easily implemented on top of existing systems such as OAuth [10]. Lastly, we have barely scratched the surface of the vast literature on capabilities, and the various research results on capabilities should be investigated to determine if they can be applied to federated identity-style systems.

What we have presented here is not a new system for federated identity, but simply viewing existing systems such as BrowserID and OpenID Connect through the lens of capabilities. We claim that capabilities actually simplifies the apparatus needed by developers and maximizes privacy for end-users. Also, unlike many other proposed privacy-enhancing federated identity schemes, capabilities are easy to implement on top of existing systems as, in their most basic form, capabilities can be considered to be digital signatures provided by an identity provider, although the higher level of privacy required by blind signatures and zero-knowledge proofs will require more complex signature schemes. Nonetheless, federated identity and the minimization of user's data disclosure are fully within the simple and powerful framework of capabilities, a framework that can hopefully be at some point generalized as an identity system and permissions component for the entire Web.

6 Acknowledgments

W3C/MIT would like to thank the Northrop Grumman Cybersecurity Research Consortium for funding this work.

References

1. Adida, Ben: BrowserID (Mozilla Personae). https://browserid.org/ (2012)
2. Bradley, J., Recordon, D., Jones, M., and Sakimura, N.: OpenID Connect. http://openid.net/connect/ (2012)

[8] https://www.torproject.org/

3. Camenisch, J. and Lysyanskaya, A.: Signature Schemes and Anonymous Credentials from Bilinear Maps. In Proceedings of Advances in Cryptology (CRYPTO 2004), Annual International Cryptology Conference, M. Franklin (Ed.), Springer-Verlag, London, UK, pp. 56-72 (2004).
4. Camenisch, J. and Lysyanskaya, A.: An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT '01), Birgit Pfitzmann (Ed.). Springer-Verlag, London, UK, pp. 93-118 (2001).
5. Camenisch, J. and Van Herreweghen, E. Design and implementation of the Idemix anonymous credential system. In Proceedings of the 9th ACM conference on Computer and Communications Security (CCS '02), V. Atluri (Ed.). ACM, New York, NY, USA, pp. 21-30 (2002).
6. Chander, A., Mitchell, J., and and Dean, D.: A State-Transition Model of Trust Management and Access Control. In Proceedings of the 14th IEEE workshop on Computer Security Foundations (CSFW '01). IEEE Computer Society, Washington, DC, USA. (2001).
7. Dey, A. and Weis, S.: PseudoID: Enhancing Privacy for Federated Login. Hot Topics in Privacy Enhancing Technologies, pp. 95-107. http://research.google.com/pubs/pub36553.html (2010).
8. Fiege, U., Fiat, A., and Shamir, A.. 1987. Zero knowledge proofs of identity. In Proceedings of the ACM Symposium on Theory of Computing (STOC '87), Alfred V. Aho (Ed.). ACM, New York, NY, USA, pp. 210-217 (1987).
9. Hardt, D.: OpenID Authentication 2.0 https://openid.net/specs/openid-authentication-2_0.html (2007).
10. Hardt, D.: OAuth 2.0 Authorization Protocol. IETF RFC. https://tools.ietf.org/html/draft-ietf-oauth-v2-31 (2012).
11. Laurie, B.: Access Control http://www.links.org/files/capabilities.pdf (2008).
12. McCallister, E., Grance, T. and Scarfone, K.: Guide to Protecting the Confidentiality of Personally Identifiable Information (PII). P 800-122. Technical Report. NIST, Gaithersburg, MD, United States (2010).
13. Shepard, L.: Facebook Connect. http://www.facebook.com/help/?page=229348490415842 (2012).
14. Story, H.: WebID. http://webid.info (2012).