Kevin Corre*, Olivier Barais, Gerson Sunyé, Vincent Frey, and Jean-Michel Crom

Why can't users choose their identity providers on the web?

Abstract: Authentication delegation is a major function of the modern web. Identity Providers (IdP) acquired a central role by providing this function to other web services. By knowing which web services or web applications access its service, an IdP can violate the end-user privacy by discovering information that the user did not want to share with its IdP. For instance, WebRTC introduces a new field of usage as authentication delegation happens during the call session establishment, between two users. As a result, an IdP can easily discover that Bob has a meeting with Alice. A second issue that increases the privacy violation is the lack of choice for the end-user to select its own IdP. Indeed, on many web-applications, the end-user can only select between a subset of IdPs, in most cases Facebook or Google. In this paper, we analyze this phenomena, in particular why the end-user cannot easily select its preferred IdP, though there exists standards in this field such as OpenID Connect and OAuth 2? To lead this analysis, we conduct three investigations. The first one is a field survey on OAuth 2 and OpenID Connect scope usage by web sites to understand if scopes requested by websites could allow for user defined IdPs. The second one tries to understand whether the problem comes from the OAuth 2 protocol or its implementations by IdP. The last one tries to understand if trust relations between websites and IdP could prevent the end user to select its own IdP. Finally, we sketch possible architecture for web browser based identity management, and report on the implementation of a prototype.

Keywords: OAuth, OpenID, SSO, web browser, identity, trust, user choice, WebRTC, communication privacy

DOI 10.1515/popets-2017-0029
Received 2016-11-30; revised 2017-03-15; accepted 2017-03-16.

***Corresponding Author: Kevin Corre:** Orange Labs/IRISA, kevin.corre@irisa.fr
Olivier Barais: IRISA/INRIA, olivier.barais@irisa.fr
Vincent Frey: Orange Labs, vincent.frey@orange.com
Jean-Michel Crom: Orange Labs, jeanmichel.crom@orange.com
Gerson Sunyé: INRIA, gerson.sunye@inria.fr

1 Introduction

The Web offers a vast range of applications and services, on which users share personal information. To protect these informations, secure access to web services must be guaranteed. But site-by-site account management is a burden for users [1]. It may often result in insecure login, for instance due to the reuse of the same password across multiple websites [2], a situation known as password fatigue. Web based Single-Sign-On (SSO) and authorization delegations are interesting alternatives to cumbersome site-specific accounts. Indeed, users are more prone to use website services if they do not have to create new accounts nor to manually fill personal information. Moreover, SSO decreases the sensitivity of data stored by websites as authentication and access tokens are specific to a particular context and can be revoked, contrary to login/password pair.

Third-party Identity Providers (IdP) acquire a central position on the web by providing SSO services. They are responsible for managing both user security and privacy. As a consequence, IdPs gain the ability to track their user's actions and monetize this information, making users face a tradeoff between usability and privacy [3].

Due to implementation constraints (cost and page space), websites only propose few sign-in options, favoring big IdPs with large user bases over medium or small one. On one hand, users with concerns for their privacy may not trust available IdPs. As a result these users face a dilemma: use an untrusted IdP or create yet another website-specific account. On the other hand, IdPs implementing privacy preserving policies and solutions have to convince individual RPs to implement them. Privacy friendly techniques are an active research topic. But to see them used, it is necessary to allow users to choose actors they trust to respect the privacy of their data.

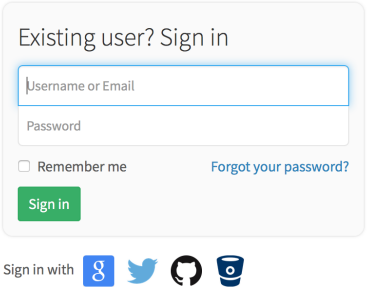
Although technical solutions to let user authenticate with the IdP of their choice exist, these are rarely adopted. We postulate that probable reasons hampering the adoption of IdP discovery mechanism by websites are unadapted protocols/API, incomplete implementa-

tions, and the necessity of trust relationship between websites and IdPs.

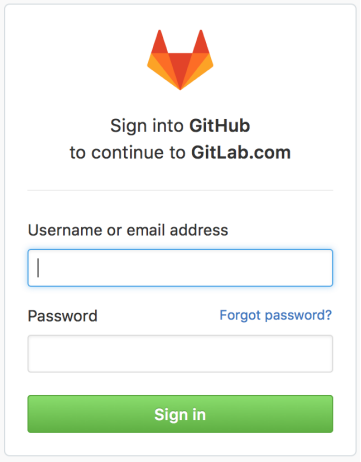
In this paper we investigate the current implementation and usage of the latest SSO protocols, OAuth 2 and OpenID Connect (OIDC), in order to determine the reasons and possible solutions to the non-usage of user defined IdP solutions. We first give background on authentication and authorization delegation, and OAuth 2/OIDC in Section 2. We also showcase how authentication delegation may be a risk to user personal data with the example of the WebRTC identity specification in Section 2.2. In particular, we show that users may be drastically limited in their choice of an IdP, and that they may not be correctly informed that the IdP may access call information.

We then present our investigation in Section 3. In that section, we propose three hypothesis that could explain the absence of IdP discovery on the web. To verify these hypothesis, we conduct a data collection experiment using a browser extension. We visit SSO services used on the top-500 worldwide websites. The browser extension then collects OAuth 2/OIDC URL requests in the navigation history. These requests allow us to discover a large number of websites using OAuth 2/OIDC and the type of information requested to supported IdPs. Results show that state of the art protocols could adequately answer current RPs requests but that IdPs do not implement functionalities for IdP discovery mechanisms.

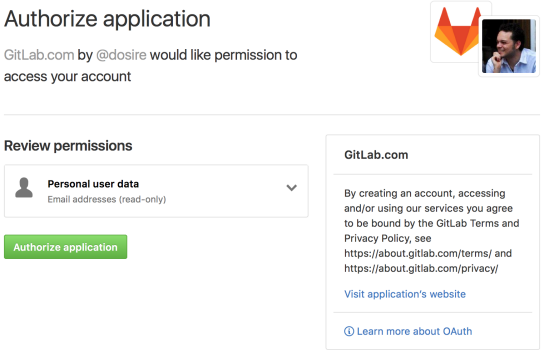
Our last contribution sketches the requirement and advantage of an identity-enabled web-browser in Section 4. We discuss why native identity support would bring much benefits to users and that an identity web API could ease the deployment of IdP discovery on website. We report on the prototype solution we implemented. Finally, we conclude and discuss our future work.



(a) GitLab sign in page with a selection of IdPs



(b) Redirection to GitHub sign in page



(c) Github consent page to share email with GitLab

Fig. 1. Authentication and authorization delegation process involving Gitlab as RP and GitHub as IdP

2 Delegated authentication and privacy

2.1 Authentication and authorization delegation, OAuth 2

Authentication delegation allows users to create an account and to sign-in on a Relying Party website (RP), by authenticating on a third-party IdP. Usually, users

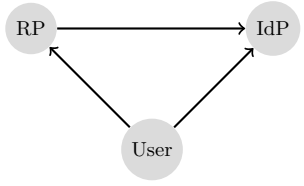


Fig. 2. Trust relations in authorization delegation

have the choice of creating and authenticating with a login/password local to the web site they want to use. However creating and maintaining accounts over a large amount of sites is a cumbersome activity. In order to avoid this task, SSO solutions allow users to use a main identity across multiple sites. SSO solutions are often provided by email providers or social networks, services historically in the role of providing an identity.

The process of authentication delegation involves the choice of an IdP over a list of implemented services offered by the RP, as shown in Figure 1a. Alternatively, discovery services, such as with OpenID [4] or BrowserID [5], may be used by the RP to discover the IdP of the user. Once the user has selected his IdP, he is redirected to a sign-in page on the IdP where he can authenticate himself, as presented in Figure 1b. Once authenticated, the user is again redirected to the RP with an identity token asserting the user authentication by the IdP.

Additionally, profile information (name, email, address, etc.) may be returned along with the identity token. Authorization delegation lets the user authorize a RP to access private information contained on a Resource Server (RS). The process is managed by an Authorization Server (AS) and involves a similar process to the authentication delegation, in which the RP receives an Access Token. This token can then be consumed by the RP to access user data on the RS. In practice with web sign-in, IdP, RS, and AS are often the same actor. For the sake of simplicity, we do not distinguish them in this paper. Most RPs often require user profile information to complete their local profile. Thus, authentication and authorization delegation are often mixed together in a single process.

OAuth 2 is an authorization delegation protocol. In the OAuth 2 protocol, the RP asks for specific authorization by requesting scopes. Once redirected to the IdP page, the user is presented with the requested scopes and given the choice to grant or refuse authorization. Figure 1c shows the GitHub consent page on which the user is presented with a choice to share his email with GitLab RP. Depending on the implementation, authorization may be individually given by the user, however it is more often an all or nothing choice. OpenID Connect (OIDC) is an identity layer built on top of OAuth 2. OIDC specifies the *OpenID* scope to request an ID Token and standardizes several other scopes related to user identity (profile, email, etc.), as well as the endpoint on which to access these informations.

Authentication and authorization delegation protocols pose a risk to user privacy as they require sharing private information between domains. In this situation, users need to trust both actors, the IdP and the RP, with their data. The RP may need to trust the IdP, regarding the validity of the provided resources and assertions. For instance, the validity of profile information or the strength of authentication process. This simple trust model is summarized in Figure 2, where arrows represent relations of type A trust B.

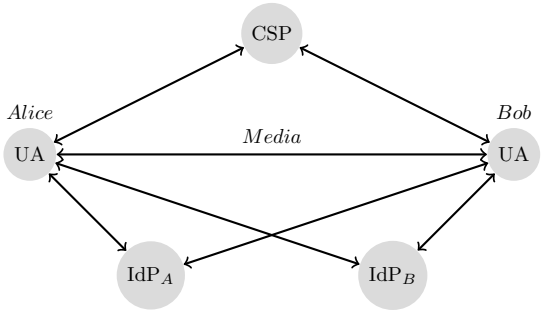


Fig. 3. A call with IdP-based identity and a single communication service provider

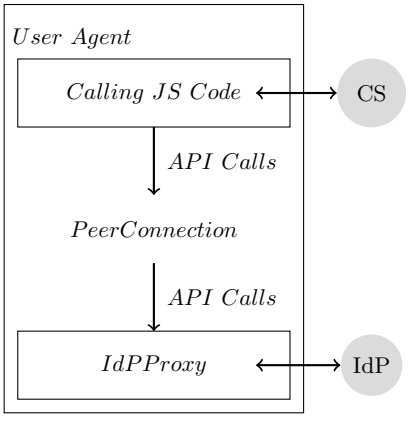


Fig. 4. WebRTC Identity Architecture
API calls are the setIdentityProvider function, offered by the PeerConnection object to the JS client, and the generateAssertion and validateAssertion functions, offered by the IdP Proxy to the PeerConnection. The protocol used between the IdP Proxy and the IdP is unspecified.

2.2 Privacy issue, the example of WebRTC identity

2.2.1 WebRTC identity

WebRTC [6] is a web API, developed by the W3C and the IETF, which supports P2P audio-video calling and

data sharing. It allows for real-time communications between web browsers, from within a webpage. The envisioned use-cases are audio conferencing, e-commerce support, and personal or enterprise communications. Due to the ease of deploying a WebRTC communication service, a simple web server being enough, it is expected to see the emergence of a larger number of WebRTC enabled web sites.

WebRTC communication setup can be decomposed into three different paths as shown in Figure 3. The Signaling path initializes the communication between Alice and Bob User-Agents (UA) through one or more Communication Service Provider (CSP) servers. On this path, Alice and Bob exchange call offers and answers Session Description Protocol (SDP) messages in order to setup the Media path. The Media path is a peer-to-peer encrypted connection between Alice and Bob used to exchange audio, video, or data. Optionally, one or two Identity paths are used to generate and verify Identity Assertions. Identity Assertions are transmitted inside SDP messages. Their role is to assert that media path encryption keys are used by users that are authenticated by an IdP.

The WebRTC Identity architecture [7] specifies the IdP Proxy component. This component serves as an interface between the WebRTC PeerConnection object and the IdP. The IdP Proxy is available at a standardized location on the IdP domain. Before making a SDP call offer or answer, the PeerConnection calls the IdP Proxy to generate an assertion covering a set of keys. After the IdP authenticated the user, the identity assertion is returned. This assertion, along with the IdP Proxy location on the IdP domain, is then included in the SDP message. This allows users to discover IdP Proxy location without prior knowledge or relationship with the IdP. On receiving a SDP call offer or answer containing an Identity Assertion, the PeerConnection object downloads the IdP Proxy from the specified location. It then calls its assertion verification function. If successful, the IdP Proxy returns the key fingerprint and the user identity (for instance an email address). This ensures that no man in the middle attack is being set, and that the user claiming the identity was authenticated by the IdP. The WebRTC Identity Architecture is shown in Figure 4.

The WebRTC specification allows users to choose a default IdP in their UA preferences if none was specified by the CSP. But implicitly, the default IdP should still be somehow compatible with CSP implementations in order to handle the authentication procedure, as specified in the Identity section of the WebRTC specification. Otherwise, this would result in a call failure as the IdP Proxy would not be able to produce an Identity Assertion. If the CSP sets an IdP to use for a WebRTC call, we would expect it to be the same IdP used to sign into the CSP's website. Indeed, the CSP must be able to handle the authentication procedure and needs to know that the user has an active account on the IdP. Besides, from a user perspective, it is important that the identity presented on the CSP user interface and the one received in the identity assertion are consistent with each other.

In practice, the choice of IdP would thus be defined by the CSP and limited in the same way as for common authentication delegation services, as explained in Section 2.1.

2.2.2 Call tracking by the IdP

While the Identity Assertion ensures secure communication by exchanging user trust in the CSP with user trust in IdP (or in addition to), it also comes at a cost as the IdP gains the ability to track any user call covered by an Identity Assertion.

The IdP Proxy code is deployed on both sides of the call, and subject to the same sandbox restrictions on both sides. This implies that an IdP Proxy can place and read cookies on a user's UA verifying an assertion, i.e. not having actively accessed the IdP domain. Such IdP could track a user's call history, under an anonymous identity, with other user authenticated with the IdP. Eventually this anonymous identity could be linked with a known identity, and the call history versed into an existing profile.

The IdP can also track user calls across multiple CSP, as long as they use, or are called with, the same IdP. However it does not appear that the IdP is able to know on which CSP the call happens, as the PeerConnection interfaces between calling sites and the IdP Proxy. Except if the user authentication procedure is used.

To reuse the classification proposed by Vapen et al. and presented in Section 2.3, this ability to track user call history could at least be classified as $R+$, or $R++$ regarding the verifying user, as it would fall under the *Friend's data* class of privacy risk. However, user may not have authorized or even be aware of such tracking capacity.

Let us follow the example of user signing in with an OAuth 2 service onto a CSP. In order to access the service, the user would give consent to authorization requested by the CSP. However, in the WebRTC speci-

fication, the Identity Assertion is not requested by the CSP but by the IdP Proxy, who plays the role of RP. As the IdP Proxy is from the same domain as the IdP, the Identity Assertion generation could be done outside of any authorization flow. For instance, a simple authentication check between the User Agent and the IdP would be enough.

Even if the exchange between the IdP Proxy and the IdP where covered by an authorization flow, this authorization would have been given by the user to the global IdP Proxy RP, rather than for a particular CSP. Any CSP could trigger a WebRTC IdP authentication without asking or notifying the user, as it is the CSP's client-side JavaScript which sets the WebRTC identity parameters.

The verifying user would get no control over the IdP Proxy as it is an automatic process controlled by the UA. He would only be informed after the UA notifies the success of the Identity Assertion verification by the IdP. This can be compared to email, where the recipient cannot prevent the sender mail server from learning the email destination. Also note that the notification by the UA of the Identity Assertion verification is optional, the user may not get it at all.

We can also observe that the information sharing relation is here reversed between the IdP and the CSP. In a classical authorization delegation, the user authorizes the IdP to share resources with the RP. However in this case, the CSP could be considered as being in possession of call information and would indirectly share them with the IdP, by adding it to the call setup.

To summarize, the WebRTC authentication mechanism can be rated as $R+$ but may trigger R- or no notification on the user side. Authorization for the IdP to access call information may have been given on a global scale, or not given at all. And the users may not even be aware of the IdP during the call setup.

2.3 Web SSO usage and related work

In 2015, Vapen et al. studied the identity management landscape on the web [8]. In their study, they classified the type of information shared by IdP to RP in five classes: basic information, personal information, created content, friend's data, and a transversal action class. In addition, they defined semi-ordered risk types classes, build as conjunction of information shared classes. These classes range from R- to RA++ risk levels, A denoting action authorization. Our study, presented in Section 3, uses a similar but simplified classification. Our classification defines three classes: authentication, profile, and specialized, characterizing the functional requirements of RP. In this classification, RP from the personal, created, friend's data, and action classes would belong to the specialized class.

They also show that in practice RPs offer few choices of IdP to their users, with 47 % offering only one IdP, and 19 % offering four or more IdPs. Proportions confirmed by our own observations.

These observations also confirm that the situation profit to a few IdPs trusting the top ranks, with Facebook as the number one, followed by Google and Twitter. It is clear that users lack choice when choosing an IdP. This can lead to trust issue, especially due to the dominant position of a few providers.

In 2011, Sun et al. conducted an empirical investigation to evaluate the acceptance by users of Web SSO, and OpenID in particular [9]. In this study, a large number of participants expressed concerns over privacy and trust in RPs (40 % and 36 % resp.). From their findings, they recommend that browser makers should provide an integrated identity support to their user, giving them a consistent, intuitive, and trustworthy user experience.

3 RP-IdP relationship investigation

In Section 2.2, we presented how the WebRTC identity mechanism allows IdPs to track user calls without explicit user's authorization or awareness. This mechanism mainly serves to leverage the user's trust in the IdP, in order to prove the CSP's goodwill regarding a man in the middle attack.

Furthermore, as explained in Section 2, users are also presented with a very limited choice of IdPs when signing in with authentication delegation. As WebRTC IdP and SSO IdP would be the same, this limitation may impact WebRTC scenarios too. Users may have more trust in IdPs whose business model does not rely on selling personal data or which they would host themselves [10]. However, current SSO implementations do not permit users to make this choice.

We develop three hypothesis regarding the reasons of these limitations and conduct an investigation on the type of relations between IdPs and RPs to verify them. Our hypothesis are the following:

– H1: API and data format divergences in implementations require specific implementations for each IdP on the RP's side.

– H2: Protocol limitations induce implementation cost.
– H3: Trust relationships between RP and IdP impose manually configured relations.

3.1 OAuth 2/OIDC scope usage collection

To evaluate the impact of the first hypothesis, we implement a browser extension to parse the browser navigation history and look for OAuth 2 and OIDC authorization request URLs. These requests are characterized as OAuth 2 requests by several keyword parameters and contain information identifying the RP making the request, the IdP to which the request is destined and the scopes requested by the RP.

```
https://accounts.google.com/o/oauth2/auth?
      client_id=74[...].googleusercontent.com&
      response_type=code&
      redirect_uri=http://www.dailymail.co.uk/
      registration/signin/google.html&
      scope=email+https://www.googleapis.com/
      auth/plus.login&
      [...]
```

The URL above is an OAuth 2 request for a *accounts.google.com* (URL Domain Name) authorization following the OAuth 2 code flow (response_type=code). The request comes from the client *74658[...].apps.googleusercontent.com*, also identifiable through the redirect_uri parameter as *dailymail.co.uk*. This client and the *facebook.com* client, registered as *146[...]95*, share the same redirect_uri domain name. We thus consider them to originate from the same RP. Requested scope are *email* and *https://www.googleapis.com/auth/plus.login*.

Note that these URLs do not contain private information regarding the user as they are accessed before the user is identified or authenticated.

To collect data, we manually visit each of the top 500 from the Alexa ranking[1] and try to use each one of the SSO solutions offered by these websites. The visited URLs are recorded and parsed by our extension. Finally, we use data on collected RPs and IdPs to also evaluate the impact of hypothesis H2 and H3.

As our data-collection search is focused on OAuth 2, we do not collect requests for IdPs using other protocols, in particular Twitter and Amazon. However we scarcely encounter RP offering only these IdPs, as a result the large majority of visited RP is captured by our extension. In total, we observe 103 unique RPs and 23 OAuth 2 provider's domain names. The two biggest observed IdPs are undoubtedly Facebook and Google, respectively serving 63 and 52 RPs. The third most observed IdP is Twitter with 30 request URLs, but it uses OAuth 1 and as such is not included in our data collection.

While our claim that users are offered a limited choice of IdPs is confirmed by our study, we globally observe three different patterns based on the RPs geographical origin. Firstly, Chinese websites only offer to login through Chinese SSO solution such as QQ.com. Technically, they also often offer phone authentication through a QR code. Occidental websites, *i. e.* North-American and European, mostly offer to login through one or both of the top two SSO platforms, sometimes with a third solution. Finally, Russian websites offer the largest number of solutions as they include Russian SSO, *e. g.* VK.com, and occidental SSO often not limited to the top three providers. Others regions were not represented in sufficient numbers to draw any conclusions.

3.2 API and data format limitation

The RPs not only require user authentication but also authorization to access protected resources. These resources may vary in nature, and may not share a standardized data format when of the same type. One reason for RP not to allow signing in with any IdP could be that they require specific resources, which are not available on any IdP.

We classify RP in three categories: authentication, profile, and specialized, in function of the scope observed in OAuth 2 authorization requests. Authentication classed RPs only require an assertion that the user got authenticated by the IdP. As such, any IdP could be able to serve for such RP, given a standardized assertion (*e. g.* a signed JSON Web Token [11]). RPs classified as profile require basic user profile information, most often to complete their own database and provide a personalized user experience. IdP would need to give access to resources in a standardized data format in order to avoid a specific implementation on RP side. Finally, specialized RPs require specific resources, for instance, write access on a user shared repository. By definition, services provided by RPs of this class are specialized to use

1 http://www.alexa.com/topsites

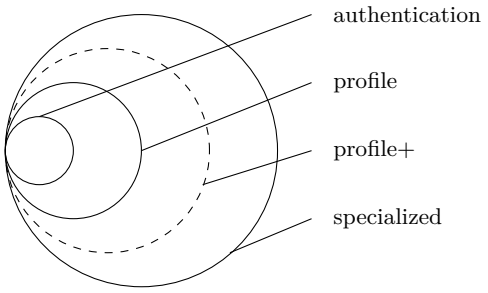


Fig. 5. Relying parties' classifications

resources from a few IdPs, each one requiring its own implementation. They cannot be generalized to cover a broad range of IdPs.

Figure 5 presents these classes, ordered by specificity. Indeed, access to a particular API is more specific than accessing generic profile information. Similarly, getting access to a user name and phone number is more specific than authentication, which could be abstracted to a boolean, *i. e.* authenticated or not. Although special cases may exist, *e. g.* access to authorized resources without authentication of the user, this classification allows us to define which RP could in theory accept any IdP, and which one would be bound to a particular API.

We classify each RP-IdP relationships, *i. e.* each collected client_id, into one of our three classes. We evaluate from the available documentation if the requested scopes give access to a specialized API, a profile information API, or an authentication proof. Differences between classes may be blurry, for instance email and friends list can be considered as user profile informations. However, email alone is often used as a unique user identifier in conjunction with a proof of authentication. OIDC defines a list of user-information claims and their types, we consider this as the standard for user information. Other type of informations and API, *e. g.* friends lists or cloud access, are non-standardized and not available from every IdP. We classify these requests outside of the profile and authentication classes as specialized. Ultimately we define our classes as:

- **Authentication:** requests only subscriber identifier scopes, *e. g.* 'openid' or email.
- **Profile:** requests scopes giving access to user informations equivalent to OIDC's standard claims.
- **Specialized:** requests not classified as authentication or profile.

Some RPs offer to sign in with multiple IdPs and may require different types of information for each IdP. As a result, they may be classified differently for each of their IdP. Based on common redirect_uri domain names, we grouped identified clients into unique RPs. For each RP, we attributed a minimum (MIN) and a maximum (MAX) classes. As an example, a RP classified as Authentication/Profile requires to access profile information on some of the offered IdP options, but only requires an authentication proof and a unique identifier on at least one offered IdP. RPs offering only a single IdP were classified as MIN/-.

3.2.1 Result analysis

Our observations, summarized in Table 1, reveals that a majority of RPs, 58 % of 103, can be classified as Authentication or Profile. MAX-classed RPs, 40 % in total, are the one providing support for multiple IdPs. In our observations, 24 out of 56 RPs with a Specialized class are classified with a MIN class of Authentication or Profile. This double classification demonstrates that while some websites require Specialized services, they also adapt to support other IdPs offering less resources. Note that this also leads to different privacy risk level for the user. On the other hand, 34 % of observed RP are rated as MIN-Specialized.

The large majority of specialized RPs use resources from Online Social Network (OSN) such as friends list, user likes, and extended profile informations. While data from different social networks can be similar from a conceptual point-of-view, the format and API used to access these data depend on the functionalities and concepts offered by social network. In these cases, the user fully depends on implementation choices by RPs.

Although a standardized protocol, OAuth 2 lets the IdPs define their scope and data format. As a result, IdPs may provide similar results in different scopes and data format. For instance, we observed six different scopes for email access and seven different scopes for basic profile information. This gives an indication on the lower bound of the implementation work that RP must complete to support these IdPs.

OIDC solves this problem by standardizing basic profile claims (*e. g.* profile, email, address, ...) as well as the scopes, endpoint, and data format to retrieve these informations. From our observation, only four IdPs out of twenty where implementing OIDC, as reported on Table 2. Notably Google is one of the OIDC provider

Min/Max Classes	Observed	Risk class
Authentication/-	10% (10)	$R-$
Authentication/Auth	1% (1)	$R-$
Authentication/Profile	9% (9)	$R-$
Authentication/Special	6% (6)	$[R-; RA+]$
Profile/-	13% (13)	$R-$
Profile/Profile	2% (2)	$R-$
Profile/Special	17% (18)	$[R-; RA+$
Specialized/-	26% (27)	$[R; RA+]$
Specialized/Special	5% (5)	$[R; RA+]$
No Scope	11% (11)	
Total	100% (102)	

Table 1. Observed relying parties' classes
Some RPs offer to sign in with a single IdP, we classify them with a single class, *e. g.* Profile/-. Other RPs offer multiples IdPs, their relations with these IdPs may belong to a single class, *e. g.* Profile/Profile, or different classes, *e. g.* Auth/Special. For these RPs we show the minimum and maximum classes they offer. Risk class refer to privacy risk classes as defined by Vapen et al. [8]. When a RP has different Risk class due to multiples RP-IdPs relationships, we give an interval for the Risk classification.

and serves 50 % of observed RPs while Facebook does not implement it and serves 63 % of observed RPs. Other IdPs serve less than 6 % of RPs each.

However, out of the 52 RPs using Google's SSO, only 22 request standard OIDC scopes, and from these 22 only 10 arequest OIDC ID Token. Surprisingly, we also observe that out of the 34 RPs using the Google API scopes, 19 were using deprecated scopes.

RPs of MIN-Authentication and MIN-Profile classes represent 58 % of all observed RPs. We defined these classes as being equivalent to claims covered by OIDC requests. It appears that from our observations, the quantity and type of information shared under these scopes would be sufficient to login into a majority of websites. As a result, we conclude that current standards for API and data format do not appear to be a hindrance to SSO interoperability.

However, there is a clear lack of implementation of these standards from big IdPs, *e. g.* Facebook not implementing OIDC or Twitter not implementing OAuth 2. But the lack of implementation effort also comes from RPs, as a not negligible proportion of them did not update their SSO implementations to current versions. Implementation cost may be a reason, but we also note that studied websites come from the most 500 visited websites.

Regarding scopes for sharing OSN data such as friends list. We observe that 43% of RPs using OSN data also implemented other IdPs without requesting OSN data. This is even the case when such data are available, *e. g.* with accounts.google.com. On one hand, a standard format for OSN data could allow more interoperability between RPs and OSN based IdPs. But on the other hand, in these cases since RPs can accept non-OSN IdPs, the sharing of OSN data appears to be non-mandatory. The possibility to opt-out of consent for data sharing is however often not offered or not clearly advertised.

3.3 Protocol specification and implementation limitation

In order to support signing-in on a RP with any IdP, the protocol must offer discovery mechanism. This mechanism allows the RP to find the IdP's protocol parameters from a URI provided by the user. Depending on the protocol this URI may be for instance the user identifier, or a resource location on the IdP. Without discovery, the RP may not know which endpoint to use on the IdP, or which public keys and algorithms to use to verify information provided by the user or the IdP. Additionally, the protocol should also allow interactions between IdP and RP without prior manual configuration. Some protocols require the RP to possess credentials to be authenticated by the IdP. In this case a dynamic registration process must take place before any further interactions.

For instance, standard OAuth 2 recommend that the RP possess a client identifier and secret for authentication to retrieve access token. The use of an unregistered client is not excluded by the specification, but our investigation did not reveal any use of this. Similarly, OIDC requires the RP to be authenticated to get access and identity tokens. The current OAuth 2 version does not specify dynamic registration mechanism, but OIDC optionally offers discovery [12] and dynamic registration [13]. OIDC discovery uses Web Finger [14] to find user's IdP from the user identifier, and standardises IdP's metadata location. The metadata are in turn used to find, if available, dynamic registration endpoint. However, both discovery and dynamic registration extensions are optional. RFC 7591 proposes to generalise the specifications of OIDC discovery and dynamic registration to the broader OAuth 2 specification.

Table 2, summarizes the observed OAuth 2 IdPs. In total, we collect 23 unique provider domain names. Out of those, 15 are not requested with a scope parameter, and are not included in our OIDC features investigation. For instance the rambler.ru request to Instagram

Provider	RPs	OIDC	Metadata
www.facebook.com	63	×	×
accounts.google.com	52	✓	✓
oauth.vk.com	6	×	×
graph.qq.com	5	×	×
login.live.com	3	×	✓
account.live.com	3	×	×
www.linkedin.com	3	×	×
connect.ok.ru	3	×	×
login.microsoftonline.com	1	✓	×
services.Adobe.com	1	✓	×
github.com	1	×	×
feedly.com	1	×	×
www.livejournal.com	1	×	×
connect.mail.ru	1	×	×
open.weixin.qq.com	1	×	×
api.weibo.cn	1	×	×
mixi.jp	1	✓	×
oauth.riotgames.co	1	✓	×
Total	103	5	3

Table 2. Observed OIDC and discovery features Implementations

do not contains a scope parameter, but Instagram still asks some authorization to the user. On the other 18 unique provider domain names, only 5 are used with an openid scope, indicating implementation of OIDC.

No observed OIDC provider implement dynamic registration or Web Finger, and we find that only two out of five offer openid-configuration metadata. We are not able to test Web Finger in every case as some of these services do not offer a look-up compatible user identifier.

Again, existing protocols offer discovery and RP registration capabilities but implementations are missing these optional specifications. This mechanism is nonetheless compatible with manual configurations of IdPs. As such RPs could nonetheless implement it to demonstrate interest and support IdP allowing dynamic registration.

However, this would impose an additional component to add to the login page and an associated implementation. Users would also need to know and enter their OIDC identifier or provider for the discovery mechanism, which may not always be obvious. There is clearly an usability limitation compared to the "click to signin" use of IdP button.

3.4 Trust relations

As we described in Section 2, a trust relationship may exists between RPs and IdPs. For instance a RP may expect verified profile information from a governmental institution or a secure authentication process with a two-factor authentication. Level of Assurance (LoA) are usually used to characterise the strength of an identity process during the user enrolment and authentication. Similarly an IdP may trust a RP, or more particularly monetise access to an API. This implies that RP and IdP got into an agreement involving registration of payment methods which is out of scope of existing dynamic registration specification.

These trust relations can be either implicit or explicit. Implicit relations are difficult to characterize as they are not clearly visible from the user point of view. For instance, the web site service-public.fr[2], a website offering direct access to french governmental service, only offers to login with account from other public services such as the tax department, the social security service, or the national postal service. Other implicit trust relations may be due to strategic decisions, for instance limiting RP access to IdP of the same company. In such scenarios, the RP would not allow the user to choose any IdP. This is the case for developer.microsoft.com, as it only allows login through the login.windows.net IdP, a Microsoft service.

Explicit relations may be more easily characterized as the RP clearly request a solid information. For instance, Orange's OIDC IdP[3] offers the scope *form_filling*. This scope substitutes to OIDC profile scope and allows the RP to access qualified Orange information, for instance the telephone number linked to the user subscription. Such relation may either fit into the specialized or profile class, depending on the RP will or capacity to accept generic information, such as standard OIDC profile. We classify these relations as profile+, as shown in Figure 5. Explicit trust relations may also be linked to a contract between the RP and IdP. Note that other common scope may also be subject to agreement, though it is impossible to determine without investigating actual IdP API terms of use.

Similarly, OIDC RPs can verify if the email was verified by the IdP through the *email_verified* boolean value. As this verification is done on the server side, it is not visible from the user point of view.

Trusting a presented identity and the associated authentication process is a complex, and sometimes subjective matter, especially on the web. For instance, social networks such as Facebook and services using them,

2 Outside of our study.
3 Outside of our study.

claim their identity to be trusted. These claims are backed by social relationships, evaluation between users, or real-name usage policies. But stricter organisation, such as governments and banks, would not accept these identities. As we explained at the beginning of this section, identity trust level are standardized by several transnational frameworks, called Level of Assurance. OpenID Connect allows to request a specific authentication level with the Authentication Context Class Reference (ACR) parameter, referring to the ISO entity authentication assurance framework [15]. Data collected during our investigation (see section 3.1) did not reveal ACR parameter usage in OIDC requests.

Out of the hundred and three observed RPs, we estimate that fourteen have an implicit trust relationship with their IdP. In most cases, these RPs only offer a single compatible IdP from the same company. Four of these RPs are also classified as Specialized/-, indicating that they would not be able to accept any IdP in the first place. We found no occurrence of an explicit trust relation in our panel.

Regarding our third hypothesis, it appears difficult to judge if trust is an issue that would impose manual configurations of IdP/RP relationships. To some extent, a decision to support a particular IdP can be considered as an implicit trust decision. But whether RPs are willing to trust other IdPs, in order to offer more control to their user, remains an open question. It seems to us, that a solution to simplify the discovery and registration of IdP endpoint, should nonetheless give RPs the option to control the range of compatible IdPs and the authentication strength for trust reasons.

4 Proposal for a web based identity browser management and API

Frameworks for authorization delegation give users control over the information they want to share to other websites. Used as SSO solutions, they force users to choose between setting up a new account or sharing some private information. While IdPs should offer privacy-friendly solutions, websites also have a responsibility as clients of these solutions. As such they should allow login through trusted and privacy-friendly IdPs. However, as presented in Section 2.3, users often don't have much choice regarding IdPs.

In Section 3, we considered several hypothesis that could prevent the implementation of IdP discovery mechanisms. From our observation, we conclude that the OIDC standard could provide this feature to at least 58% of observed websites. However we also found several hindrance to this solution:

- the lack of implementation by IdPs and RPs.
- the difficulties for users of knowing and configuring their identifiers in the discovery process.

As a solution to these issues, we propose to add identity management functionalities to web-browser through a Web API.

Several tentatives have been made to develop Internet's "Missing Identity Layer" [16], without clear success. As an example, Windows Cardspace [17], first released in 2007, was Microsoft's solution to propose an integrated identity management experience to Windows' users. This identity metasystem allowed users to select InfoCards to prove identity claims, *e. g.* name, age, to requesting applications.

An identity enabled web browser should let users configure several identity for signing into webpages. From a user perspective, the browser would provide functionalities and associated interfaces to define new identities, register passwords, display login prompt, and define website preferences, *i. e.* which identity to use with which web site. Some of these functionalities are already provided by web browser, *e. g.* login and password storage, but without the coherency of a full identity management experience.

Recent trends in web browser development tend to indicate that browser makers are looking to offer a more complete browser experience. Personal preference synchronization, official plugins bringing new functionalities, *e. g.* Firefox Hello[4], although being discontinued, offers a web-calling service integrated in Firefox. Or simply access to an application marketplace. Particularly related to our interest, Google Chrome offers an OAuth 2 API[5] to Chrome Apps (third party applications running inside Chrome). This API implies that users can use a Chrome interface to choose an identity, sign-in, and modify some of their informations such as their profile picture. However, Chrome only offers integration with Google's identity.

WebRTC identity specification also offers some kind of browser identity, but limited to user-to-user authen-

4 https://www.mozilla.org/en-US/firefox/hello/

5 https://developer.chrome.com/apps/app_identity

tication. The specifications offers abstract authentication protocol and a discovery functionality. In itself the WebRTC specification pose some interesting concepts, but is limited in its capacities and interaction capability with the user. We also note that its implementation in web browser is lacking behind other WebRTC components.

4.1 Possible architecture for browser identity

We envision two possible architectures for a web browser offering an identity API to websites and identity management functionalities to users. We think that the main requirement for such a solution would be to let user choose their own identity service in order raise their trust level in web SSO, including privacy concerns. This should of course come with appropriate security on the device or browser, to store and manipulate token and credentials.

4.1.1 Discovery and registration through the browser

The first architecture, and possibly the lighter one, would be to use the browser in a pre-OAuth step. The user would first register his identity parameters and select the one to use when prompted by the browser. In turn, the browser would offer discovery and dynamic-registration services to web sites through a web API. As this solution build on top of existing protocols and given the results of our investigation we estimate that it could be implemented by a large proportion of web site using SSO (see Section 3).

However, this architecture binds the API to a single protocol suite, *i. e.* OAuth 2/OIDC dynamic registration. This is a major issue as it may be incompatible with future identity protocol evolutions.

4.1.2 Web authentication proxy

The second solution is more integral. In this architecture the browser would serve as an abstracting interface between the web site and the identity protocol in use. WebRTC identity introduces this mechanism with the IdP Proxy component which offers identity assertion generation and verification functions for the purpose of web-calling authentication (see Section 2.2). While WebRTC identity only offers authentication through an identity assertion, it could be expanded to offer more resources in a user to website relation.

As proxy for authentication and authorization services, websites would consume user resources, *i. e.* authentication assertion, profile information, through a web API, as shown in Figure 6. In this situation, the browser would be responsible for authorization management of these resources. HTTPS could be used for RP authentication, leveraging existing browser security implementation. User profile resources could be stored by local caching on the browser, or directly accessed on the IdP through the browser. Following the result of our investigation, this browser API should support RP of class authentication and profile.

For discovery, an IdP proxy similar to the one defined by WebRTC identity would be downloaded by the browser from a standard location on the IdP. This proxy component would serve the purpose of abstracting the protocol used by the IdP, to fit the browser API and assertion format.

A standard data format for user profile information should be defined for website to consume. We note that this work is already covered by the OIDC specification. The issue with this architecture may be in defining the browser to IdP Proxy API. Indeed, this interface should support existing protocols (or a subset) and still keep enough flexibility for future evolutions. On the other hand, this standardization process would take place between identity providers and browser makers. For website developers, the exposed web API would be stable and the standardization process transparent.

4.2 Prototype implementation

To evaluate the practicability in terms of use and development of our solutions, we implement a prototype for the Web authentication proxy architecture[6].

This architecture relies on three components interacting together: (a) a web-browser modification providing a JavaScript API to website and a user interface for identity selection, (b) a website –client and server side– requesting an authentication from the user and able to understand and verify the provided identity assertion, and (c) an IdP Server able to provide a suitable identity assertion through a WebRTC compatible IdP Proxy.

Figures 7 and 7 show sequence diagrams representing the interactions between the different actors of our

6 https://github.com/Sparika/WebConnect

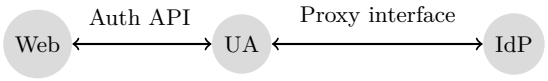


Fig. 6. Browser (UA) as proxy between web-site and IdP.

implementation. In Figure 7, after the user requested to login, the JavaScript (JS) client code calls the UA connect function which returns a promise[7] for an identity assertion. The UA then asks the user to choose one of its registered identity provider and then proceed to instantiate the corresponding IdP Proxy. Interactions with the IdP Proxy follows the WebRTC identity specification [6, 7]. Once the IdP Proxy returns the identity assertion to the UA, the UA resolves the connect promise. The website client thus receives the assertion.

Figure 7 shows the identity assertion verification. After the website client receives the identity assertion, the assertion is passed to the website server. In our example, the client requests a login URL with a HTTP GET method and passes the assertion as a query parameter. The server then extracts the JKU (JSON Web Key Set URL) parameter from the identity assertion header, and get the public key from the IdP over HTTPS at the provided location. This public key is then used to verify the assertion signature. The format and verification procedure of the assertion are detailed in Section 4.2.2. Once the assertion authenticity and integrity is confirmed, the server logs in the user and responds to the client GET.

4.2.1 Browser modification

We develop a browser extension to implement the browser modification. This solution was chosen for its simplicity in comparison to browser source modifications. A browser extension does not expose function to the global window scope. To simulate access to a standard Web API, we offer a JS file exposing our API functions to website client code. This script can then communicate with the extension code through the postMessage API.

The API offers two functions: *connect(request)* and *register(iss, sub, type, name)*. The register function creates an entry for an identity in the browser. It is used by IdP to let users store identity cards in the browser. *iss* and *type* are used to retrieve the IdP Proxy following the procedure in WebRTC identity. *sub* is an identifier for the user on the IdP, while *name* is the display name of the user on the identity card. The *connect* function is called by website to request an identity assertion in order to log in the user. The request parameter is a JSON object used to convey request parameters, *e. g.* scope.

Technically, our extension implements a graphical user interface for identity selection and configuration on top of the PeerConnectionIdP Firefox module. As we leverage and reuse the WebRTC identity specification, we developed our prototype for Firefox which is the only browser to support it.

4.2.2 Identity provider implementation

The role of the IdP is to provide an IdP Proxy at a standard location, and through it, authenticate the user and return an identity assertion. Our IdP implementation uses an OIDC server written with the NodeJS framework, modified to serve IdP Proxy. We implement our IdP Proxy as an OIDC client, which follows the OIDC implicit flow to interacts with the server.

The returned assertion is thus a signed JSON Web Token (JWS) [11]. In WebRTC, the party wanting to verify the assertion validity can download the IdP Proxy and call the verifyAssertion function. However, as we wanted to avoid IdP Proxy sandboxing on the website server, we used the JKU header in the JWS assertion. This allows a verifying party to retrieves, from the IdP, the public key used to sign the assertion and verify the JWS authenticity.

As standard for OIDC, the assertion payload contains Issuer Identifier (iss) and Subject Identifier (sub), respectively identifying the IdP and the user to the requesting website. The payload may also include OIDC user-info claims, such as name, address, or email.

4.2.3 Website implementation

Besides calling the API to get an identity assertion, a compatible website must also be able to understand and verify the assertion. In our implementation, the JS code on the client side sends the assertion to its backend server for verification and login. This is done by a GET to a login URL with the assertion transmitted as a URL query parameter.

7 http://www.ecma-international.org/ecma-262/6.0/#sec-promise-objects

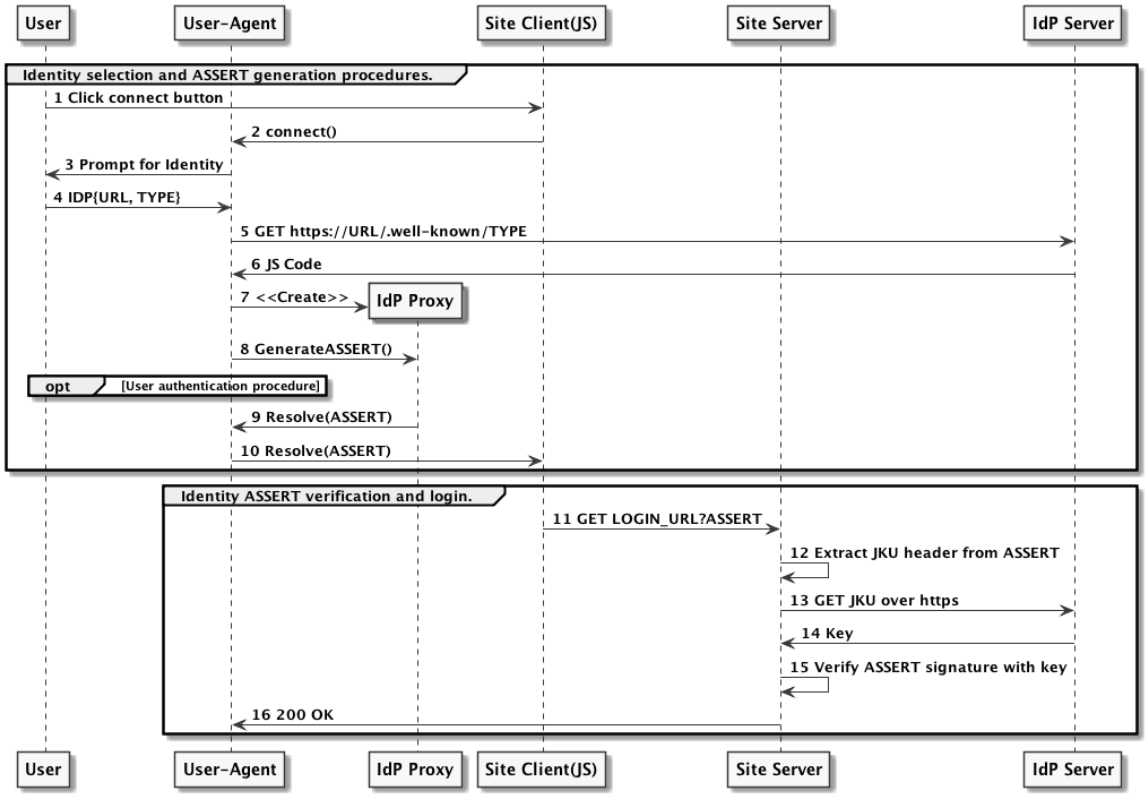


Fig. 7. Identity assertion management sequence diagram.

The assertion authenticity is then verified by the server. To do so, developer can use several libraries for JWS support. For our prototype we extended a JWS strategy for Passport[8] –a popular NodeJS authentication library– with support for JKU verification. Once the assertion has been verified, the server can extract relevant information from it and lookup for existing users in its database. If no user exists, the server creates a new entry on the fly. Thus this procedure serves the dual purpose of signing in and logging in. Ultimately, the user is returned to the relevant page through the HTTP response.

4.2.4 Security analysis

In comparison to a standard OAuth 2 flow, our implementation introduced two major changes that may have security implications. We discuss these changes in this section.

Firstly, in order to verify the validity of claims covered by a received assertion, the RP must verify the assertion's signature. This signature would have been produced by the IdP using a key pair exchanged with the RP during the registration process. In our solution we replaced the registration process, including the key exchange, by a verification of the JKU's origin. However, the OpenID Connect specification states "ID Tokens SHOULD NOT use the JWS [...] jku, or jwk header parameter fields". From IETF mail archives[9], it appears that assertions claims, including the iss and jku parameters, are considered to be self asserted until verified by a trusted key. To solve this issue, additional constraint could be added to the key's origin verification. For instance, using a standard well-known[18] path for the JKU URL also matching the ISS domain would prevent attacker from specifying any key.

Secondly, assertion manipulated by the javascript client code, and returned by the API's promise response, would be added to the page's global. The assertion could thus be read, and used, by a malicious cross-origin script embedded on the same page. This issue is similar to what can happen on an OAuth 2 implicit flow, where the

8 http://passportjs.org/

9 https://www.ietf.org/mail-archive/web/jose/current/msg03929.html

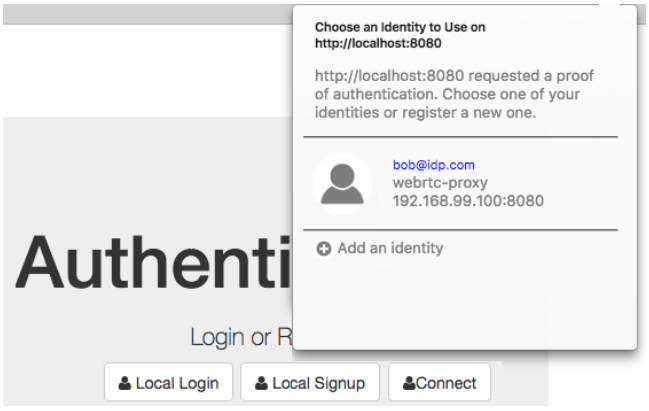


Fig. 8. Prototype user interface for the Authentication Web API.

client directly receives an access token. In OAuth 2 we would use a code flow to instead let the client exchange a code and authentication with the token endpoint to get the token. But as in our solution the RP/client cannot be authenticated by the IdP, the code flow could not be used. A solution could be to use a sort of code exchange, leveraging TLS mutual authentication between the RP and IdP. Alternatively, the browser could protect the assertion from the Javascript code and transmit it directly to the RP sever. Action 10 and 1, respectively from Figure 7 and 7, would be replaced by a single message from the User-Agent to the Site Server. The website redirection uri would be passed as a parameter to the connect function, action 2 on Figure 7.

4.2.5 Usability

From the end-user perspective the overhead is quite limited. Compared to current authentication process, the main addition is a one time configuration of the web-browser. This configuration can be done in a single action with the *register* function of the API. Identity selection on login request is also similar to current SSO solutions. Preferences storage by the browser may help reduce it further. However, the user-experience does not constitute our field of expertise and we did not conducted user studies.

Regarding developer, we also believe the additional work to be limited. Table 3 compares the amount of new code lines to the total for each modules of our prototype implementation. In proportion, the biggest task is to develop the browser modification and IdP Proxy, which are new concepts. These developments would however

Module	Total code lines	New code lines
Firefox Addon	0	417
IdP Proxy	0	197
Client site (.js)	693	66
Client site (.conf)	457	70
Passport JWS	1242	60
Total	2392	810

Table 3. Code lines written for the prototype implementation

be done once by browser makers and IdPs. On the client website, the main task is to configure the new authentication method and verify the assertion authenticity. However as we noted, library for JWS and OIDC ID Token support already exists. Modifying the existing JWS verification library required 60 new lines over a total 1242 code lines, while configuring the new strategy required 70 code lines, mostly copy-pasted from other strategies.

5 Conclusion and future work

In this paper, we showed how users are drastically limited in their choice of an IdP and as a result, unable to use a trusted IdP of their choice. Using the WebRTC identity specification as an example, we demonstrated the type of privacy risks that user may be exposed to in the near future. Postulating that allowing users to choose freely their IdP would help solve these issues, we investigated the reasons that may prevent the usage of IdP discovery mechanisms.

Our results show that while state of the art protocols (OAuth 2 and OIDC) prove to be adequate, lack of implementation by IdPs, and more particularly the non-implementation of OIDC optional Discovery and Dynamic Registration specifications is a cause of this problem. It seems to us that a push for these implementations could come from RPs and smaller IdPs.

Providing an integrated identity management on web browser is also a key to reduce the implementation cost for RPs and to provide a fluid and simple user experience. To this end we proposed an architecture to place the browser in an active role between RPs and IdPs, leveraging its trusted position. We implemented a prototype and reported on the solutions adopted and security issues encountered during its development.

As future work, we intend to extend our browser extension collecting OAuth request URL. We will then distribute it to the public in order to aggregate OAuth

usage and in return offer recommendations on privacy preserving identity solutions. We will also write a technical draft specification to disseminate our browser identity solution. Additional researches are required in order to facilitate this work. In particular, we will conduct user studies to evaluate the impact on SSO acceptance of our solution.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 645342, project reTHINK.

References

[1] L. Lynch, "Inside the identity management game," *IEEE Internet computing*, vol. 15, no. 5, p. 78, 2011.

[2] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 657–666.

[3] A. Jøsang, M. A. Zomai, and S. Suriadi, "Usability and privacy in identity management architectures," in *Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68*. Australian Computer Society, Inc., 2007, pp. 143–152.

[4] D. Reed, L. Chasen, and W. Tan, "Openid identity discovery with xri and xrds," in *Proceedings of the 7th symposium on Identity and trust on the Internet*. ACM, 2008, pp. 19–25.

[5] D. Mills, "Introducing browserid: a better way to sign in," 2011.

[6] "WebRTC 1.0: Real-time Communication Between Browsers." [Online]. Available: https://www.w3.org/TR/webrtc/

[7] E. Rescorla, "WebRTC Security Architecture," 2015.

[8] A. Vapen, N. Carlsson, A. Mahanti, and N. Shahmehri, "Information sharing and user privacy in the third-party identity management landscape," in *IFIP International Information Security Conference*. Springer, 2015, pp. 174–188.

[9] S.-T. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov, "What Makes Users Refuse Web Single Sign-on?: An Empirical Investigation of OpenID," in *Proceedings of the Seventh Symposium on Usable Privacy and Security*, ser. SOUPS '11. ACM, 2011, pp. 4:1–4:20.

[10] M. Sporny, T. Inkster, H. Story, B. Harbulot, and R. Bachmann-Gmür, "Webid 1.0: Web identification and discovery," *Editor's draft, W3C*, 2011.

[11] M. Jones, J. Bradley, and N. Sakimura, "Json web signature (jws), rfc 7515," Internet Engineering Task Force (IETF), Tech. Rep., 2014.

[12] N. Sakimura, J. Bradley, and M. Jones, "OpenID Connect Discovery 1.0," 2013.

[13] ——, "OpenID Connect Dynamic Client Registration 1.0," 2013.

[14] P. Jones, J. Smarr, G. Salgueiro, and M. Jones, "Webfinger," 2013.

[15] "ISO/IEC 29115:2013 - Information technology – Security techniques – Entity authentication assurance framework." [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45138

[16] K. Cameron, "The laws of identity," *Microsoft Corp*, 2005.

[17] "Introducing Windows CardSpace." [Online]. Available: https://msdn.microsoft.com/en-us/library/aa480189.aspx

[18] M. Nottingham and E. Hammer-Lahav, "Defining well-known uniform resource identifiers (uris)," Tech. Rep., 2010.