User Identity for WebRTC Services: A Matter of Trust

Secured interpersonal communications should come with asserted user identities and trust between the involved parties. No single trust model exists in Web Real-Time Communications services, so neither is there a single identity architecture. The authors discuss different models for provisioning user identity and their impact on user privacy.

Victoria Beltran and Emmanuel Bertin
Orange Labs

Noël Crespi
Institut Mines-Telecom

"On the Internet, nobody knows you're a dog" was the caption for one of the most famous *New Yorker* cartoons (drawn by Peter Steiner in 1993). More than 20 years later, assurance of user identity remains a challenge, and user impersonations are more frequent. Robocalls are so numerous in the US that the Federal Communications Commission (FCC) has addressed the problem by organizing a contest to combat them (http://robocall.challengepost.com). Now, Web Real-Time Communications (WebRTC)[1] is adding Web users to the already fragmented landscape of communication services.

WebRTC is a disruptive HTML5 technology driven by Internet companies such as Google and Mozilla, as well as by telecom equipment manufacturers such as Cisco and Ericsson.[2] It aims to offer Web developers native browser tools for inserting real-time media exchange into their webpages, including browser-to-browser audio and video communication as well as data exchange (for example, for instant messaging or file transfer). In practical terms, this means users need no longer download, install, and manually configure an application or use some proprietary plug-in in the browser to communicate. The browser makers integrate all needed functions: codecs, management of exchanged streams, APIs, and so on. WebRTC leaves the signaling plan to service developers, who are free to implement a standard (such as the Session Initiation Protocol [SIP] or the Extensible Messaging and Presence Protocol [XMPP]) or proprietary call-establishment protocol. This constitutes a new signaling paradigm based on a shared JavaScript code between peers rather than on a standardized protocol.[3] The only requirement is that the session negotiation data be represented by the Session Description Protocol (SDP).[4]

WebRTC's rise has generated several questions, including how users will verify the identity of other participants in a WebRTC communication and how to establish trust in users' identities. Here, we provide a wider vision of this topic by discussing how identity models and protocols apply to WebRTC. We also introduce different trust models and describe their impact on user privacy.

Identity for Communication Services

Whether a dedicated protocol or a JavaScript code handles the signaling, some things don't really change: we're talking about interpersonal communication. As usual, Alice is calling Bob (both Alice and Bob could, of course, be services, as with after-sales interactive voice-response systems). We therefore must first consider the usual way identity is managed for communication services.

In the telephony world, identity is generally managed in a shared way between providers. An identity is tied to a publicly known address such as a phone number or SIP URI that enforces international standardized rules common for all communication providers (CPs). These addresses play two roles: in the signaling role, they route the session initiation message to the callee; in the identity role, they let the callee identify the caller and vice versa. Identity management is coupled here with call routing. Authentication is usually achieved within the signaling plane, with password- or certificate-based mechanisms. As far as trust is concerned, both Alice and Bob trust their communication providers, and the providers trust each other.

Identity on the Web

Things are quite different on the Web. User identity is basically an immutable link between a profile (such as a name, email address, and phone number), authorization rules, and credentials (for example, login/password). Identities only let users log into a service; they aren't used for routing purposes. They aren't specified by common rules but are the responsibility of the individual service provider. Although users trust their Web services, global trust and interoperability between providers don't exist.

Nevertheless, widespread single-sign-on (SSO) systems (such as Facebook Connect) let Web services retrieve user identity information from a dedicated provider.[5] Here, user authentication is delegated from one service, known as the *relying party* (RP), to a third-party service, known as the *identity provider* (IdP). SSO protocols basically define the interactions between the RP, browser, and IdP that let the RP receive identity assertions. To date, the Web uses only a few SSO protocols to any extent, mainly OAuth2.0[6] and OpenID Connect (OIDC; http://openid.net/specs/). A more recent approach that hasn't yet seen wide adoption is BrowserID (https://github.com/mozilla/id-specs/blob/prod/browserid/index.md). With all the protocols, regardless of the SSO protocol, users trust their IdP on one side (to manage their identities) and their service providers on the other (to access their profile and provide the desired service).

Identity Provision in WebRTC

Let's now focus on communications over the Web. User identities in WebRTC services can be involved in two different activities:

- the authentication process by which the service — here, the CP — authorizes the user to log in; and
- the verification of the peer users' identity within a call (or any data transfer).

Although user login is a matter for each CP (handled via the usual authentication mechanisms), the RTCWEB working group (WG; http://tools.ietf.org/wg/rtcweb/charters) is addressing methods to enable verifying user identities. In particular, the main question is how to provide Bob with Alice's identity and vice versa so both of them can reliably decide to accept or reject the call; this is commonly known as *identity provision*. Readers unfamiliar with WebRTC would respond immediately that the CP authenticates users and hence provides their identity. This isn't the case, at least not as the IETF considers it. The RTCWEB WG proposes decoupling identity provision from CPs via a third-party IdP, so trust between users is only built on an external entity.[7]

Having a new player in the WebRTC model (the IdP) means new issues arise — that is, how to retrieve, deliver, and verify each participant's identity (see Figure 1).

Identity delivery must be the CP's responsibility because it's in charge of the signaling path between Alice and Bob. WebRTC provides browsers with a JavaScript interface for session negotiation based on SDP, so the most reasonable solution has been to include user identities as any other session negotiation parameter in SDP objects. The retrieval and verification of user identities will depend rely on the SSO protocol, as we explain later.

User Identities with SSO Protocols

SSO protocols were conceived for user login purposes rather than identity provision.

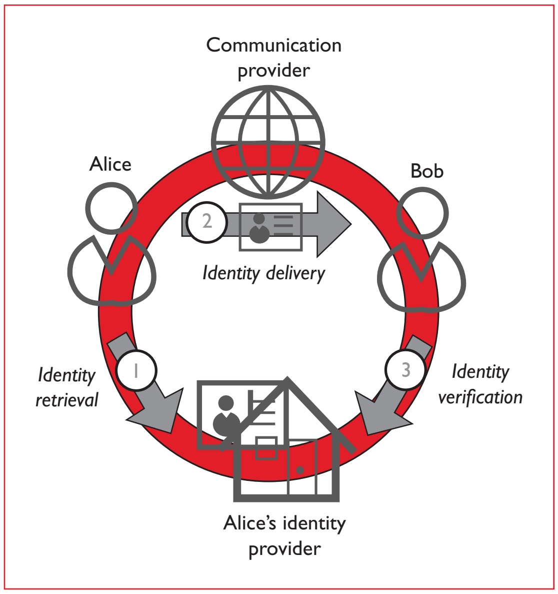

Figure 1. Collaboration circle for user provision. (1) Alice retrieves her identity from her identity provider (IdP); (2) this identity is sent to Bob through the communication provider (CP), and (3) Bob verifies Alice's identity with her IdP.

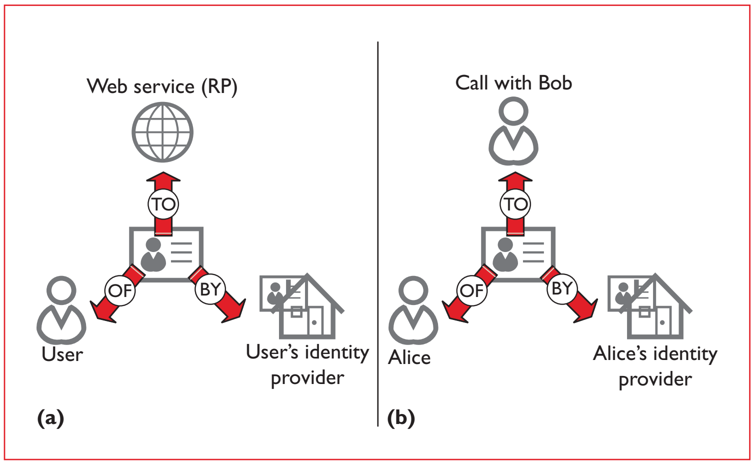

Figure 2. Conceptual relationships of user identities in (a) single-sign-on and (b) Web Real-Time Communications (WebRTC): ownership of the identity (of), provider of the identity (by), and target of the identity (to).

Adopting them for the latter undoubtedly means a conceptual shift that could affect user privacy if system architects aren't aware of it. User identities for login purposes correspond to the conceptual model depicted in Figure 2a. User identities' "of," "to," and "by" relationships are presented, if they are, by each SSO protocol in a different way. For instance, OAuth2.0 doesn't define the identity format, but only the interactions the RP must undertake to obtain user authorization in the form of an access token. Once the RP has obtained this token, it can request the user's identity information from the IdP. Thus, this protocol does not explicitly represent the model in Figure 2a. Nevertheless, the service can use HTTPS at the transport level to authenticate the IdP's certificate (that is, the "by" relation).

With OIDC and BrowserID, user identities explicitly represent the triple relationship indicated in Figure 2a. In OIDC, A JSON structure called ID Token includes the user's identity information, the target service, and, optionally, the IdP's signature. In BrowserID, a user identity is composed of two parts: the IdP generates one part to state that the user has been authenticated. This part mainly contains the user's email address and public key, and the IdP's signature. The browser itself generates the other part to state that the user authorizes the service to access his or her identity. This part specifies the target service and contains the user's signature.

User identities for identity provision in WebRTC services involve a conceptual change, as Figure 2b shows. Because Alice wishes to talk to Bob, her identity assertion is targeted to Bob and, more technically speaking, to the call being negotiated with Bob instead of a Web service. Thus, Alice's identity should carry enough information to ensure that her identity is delivered only to Bob and only for the current media flow (and not for any future media flow), thereby ensuring the "to" relationship in Figure 2b.

To associate Alice's identity with the call to Bob, the IETF has proposed a cryptographical binding between Alice's identity and her key used for the Datagram Transport Layer Security (DTLS) handshake on the media plane.[7] This handshake lets Alice and Bob be sure that they are going to talk to the same user with whom they negotiated the communication session. During call negotiation, they exchange the fingerprint of their keys. Each participant can therefore verify that the other's key in the media plane corresponds to the received fingerprint. How user identities are actually linked to call fingerprints still isn't obvious, and the RTCWEB WG only illustrates BrowserID and OAuth2.0, as we explain in detail later.

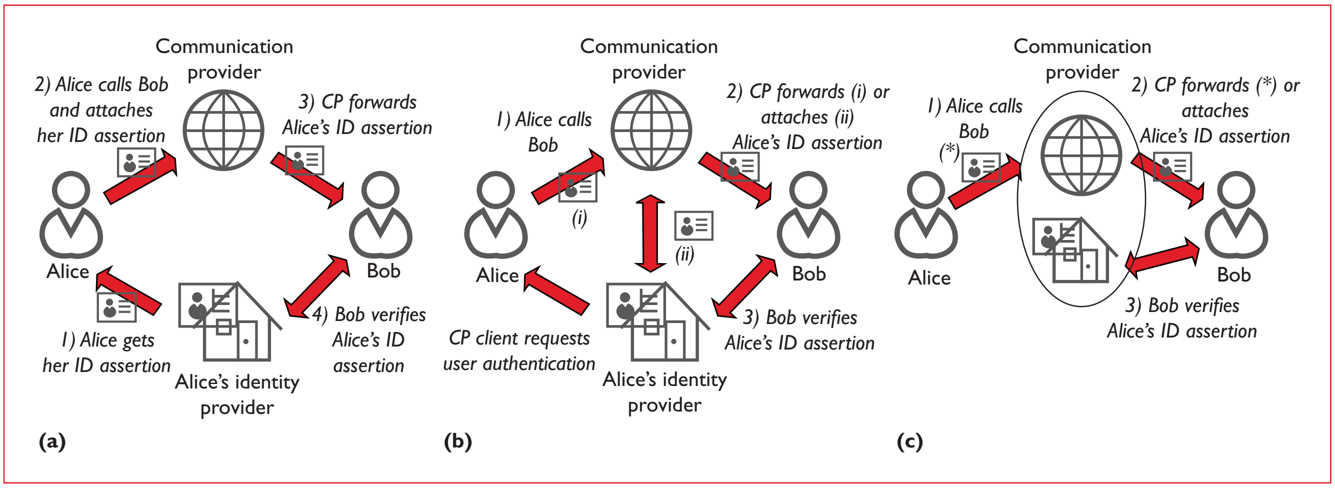
Communication provider
2) Alice calls Bob and attaches her ID assertion
3) CP forwards Alice's ID assertion
Alice
Bob
1) Alice gets her ID assertion
Alice's identity provider
4) Bob verifies Alice's ID assertion
(a)
Communication provider
1) Alice calls Bob
(i)
(ii)
2) CP forwards (i) or attaches (ii) Alice's ID assertion
Alice
Bob
CP client requests user authentication
Alice's identity provider
3) Bob verifies Alice's ID assertion
(b)
Communication provider
1) Alice calls Bob
(*)
2) CP forwards (*) or attaches Alice's ID assertion
Alice
Bob
3) Bob verifies Alice's ID assertion
(c)

Figure 3. Trust models for identity provision in Web Real-Time Communications: (a) nontrust; (b) partial trust using (i) OpenID Connect (OIDC) implicit flow and (ii) OIDC authorization code flow; and (c) full trust, in which (*) the ID assertion is downloaded from the communication provider (CP) and attached by the client.

The WebRTC Identity Model: BrowserID and OAuth2.0

The RTCWEB WG proposes a model[7] that completely excludes the CP from the retrieval and verification of user identities (see Figure 3a). The user's browser obtains the user identity assertion from the IdP and attaches it to the SDP offer. The CP merely forwards the identity. This model relies on a generic communication protocol implemented on the user's browser that lets the browser retrieve user identity assertions from the IdP and verify them. This protocol defines JavaScript Object Notation (JSON)-based interactions between a *PeerConnection* object (that is, the browser object that manages the interactions with the remote peer) and an object called the *IdP Proxy*. An IdP Proxy is a piece of code that the user's browser downloads from a well-known URL in the IdP's domain.

SSO protocols can be classified into two modes of operation: browser-centric and RP-centric as shown in Figure 4. The WebRTC identity model perfectly fits into the former, whose only representative is BrowserID. With this protocol, the RP and IdP are totally decoupled. The user's browser requests that the IdP authenticate the user and then delivers the authentication assertion to the RP. The browser generates the final user's identity after receiving the user's authentication assertion from the IdP. In WebRTC services, this browser-centric mode allows the browser to cryptographically bind the call fingerprint to the user identity locally (by a BrowserID code downloaded from a trusted Web service). It can be easily accomplished by including the fingerprint in the user's identity assertion that is signed with the user's public key by the browser. There is no need to send the fingerprint to the IdP.

Conversely, the RP-centric operation mode of SSO protocols in Figure 4b requires the IdP to authorize the RP, for which the RP and IdP must directly communicate. OAuth2.0, along with other protocols such as OpenID and OIDC, corresponds to this operation mode. Although this approach doesn't match with the browser-centric model that the RTCWEB WG promotes, the WG mentions a possible circumvention of OAuth2.0 to fit with this model. In OAuth2.0, the RP's client code executes on the browser and communicates with the IdP to get the user's authorization. This can be an authorization code that the RP's server side uses to retrieve the user's identity. To adapt OAuth2.0 to WebRTC, the user's browser — and more specifically, the IdP Proxy — rather than the service could work as the RP toward the IdP. The IdP Proxy would add the call's fingerprint to the authentication request sent to the IdP. The IdP would authenticate Alice, register the fingerprint as part of her identity, and return an authorization code. The PeerConnection object would attach the authorization code to the SDP offer. Thus, when Alice calls Bob, Bob will receive not Alice's identity but rather the authorization code to get her identity from the IdP. As part of Alice's identity, Bob will get the call's

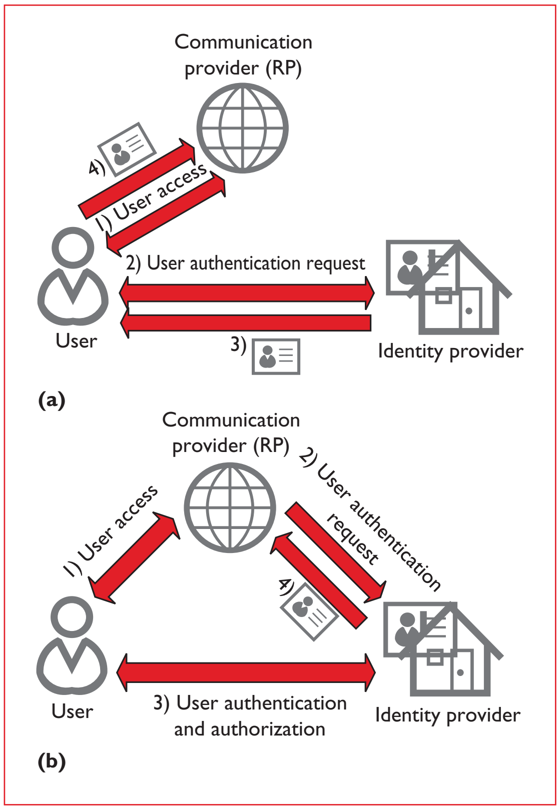
Communication provider (RP)
4)
1) User access
2) User authentication request
User
3)
Identity provider
(a)
Communication provider (RP)
2) User authentication request
1) User access
4)
User
3) User authentication and authorization
Identity provider
(b)

Figure 4. Operation models of single sign-on (SSO) protocols (logical flows) in (a) browser-centric mode and (b) relying party (RP)-centric mode. The figure doesn't show implementation-specific flows (such as HTTP redirection).

fingerprint that Alice's browser registered at the IdP. How the IdP is going to verify that Bob has been given authority to see Alice's identity (that is, the RP has delegated its authorization to him) is still doubtful.

OpenID Connect: Why Not?

OIDC constitutes a set of extensions on top of OAuth2.0, mainly enhancements on security and identity formats. This protocol therefore fits into the RP-centric operation mode in Figure 4b. Although the WebRTC security architecture hasn't yet considered this protocol, several of its features make it a better candidate than OAuth2.0. As mentioned, OIDC defines a JSON structure called ID Token that carries information about the user's authentication event. The IdP generates an ID token as a result of a successful user authentication. ID tokens enable the browser-centric approach in Figure 3a more easily than OAuth2.0 authorization codes. OIDC authentication requests can indicate which information should be included in the ID token. Such information can be standard (that is, OIDC-defined standard attributes for user profile information) or not. Authentication requests can even set the value of the information to be returned into the ID token. It is therefore possible to request that the IdP return certain information signed in an ID token. Thus, to bind the user identity (that is, an ID token) cryptographically with the call fingerprint, the fingerprint can be added to the authentication request as a URL parameter (specifically, inside the "claims" parameter; see http://openid.net/specs/openid-connect-core-1_0.html). Moreover, IdPs can return ID tokens directly to the user's browser without requiring a previous authorization code. Thus, the PeerConnection object could attach the ID token, instead of an authorization code as in OAuth2.0, to the SDP object. Because the IdP can sign ID tokens, Bob would be able to verify that Alice's IdP has generated her identity.

In comparison to BrowserID, OIDC has two main advantages for the WebRTC security model. First, identity isn't linked to an email address. This can improve user privacy properties, as we discuss in detail later. Second, only the IdP generates identity assertions to each RP. On the contrary, in BrowserID, a local code on the browser that's totally independent from the IdP generates the final identity assertion. Thus, OIDC (like OAuth2.0) can better enforce authorization rules through a trustworthy IdP.

Table 1 summarizes how BrowserId, OAuth2.0, and OIDC represent the relationships from Figure 2b. In this table, "call info" indicates the call's fingerprint and any other information needed to uniquely identify a call with the user. (The call fingerprint doesn't necessarily uniquely identify the call because DTLS keys are potentially reusable.)

To Trust or Not to Trust?

The identity provision model for WebRTC was conceived to work "without trusting the signaling site."[7] But what does this trust, or rather lack of it, mean? To the RTCWEB WG, it means that the CP isn't going to participate at all in retrieving and verifying user identities (see Figure 3a). Nevertheless, because it's in charge of the signaling path, the CP can know the user's identity (including any personal information in it) without the user's consent. Furthermore, it isn't clear that this non-trust model will be able to meet requirements in

Table 1. Triple relationship of identity assertions in single-sign-on protocols.

Protocol	Relationship		
	Of *Alice*	By *identity provider (IdP)*	To *call with Bob*
BrowserID	Email and Alice's signature in identity assertion	IdP's signature	Call info signed by Alice
OAuth2.0	Alice's profile at the IdP	HTTPS certificate	Call info in Alice's profile at the IdP
OpenID Connect (OIDC)	Alice's attributes in identity assertion	IdP's signature	Call info signed by the IdP

real-world scenarios, including those the RTCWEB WG has addressed (for example, emulating a telephony terminal, which implies that a user's CP asserts his or her identity to place and receive calls with the PSTN).[8] The user should have the last word about whether to trust the CP, and this will depend on each service case. To provide a broader view on identity provision, we describe two alternative models: partial and full trust. We focus here on a case in which both users belong to the same CP. Other scenarios for interoperability between CPs are currently out of our scope.

Partial-Trust Model

As Figure 3b illustrates, the CP might be involved in obtaining the user's identity from the IdP. The CP's client code on the user's browser would request that the IdP authenticate the user. This model would be implemented by SSO protocols such as OAuth2.0 or OIDC. These protocols allow either the client part (that is, a JavaScript code) or the server part of the CP to receive user identities via implicit or authorization code flows,[6] respectively, as Figure 3b shows. In a case where the CP's client code receives the user identity, it would pass this identity to the PeerConnection object for attachment to the call SDP. It might also be possible to have a solution based on the existence of an IdP Proxy. This would, however, require passing protocol-specific parameters to the proxy (for example, the CP's application ID). In a case where the CP server receives the user identity, it would attach the identity to the call request on the network side. The CP server would therefore have to bind cryptographically the fingerprint of the call request's SDP to the identity. Regardless of the operation flow, as previously mentioned, OIDC can enable the binding between the user identity and the fingerprint by sending this information as a URL parameter of the authentication request.

This model enables the CP to control the identity-provisioning process. A CP can choose to work only with a few preferred IdPs, for example, to guarantee that the identities of all its users come from a trustworthy IdP, or because this IdP will provide the CP with more personal information. Moreover, in specific cases such as corporate communications (with the enterprise acting as a CP), the choice of IdP will surely be subject to security policies, which the CP can enforce.

Full-Trust Model

This model is the most familiar to Web users — that is, placing full trust in the CP. In this case, users authenticate to the CP and trust it to deliver their identities to other users. Thus, the CP plays the role of IdP by generating user identity assertions (see Figure 3c). The CP could attach the user identity to the SDP on the server side. However, this would require that the server take the fingerprint of the call's SDP and bind it to the identity. Conversely, the CP's client code could download the user identity onto the browser and from there bind the identity to the call request's fingerprint. This second solution doesn't require that the CP server process SDPs.

This model provides obvious simplicity advantages for big providers that can compete on both the identity and communications markets such as Facebook or Google. This case would be somewhat similar to legacy communication services, in which service providers manage both signaling and identity, and will thus be perfectly adapted for CPs providing in and out PSTN calls with asserted identities.

User Privacy

Let's now compare these models as regards user privacy. Encryption is built into WebRTC: a browser-to-browser encrypted media channel relies on a DTLS-Secure RTP key exchange — there is actually no way to send unencrypted media. However, data confidentiality and integrity isn't enough for a secure system. What does

Table 2. User privacy properties in identity provision models.

Identity provision model	Identification	Anonymity to participants	Anonymity to communication provider (CP)	Identity unlinkability	Identity confidentiality	CP unlinkability
Nontrust (browserID)	✓	—	—	—	—	✓
Nontrust (OAuth2.0/ OpenIDConnect)	✓	✓	✓	✓	—	✓
Partial trust (OAuth2.0/OIDC)	✓	✓	✓	✓	—	—
Full trust (no single sign-on)	✓	✓	—	—	—	n/a

it matter that you have confidential data on a secure channel if you can't be sure whom you're talking to? Your well-encrypted information might go to the wrong person.

Although data security provides confidentiality and integrity to the multimedia data exchanged between users, user privacy deals with user identities and their associated profile information. The RTCWEB WG hasn't really undertaken user privacy; it focuses mainly on data security. We identify six privacy properties in relation to user identities:

- *Identification* is a user's ability to verify that another participant in a communication is the person the user thinks he or she is.
- *Anonymity to participants* is the user's ability to remain anonymous to another communication participant. The IdP might generate anonymous identities that hide a user's real identity while asserting that it has been authenticated.
- *Anonymity to CP* is the user's ability to remain anonymous while authenticating to a CP.
- *Identity unlinkability* is the CP's inability to track user activity based on users' identities across communications.
- *Identity confidentiality* is a user's ability to send his or her identity information encrypted to thus have confidentiality from the CP.
- *CP unlinkability* is the IdP's inability to correlate a user across different CPs.

The ability of WebRTC services to accomplish these privacy properties depends on the identity provision mode (that is, nontrust, partial trust, or full trust) and the underlying SSO protocol (mainly BrowserId, OAuth2.0, or OIDC). Table 2 shows the user privacy properties that each identity provision mode offers.

Obviously, all the identity provision modes can identify users. Not all, however, can provide anonymity. Because BrowserID doesn't provide user anonymity (it requires public email addresses as user identifiers), neither does the nontrust model with this protocol. Likewise, identity unlinkability isn't possible because user identity assertions can be correlated via email address. Regardless of the SSO protocol used, the nontrust model always provides CP unlinkability.

OAuth2.0 and OIDC overcome BrowserID's limitations in the nontrust model by providing user anonymity and identity unlinkability. With OAuth2.0, these privacy properties are the IdP's responsibility. The protocol specification is independent of user identifiers, and its architecture doesn't require permanent user identifiers that could be used as keys to track usage. OIDC specifies the use of unique user identifiers that are local to the IdP and different than any user's contact address. Moreover, OIDC defines the use of pairwise user identifiers. The IdP can generate a different pairwise user identifier for each CP; hence, tracking the user identity assertions over the Web is impossible.

The partial-trust model shares all the privacy properties of OAuth2.0 and OIDC in the nontrust model except for CP unlinkability. Because this model involves the CP in identity provision, the IdP can track the CPs the user visits.

In the full-trust model, because the CP authenticates users and adds their identities to the call negotiation, identity unlinkability and user anonymity to the CP are impossible. However, the CP could support user anonymity to participants. Because no third-party IdP exists, the property of CP unlikability doesn't apply.

No identity provision model offers a solution for identity confidentiality. As stated in the IETF working draft,[7] users who wish privacy against CPs should use separate privacy-enhancing technologies such as Tor (www.torproject.org) — although this would imply very suboptimal performances.

The concept of trust or lack of trust in CPs still isn't clear in the WebRTC arena. The IETF is proposing a particular model for identity provision from the assumption that CPs aren't trustworthy. Although this model claims to be protocol-independent, the underlying SSO protocol strongly affects the security of user identities. We prompt the community to promote a wider vision of identity in the WebRTC standardization that meets the needs of real Web applications — the requirements of identity for communication are different from those for SSO. We did not consider how an IdP creates and manages identities. This would be another story; Alice can still be a talking dog — but the same dog as last time.

References

1. S. Loreto and S.P. Romano, "Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts," *IEEE Internet Computing*, vol.16, no. 5, 2012, pp. 68–73.
2. E. Bertin et al., "WebRTC, the Day After: What's Next for Conversational Services?" *Proc. 17th Int'l Conf. Intelligence in Next Generation Networks* (ICIN), 2013, pp. 46–52.
3. A. Bernard et al., "The Future of Web Applications or How to Move into the Post Standardization Area," IETF position paper, Oct. 2010.
4. J. Rosenberg and H. Schulzrinne, *An Offer/Answer Model with the Session Description Protocol (SDP)*, IETF RFC 3264, June 2002; www.ietf.org/rfc/rfc3264.txt.
5. L. Lynch, "Inside the Identity Management Game," *IEEE Internet Computing*, vol. 15, no. 5, 2011, pp. 78–82.
6. D. Hardt, *The OAuth2.0 Authorization Framework*, IETF RFC 6749, Oct. 2012; http://tools.ietf.org/html/rfc6749.
7. E. Rescorla, "WebRTC Security Architecture," IETF Internet draft, work in progress, July 2014.
8. C. Holmberg, S. Hakansson, and G. Eriksson, "Web Real-Time Communication Use-Cases and Requirements," IETF Internet draft, work in progress, May 2014.

Victoria Beltran is a postdoctoral researcher at Orange Labs. Her main interests include context-aware applications, identity management, and Internet multimedia systems. Beltran received a PhD in telematics engineering from Universitat Politècnica de Catalunya. Contact her at victoria.beltran@orange.com.

Emmanuel Bertin is a senior service architect at Orange Labs and an adjunct professor at the Institut Mines-Telecom. His activities are focused on communication networks, service engineering, WebRTC, and Web-based services. Bertin received a PhD in computer science from the University of Paris VI. He's a member of IEEE. Contact him at emmanuel.bertin@orange.com.

Noël Crespi is a professor and program director leading the Service Architecture Lab at Institut Mines-Telecom, as well as an adjunct professor at the Korea Advanced Institute of Science and Technology (KAIST) and an affiliate professor at Concordia University, Canada. Crespi received a PhD and Habilitation in computer science from the University of Paris VI. He is the coordinator for Institut Mines-Telecom for the standardization activities at the European Telecommunications Standards Institute (ETSI) and 3GPP. Contact him at noel.crespi@mines-telecom.fr.