RTCWEB
Internet-Draft
Intended status: Standards Track
Expires: September 13, 2012

E. Rescorla
RTFM, Inc.

March 12, 2012

RTCWEB Generic Identity Provider Interface
draft-rescorla-rtcweb-generic-idp-01

Abstract

Security for RTCWEB communications requires that the communicating endpoints be able to authenticate each other. While authentication may be mediated by the calling service, there are settings in which this is undesirable. This document describes a generic mechanism for leveraging existing identity providers (IdPs) such as BrowserID or OAuth to provide this authentication service.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 13, 2012.

Table of Contents

1. Introduction

Security for RTCWEB communications requires that the communicating endpoints be able to authenticate each other. While authentication may be mediated by the calling service, there are settings in which this is undesirable. This document describes a mechanism for leveraging existing identity providers (IdPs) such as BrowserID or OAuth to provide this authentication service.

Specifically, Alice and Bob have relationships with some Identity Provider (IdP) that supports a protocol such OpenID or BrowserID) that can be used to attest to their identity. While they are making calls through the signaling service, their identities (and the cryptographic keying material used to make the call) is authenticated via the IdP. This separation isn't particularly important in "closed world" cases where Alice and Bob are users on the same social network, have identities based on that network, and are calling using that network's signaling service. However, there are important settings where that is not the case, such as federation (calls from one network to another) and calling on untrusted sites, such as where two users who have a relationship via a given social network want to call each other on another, untrusted, site, such as a poker site.

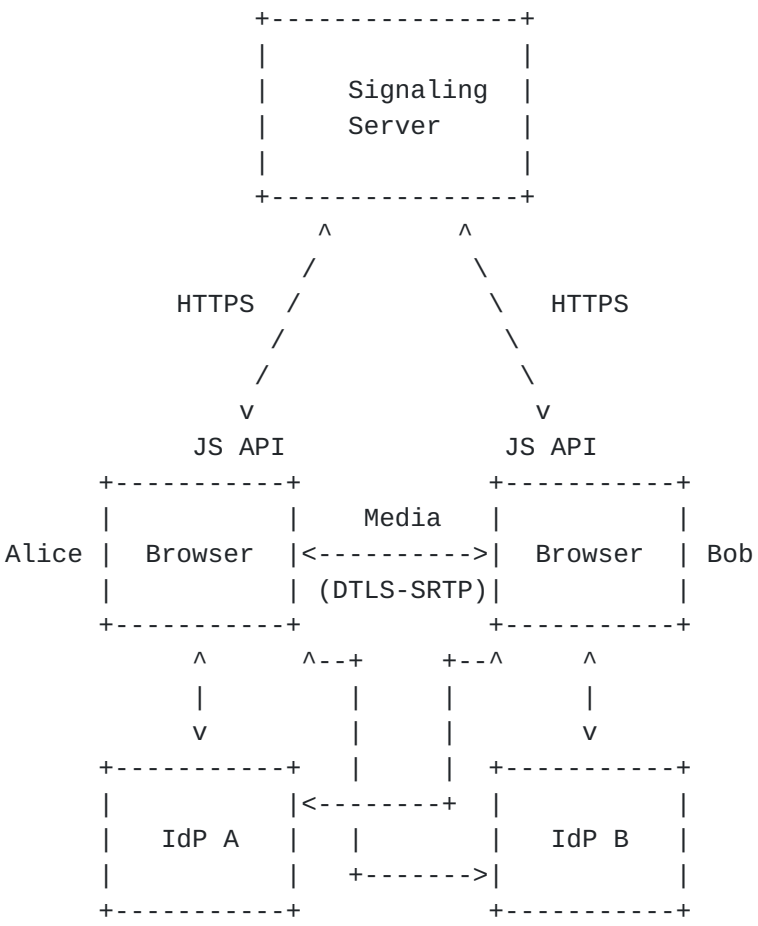


Figure 1: A call with IdP-based identity

Figure 1 shows the basic topology. Alice and Bob are on the same signaling server, but they additionally have relationships with their own IdPs. Alice has registered with IdP A and Bob has registered with IdP B. Note that nothing stops these IdPs from being the same, or indeed from being the same as the signaling server, but they can also be totally distinct. In particular, Alice and Bob need not have identities from the same IdP.

Starting from this point, the mechanisms described in this document allow Alice and Bob to establish a mutually authenticated phone call. In the interest of clarity the remainder of this section provides a brief overview of how these mechanisms fit into the bigger RTCWEB calling picture. For a detailed description of the relevant protocol elements and their interaction with the larger signaling protocol see [I-D.ietf-rtcweb-security]. When Alice goes to call Bob, her browser (specifically her PeerConnection object) contacts her IdP on her behalf and obtains an assertion of her identity bound to her certificate fingerprint. This assertion is carried with her signaling messages to the signaling server and then down to Bob.

Bob's browser verifies the assertion, possibly with the cooperation of the IdP, and can then display Alice's identity to Bob in a trusted user interface element. If Alice is in Bob's address book, then this interface might also include her real name, a picture, etc.

When/If Bob agrees to answer the call, his browser contacts his IdP and gets a similar assertion. This assertion is sent to the signaling server as part of Bob's answer which is then forwarded to Alice. Alice's browser verifies Bob's identity and can display the result in a trusted UI element. At this point Alice and Bob know each other's fingerprints and so they can transitively verify the keys used to authenticate the DTLS-SRTP handshake and hence the security of the media.

The mechanisms in this document do not require the browser to implement any particular identity protocol or to support any particular IdP. Instead, this document provides a generic interface which any IdP can implement. Thus, new IdPs and protocols can be introduced without change to either the browser or the calling service. This avoids the need to make a commitment to any particular identity protocol, although browsers may opt to directly implement some identity protocols in order to provide superior performance or UI properties.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Trust Relationships: IdPs, APs, and RPs

Any authentication protocol has three major participants:

Authenticating Party (AP): The entity which is trying to establish its identity.

Identity Provider (IdP): The entity which is vouching for the AP's identity.

Relying Party (RP): The entity which is trying to verify the AP's identity.

The AP and the IdP have an account relationship of some kind: the AP registers with the IdP and is able to subsequently authenticate directly to the IdP (e.g., with a password). This means that the

browser must somehow know which IdP(s) the user has an account relationship with. This can either be something that the user configures into the browser or that is configured at the calling site and then provided to the PeerConnection by the calling site.

At a high level there are two kinds of IdPs:

Authoritative: IdPs which have verifiable control of some section of the identity space. For instance, in the realm of e-mail, the operator of "example.com" has complete control of the namespace ending in "@example.com". Thus, "alice@example.com" is whoever the operator says it is. Examples of systems with authoritative identity providers include DNSSEC, RFC 4474, and Facebook Connect (Facebook identities only make sense within the context of the Facebook system).

Third-Party: IdPs which don't have control of their section of the identity space but instead verify user's identities via some unspecified mechanism and then attest to it. Because the IdP doesn't actually control the namespace, RPs need to trust that the IdP is correctly verifying AP identities, and there can potentially be multiple IdPs attesting to the same section of the identity space. Probably the best-known example of a third-party identity provider is SSL certificates, where there are a large number of CAs all of whom can attest to any domain name.

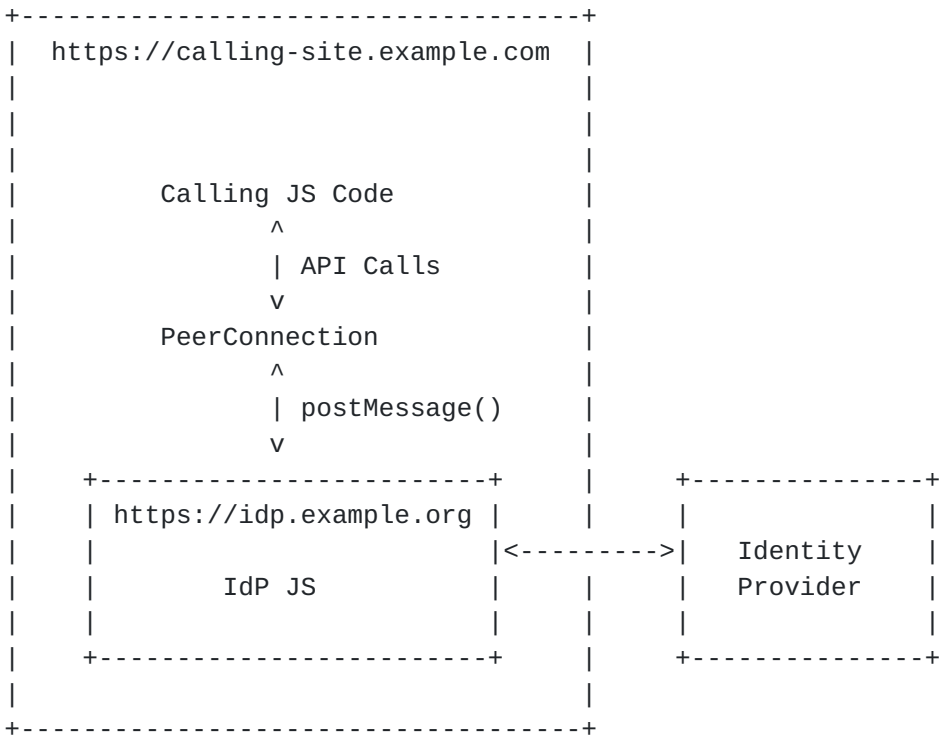
If an AP is authenticating via an authoritative IdP, then the RP does not need to explicitly trust the IdP at all: as long as the RP knows how to verify that the IdP indeed made the relevant identity assertion (a function provided by the mechanisms in this document), then any assertion it makes about an identity for which it is authoritative is directly verifiable.

By contrast, if an AP is authenticating via a third-party IdP, the RP needs to explicitly trust that IdP (hence the need for an explicit trust anchor list in PKI-based SSL/TLS clients). The list of trustable IdPs needs to be configured directly into the browser, either by the user or potentially by the browser manufacturer. This is a significant advantage of authoritative IdPs and implies that if third-party IdPs are to be supported, the potential number needs to be fairly small.

4. Overview of Operation

In order to provide security without trusting the calling site, the PeerConnection component of the browser must interact directly with the IdP. In this section, we describe a standalone mechanism based

on IFRAMEs and postMessage(), however, most likely this will eventually be superceded by WebIntents <http://www.webintents.com/>. [[OPEN ISSUE: I've been looking at WebIntents and I believe that it can be made to work but may require some modifications. I am currently studying the problem. More analysis to come.]]]].



When the PeerConnection object wants to interact with the IdP, the sequence of events is as follows:

1. The browser (the PeerConnection component) instantiates an IdP proxy (typically a hidden IFRAME) with its source at the IdP. This allows the IdP to load whatever JS is necessary into the proxy, which runs in the IdP's security context.
2. If the user is not already logged in, the IdP does whatever is required to log them in, such as soliciting a username and password.
3. Once the user is logged in, the IdP proxy notifies the browser (via postMessage()) that it is ready.
4. The browser and the IdP proxy communicate via a standardized series of messages delivered via postMessage. For instance, the browser might request the IdP proxy to sign or verify a given identity assertion.

This approach allows us to decouple the browser from any particular identity provider; the browser need only know how to load the IdP's JavaScript--which is deterministic from the IdP's identity--and the

generic protocol for requesting and verifying assertions. The IdP provides whatever logic is necessary to bridge the generic protocol to the IdP's specific requirements. Thus, a single browser can support any number of identity protocols, including being forward compatible with IdPs which did not exist at the time the browser was written.

5. Protocol Details

5.1. General Message Structure

Messages between the PeerConnection object and the IdP proxy are formatted using JSON [RFC4627]. For instance, the PeerConnection would request a signature with the following "SIGN" message:

```
{
  "type":"SIGN",
  "id": "1",
  "message":"012345678abcdefghijkl"
}
```

All messages MUST contain a "type" field which indicates the general meaning of the message.

All requests from the PeerConnection object MUST contain an "id" field which MUST be unique for that PeerConnection object. Any responses from the IdP proxy MUST contain the same id in response, which allows the PeerConnection to correlate requests and responses.

Any message-specific data is carried in a "message" field. Depending on the message type, this may either be a string or a richer JSON object.

5.1.1. Errors

If an error occurs, the IdP sends a message of type "ERROR". The message MAY have an "error" field containing freeform text data which containing additional information about what happened. For instance:

```
{
  "type":"ERROR",
  "error":"Signature verification failed"
}
```

Figure 2: Example error

5.2. IdP Proxy Setup

In order to perform an identity transaction, the PeerConnection must first create the IdP proxy. While the specific technical mechanism used is left up to the implementation, the following requirements MUST be met for security and interoperability.

- Any JS MUST run in the IdP's security context.
- The usual browser sandbox isolation mechanisms MUST be enforced with respect to the IdP proxy.
- JS running in the IdP proxy MUST be able to send and receive messages to the PeerConnection object using postMessage.
- Either window.parent or window.opener MUST be set such that messages sent with postMessage() arrive at the PeerConnection object. If both variables are set, they MUST be the same.
- Messages sent by the PeerConnection object MUST have their .origin value set to "rtcweb::://idp-interface". [TBD]

One mechanism for implementing the IdP proxy is as a hidden (CSS "display=none") IFRAME with a URI as determined in Section 5.2.1. The PeerConnection component will of course need to specially arrange for the origin value to be set correctly; as dicussed in Section 5.6, the fact that ordinary Web pages cannot set their origins to "rtcweb://..." is an essential security feature.

Initially the IdP proxy is in an unready state; the IdP JS must be loaded and there may be several round trips to the IdP server, for instance to log the user in. Thus, the IFRAME's "onready" property is not a reliable indicator of when the IdP IFRAME is ready to receive commands. Instead, when the IdP proxy is ready to receive commands, it delivers a "ready" message via postMessage(). As this message is unsolicited, it simply contains:

```
{ "type":"READY" }
```

Once the PeerConnection object receives the ready message, it can send commands to the IdP proxy.

5.2.1. Determining the IdP URI

Each IdP proxy instance is associated with two values:

domain name: The IdP's domain name
protocol: The specific IdP protocol which the IdP is using. This is a completely IdP-specific string, but allows an IdP to implement two protocols in parallel. This value may be the empty string.

Each IdP MUST serve its initial entry page (i.e., the one loaded by

the IdP proxy) from the well-known URI specified in "/.well-known/idp-proxy/<protocol>" on the IdP's web site. This URI MUST be loaded via HTTPS [RFC2818]. For example, for the IdP "identity.example.com" and the protocol "example", the URL would be:

```
https://example.com/.well-known/idp-proxy/example
```

5.2.1.1. Authenticating Party

How an AP determines the appropriate IdP domain is out of scope of this specification. In general, however, the AP has some actual account relationship with the IdP, as this identity is what the IdP is attesting to. Thus, the AP somehow supplies the IdP information to the browser. Some potential mechanisms include:

- Provided by the user directly.
- Selected from some set of IdPs known to the calling site. E.g., a button that shows "Authenticate via Facebook Connect"

5.2.1.2. Relying Party

Unlike the AP, the RP need not have any particular relationship with the IdP. Rather, it needs to be able to process whatever assertion is provided by the AP. As the assertion contains the IdP's identity, the URI can be constructed directly from the assertion, and thus the RP can directly verify the technical validity of the assertion with no user interaction. Authoritative assertions need only be verifiable. Third-party assertions also MUST be verified against local policy, as described in Section 5.4.1.

5.3. Requesting Assertions

In order to request an assertion, the PeerConnection sends a "SIGN" message. Aside from the mandatory fields, this message has a "message" field containing a string. The contents of this string are defined in [I-D.ietf-rtcweb-security], but are opaque from the perspective of this protocol.

A successful response to a "SIGN" message contains a message field which is a JS dictionary dictionary consisting of two fields:

idp: A dictionary containing the domain name of the provider and the protocol string

assertion: An opaque field containing the assertion itself. This is only interpretable by the idp or its proxy.

Figure 3 shows an example transaction, with the message "abcde..." being signed and bound to identity "ekr@example.org". In this case,

the message has presumably been digitally signed/MACed in some way that the IdP can later verify it, but this is an implementation detail and out of scope of this document. Line breaks are inserted solely for readability.

```
PeerConnection -> IdP proxy:
  {
    "type":"SIGN",
     "id":1,
     "message":"abcdefghijklmnopqrstuvwyz"
  }

IdPProxy -> PeerConnection:
  {
    "type":"SUCCESS",
    "id":1,
    "message": {
      "idp":{
        "domain": "example.org"
        "protocol": "bogus"
      },
      "assertion":\"{\"identity\":\"bob@example.org\",
                    \"contents\":\"abcdefghijklmnopqrstuvwyz\",
                    \"signature\":\"010203040506\"}"
    }
  }
```

Figure 3: Example assertion request

5.4. Verifying Assertions

In order to verify an assertion, an RP sends a "VERIFY" message to the IdP proxy containing the assertion supplied by the AP in the "message" field.

The IdP proxy verifies the assertion. Depending on the identity protocol, this may require one or more round trips to the IdP. For instance, an OAuth-based protocol will likely require using the IdP as an oracle, whereas with BrowserID the IdP proxy can likely verify the signature on the assertion without contacting the IdP, provided that it has cached the IdP's public key.

Regardless of the mechanism, if verification succeeds, a successful response from the IdP proxy MUST contain a message field consisting of a dictionary/hash with the following fields:

identity The identity of the AP from the IdP's perspective. Details of this are provided in Section 5.4.1

contents The original unmodified string provided by the AP in the original SIGN request.

Figure 4 shows an example transaction. Line breaks are inserted solely for readability.

```
PeerConnection -> IdP Proxy:
  {
    "type":"VERIFY",
    "id":2,
    "message":\"{\"identity\":\"bob@example.org\",
                \"contents\":\"abcdefghijklmnopqrstuvwyz\",
                \"signature\":\"010203040506\"}"
  }

IdP Proxy -> PeerConnection:
  {
   "type":"SUCCESS",
   "id":2,
   "message": {
     "identity" : {
       "name" : "bob@example.org",
       "displayname" : "Bob"
     },
     "contents":"abcdefghijklmnopqrstuvwyz"
   }
  }
```

Figure 4: Example assertion request

5.4.1. Identity Formats

Identities passed from the IdP proxy to the PeerConnection are structured as JSON dictionaries with one mandatory field: "name". This field MUST consist of an RFC822-formatted string representing the user's identity. [[OPEN ISSUE: Would it be better to have a typed field?]] The PeerConnection API MUST check this string as follows:

1. If the RHS of the string is equal to the domain name of the IdP proxy, then the assertion is valid, as the IdP is authoritative for this domain.
2. If the RHS of the string is not equal to the domain name of the IdP proxy, then the PeerConnection object MUST reject the assertion unless (a) the IdP domain is listed as an acceptable

third-party IdP and (b) local policy is configured to trust this IdP domain for the RHS of the identity string.

Sites which have identities that do not fit into the RFC822 style (for instance, Facebook ids are simple numeric values) SHOULD convert them to this form by appending their IdP domain (e.g., 12345@identity.facebook.com), thus ensuring that they are authoritative for the identity.

The IdP proxy MAY also include a "displayname" field which contains a more user-friendly identity assertion. Browsers SHOULD take care in the UI to distinguish the "name" assertion which is verifiable directly from the "displayname" which cannot be verified and thus relies on trust in the IdP. In future, we may define other fields to allow the IdP to provide more information to the browser.

5.4.2. PostMessage Checks

Because the PeerConnect object and the IdP proxy communicate via postMessage(), it is essential to verify that the origin of any message (contained in the event.origin property) and source (contained in the event.source) property are as expected:

- For messages from the PeerConnection object, the IdP proxy MUST verify that the origin is "rtcweb://idp-interface" and that the source matches either window.opener or window.parent. If both are non-falsey, they MUST be equal. If any of these checks fail, the message MUST be rejected. [[OPEN ISSUE: An alternate (more generic) design would be to not check the origin here but rather to include the origin in the assertion and have it checked at the RP. Comments?]]
- For messages from the IdP proxy, the PeerConnection object MUST verify that the origin matches the IdP's origin and that the source matches the window/IFRAME opened for the IdP proxy.

If any of these checks fail, the message MUST be rejected. In general, mismatches SHOULD NOT cause transaction failure, since malicious JS might use bogus messages as a form of DoS attack.

5.4.3. PeerConnection API Extensions

5.4.3.1. Authenticating Party

As discussed in Section 3, the AP's IdP can either be configured directly into the browser or selected from a list known to the calling site. We anticipate that some browsers will allow configuration of IdPs in the browser UI but allow the calling application to provide new candidate IdPs or to direct the selection

of a known one. Thus, one model would be:

- If a IdP is provided by the calling application use that.
- If no IdP is provided, and one is configured, use that.
- If no IdP is provided or configured, do nothing.

Implementations MAY also wish to have configuration settings override the calling application's preferences.

APIs for PeerConnection configuration are as-yet unsettled, but it MUST be possible to specify the following parameters to the PeerConnection.

- The IdP domain.
- The users expected identity (if known) [this allows selection between multiple candidate identities with the same IdP.]

5.4.3.2. Relying Party

Because the browser UI must be responsible for displaying the user's identity, it isn't strictly necessary to have new JS interfaces on the relying party side. However, two new interfaces are RECOMMENDED.

When a message is provided to the PeerConnection API with processSignalingMessage() with an assertion that cannot be verified, there is a need for some sort of error indicating verification failure. [Note: I don't see an interface for any other kind of parse error, so I'm not sure what to imitate here.]

A new attribute should be added to indicate the verification status. For instance:

```
readonly attribute DOMString verifiedIdentity;
```

The attribute value should be a JS dictionary indicating the identity and the domain name of the IdP, such as:

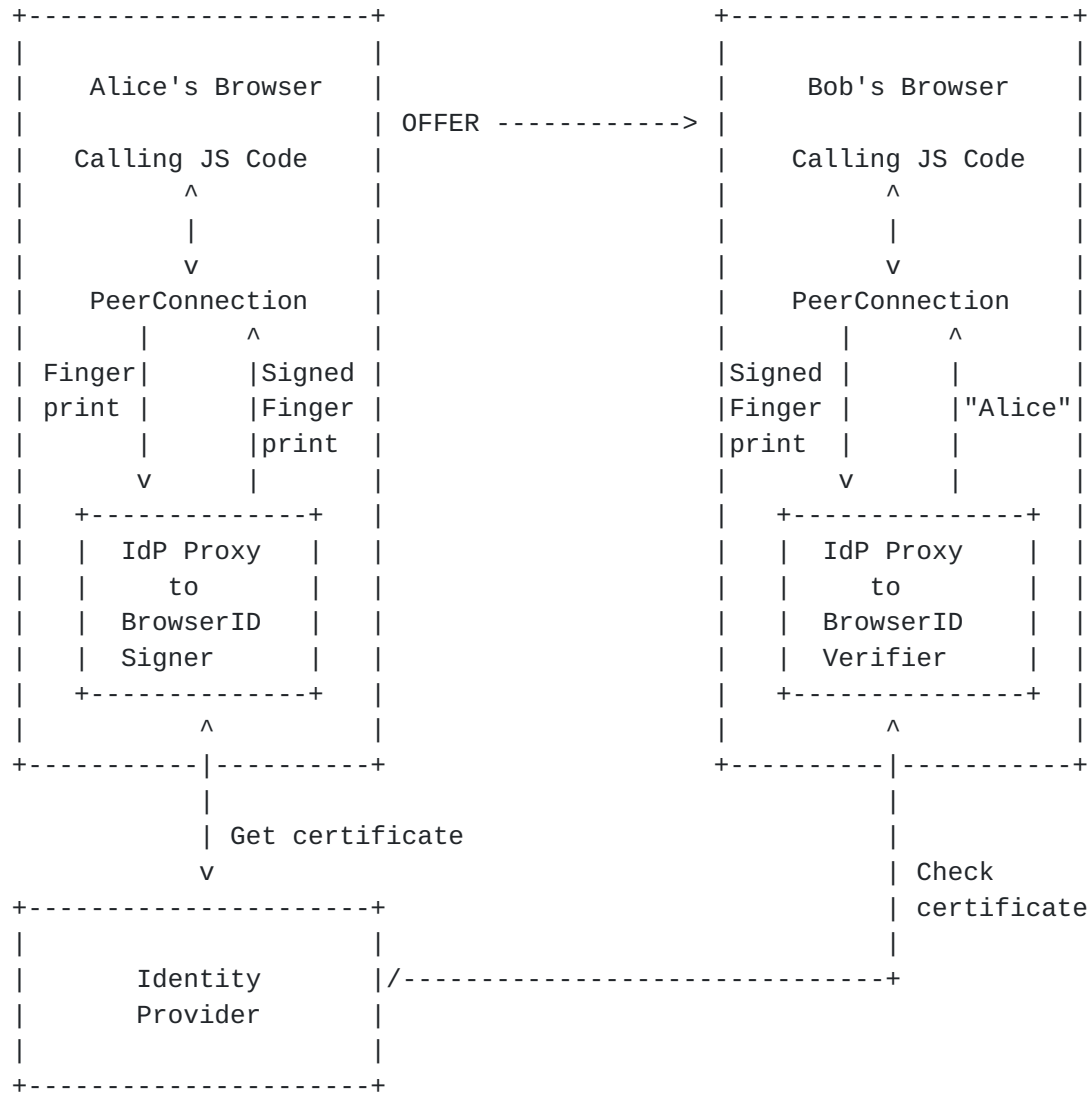
```
{
  "identity" : "ekr@example.org",
  "idp": "example.org"
}
```

5.5. Example Bindings to Specific Protocols

This section provides some examples of how the mechanisms described in this document could be used with existing authentication protocols such as BrowserID or OAuth. Note that this does not require browser-level support for either protocol. Rather, the protocols can be fit into the generic framework. (Though BrowserID in particular works better with some client side support).

5.5.1. BrowserID

BrowserID [https://browserid.org/] is a technology which allows a user with a verified email address to generate an assertion (authenticated by their identity provider) attesting to their identity (phrased as an email address). The way that this is used in practice is that the relying party embeds JS in their site which talks to the BrowserID code (either hosted on a trusted intermediary or embedded in the browser). That code generates the assertion which is passed back to the relying party for verification. The assertion can be verified directly or with a Web service provided by the identity provider. It's relatively easy to extend this functionality to authenticate RTCWEB calls, as shown below.



The way this mechanism works is as follows. On Alice's side, Alice goes to initiate a call.

1. The calling JS instantiates a PeerConnection and tells it that it is interested in having it authenticated via BrowserID (i.e., it provides "browserid.org" as the IdP name.)
2. The PeerConnection instantiates the BrowserID signer in the IdP proxy
3. The BrowserID signer contacts Alice's identity provider, authenticating as Alice (likely via a cookie).
4. The identity provider returns a short-term certificate attesting to Alice's identity and her short-term public key.
5. The Browser-ID code signs the fingerprint and returns the signed assertion + certificate to the PeerConnection.

6. The PeerConnection returns the signed information to the calling JS code.
7. The signed assertion gets sent over the wire to Bob's browser (via the signaling service) as part of the call setup.

Obviously, the format of the signed assertion varies depending on what signaling style the WG ultimately adopts. However, for concreteness, if something like ROAP were adopted, then the entire message might look like:

```
{
  "messageType":"OFFER",
  "callerSessionId":"13456789ABCDEF",
  "seq": 1
  "sdp":"
v=0\n
o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
s= \n
c=IN IP4 192.0.2.1\n
t=2873397496 2873404696\n
m=audio 49170 RTP/AVP 0\n
a=fingerprint: SHA-1 \
4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB\n",
 "identity":{
      "idp":{     // Standardized
         "domain":"browserid.org",
         "method":"default"
      },
      "assertion":   // Contents are browserid-specific
        "\"assertion\": {
          \"digest\":\"<hash of the contents from the browser>\",
          \"audience\": \"[TBD]\"
          \"valid-until\": 1308859352261,
         },
         \"certificate\": {
           \"email\": \"rescorla@example.org\",
           \"public-key\": \"<ekrs-public-key>\",
           \"valid-until\": 1308860561861,
         }" // certificate is signed by example.org
      }
}
```

Note that while the IdP here is specified as "browserid.org", the actual certificate is signed by example.org. This is because BrowserID is a combined authoritative/third-party system in which browserid.org delegates the right to be authoritative (what BrowserID calls primary) to individual domains.

On Bob's side, he receives the signed assertion as part of the call setup message and a similar procedure happens to verify it.

1. The calling JS instantiates a PeerConnection and provides it the relevant signaling information, including the signed assertion.
2. The PeerConnection instantiates the IdP proxy which examines the IdP name and brings up the BrowserID verification code.
3. The BrowserID verifier contacts the identity provider to verify the certificate and then uses the key to verify the signed fingerprint.
4. Alice's verified identity is returned to the PeerConnection (it already has the fingerprint).
5. At this point, Bob's browser can display a trusted UI indication that Alice is on the other end of the call.

When Bob returns his answer, he follows the converse procedure, which provides Alice with a signed assertion of Bob's identity and keying material.

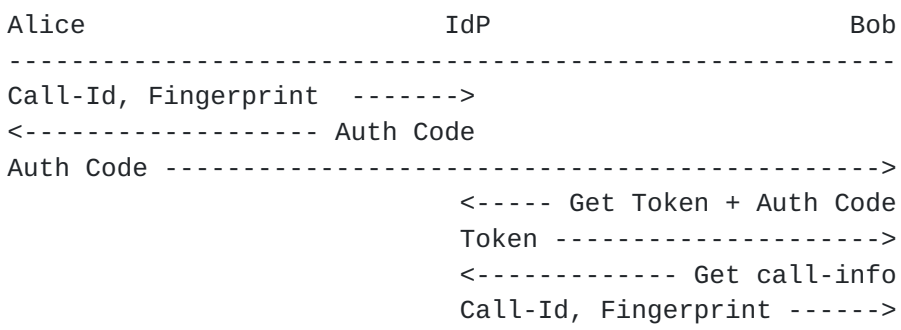
5.5.2. OAuth

While OAuth is not directly designed for user-to-user authentication, with a little lateral thinking it can be made to serve. We use the following mapping of OAuth concepts to RTCWEB concepts:

OAuth	RTCWEB
Client	Relying party
Resource owner	Authenticating party
Authorization server	Identity service
Resource server	Identity service

Table 1

The idea here is that when Alice wants to authenticate to Bob (i.e., for Bob to be aware that she is calling). In order to do this, she allows Bob to see a resource on the identity provider that is bound to the call, her identity, and her public key. Then Bob retrieves the resource from the identity provider, thus verifying the binding between Alice and the call.



This is a modified version of a common OAuth flow, but omits the redirects required to have the client point the resource owner to the IdP, which is acting as both the resource server and the authorization server, since Alice already has a handle to the IdP.

Above, we have referred to "Alice", but really what we mean is the PeerConnection. Specifically, the PeerConnection will instantiate an IFRAME with JS from the IdP and will use that IFRAME to communicate with the IdP, authenticating with Alice's identity (e.g., cookie). Similarly, Bob's PeerConnection instantiates an IFRAME to talk to the IdP.

5.6. Security Considerations

This mechanism relies for its security on the IdP and on the PeerConnection correctly enforcing the security invariants described above. At a high level, the IdP is attesting that the user identified in the assertion wishes to be associated with the assertion. Thus, it must not be possible for arbitrary third parties to get assertions tied to a user or to produce assertions that RPs will accept.

5.6.1. PeerConnection Origin Check

Fundamentally, the IdP proxy is just a piece of HTML and JS loaded by the browser, so nothing stops a Web attacker o from creating their own IFRAME, loading the IdP proxy HTML/JS, and requesting a signature. In order to prevent this attack, we require that all signatures be tied to a specific origin ("rtcweb://...") which cannot be produced by a page tied to a Web attacker. Thus, while an attacker can instantiate the IdP proxy, they cannot send messages from an appropriate origin and so cannot create acceptable assertions. [[OPEN ISSUE: Where is this enforced?]]

5.6.2. IdP Well-known URI

As described in Section 5.2.1 the IdP proxy HTML/JS landing page is located at a well-known URI based on the IdP's domain name. This

requirement prevents an attacker who can write some resources at the IdP (e.g., on one's Facebook wall) from being able to impersonate the IdP.

5.6.3. Security of Third-Party IdPs

As discussed above, each third-party IdP represents a new universal trust point and therefore the number of these IdPs needs to be quite limited. Most IdPs, even those which issue unqualified identities such as Facebook, can be recast as authoritative IdPs (e.g., 123456@facebook.com). However, in such cases, the user interface implications are not entirely desirable. One intermediate approach is to have special (potentially user configurable) UI for large authoritative IdPs, thus allowing the user to instantly grasp that the call is being authenticated by Facebook, Google, etc.

5.7. Web Security Feature Interactions

A number of optional Web security features have the potential to cause issues for this mechanism, as discussed below.

5.7.1. Popup Blocking

If the user is not already logged into the IdP, the IdP proxy may need to pop up a top level window in order to prompt the user for their authentication information (it is bad practice to do this in an IFRAME inside the window because then users have no way to determine the destination for their password). If the user's browser is configured to prevent popups, this may fail (depending on the exact algorithm that the popup blocker uses to suppress popups). It may be necessary to provide a standardized mechanism to allow the IdP proxy to request popping of a login window. Note that care must be taken here to avoid PeerConnection becoming a general escape hatch from popup blocking. One possibility would be to only allow popups when the user has explicitly registered a given IdP as one of theirs (this is only relevant at the AP side in any case). This is what WebIntents does, and the problem would go away if WebIntents is used.

5.7.2. Third Party Cookies

Some browsers allow users to block third party cookies (cookies associated with origins other than the top level page) for privacy reasons. Any IdP which uses cookies to persist logins will be broken by third-party cookie blocking. One option is to accept this as a limitation; another is to have the PeerConnection object disable third-party cookie blocking for the IdP proxy.

6. References

6.1. Normative References

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for RTC-Web", draft-ietf-rtcweb-security-01 (work in progress), October 2011.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "RTCWEB Security Architecture", draft-ietf-rtcweb-security-arch-00 (work in progress), January 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

6.2. Informative References

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

Author's Address

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com