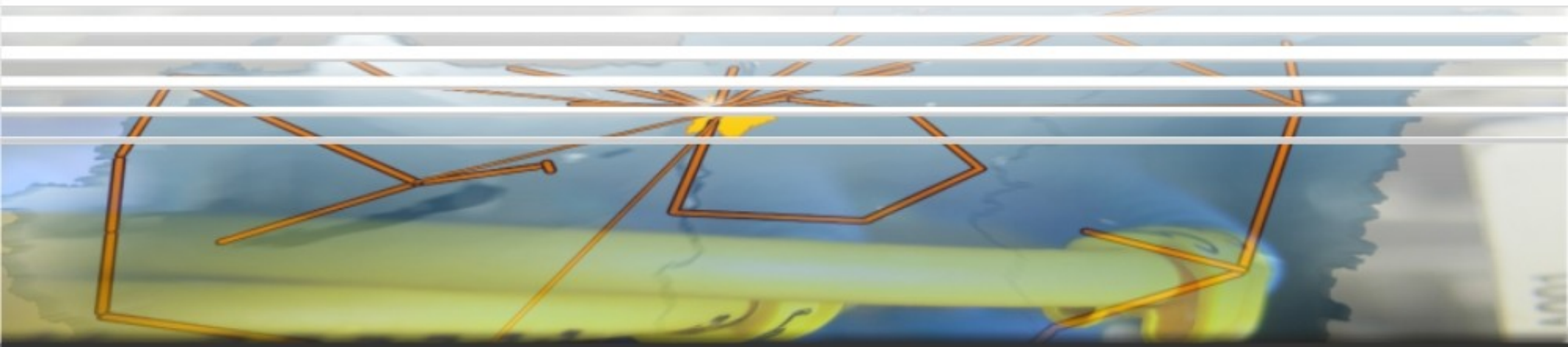


WebRTC Identity in SAML Federations



May 19, 2015
Budapest / Hungary

Mihály Mészáros
NIIF Institute



Agenda

- Education & Research Identity Federations
 - SAML Terminology
 - Overview of SAML auth call flow
- WebRTC Identity model
 - WebRTC(W3C) API, RTCWEB(IETF) security model
 - Terminology
 - Example call flow
 - Trust model
- Demonstrator
- Under the hood details
 - JSON Web Token
 - Trust chain between Federated Auth(SAML) and WebRTC

Why Identity/authentication is important in RTC? (education in focus)

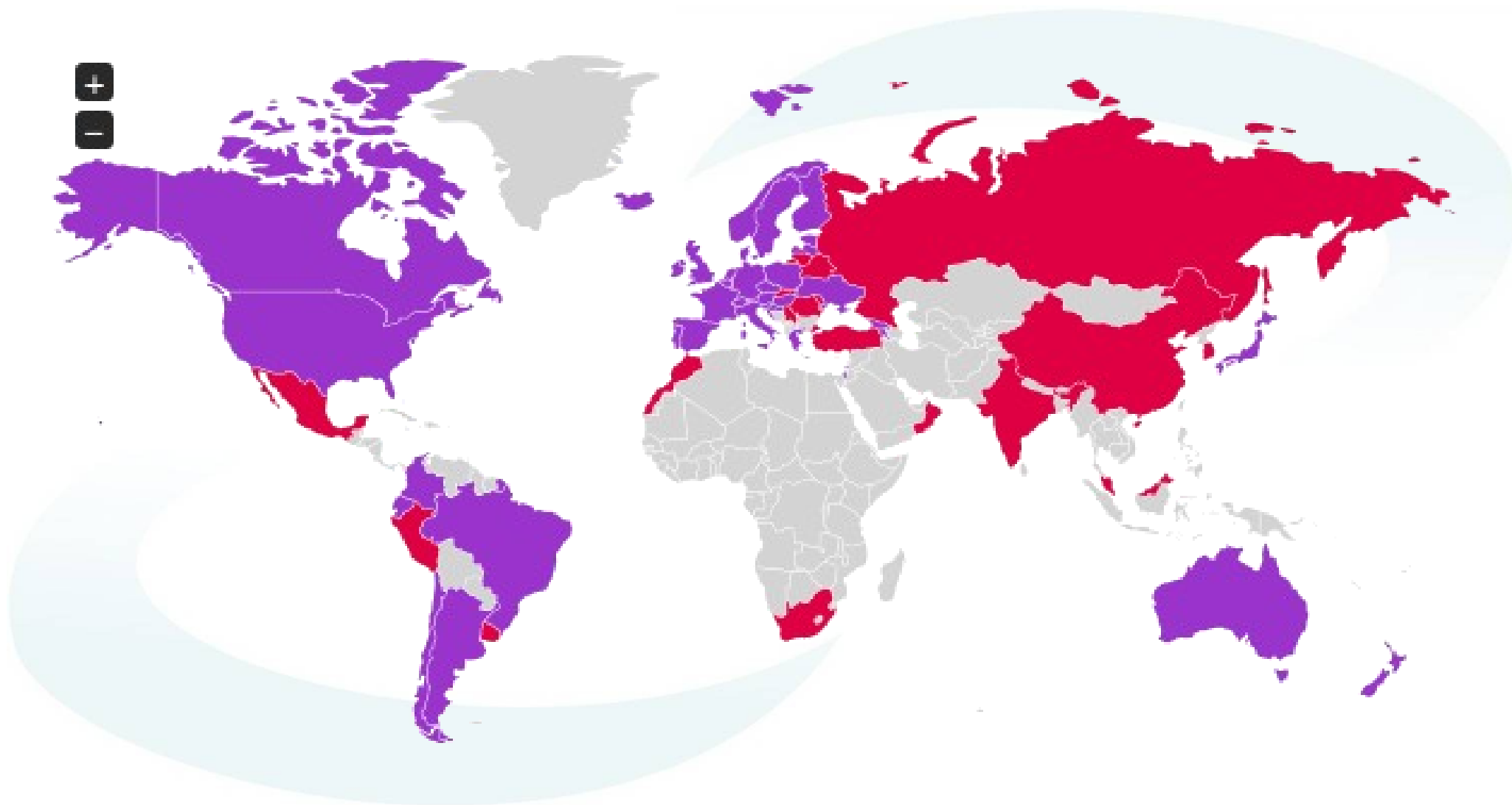
- If we hear or see each-other why it is important?
 - We can identify the other party.
- Some use-cases where Identity could give a plus
 - In a University VoIP / Video Conference network.
 - Connection with SIP RFC4474
 - e-Learning consultations
 - Meeting with an Unknown Professor far away
 - Share research results e.g. VC with a publisher
 - Speak to a Service Provider where trust matters
 - e.g. Bank, etc.
 - Remote interviews for grants/scholarship
 - Any work group/task force meeting where participants have never met before
 - Etc.

EduID/HREF, eduGAIN, REFEDS

- eduID.hu/HREF – Hungarian identity federation
 - Hungarian Research and Educational Federation
 - HREF is a SAML2-based Identity Federation of Hungarian higher education and research institutions, public collections and other content providers
- EduGAIN – The European AAI Federation and beyond
 - Interconnects identity federations around Europe
 - One of the GÉANT services
- REFEDS - World wide
 - Research and Education Federations
 - REFEDS is a community of identity federations from around the world



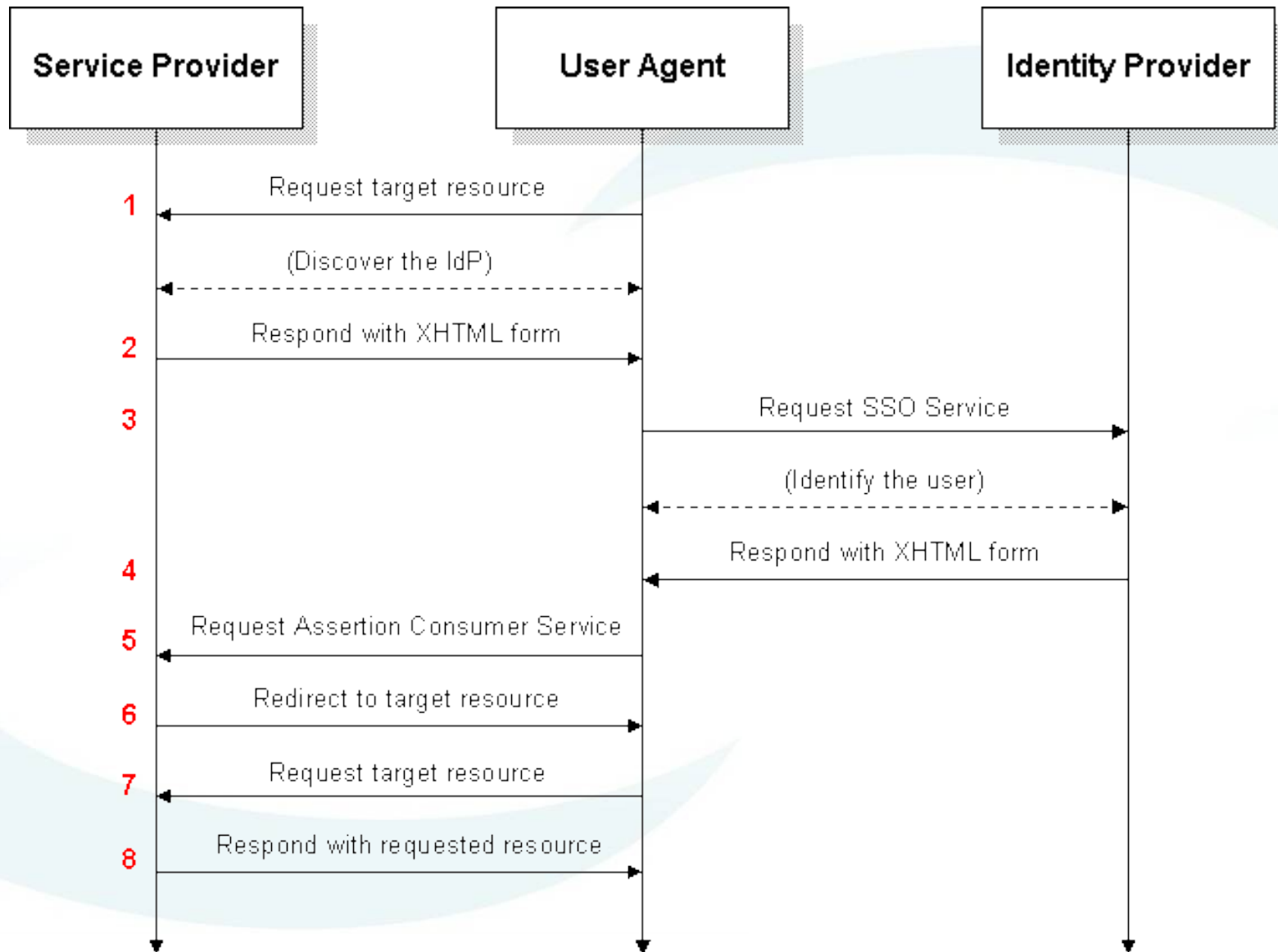
REFEDS



SAML Terminology

- The principal (typically a user)
- IdP: An Identity Provider (IdP), also known as Identity Assertion Provider, is responsible for
 - providing identifiers for users looking to interact with a system, and
 - asserting to such a system that such an identifier presented by a user is known to the provider, and
 - possibly providing other information about the user that is known to the provider.
- SP: The service provider (SP) requests and obtains an identity assertion from the identity provider. On the basis of this assertion, the service provider can make an access control decision – in other words it can decide whether to perform some service for the connected principal.

SAML HTTP POST binding



Typical simple SAML login example

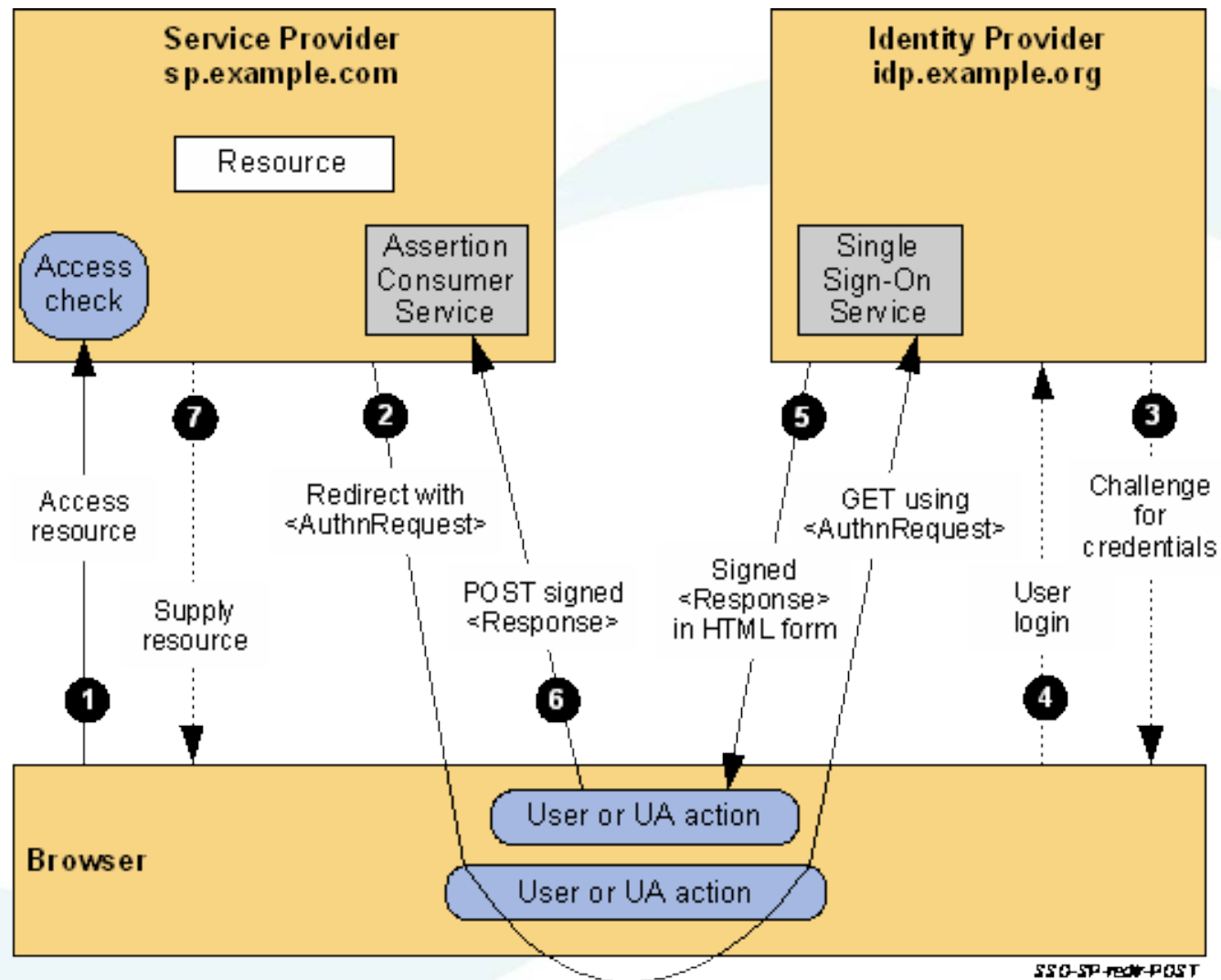


Image source:

http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02_html_m1960c26f.gif

Important barriers for WebRTC integration

- SAML IdP and SP mutually authenticate each other:
 - The IdP Assertion is signed by the IdP.
 - In WebRTC we need a signed identity from the federation, so this isn't a barrier of course. We need this signed assertion.
 - Commonly, the Response object (that encapsulates the Assertion) is signed, but furthermore in order to prevent browser-based attacks the IdP could encrypt the entire SAML response Object (the envelope around the assertion) with the SP public key, so only the SP could access the response and so the assertion.
 - So to fetch identity assertion from IDP directly could be a problem!
- There is no SAML standard way to get signed assertion directly from IdP if we aren't an SP.
- It is not possible to run a Web App code in a browser that act as a SAML SP.

The challenge: WebRTC Identity & SAML

Is it possible?



Goal: Authenticated RTC / VC even on a not fully trusted web site

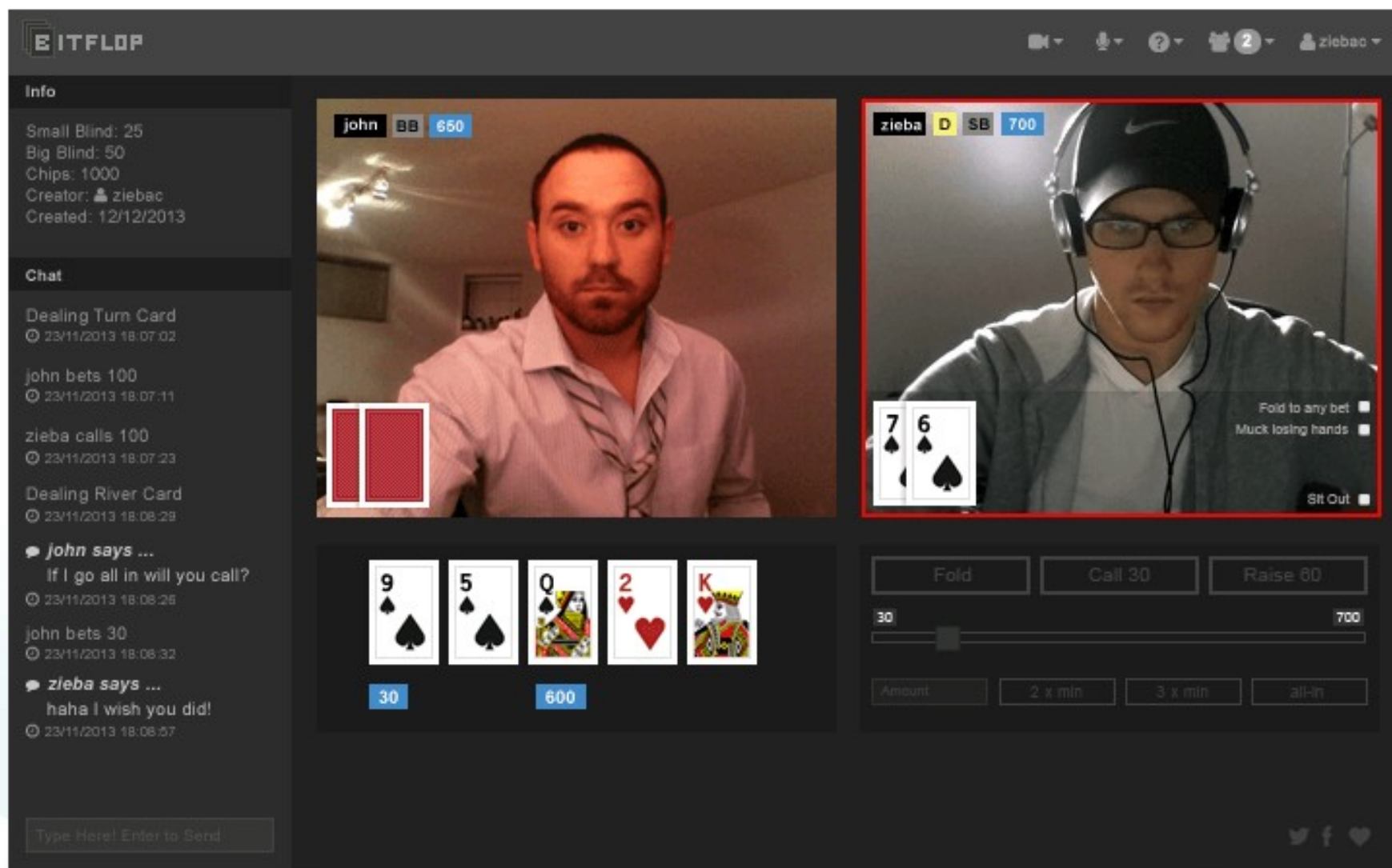


Image Source: <https://bitflop.me/img/bitflop-webrtc-webcam-poker.png>

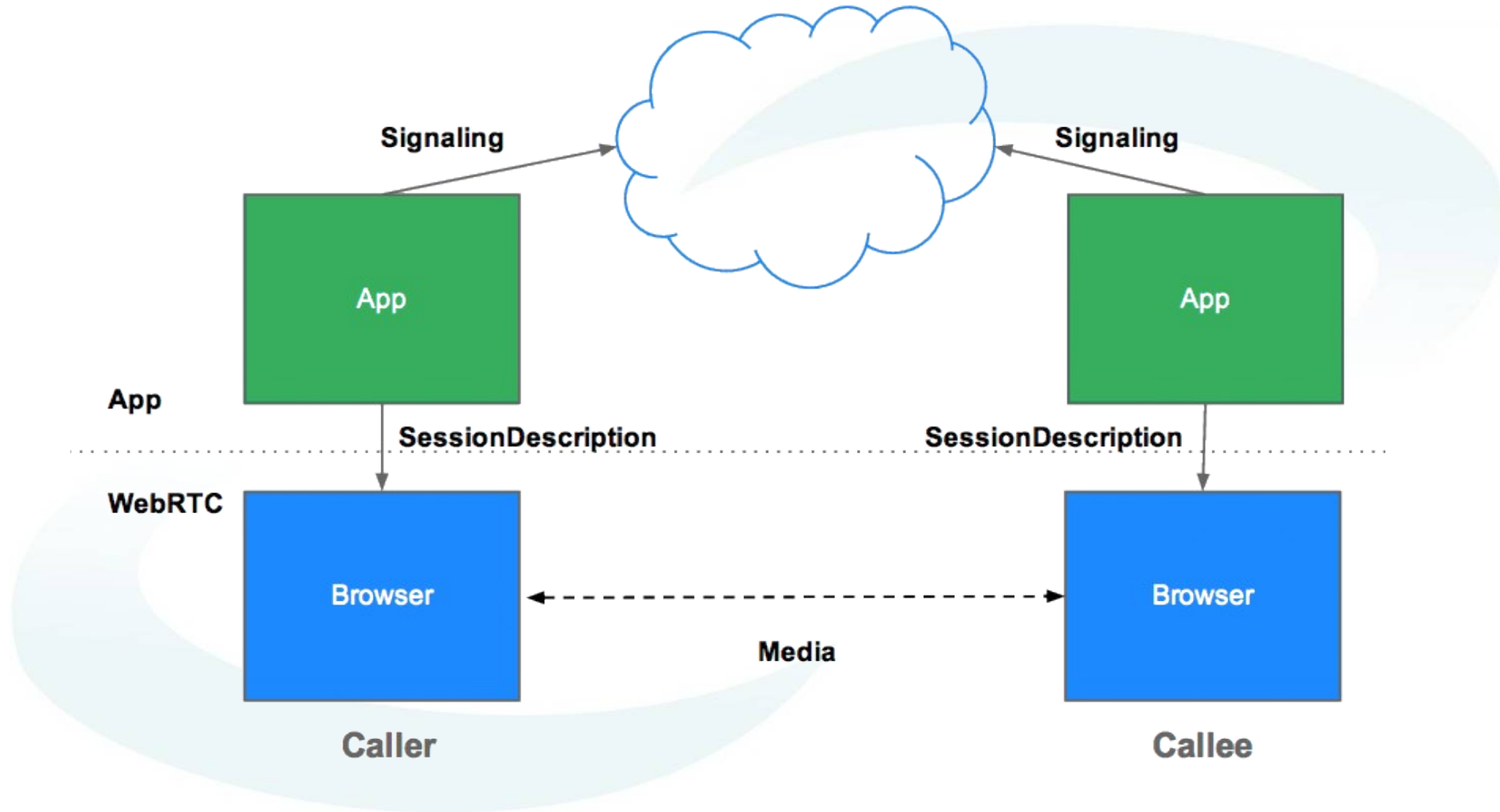
Trust in Browser only

(slide source: EKR rtcweb13.pdf)

- Browser acts as the Trusted Computing Base
 - Only piece of the system user can really trust
 - Job is to enforce user's desired security policies
- Authenticated entities
 - Identity is checked by the browser (sometimes transitively)
- Unauthenticated entities
 - Random other network elements who send and receive traffic
- Authenticated $\neq$ trusted
 - authentication is the basis of trust decisions

JavaScript Session Establishment Protocol (JSEP)

IETF RTCWEB Workgroup



W3C WebRTC Identity API

- API Identity related part
 - PeerConnection functions
 - `setIdentityProvider(provider, protocol)`
 - `getIdentityAssertion()`
 - PeerConnection event handlers
 - `onpeeridentity`
 - `onidentityresult`
 - `onidpassertionerror`
 - `onidpvalidationerror`
 - PeerConnection Attribute
 - `PeerIdentity` (readonly)

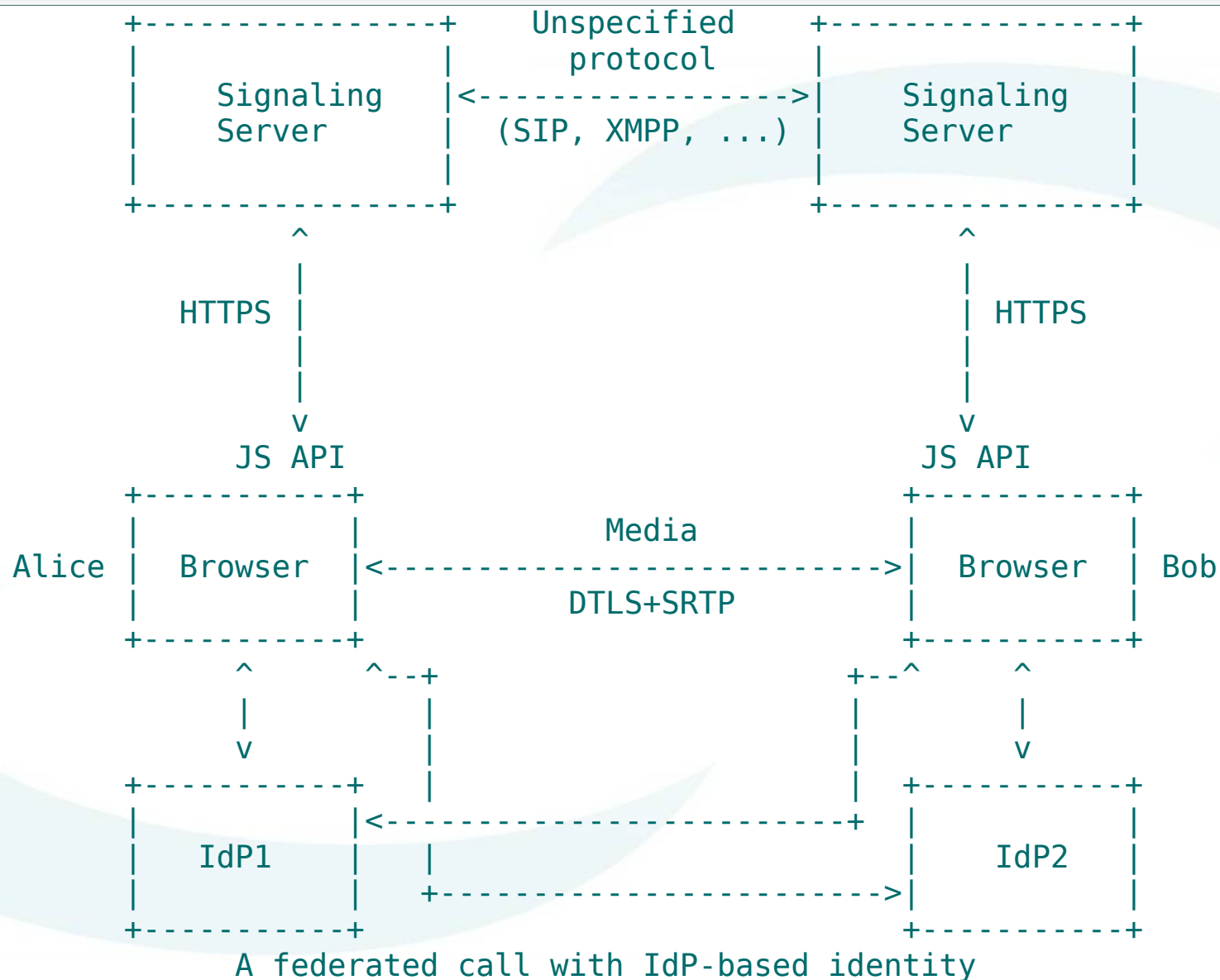
Requesting Identity Assertions (W3C WebRTC API)

- The `RTCPeerConnection` instantiates an IdP proxy as described in Identity Provider Selection section and waits for the IdP to signal that it is ready.
- The `RTCPeerConnection` sends a "SIGN" message to the IdP proxy. This message includes the material the `RTCPeerConnection` desires to be bound to the user's identity.
- If the user has been authenticated by the IdP, and the IdP is willing to generate an identity assertion, the IdP generates an identity assertion. This step depends entirely on the IdP. The methods by which an IdP authenticates users or generates assertions is not specified, though this could involve interacting with the IdP server or other servers.
- The IdP proxy sends a response containing the identity assertion to the `RTCPeerConnection` over the message channel.
- The `RTCPeerConnection` may store the identity assertion for use with future offers or answers.
- If the identity request was triggered by a `createOffer()` or `createAnswer()`, then the assertion is inserted in the offer/answer SDP.

Verifying Identity Assertions (W3C WebRTC API)

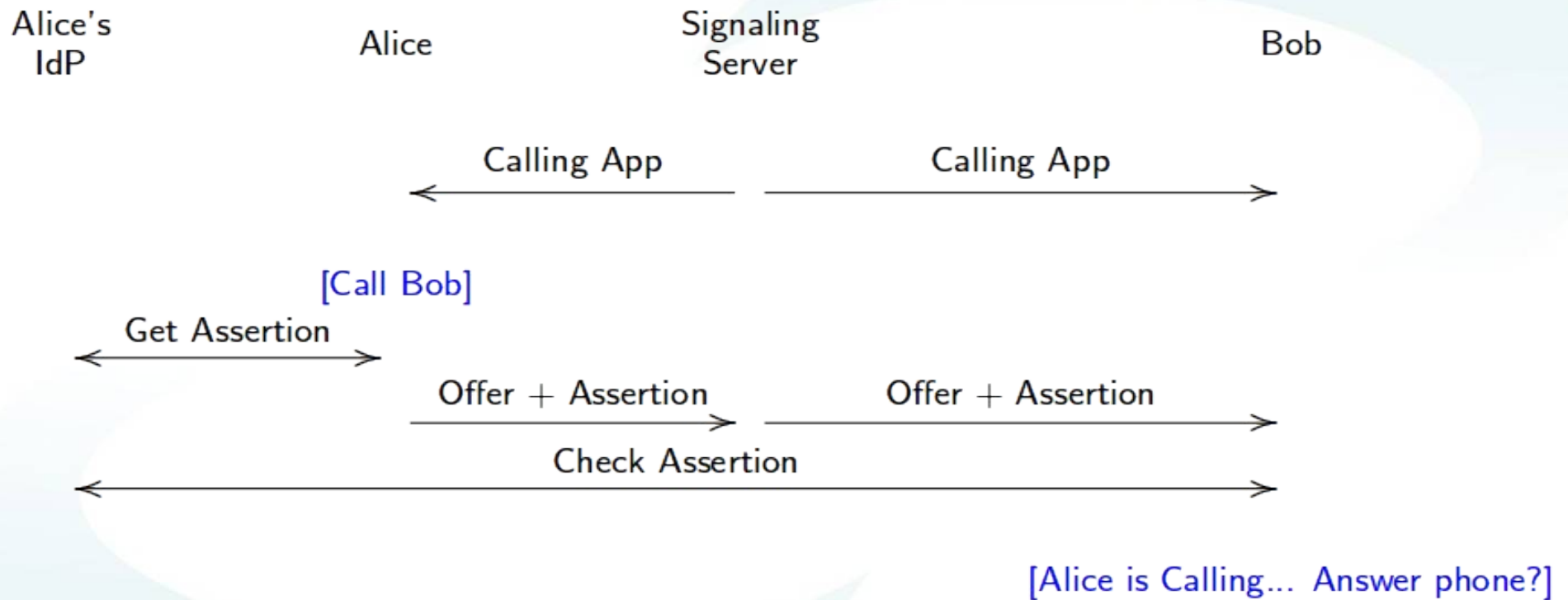
- The `RTCPeerConnection` instantiates an IdP proxy as described in Identity Provider Selection section and waits for the IdP to signal that it is ready.
- The `RTCPeerConnection` sends a "VERIFY" message to the IdP proxy. This message includes the assertion from the offer/answer which is to be verified.
- The IdP proxy verifies the identity assertion (depending on the authentication protocol this could involve interacting with the IDP server).
- Once the assertion is verified, the IdP proxy sends a response containing the verified assertion results to the `RTCPeerConnection` over the message channel.
- The `RTCPeerConnection` validates that the fingerprint provided by the IdP in the validation response matches the certificate fingerprint that is, or will be, used for communications. This is either by:
 - Ensuring that there is only a single `a=fingerprint` value across all media sections in the SDP that matches the fingerprint provided by the IdP.
- Waiting for all DTLS connections to be established and checking that the certificate fingerprints on all connections matches the one provided by the IdP.
- The `RTCPeerConnection` validates that the domain portion of the identity matches the domain of the IdP as described in [RTCWEB-SECURITY-ARCH].
- The `RTCPeerConnection` stores the assertion in the `peerIdentity` attribute and raises a simple event named `peeridentity` at itself. The assertion information to be displayed must contain the domain name of the IdP as provided in the assertion.
- The browser may display identity information to a user in browser UI. Any user identity information that is displayed in this fashion must use a mechanism that cannot be spoofed by content.

RTCWEB Security architecture Overview



Offer (Fingerprint + Assertion)

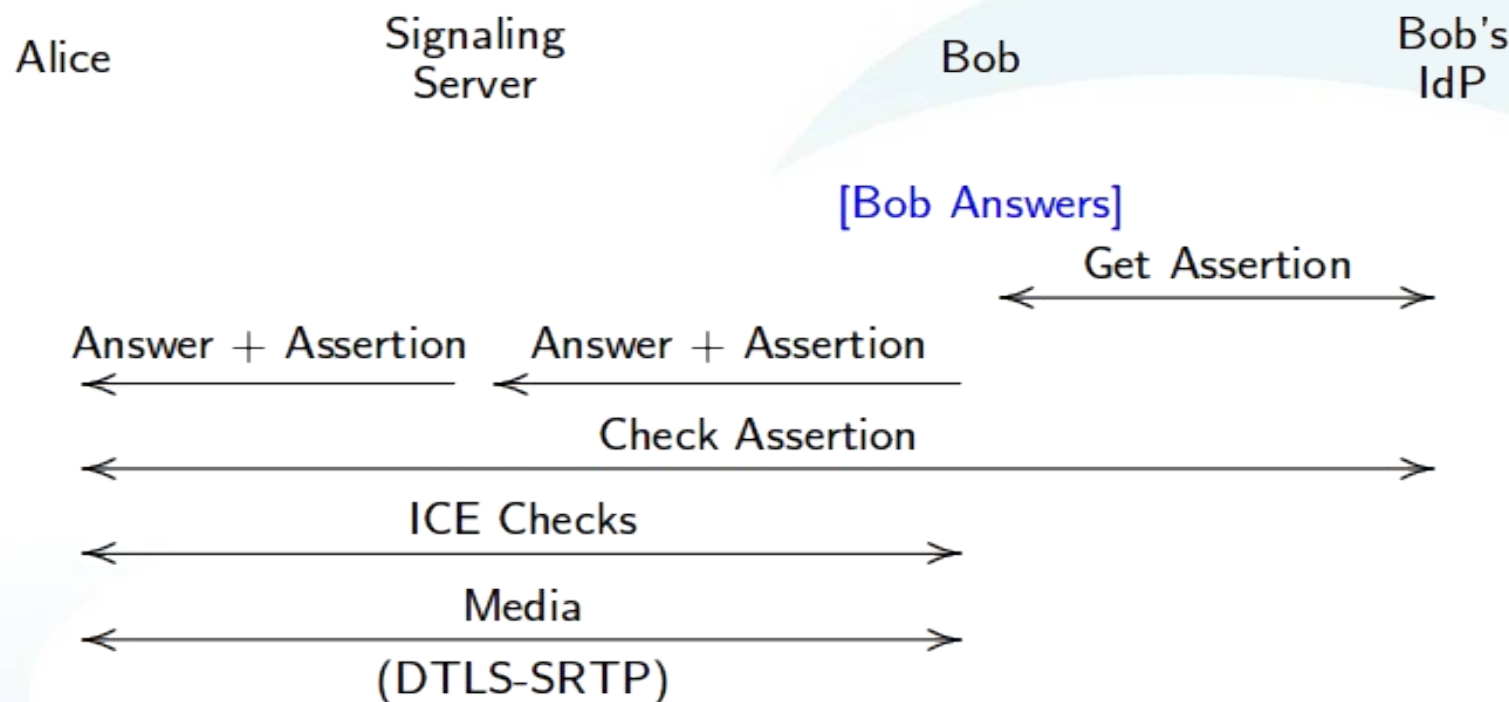
Call Flow (I)



Origin: <http://www.ietf.org/proceedings/82/slides/rtcweb-13.pdf>

Answer (Fingerprint+Assertion)

Call Flow (II)



Origin: <http://www.ietf.org/proceedings/82/slides/rtcweb-13.pdf>

SDP (Session Description Protocol)

```
V=0  
o=mozilla...THIS_IS_SDPARTA-37.0.2 6918706470784164318 0 IN IP4 0.0.0.0  
S=-  
T=0 0  
a=fingerprint:sha-256 93:C0:23:2F:A2:00:00:0D:51:AC:D2:54:65:F4:3B:7D:92:DC:88:33:51:23:40:72:91:83:5B:01:2F:50:78:3F  
a=group:BUNDLE sdparta_0 sdparta_1 sdparta_2  
a=ice-options:trickle  
a=identity:eyJpZHAiOiOnsiZG9tYWluIjoibmlpZi5odSIIsInByb3RvY29sIjoiaWRwLmh0bWwifSwiYXNzZXJ0aW9uIjoiZXlKaGJHY2lPaUpTVXpJMUM5pSXNJblI1Y0NjNklrcFhVeUo5LmV5SmpiMjUwWlc1MGV5STZleUptYVc1bGlpYSndjbWx1ZENjNlczc2lZV3huYjNKcGRHaHRJam9pYzJoExUSTFOaUlzSW1ScFoyVnpkQ0k2SWprek9rTXdPakl6T2pkRU9qa3lPa1JET2pnNE9qTXpPalV4T2pJek9qUXdPamN5T2preE9qZ3pPalZDT2pBeE9qSkdPalV3T2pjNE9qTkdlbjFkZWlnMEdaTDl3d0RBVGRRTWtFWllmdlNVTTJGUmd5R09WSGgzRmpnc2FPZklkRnFsNUx6azBFbnRVOTNQOUlCQ0xzOWtia3V1c0V1S25YRGVNLTNINWFmdTJvZl9CTlZjUnBzMmdBdlNBbVR6Sl1tcEpqMFExtbmV0TmtVT1huZE9HLUIzT3ZGb3QwZVNNENlZSNudhb2wycGduS3FSTktOd3dacEZ1eUZzbFRodHJIIdGNiT19WV3o4QnZpTThKS250dExwd1JxNUhMX2ZLTlRCNzFDYkoyWmh5WXU1UEdwWDhxcXJMWClpbm5YSFY3RnhOttH5OHdrLWd5cnRZazVnbFlZeUFrcTVqZklSXzRzWER5d19Qc1BWTWlaZXRltenVGV3BQTzVFWlJJYR0ZprjFET0o4Q0Q3Z3Zta2dUdlBXSWpkemtBIIn0=  
m=audio 9 RTP/SAVPF 109 9 0 8  
c=IN IP4 0.0.0.0  
a=sendrecv  
a=extmap:1 urn:iETF:params:rtp-hdrext:ssrc-audio-level  
a=ice-pwd:3969306b18d3deed8c120a8bdec02112  
a=ice-ufrag:93f1515f  
a=mid:sdparta_0  
a=rtp-mux  
a=rtpmap:109 opus/48000/2  
a=rtpmap:9 G722/8000/1  
a=rtpmap:0 PCMU/8000  
a=rtpmap:8 PCMA/8000  
a=setup:actpass  
a=ssrc:2076263852 cname:{074bac48-3b26-4f2c-9cf9-06021ca088be}  
m=video 9 RTP/SAVPF 120 126 97  
...
```

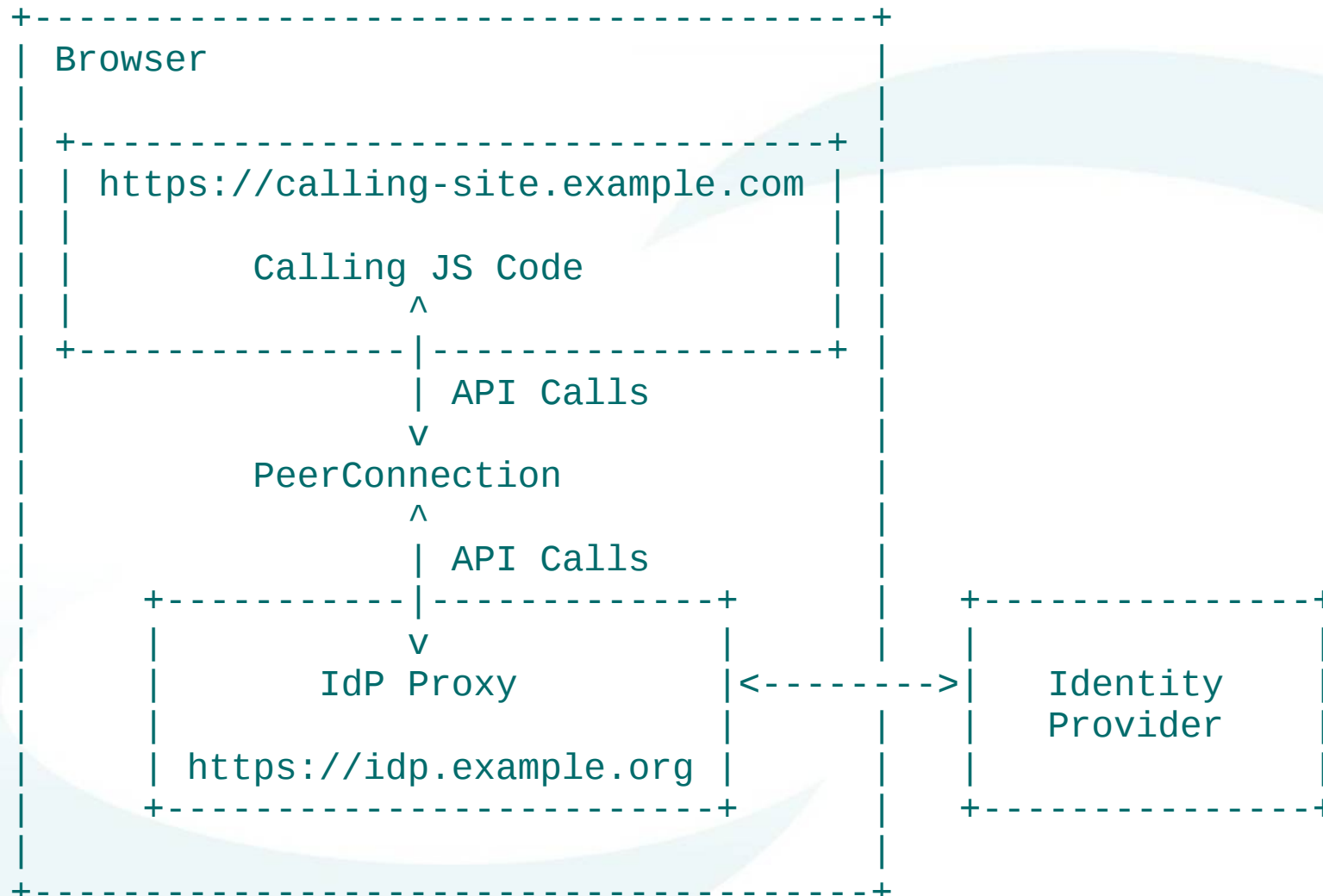
Base64decode (a=identity)

```
a=identity:eyJpZHAiOnsiZG9tYWluIjoibmlpZi50dSIiInByb3RvY29sIjoiaWRwLmh0bWwifSwiYXNzZXJ0aW9uIjoiaXNlKaGJHY2lPaUpTVXpJMU5pSXNjbGl1Y0NjNkIrcFhVeUo5LmV5SmpiMjUwWlc1MGN5STZleUptYVc1blpYSndjbWx1ZENjNlczc2lZV3huYjNKcGRHaHRJam9pYzJoaExUSTF0aUlzSW1ScFoyVnpkQ0k2SWprek9rTXdPakl6T2pKR09rRXlPakF3T2pBd09qQkVPalV4T2tGRE9rUXlPalUwT2pZMU9rWTBPak5DT2pkRU9qa3lPa1JET2pnNE9qTXpPalV4T2pJek9qUXdPamN5T2preE9qZ3pPalZDT2pBeE9qSkdPalV3T2pjNE9qTkdBjFkZlN3aWFXUmxiYjJwZUhraU9pSnRhWE5wUUc1cGFhWVhSFVpZlEuSTVQdGhKNFFDT05TOFVXd2500UUh3MEdaTDl3d0RBVGRRtWtFWllmdlNVTTJ6Umd5R09WSGgzRmpnc2FPZklkRnFsNUx6azBFbndVOTNQOUlCQ0xZWt1a3V1c0V1S25YRGVNLTNINWFmdTJvZl9CTlZjUnB3MmdBdlNBbVR6Sl1tcEpgMFETdmV0TmtVT1huZE9HLUIzT3ZGb3QwZVNENlZSNUdhd2wycGduS3FSTktOd3dacEZ1eUZZbFRodHJIdGNIi19wV3o4QnZpTThKS250dExwd1JxNUhMX2ZLTlRCNzFDYkoyWmh5WXU1UEdwWDhXcXJMWc1ybm5YSFY3RnhoTTh5OHdrLWd5cnRZazVnbFlZeUFrcTVqZklSXzRzWER5d19Qc1BWTW1aZXltenVGV3BQTzVFWlJYR0ZpRjFET0o4Q0Q3Z3Zta2dUdlBXSWpkemtBIn0=
```

Base64decoded identity:

```
{
  "idp": {
    "domain": "niif.hu",
    "protocol": "idp.html"
  },
  "assertion":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUEJ9.eyJjb250ZW50cyI6eyJmaW5nZXJwcmVudCI6W3siYWxnb3JpdGhtIjoic2hhLTl1NiIsImRpZ2VzdCI6IjxzOkMwOjIzOjJGOGkEY0jAw0jAw0jBE0jUxOkFD0kQy0jU00jY10kY00jNC0jdE0jkyOkRD0jg40jMz0jUx0jIz0jQW0jcyc0jKx0jgz0jVC0jAx0jJG0jUw0jc40jNGIn1dfSwiaWRlbmRpdHkiOiJtaXNpQG5paWYuaHUifQ.I5PthJ4QCONS8UwwnN9Hw0GZL9wwDATdkMKEZYfvSUM2zRgyGOVHh3FjgsaOfIdFql5Lzk0EnwU93P9IBCLY9kbkuusEuKnXDeM-3H5afu2of_BNVcRpw2gAvSAmtzJYmpJj0Q-vetNkUOXndOG-B30vFot0eSD6VR5Gaol2pgnKqRNKNwwZpFuyFYlThtrHtcbo_VWz8BviM8JKnNtLVwRq5HL_fKNTB71CbJ2ZhyYu5PGpX8WqrLX-rnnXHV7FxhM8y8wk-gyrtyK5glYYyAkq5jfIR_4sXDyw_PsPVMMZeymzuFWpPO5EZRXGFIF1DOJ8CD7gvmkgTvPWijdzkA"
```

WebRTC App & Sandboxed IdP Proxy & IdP



IdP Proxy

<https://example.org/.well-known/idp-proxy/protocol>

In order to provide security without trusting the calling site, the PeerConnection component of the browser must interact directly with the IdP. The details of the mechanism are described in the W3C API specification, but the general idea is that the PeerConnection component downloads JS from a specific location on the IdP dictated by the IdP domain name. That JS (the "IdP proxy") runs in an isolated security context within the browser and the PeerConnection talks to it via a secure message passing channel.

Note that there are two logically separate functions here:

- Identity assertion generation.
- Identity assertion verification.

IdP Proxy

- This approach is flexible
- Glue between the AAI and WebRTC
 - Allows multiple AAI mechanism
 - In RFC example for
 - BrowserID
 - OAuth
 - RFC mentioning other Identity Provider technologies:
 - Federated Google Login
 - Facebook Connect
 - OpenID
 - WebFinger
 - The beauty of this concept that the two parties could use different IdP technologies to prove their Identity to each other!

WebRTC Identity with BrowserID

(Example from security model)

Example IdP Interaction: BrowserId

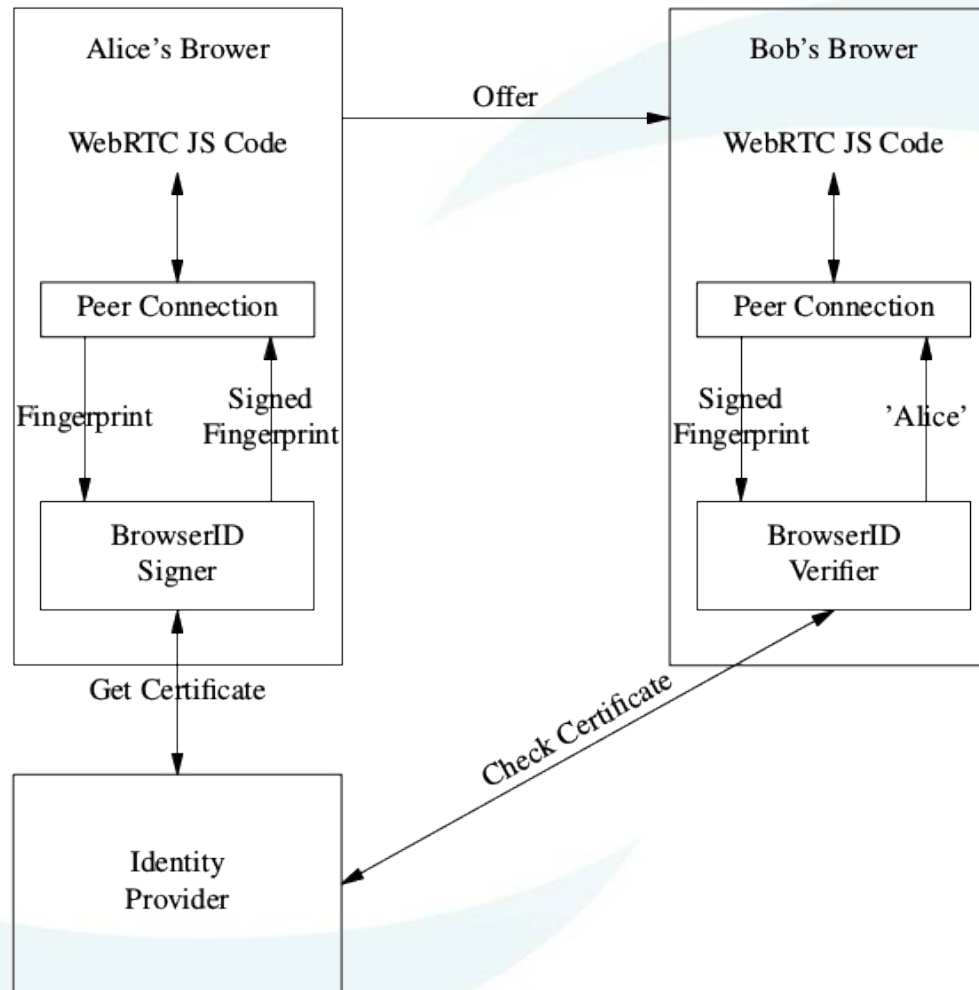


Image Source: <http://www.ietf.org/proceedings/82/slides/rtcweb-13.pdf>

WebRTC Identity with OAuth

(Example from security model)

Mapping of OAuth concepts to WebRTC concepts:

+-----+	+-----+
OAuth	WebRTC
+-----+	+-----+
Client	Relying party
Resource owner	Authenticating party
Authorization server	Identity service
Resource server	Identity service
+-----+	+-----+

Alice	IdP	Bob

Call-Id, Fingerprint	----->	
<-----	Auth Code	
Auth Code	----->	
	<-----	Get Token + Auth Code
	Token	----->
	<-----	Get call-info
	Call-Id, Fingerprint	----->

SAML+ WebRTC Identity Demonstrator

<https://goo.gl/WpXR0W>

<https://ficko.vvc.niif.hu/>

Firefox version:37.0.2



Under the Hood

- WebRTC is a moving target.
 - Identity is in Draft and Eric Rescorla (EKR) author of the webrtc security model works also in Mozilla team on implementation of the WebRTC Identity in Firefox
 - Even on the only WebRTC identity implementation, this feature is not enabled by default!
 - The security model draft is slightly changing version to version. Iterating like a feedback mechanism according the gathered implementation experiences and knowledge.
- Firefox implemented only the WebRTC security model identity part.
 - At least AFAIK
- Enable WebRTC Identity in Firefox:
 - `about:config`
 - Identity enable
 - `media.peerconnection.identity.enabled>true`
 - To debug sandbox application enable the following too:
 - `devtools.debugger.remote-enabled>true`
 - `devtools.chrome.enabled>true`
 - `devtools.inspector.remote>true`

Design concept of the demo IdP proxy code

- Assertion + encryption key fingerprint are glued together in a signed JSON Web Token (JWT) bundle.
- Signature is based on asymmetric cryptography. So with a public-private X.509 key pair.
 - The private key is not stored in IdP Proxy code!
 - Signing server side not in the browser.
- To connect the two worlds (SAML & WebRTC)
 - The (identity domain part) SP's private key is used to sign the JWT.
 - Federation metadata could be used to validate the signature.
- User Authentication is required to access the JWT Signer Web Application service. That has only one input the RTC communication encryption key fingerprint.
 - Signing code could be fully integrated in SP as module in case Semplesamlphp.
 - Or the same service could be an app on top of Shibboleth

Terminology, Identity & Proxy URL examples

- According WebRTC security model
 - The Identity Provider (IdP):
 - JWT Signer App on top of SAML SP
 - IdP Proxy:
 - HTML and JS Glue code in a sandbox (From FF38 only JS)
- My own terminology
 - JSON Web Token Signer Service:
 - PHP code cryptographically binding the encryption key fingerprint to the identity
- Examples
 - Identity:
 - misi@niif.hu
 - IdP Proxy location URL:
 - <https://niif.hu/.well-known/idp-proxy/default>

Trust chain between Federated Auth(SAML) and WebRTC,

- Sign

- SAML IdP → SAML SP
 - mutual cert based authN
 - Signed assertion verification
- SAML SP → JWT signer service
 - On server the application receives the identity attribute. e.g. EPPN
 - Environment for Shibboleth or PHP var for SimpleSamlPhp (SSP)
 - Receives in HTTP from sandbox the RTC encryption key fingerprint.
 - Signs with the SAML SP private key

- Verify

- Get the federation Metadata
- Extracts the identity domain part SP public key
- Verify the signature of the JSON Web Token (JWS) based on it

SDP

[illegible]

Base64decode (a=identity)

a=identity:eyJpZHAiOnsiZG9tYWluIjoibmVlZi50dSIiInByb3RvY29sIjoiaWRwLmhf0bWwifSwiYXNzZXJ0aW9uIjoiaXNlKaGJHY2lPaUpTVXpJMU5pSXNjb1I1Y0NjNklrcFhVeUo5LmV5SmpiMjUwWlc1MGN5STZleUptYVc1blpYSndjbWx1ZENjNlczcz2lZV3huYjNKcGRHaHRJam9pYzJoaExUSTF0aUlzSW1ScFoyVnphQ0k2SWprek9rTXdPakl6T2pKR09rRXlPakF3T2pBd09qQkVPalV4T2tGRE9rUXlPalUwT2pZMU9rWTBPak5DT2pkRU9qa3lPa1JET2pnNE9qTXpPalV4T2pJek9qUXdPamN5T2preE9qZ3pPalZDT2pBeE9qSkdPalV3T2pjNE9qTkdJbjFkZlN3aWFXUmxiYjJwZUhraU9pSnRhWE5wUUc1cGFhWVhSFVpZlEuSTVQdGhKNFFDT05TOFVXd2500UUh3MEdaTDl3d0RBVGRRtWtFWllmdlNVTTJ6Umd5R09WSGgzRmpnc2FPZklkRnFsNUx6azBFbndVOTNQOUlCQ0xZWt1a3V1c0V1S25YRGVNLTNINWFmdTJvZl9CTlZjUnB3MmdBd1NBbVR6S1ltcEppMFETdmV0TmtVT1huZE9HLUIzT3ZGb3QwZVNENlZSNUdhd2wycGduS3FSTktOd3dacEZ1eUZZbFRodHJIdGNiT19wV3o4QnZpTThKS250dExWd1JxNUhMX2ZLTlRCNzFDYkoyWmh5WXU1UEdwWDhXcXJMWc1ybm5YSFY3RnhoTTh5OHdrLWd5cnRZazVnbFlZeUFrcTVqZk1SXzRzWER5d19Qc1BWTW1aZXltenVGV3BQTzVFWlJYR0ZpRjFET0o4Q0Q3Z3Zta2dUdlBXSWpkemtBIn0=

Base64decoded identity:

```
{
  "idp": {
    "domain": "niif.hu",
    "protocol": "idp.html"
  },
  "assertion":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUEJ9.eyJjb250ZW50cyI6eyJmaW5nZXJwcmVudCI6W3siYWxnb3JpdGhtIjoic2hhLTI1NiIsImRpZ2VzdCI6IjxzOkMwOjIzOjJGOGkEY0jAw0jAw0jBE0jUxOkFD0kQy0jU00jY10kY00jNC0jdE0jkyOkRD0jg40jMz0jUx0jIz0jQW0jcyc0jKx0jgz0jVC0jAx0jJG0jUw0jc40jNGIn1dfSwiaWRlbnRpdHkiOiJtaXNpQG5paWYuaHUifQ.I5PthJ4QCONS8UwwnN9Hw0GZL9wwDATdkMKEZYfvSUM2zRgyGOVHh3FjgsaOfIdFql5Lzk0EnwU93P9IBCLY9kbkuusEuKnXDeM-3H5afu2of_BNVcRpw2gAvSAmtzJYmpJj0Q-vetNkUOXndOG-B30vFot0eSD6VR5Gaol2pgnKqRNKNwwZpFuyFYlThtrHtcbo_VWz8BviM8JKnNtLVwRq5HL_fKNTB71CbJ2ZhyYu5PGpX8WqrLX-rnnXHv7FxmM8y8wk-gyrtyK5glYYyAkq5jfIR_4sXDyw_PsPVMMZeymzuFWpPO5EZRXGFIF1DOJ8CD7gvmkgTvPWijdzkA"
```

JSON Web Token (JWT) decoded

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXUEIyJmJb250Z  
W50cyI6eyJmaW5nZXJwcmVudCI6W3siYWxnb3Jpd  
Ghtljoic2hhLTl1NiIsImRpZ2VzdCI6IjZkOkMwOjIzOj  
JGOkEyOjAwOjAwOjBEOjUxOkFDOKQyOjU0OjY1  
OkY0OjNCOjdBEOjkyOkRDOjg4OjMzOjUxOjIzOjQw  
OjcyOjkyOjg4OjVCOjAxOjJGOjUwOjc4OjNGIn1dfS  
wiaWRIbnRpdHkiOiJtaXNpQG5paWYuaHUifQ.I5Pth  
J4QCONS8UWwnN9Hw0GZL9wwDATdkMkEZYfv  
SUM2zRgyGOVHh3FjgsaOfldFql5Lzk0EnwU93P9IB  
CLY9kbkuusEuKnXDeM-  
3H5afu2of_BNVcRpw2gAvSAmTzJYmpJj0Q-  
vetNkUOXndOG-  
B3OvFot0eSD6VR5Gaol2pgnKqRNKNwwZpFuyFYI  
ThtrHtcbo_VWz8BviM8JKNtLVwRq5HL_fKNTB7  
1CbJ2ZhyYu5PGpX8WqrLX-rnnXHV7Fxm8y8wk-  
gyrtYk5glYYyAkq5jflR_4sXDyw_PsPVMmZeymzuF  
WpPO5EZRXXGFIF1DOJ8CD7gvmkgTvPWljdzkA
```

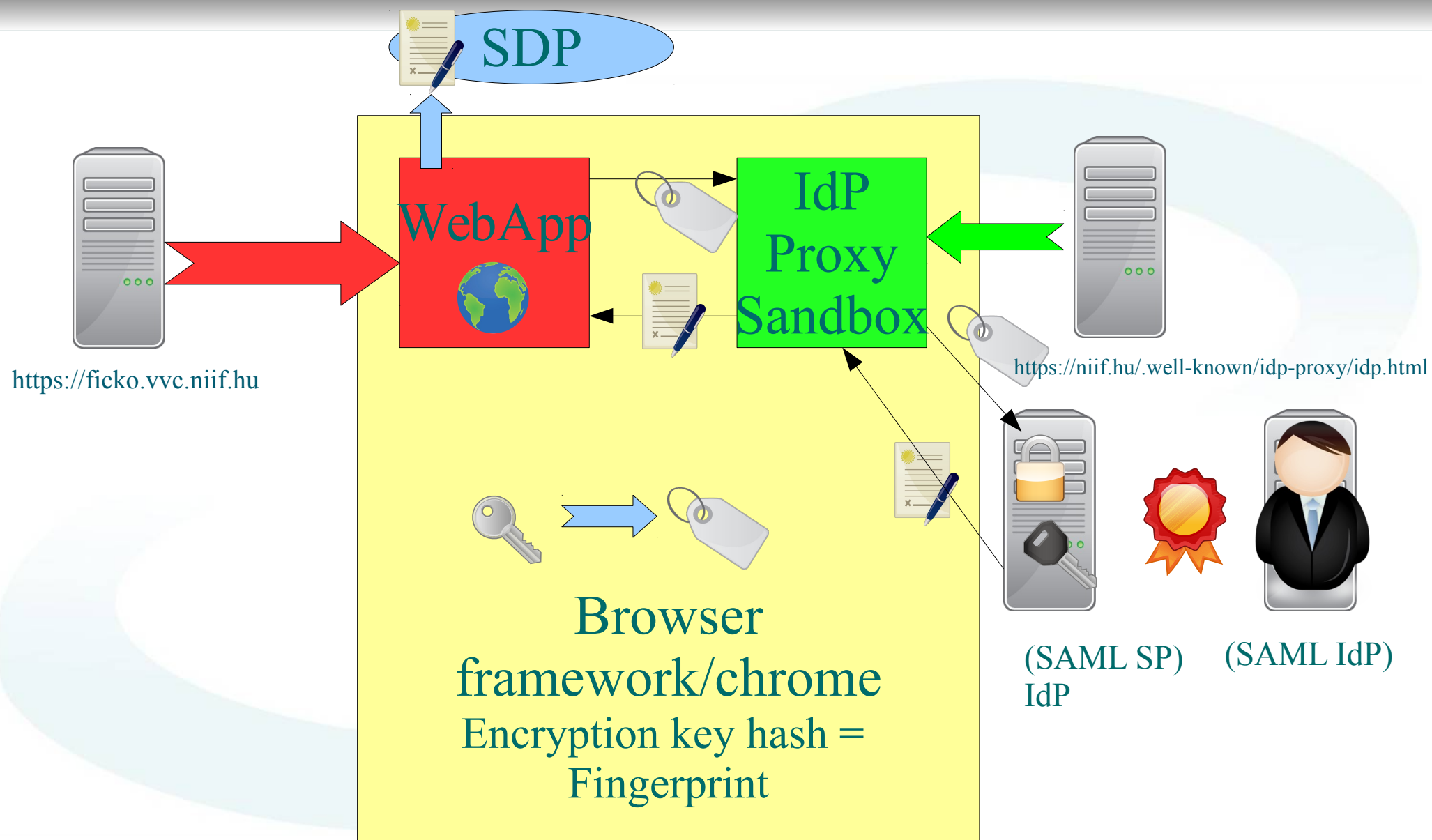
```
{  
  "alg": "RS256",  
  "typ": "JWT"  
}
```

```
{  
  "contents": {  
    "fingerprint": [  
      {  
        "algorithm": "sha-256",  
        "digest":  
          "93:C0:23:2F:A2:00:00:0D:51:AC:D2:54:65:F4:3B:7D:92:DC:88:33  
          :51:23:40:72:91:83:5B:01:2F:50:78:3F"  
      }  
    ]  
  },  
  "identity": "misi@niif.hu"  
}
```

```
RSASHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),
```

```
157LM7FfPXkXMf6h18VA/RJqs049apX/x0GC4iJbaSl+Bf70uod0  
Ualsk2IyNzkA  
jc0VaaycjZIGTlm2MrQ4bzsnyBk3OwEh2l8ZaNnutIk3  
-----END CERTIFICATE-----
```

Bind/Sign (Identity assertion/certificate + Encryption key fingerprint/hash)



Load IdP proxy code

The screenshot shows a web browser's developer tools with the Network tab selected. The left pane lists various requests, including GET requests for files like /, idp.html, favicon.ico, and core.js, and POST requests for jws.php and DS?target=... The right pane shows the details for a selected GET request to niif.hu. The request URL is https://niif.hu/.well-known/idp-proxy/idp.html, and the status code is 200 OK. The response headers are expanded, showing fields like Accept-Ranges, Cache-Control, Connection, Content-Encoding, Content-Length, Content-Type, Date, Etag, Expires, Keep-Alive, Last-Modified, Server, and Vary. The request headers are also expanded, showing fields like Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Connection, Pragma, and Cache-Control.

Method	File	Source
302 GET	/	ficko.vvc.ni
200 GET	/	ficko.vvc.ni
200 GET	idp.html	niif.hu
404 GET	favicon.ico	ficko.vvc.ni
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu

Request URL: https://niif.hu/.well-known/idp-proxy/idp.html
Request method: GET
Status code: 200 OK

Response headers (0.395 KB)

- Accept-Ranges: "bytes"
- Cache-Control: "max-age=1209600"
- Connection: "Keep-Alive"
- Content-Encoding: "gzip"
- Content-Length: "391"
- Content-Type: "text/html"
- Date: "Sun, 10 May 2015 07:16:18 GMT"
- Etag: ""9000973e-981-50c88c30c44c0""
- Expires: "Sun, 24 May 2015 07:16:18 GMT"
- Keep-Alive: "timeout=5, max=100"
- Last-Modified: "Tue, 13 Jan 2015 13:40:11 GMT"
- Server: "Apache/2.2.22"
- Vary: "Accept-Encoding"

Request headers (0.350 KB)

- Host: "niif.hu"
- User-Agent: "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0"
- Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language: "en-US,en;q=0.5"
- Accept-Encoding: "gzip, deflate"
- Connection: "keep-alive"
- Pragma: "no-cache"
- Cache-Control: "no-cache"

IdP proxy received SIGN request

new XMLHttpRequest to JWT signer service

The screenshot shows the Network tab of a web browser's developer tools. The left pane lists various requests, with the last one, a 302 POST to `jws.php`, selected. The right pane displays the details for this request.

Request Details:

- Request URL:** `https://niif.hu/.well-known/sp/jws.php`
- Request method:** POST
- Status code:** 302 Moved Temporarily

Response headers (0.598 KB):

- Cache-Control:** "no-store, no-cache, must-revalidate, post-check=0, pre-check=0"
- Connection:** "Keep-Alive"
- Content-Encoding:** "gzip"
- Content-Length:** "20"
- Content-Type:** "text/html"
- Date:** "Sun, 10 May 2015 07:16:19 GMT"
- Expires:** "Thu, 19 Nov 1981 08:52:00 GMT"
- Keep-Alive:** "timeout=5, max=96"
- Location:** "https://niif.hu/Shibboleth.sso/DS?target=https://niif.hu/.well-known/sp/jws.php"
- Pragma:** "no-cache"
- Server:** "Apache/2.2.22"
- Set-Cookie:** "PHPSESSID=tfj2l88d897me52cabtcbfu612; path=/"
- Vary:** "Accept-Encoding"
- X-Powered-By:** "PHP/5.4.39-0+deb7u2"
- access-control-allow-origin:** "https://idp.niif.hu"

Request headers (0.518 KB):

- Host:** "niif.hu"
- User-Agent:** "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0"
- Accept:** "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language:** "en-US,en;q=0.5"
- Accept-Encoding:** "gzip, deflate"
- Referer:** "https://niif.hu/.well-known/idp-proxy/idp.html"
- Content-Length:** "321"
- Content-Type:** "multipart/form-data; boundary=-----1135772284134107357628104395"
- Connection:** "keep-alive"
- Pragma:** "no-cache"
- Cache-Control:** "no-cache"

Send the fingerprint to sign

The screenshot shows the Network tab of a web browser's developer tools. The left pane lists various requests, with the last one, a POST to `jws.php` on `niif.hu`, selected. The right pane shows the request payload for this POST request.

Method	File	Source
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu
200 GET	jws.php	niif.hu
200 GET	idp.html	niif.hu
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g

Request payload:

```
1 -----1135772284134107357628104395
2 Content-Disposition: form-data; name="contents"
3
4 {"fingerprint":[{"algorithm":"sha-256","digest":"3D:15:E7:EA:0E:3C:5B:86:27:81:28:10:28:C5:E4:D8:D6:60:E6:E
5 -----1135772284134107357628104395--
6
```

niif.hu DS redirect to NIIF IdP SSO

The screenshot displays the Network tab of a web browser's developer tools. The left pane shows a list of network requests. The right pane shows the details for the selected request, which is a 302 redirect.

Method	File	Source
302 GET	/	ficko.vvc.ni
200 GET	/	ficko.vvc.ni
200 GET	idp.html	niif.hu
404 GET	favicon.ico	ficko.vvc.ni
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu

Request URL: https://niif.hu/Shibboleth.sso/DS?target=https://niif.hu/.well-known/sp/jws.php
Request method: GET
Status code: 302 Found

Response headers (0.938 KB)

- Cache-Control: "private,no-store,no-cache,max-age=0"
- Connection: "Keep-Alive"
- Content-Encoding: "gzip"
- Content-Length: "648"
- Content-Type: "text/html; charset=iso-8859-1"
- Date: "Sun, 10 May 2015 07:16:19 GMT"
- Expires: "Wed, 01 Jan 1997 12:00:00 GMT"
- Keep-Alive: "timeout=5, max=95"
- Location: "https://idp.niif.hu/simplesaml/saml2/idp/SSOSe...0099e100f919b8127f08872ec204b3493fe9825db0e85"
- Server: "Apache/2.2.22"
- Vary: "Accept-Encoding"

Request headers (0.482 KB)

- Host: "niif.hu"
- User-Agent: "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0"
- Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language: "en-US,en;q=0.5"
- Accept-Encoding: "gzip, deflate"
- Referer: "https://niif.hu/.well-known/idp-proxy/idp.html"
- Cookie: "PHPSESSID=tfj2l88d897me52cabtcbfu612"
- Connection: "keep-alive"
- Pragma: "no-cache"
- Cache-Control: "no-cache"

IdP SSO, CORS

The screenshot shows a web browser's Network tab with a list of requests on the left and the details of a selected request on the right.

Request List (Left):

Method	File	Domain
302 GET	/	ficko.vvc.ni
200 GET	/	ficko.vvc.ni
200 GET	idp.html	niif.hu
404 GET	favicon.ico	ficko.vvc.ni
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu

Request Details (Right):

Request URL: https://idp.niif.hu/simplesaml/saml2/idp/SSOService.php?SAMLRequest=fZJRb4IwFIX/Cum7tBARbYSE6cNM3C...

Request method: GET

Status code: 200 OK

Response headers (0.484 KB):

- Access-Control-Allow-Credentials: "true"
- Cache-Control: "no-store, no-cache, must-revalidate, post-check=0, pre-check=0"
- Connection: "Keep-Alive"
- Content-Encoding: "gzip"
- Content-Length: "6190"
- Content-Type: "text/html"
- Date: "Sun, 10 May 2015 07:16:19 GMT"
- Expires: "Thu, 19 Nov 1981 08:52:00 GMT"
- Keep-Alive: "timeout=5, max=100"
- Pragma: "no-cache"
- Server: "Apache/2.2.22 (Debian)"
- Vary: "Accept-Encoding"
- X-Powered-By: "PHP/5.4.39-0+deb7u2"
- access-control-allow-origin: "https://niif.hu"

Request headers (1.092 KB):

- Host: "idp.niif.hu"
- User-Agent: "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0"
- Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language: "en-US,en;q=0.5"
- Accept-Encoding: "gzip, deflate"
- Referer: "https://niif.hu/.well-known/idp-proxy/idp.html"
- Origin: "https://niif.hu"
- Cookie: "PHPSESSID=e65213b0391d9d3cc72483327ebd3bc0; ...=_3d8199112abdc9f9cd7c316af485705b094390fec9"
- Connection: "keep-alive"
- Pragma: "no-cache"
- Cache-Control: "no-cache"

SAML SSO request parameters

The screenshot shows a web browser's developer network tool. The left pane lists network requests, and the right pane shows the details of the selected request.

Method	File	Domain
302 GET	/	ficko.vvc.ni
200 GET	/	ficko.vvc.ni
200 GET	idp.html	niif.hu
404 GET	favicon.ico	ficko.vvc.ni
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu

The right pane shows the details of the selected request (SSOService.php?SAMLRequest=fZJ...). The 'Query string' tab is active, showing the following parameters:

- SAMLRequest:** "fZJRb4lwFIX/Cum7tBARbYSE6cNM3CTC9rCXpU...0++m3WqZai/HESKXW/MpxZHiEP4XiU/P8A8S8="
- RelayState:** "ss:mem:dff9a70f14f3c3df1650099e100f919b8127f08872ec204b3493fe9825db0e85"

IdP SSO SAMLResponse

The screenshot displays a web browser's developer tools interface, specifically the Network tab. The left pane shows a list of network requests, with the 200 GET request to `SSOService.php?SAMLRequest=fzj...` selected. The right pane shows the response of this request, which is an HTML document. The response content is as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>POST data</title>
</head>
<body onload="document.getElementsByTagName('input')[0].click();">
  <noscript>
    <p><strong>Note:</strong> Since your browser does not support JavaScript, you must press the button below</p>
  </noscript>
  <form method="post" action="https://niif.hu/Shibboleth.sso/SAML2/POST">
    <!-- Need to add this element and call click method, because calling submit()
    on the form causes failed submission if the form has another element with name or id of submit.
    See: https://developer.mozilla.org/en/DOM/form.submit#Specification -->
    <input type="submit" style="display:none;" />
    <input type="hidden" name="SAMLResponse" value="PHNhbWxwO1Jlc3BvbmlhHtbG5zOnNhbWxwPSJ1cm46b2FzaXM6bmFtZXNM" />
    <noscript>
      <input type="submit" value="Submit" />
    </noscript>
  </form>
</body>
</html>
```

Post IdP SSO SAMLResponse to niif.hu SP

The screenshot displays the Network tab of a web browser's developer tools. The left pane shows a list of network requests, with the final request (302 POST) selected. The right pane shows the details of this request, including the Request URL, Request method, Status code, and various headers.

Method	File	Source
302 GET	/	ficko.vvc.ni
200 GET	/	ficko.vvc.ni
200 GET	idp.html	niif.hu
404 GET	favicon.ico	ficko.vvc.ni
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu

Request URL: https://niif.hu/Shibboleth.sso/SAML2/POST
Request method: POST
Status code: 302 Found

Response headers (0.534 KB)

- Cache-Control: "private,no-store,no-cache,max-age=0"
- Connection: "Keep-Alive"
- Content-Encoding: "gzip"
- Content-Length: "198"
- Content-Type: "text/html; charset=iso-8859-1"
- Date: "Sun, 10 May 2015 07:16:19 GMT"
- Expires: "Wed, 01 Jan 1997 12:00:00 GMT"
- Keep-Alive: "timeout=5, max=94"
- Location: "https://niif.hu/.well-known/sp/jws.php"
- Server: "Apache/2.2.22"
- Set-Cookie: "_shibsession_6e69696668747470733a2f2f7777772e...db58ac7ecfc7ff3bc05; path=/; secure; HttpOnly"
- Vary: "Accept-Encoding"

Request headers (0.531 KB)

- Host: "niif.hu"
- User-Agent: "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0"
- Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language: "en-US,en;q=0.5"
- Accept-Encoding: "gzip, deflate"
- Content-Type: "application/x-www-form-urlencoded; charset=UTF-8"
- Referer: "https://niif.hu/.well-known/idp-proxy/idp.html"
- Content-Length: "10780"
- Cookie: "PHPSESSID=tfj2l88d897me52cabtcbfu612"
- Connection: "keep-alive"
- Pragma: "no-cache"
- Cache-Control: "no-cache"

Parameters: SAMLResponse, RelayState

The screenshot shows the Network tab of a web browser's developer tools. The left pane lists various requests, including GET requests for JavaScript files from crypto-js.g and kjur.github, and a POST request to niif.hu. The right pane shows the details of the selected POST request, including the SAMLResponse and RelayState parameters.

Method	File	Source
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu
200 GET	jws.php	niif.hu
200 GET	idp.html	niif.hu
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g

Filter request parameters

Form data

SAMLResponse: "PHNhbWxwOJlc3BvbnNlIHhtbG5zOnNhbmWxwP...Bc3NlcnRpb24+PC9zYW1scDpSZXNwb25zZT4="

RelayState: "ss:mem:dff9a70f14f3c3df1650099e100f919b8127f08872ec204b3493fe9825db0e85"

Redirect finally to the original request

The screenshot shows the Network tab of a web browser's developer tools. The left pane lists various requests, including GET requests for files like idp.html, favicon.ico, core.js, sha1.js, sha256.js, x64-core.js, sha512.js, base64x-1.1.js, base64.js, jsbn.js, jsbn2.js, rsa.js, rsa2.js, rsapem-1.1.js, asn1hex-1.1.js, x509-1.1.js, crypto-1.1.js, rsassign-1.2.js, keyutil-1.0.js, json_sans_eval.js, jws-3.0.js, idp-proxy.js, base64x-1.1.js, jws.php, DS?target=https://niif.hu/.well-kno..., SSOService.php?SAMLRequest=fZJ..., POST, jws.php, idp.html, and core.js. The right pane shows the details for the selected 200 GET request to jws.php.

Request URL: https://niif.hu/.well-known/sp/jws.php
Request method: GET
Status code: 200 OK

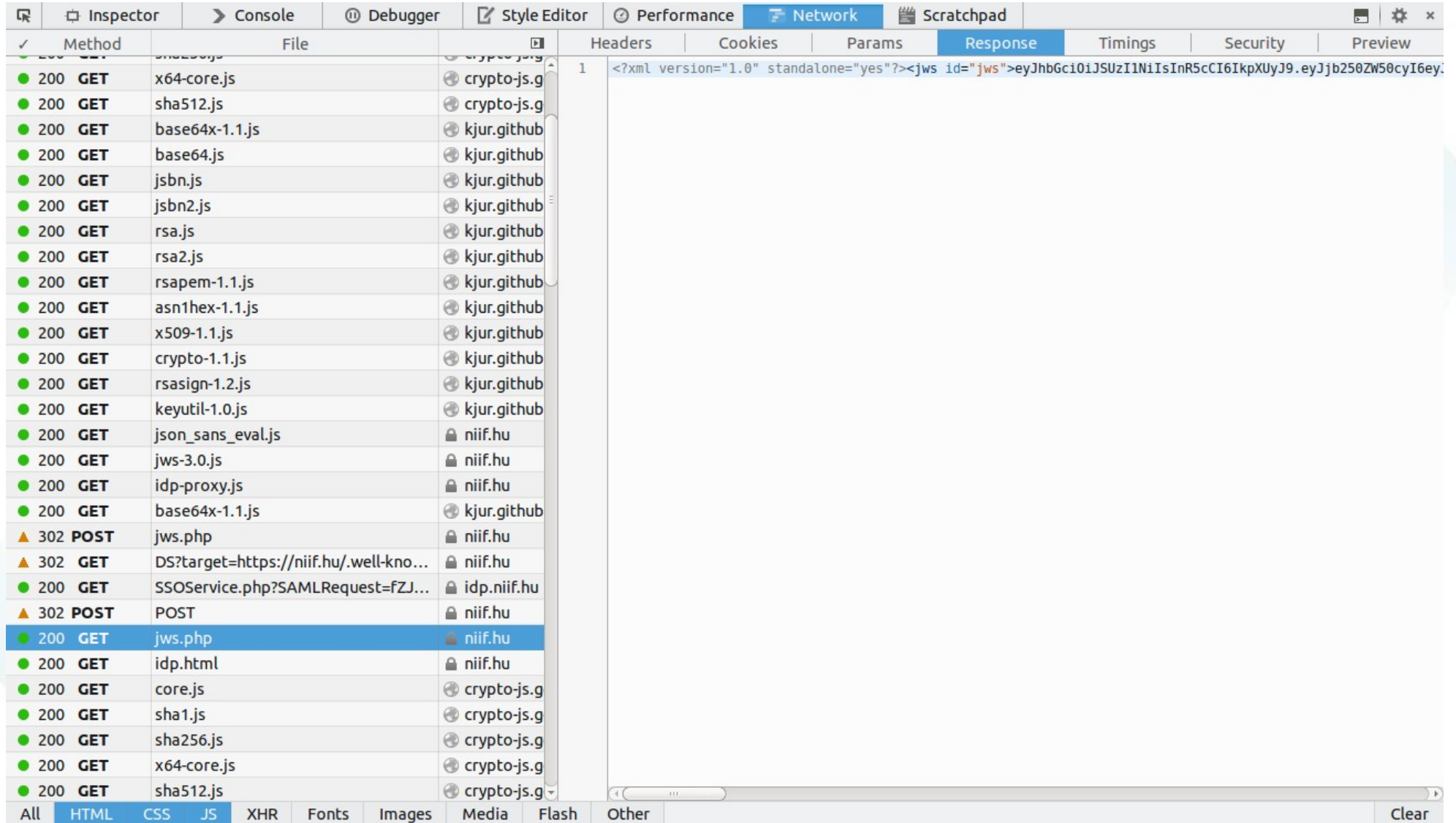
Response headers (0.390 KB):

- Cache-Control: "no-store, no-cache, must-revalidate, post-check=0, pre-check=0"
- Connection: "Keep-Alive"
- Content-Encoding: "gzip"
- Content-Length: "566"
- Content-Type: "text/html"
- Date: "Sun, 10 May 2015 07:16:19 GMT"
- Expires: "Thu, 19 Nov 1981 08:52:00 GMT"
- Keep-Alive: "timeout=5, max=93"
- Pragma: "no-cache"
- Server: "Apache/2.2.22"
- Vary: "Accept-Encoding"
- X-Powered-By: "PHP/5.4.39-0+deb7u2"

Request headers (0.604 KB):

- Host: "niif.hu"
- User-Agent: "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:37.0) Gecko/20100101 Firefox/37.0"
- Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language: "en-US,en;q=0.5"
- Accept-Encoding: "gzip, deflate"
- Referer: "https://niif.hu/.well-known/idp-proxy/idp.html"
- Content-Type: "application/x-www-form-urlencoded"
- Cookie: "PHPSESSID=tfj2l88d897me52cabtcbfu612; _shibses...26f6c657468=_3ad792a343180db58ac7ecfc7ff3bc05"
- Connection: "keep-alive"
- Pragma: "no-cache"
- Cache-Control: "no-cache"

Get the final signed JWT/JWS



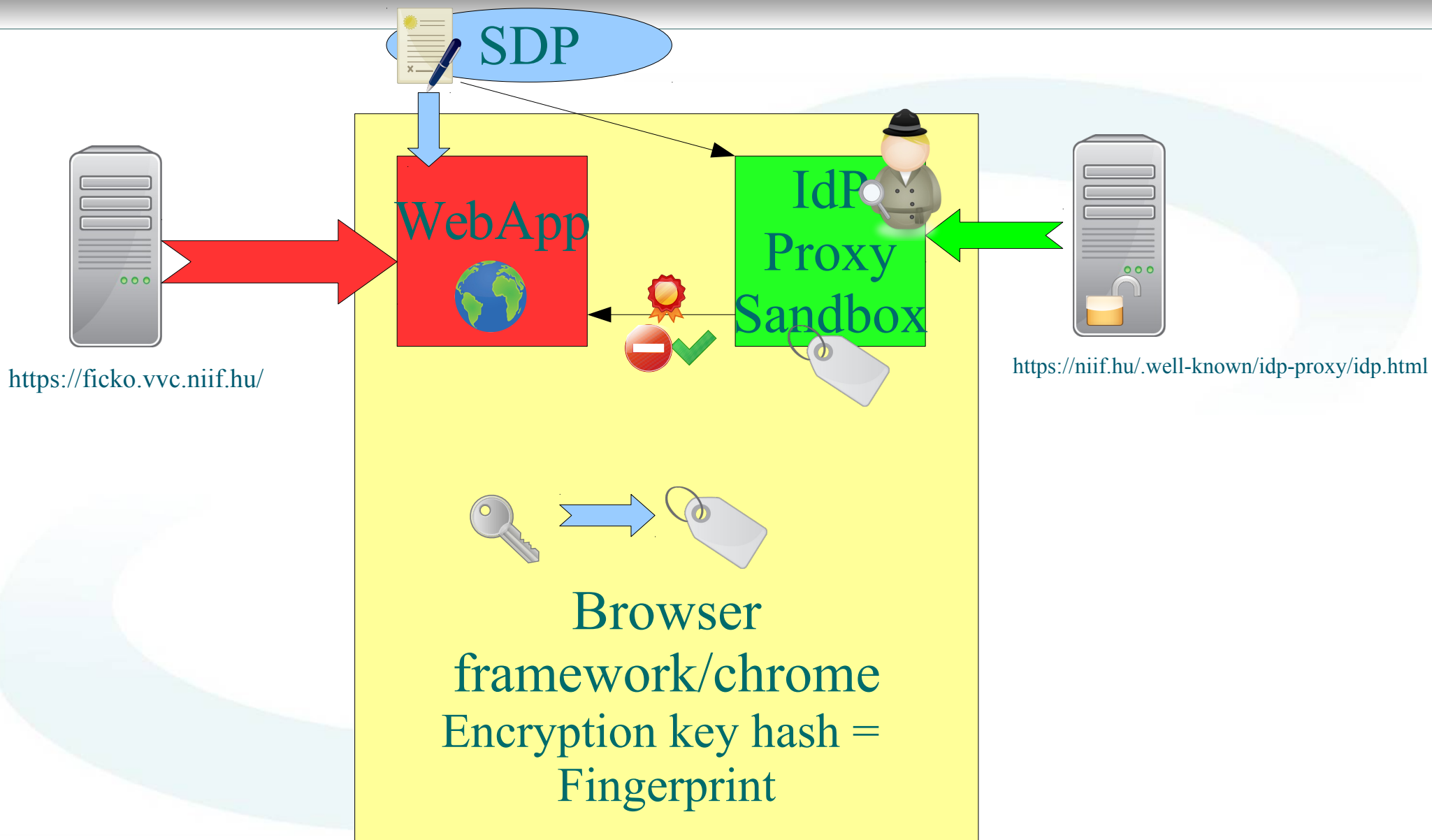
The screenshot shows the Network tab of a web browser's developer tools. The list of requests includes various JavaScript files and a SAML request. The 'jws.php' request is highlighted, and its response is displayed in the right pane.

Method	File	Source
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g
200 GET	base64x-1.1.js	kjur.github
200 GET	base64.js	kjur.github
200 GET	jsbn.js	kjur.github
200 GET	jsbn2.js	kjur.github
200 GET	rsa.js	kjur.github
200 GET	rsa2.js	kjur.github
200 GET	rsapem-1.1.js	kjur.github
200 GET	asn1hex-1.1.js	kjur.github
200 GET	x509-1.1.js	kjur.github
200 GET	crypto-1.1.js	kjur.github
200 GET	rsassign-1.2.js	kjur.github
200 GET	keyutil-1.0.js	kjur.github
200 GET	json_sans_eval.js	niif.hu
200 GET	jws-3.0.js	niif.hu
200 GET	idp-proxy.js	niif.hu
200 GET	base64x-1.1.js	kjur.github
302 POST	jws.php	niif.hu
302 GET	DS?target=https://niif.hu/.well-kno...	niif.hu
200 GET	SSOService.php?SAMLRequest=fZJ...	idp.niif.hu
302 POST	POST	niif.hu
200 GET	jws.php	niif.hu
200 GET	idp.html	niif.hu
200 GET	core.js	crypto-js.g
200 GET	sha1.js	crypto-js.g
200 GET	sha256.js	crypto-js.g
200 GET	x64-core.js	crypto-js.g
200 GET	sha512.js	crypto-js.g

The response for the 'jws.php' request is shown in the right pane:

```
<?xml version="1.0" standalone="yes"?><jws id="jws">eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXUyJ9.eyJjb250ZW50cyI6ey...
```

Verify (Identity assertion/certificate + Encryption key fingerprint/hash)



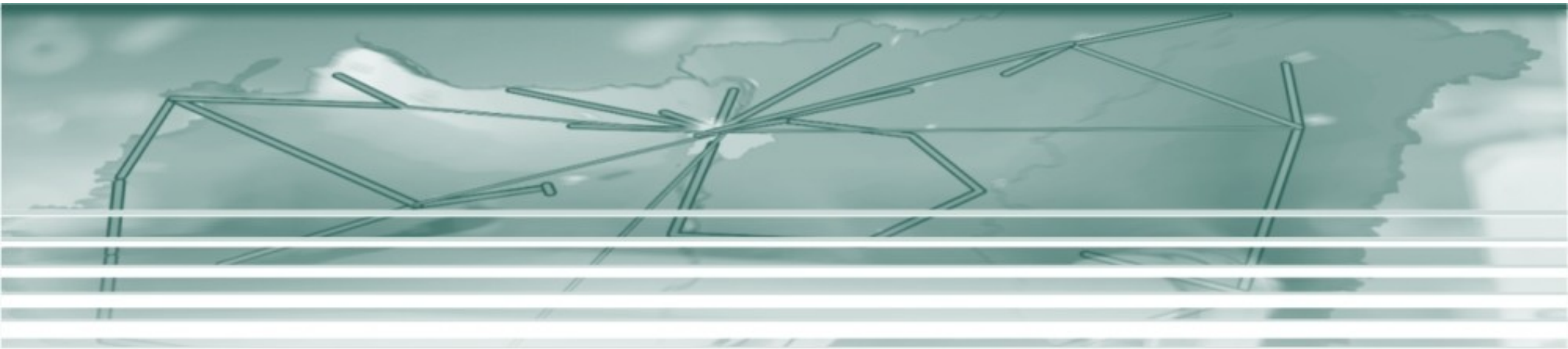
Assumptions and Limitations

- Limitations for SAML federation
 - User needs to have a valid session with his home IdP.
 - Automatic SAML attribute consent needed for automatic SP login (niif.hu)
 - SSO on IdP/SP (identity domain SP)
 - CORS on IdP side is needed.
 - Consequence of XMLHttpRequest usage for signing JWT.
 - All web server that serves any identity domain part
 - must have been deployed the identity-proxy code
 - and must have function as SAML SP.

Summary & Takeaways

- We have trusted World Wide federated AAI service deployed. There is a demand to use this already in place service with WebRTC for RTC.
- WebRTC Identity is still a moving target.
 - Security model is in draft phase. Not everybody is happy with it.
- Implementation issues
 - Lack of presentation of the verified Identity in the browser Chrome/Framework in current Firefox 37.0.2 implementation.
 - WebRTC identity lack of implementation Browsers
 - Critical Mass needed.
 - More interest needed from Web Application developers to implement it.
 - Only one attribute the Identity is handled and provided by the IdP-Proxy.
 - Lack of support for more than the identity attribute:
More attributes needed like picture URL, entitlements, group membership etc. to support more sophisticated Authentication and call accept/reject decision.
- As I proved in my demonstration, the usage of a SAML based Identity federation as WebRTC Identity Provider is pretty complicated, but it is not completely impossible!

Thank You!



Big thanks to Gyufi, Kristóf, Sitya (NIIF AAI Team)
for their continuous help and support!

References

- Demo: https://youtu.be/aeXaWDNU_sg
- WebRTC
 - <https://tools.ietf.org/html/draft-ietf-rtcweb-security-arch-11>
 - <http://w3c.github.io/webrtc-pc/#identity>
 - <http://www.ietf.org/proceedings/82/slides/rtcweb-13.pdf>
- SAML
 - <https://technical.edugain.org/status.php>
 - <https://refeds.org/resources/>
 - <http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.html>
 - http://en.wikipedia.org/wiki/SAML_2.0
- JWT
 - <http://kjur.github.io/jsjws/>
 - https://github.com/ritou/php-Akita_JOSE
 - <https://code.google.com/p/json-sans-eval/>
 - <http://jwt.io/>