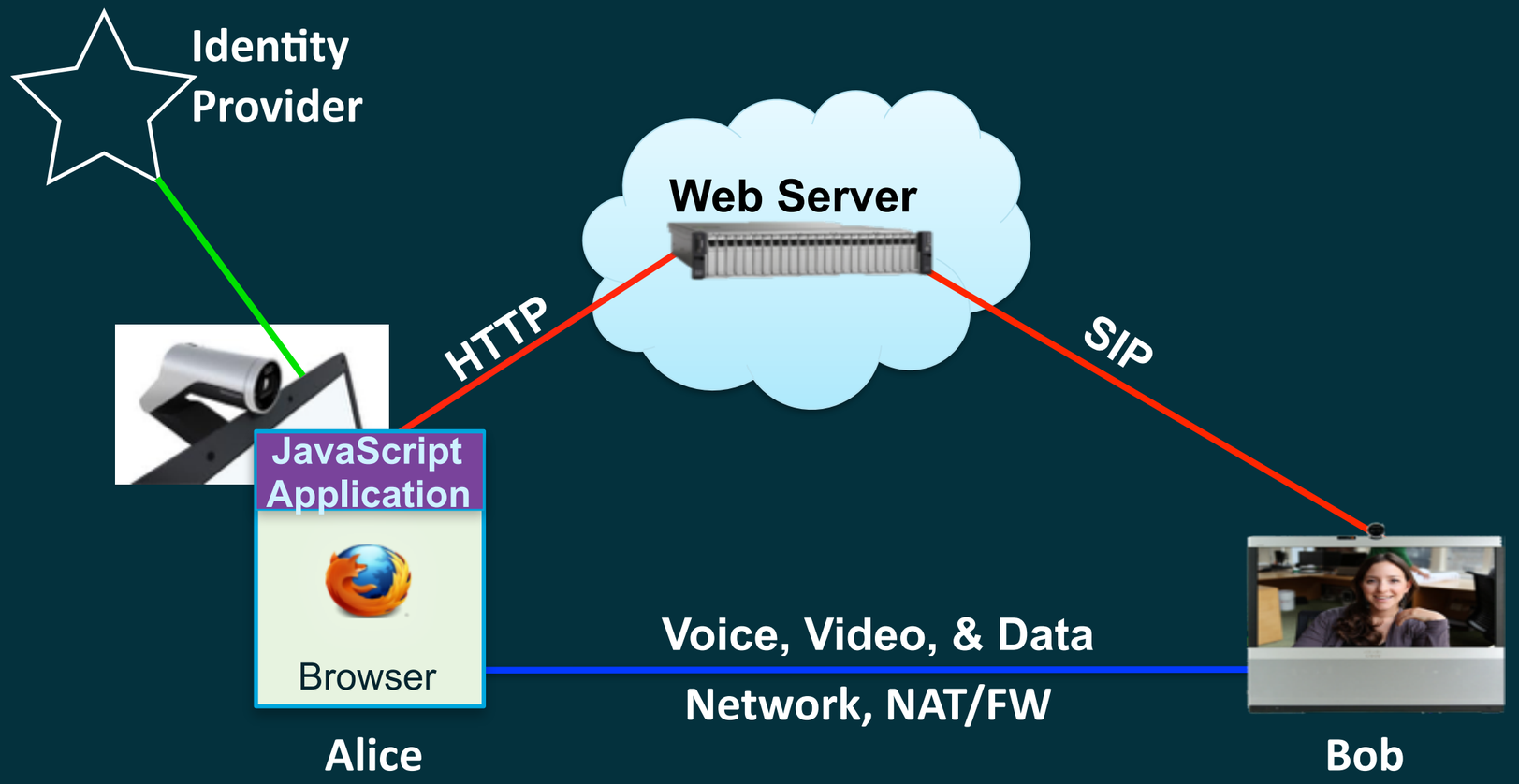
WebRTC Identity: Explained Top to Bottom

Dr. Cullen Jennings
September 24, 2015
fluffy@cisco.com

v42

Architecture



Threats

- Toll fraud
  - Unauthorized or non-billable resource utilization
- Eavesdropping
  - Listening to another's call
- Learning private information
  - Caller ID, DTMF password/accounts, calling patterns
- Session replay
  - Replay a session, such as a bank transaction
- Fake identity & Impersonation
- Hijacking calls
- Media tampering
- Denial of service
  - Hanging up other people's conversations
  - Contributing to other DOS attacks
- SPAM
  - SPIM, SPIT, and more SPAM

Who is fluffy@cisco.com

- Who is in the best position to make strong assertions about who fluffy@cisco.com is?
  - Cisco.com allocated the address fluffy to Cullen
  - They provided a way for Cullen to prove his identity with logon password, secure token card, etc.
- Who knows who cisco.com is?
  - The CA can verify with DNS registrars who has been given that name and can get appropriate contacts for it
- Cisco owns the name space
  - They can give "fluffy" to some other user

The Cake is a Lie

- Think about it, you can any two out of the following three properties:
  - Strong Encryption
  - Password Reset
  - History

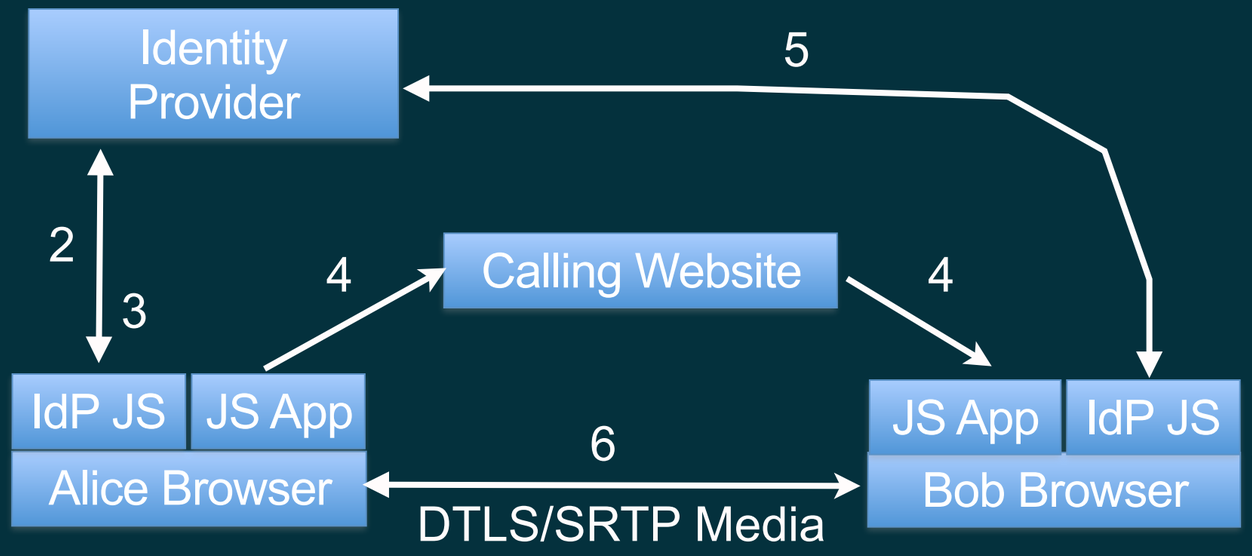
WebRTC - The big picture
Who you going to trust?

- We have to trust someone to run the namespace for users
  - Minimize this function to a "Identity Provider"
    - As future work, block chain techniques with P2P may further reduce the amount we need to trust an Identity Provider
  - Don’t require trust of website that runs the service to not reveal our media
  - Do trust it website with information about our usage patterns
    - As future work, P2P onion routing networks with appropriate random message mixing and constant rate transition may further reduce the meta data revealed

- We have to trust the people, devices, and software that have access to unencrypted media
  - Anyone can record the output of the speaker
  - Means we have to trust the web browser we are using
    - Choose wisely
    - We do not need to trust the JavaScript app we are running with the media contents

How WebRTC Works

Identity: The fluffy hand wavy explanation



1. Application is configured with identity provider(s) for the use
2. User “logs on” using protocol downloaded from identity provider in JavaScript/HTML
3. Browser get an assertion from identity provider which binds the DTLS fingerprint to the identity such as fluffy@cisco.com
4. The calling JavaScript passes the assertion to far side
5. Bob’s browser verifies the assertion with identity provider and check DTLS fingerprint matches the assertion
6. DTLS sets up keys for SRTP. SRTP encrypts traffic
7. Browser display "secure to fluffy@cisco.com"

Simple Peer to Peer Example

Each "tab" of a browser is isolated in a separate context

CommonIdentity.Org

Identity Provider

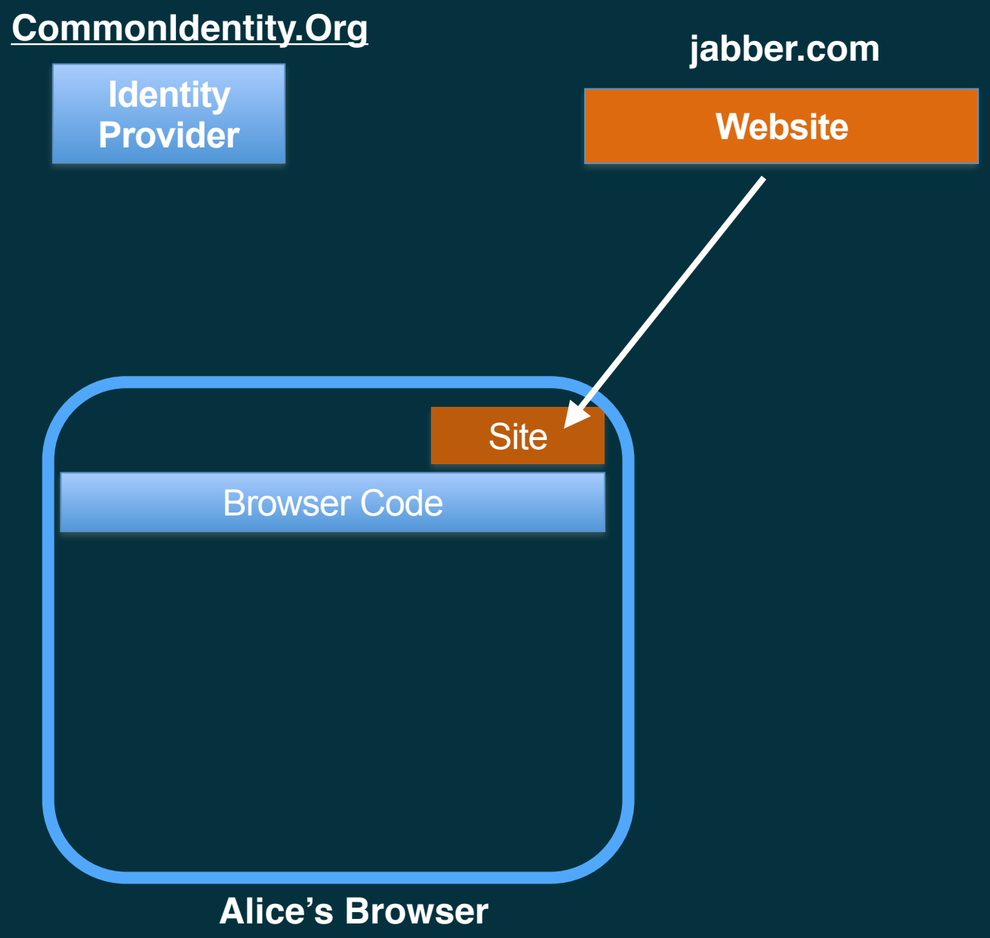
jabber.com

Website

Alice's Browser

- Identity will rely on three different contexts in the the browsers
- The browser will pass some information between them
- Blue stuff is trusted for some certain things while the orange is less trusted
- Green objects in the browser are semi private and not shared with the orange things

Browser Loads Website for jabber.Com



Site tells browser which Identity Provider to use

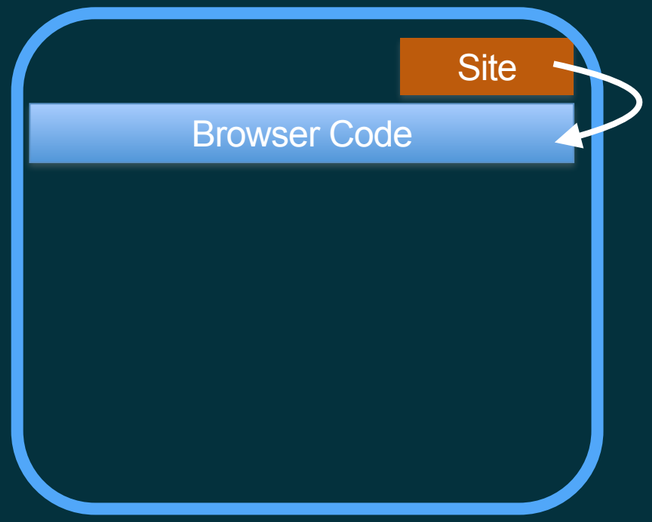
CommonIdentity.Org

Identity Provider

jabber.com

Website

1. The site might use only one identity provider or there might be a way for user to enter which Identity Provider to use



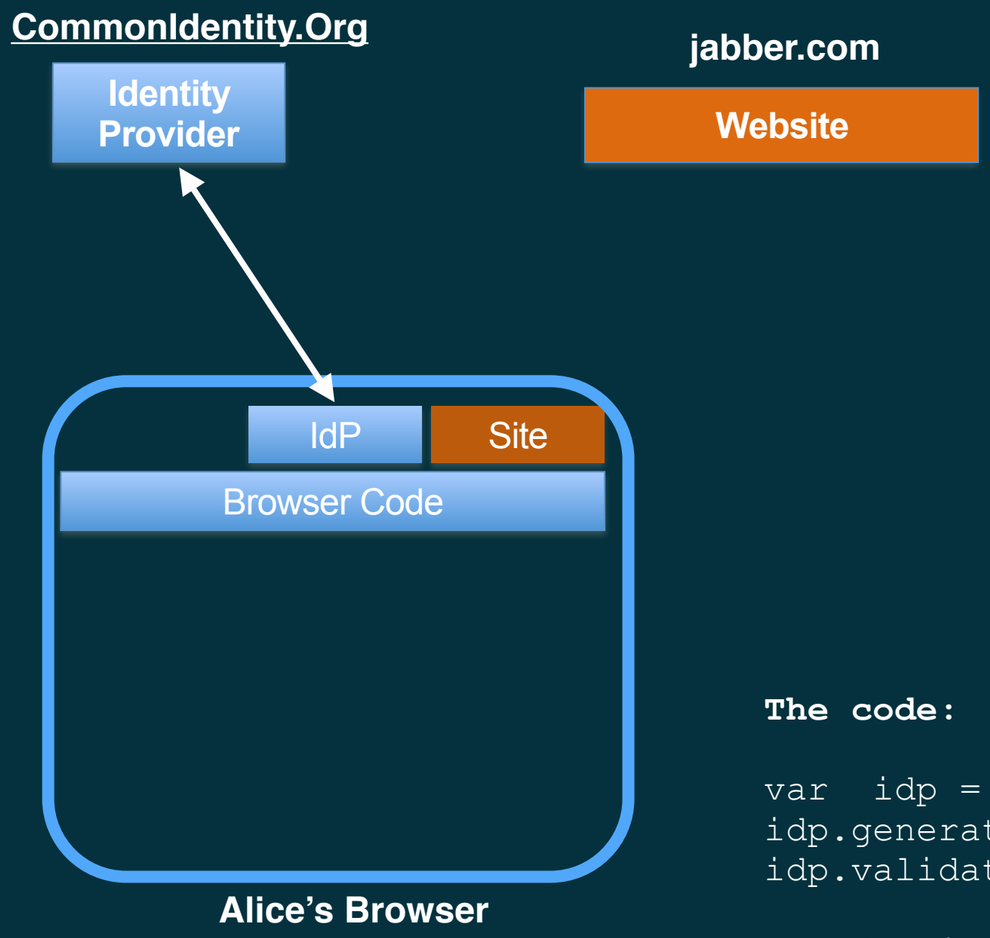
Alice's Browser

The code:

```
// first load adapter.js from https://github.com/webrtc/adapter
var configuration = { "iceServers": [] };
var pc = new window.RTCPeerConnection(configuration);

pc.setIdentityProvider("commonidentity.org","v1","fluffy");
```

Browser loads the IdP Javascript in new "hidden tab"



1. IdP JS has to be loaded over HTTPS
2. Redirects are allowed
3. The IdP JS must provides an certain API interface which the browser calls
4. The IdP code registers the API calls with so browser can call them

The code:

```
var  idp = {};
idp.generateAssertion = function( contents, origin, userNameHint ) {…}
idp.validateAssertion = function( assertion, origin ) {… }

rtcIdentityProvider.register( idp );
```

IdP tells login URL to browser, browser passes to Site

CommonIdentity.Org

Identity Provider

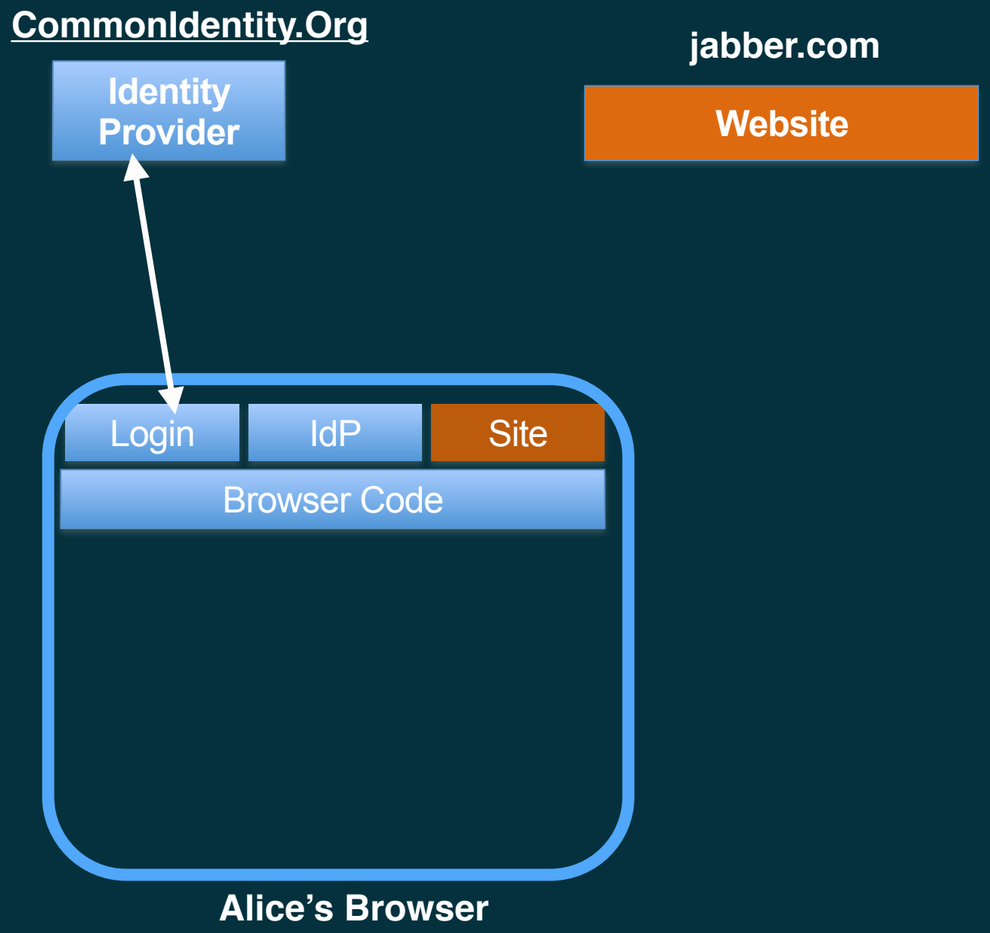
jabber.com

Website

Alice's Browser

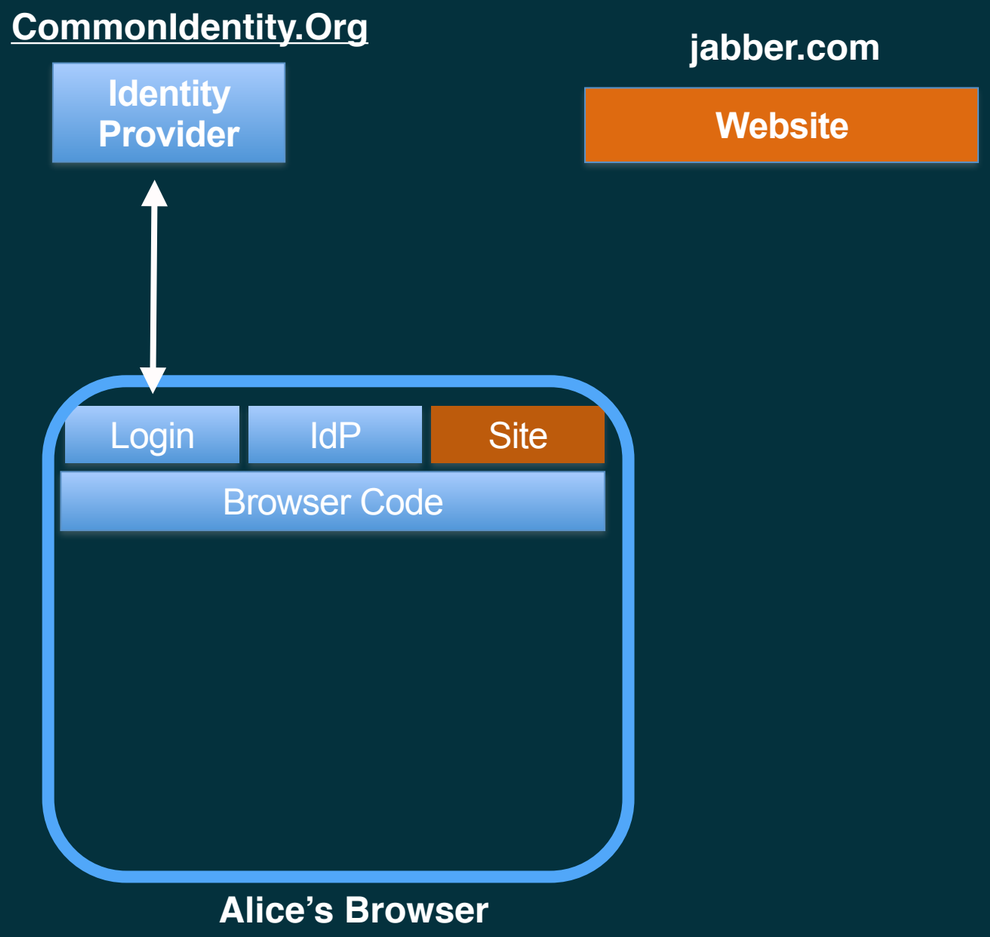
1. It's slightly more complicated how site can test to see if user is already logged on and if not get this login URL but this is a close enough simplification
2. What actually happens is site ask for assertion (more on how later) and user is not logged in so IdP returns error with loginUrl attribute that has URL to login

Site opens new tab at the login URL



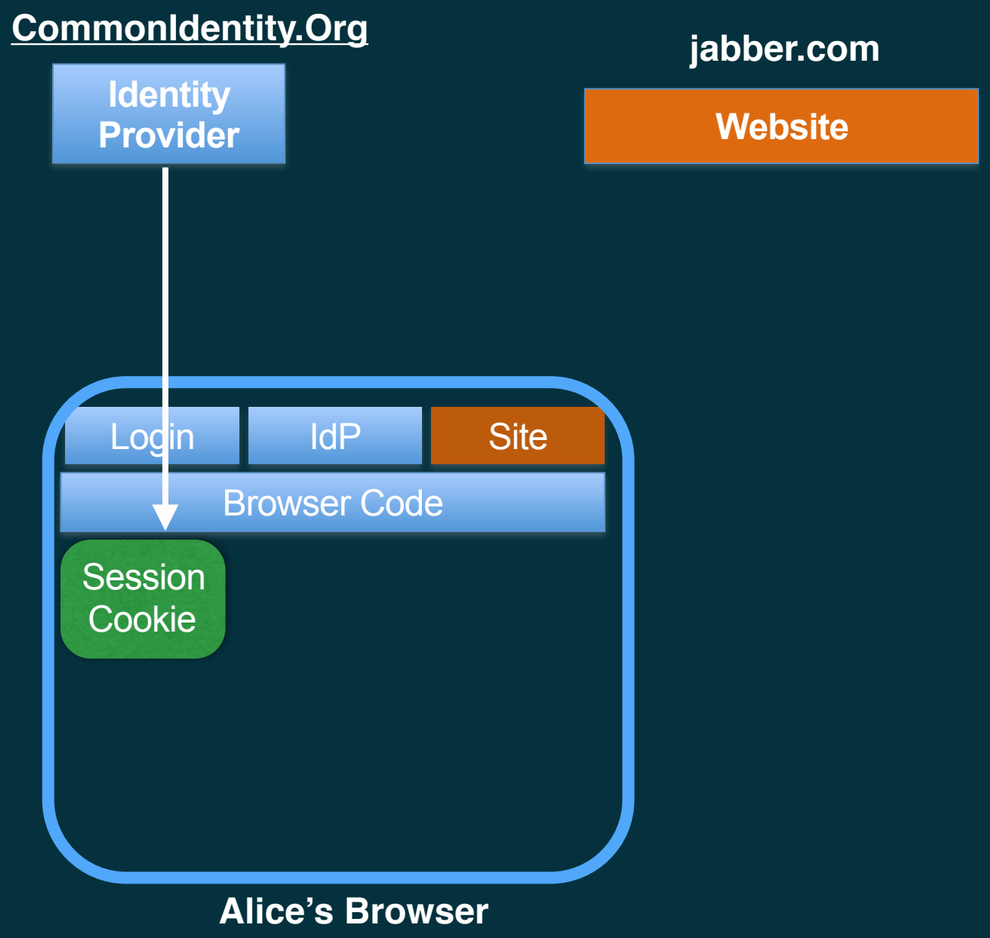
1. The site can use the URL to create popup or IFRAME that loads the login web page

Login window is displayed to user and does login dance



1. The user uses this webpage to login
2. The webpage might do things like OAuth to redirect to some SSO server but that is not shown in this flow

If user logs in, an session cookie is returned



1. The IdP and Login window both go to the same origin so this session cookie allows both of them to tell the Identity Provider which user logged on

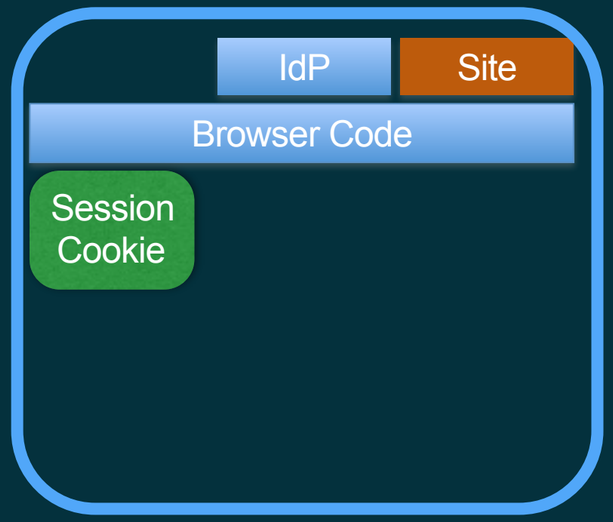
Login window goes away

CommonIdentity.Org

Identity Provider

jabber.com

Website



Alice's Browser

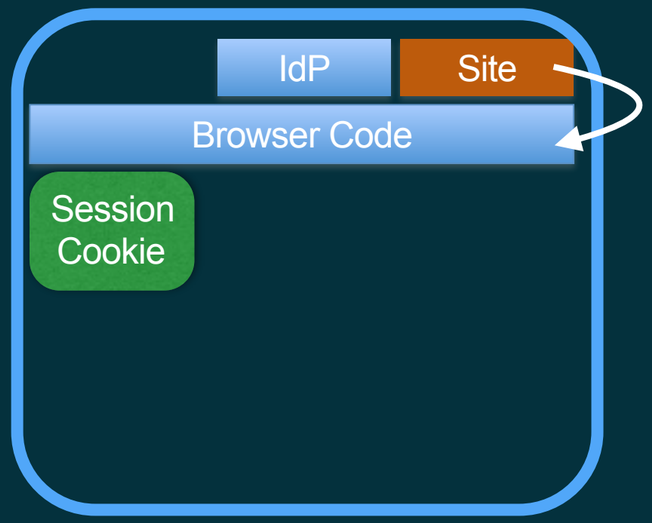
Site creates WebRTC Peer Connection in browser

CommonIdentity.Org

Identity Provider

jabber.com

Website



Alice's Browser

The code:

```
// first load adapter.js from https://github.com/webrtc/adapter
var configuration = { "iceServers": [] };
var pc = new window.RTCPeerConnection(configuration);
```

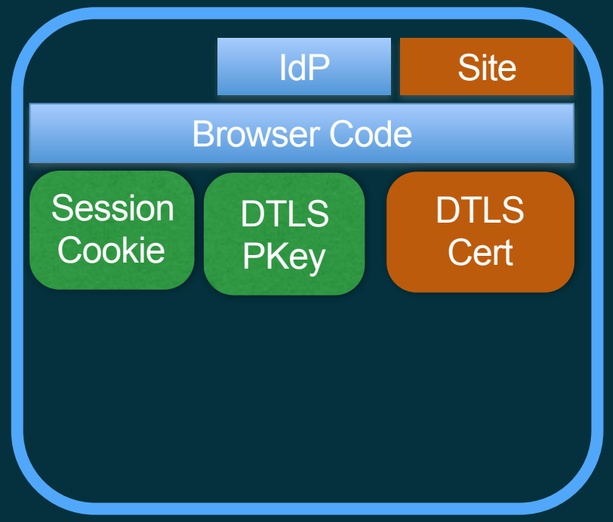
Browser creates Certificate and Key for PeerConnection

CommonIdentity.Org

Identity Provider

jabber.com

Website



Alice's Browser

1. Note the Site JS can not get the private key for the certificate
2. More than one certificate might be created with different crypto algorithms

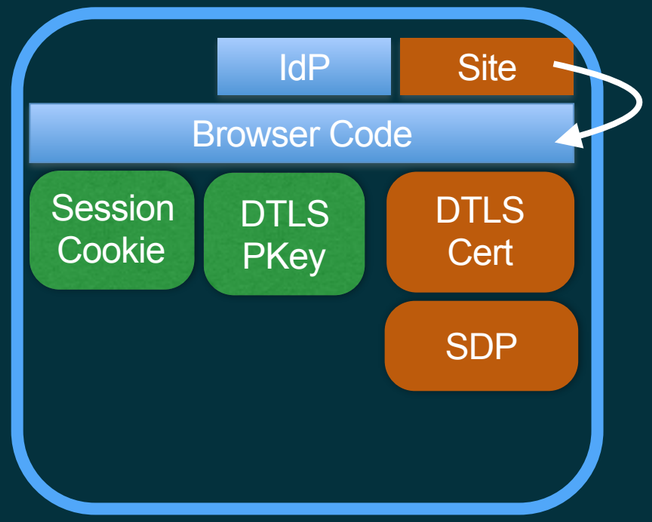
Site asks browser to create an SDP Offer

CommonIdentity.Org

Identity Provider

jabber.com

Website



Alice's Browser

The code:

```
pc.onnegotiationneeded = function () {
        pc.createOffer().then(function (offer) {
            return pc.setLocalDescription(offer);
        })
        .then(function () {
            // send the offer to the other peer
        })
        .catch(logError);
    };
```

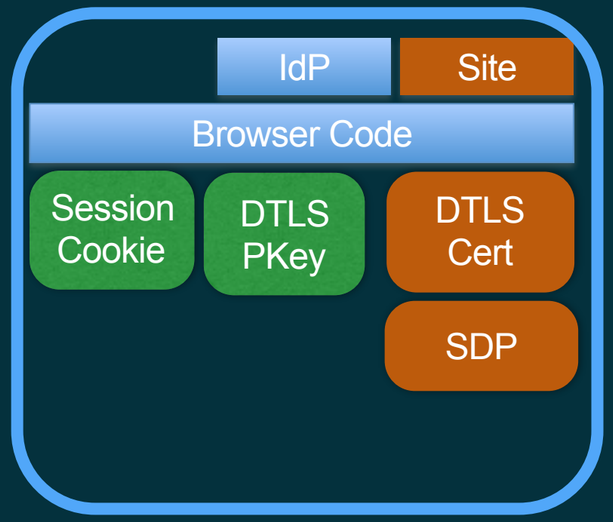
Browser computes fingerprint of cert and puts in SDP

CommonIdentity.Org

Identity Provider

jabber.com

Website



Alice's Browser

1. This adds an "a=fingerprint" lines to the SDP
2. More than one fingerprint might be added to support more than one hash algorithm and if there were multiple certificates, they would also have their own fingerprint lines

The SDP:

```
o=SDPARTA-40.0.3 6433281818588249631 0 IN IP4 0.0.0.0
s=-
t=0 0
a=sendrecv
a=fingerprint:sha-256 B1:3F:...:14:39
a=group:BUNDLE sdparta_0
a=identity:eyJhc3NlcnRpb24 .... joidjEifX0=
m=application 59974 DTLS/SCTP 5000
c=IN IP4 192.168.1.1
a=candidate:0 1 UDP 2130379007 192.168.1.1 60227 typ host
a=sendrecv
a=ice-pwd:82c30c0eb0f0af59ccb2e2a2ad1ac346
a=ice-ufrag:414c78bd
a=mid:sdparta_0
a=sctpmap:5000 webrtc-datachannel 256
a=setup:actpass
```

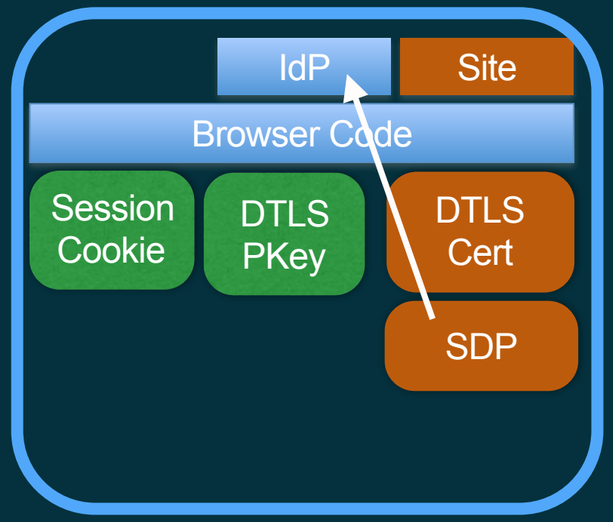
Browser requests IdP to create an assertion binding user identity to fingerprint

CommonIdentity.Org

Identity Provider

jabber.com

Website



Alice's Browser

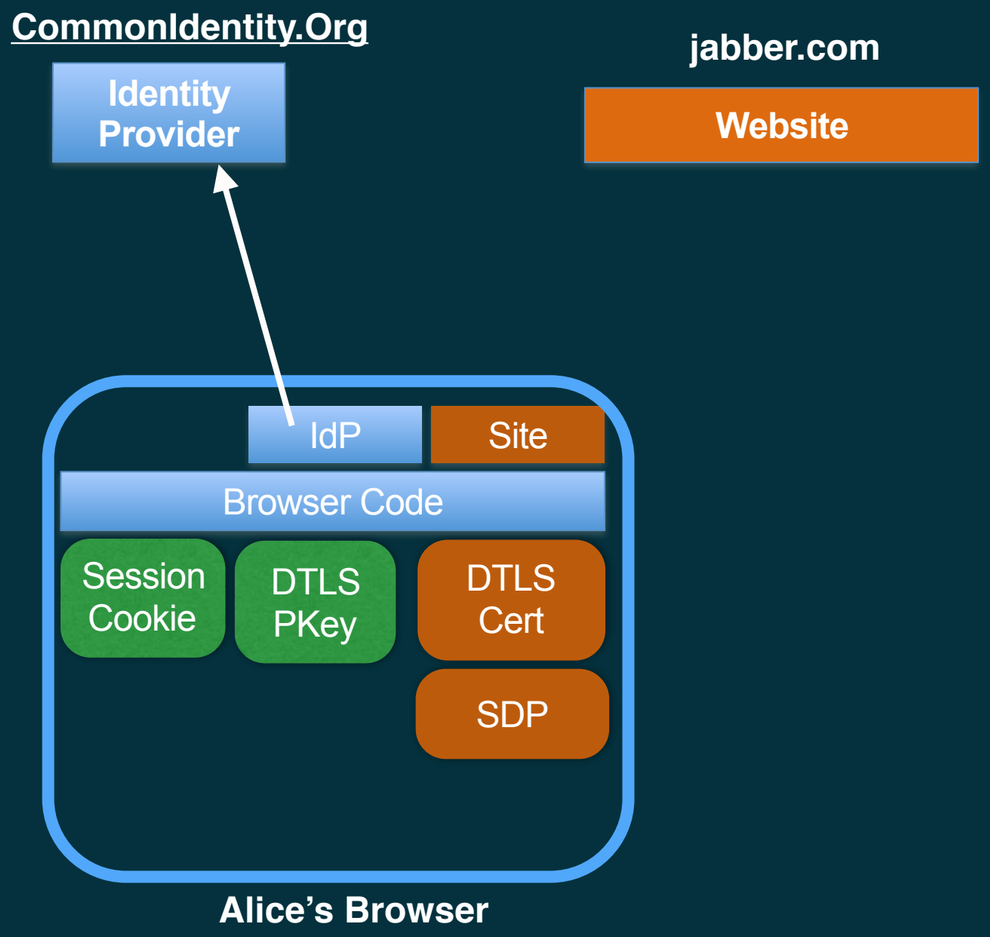
From the assertion, the IdP and Identity Provider need to be able to:

1. return the fingerprint(s) used to create it
2. provide the domain name of identity provider
3. provide the authenticated user name
4. check the assertion has not been modified

The code:

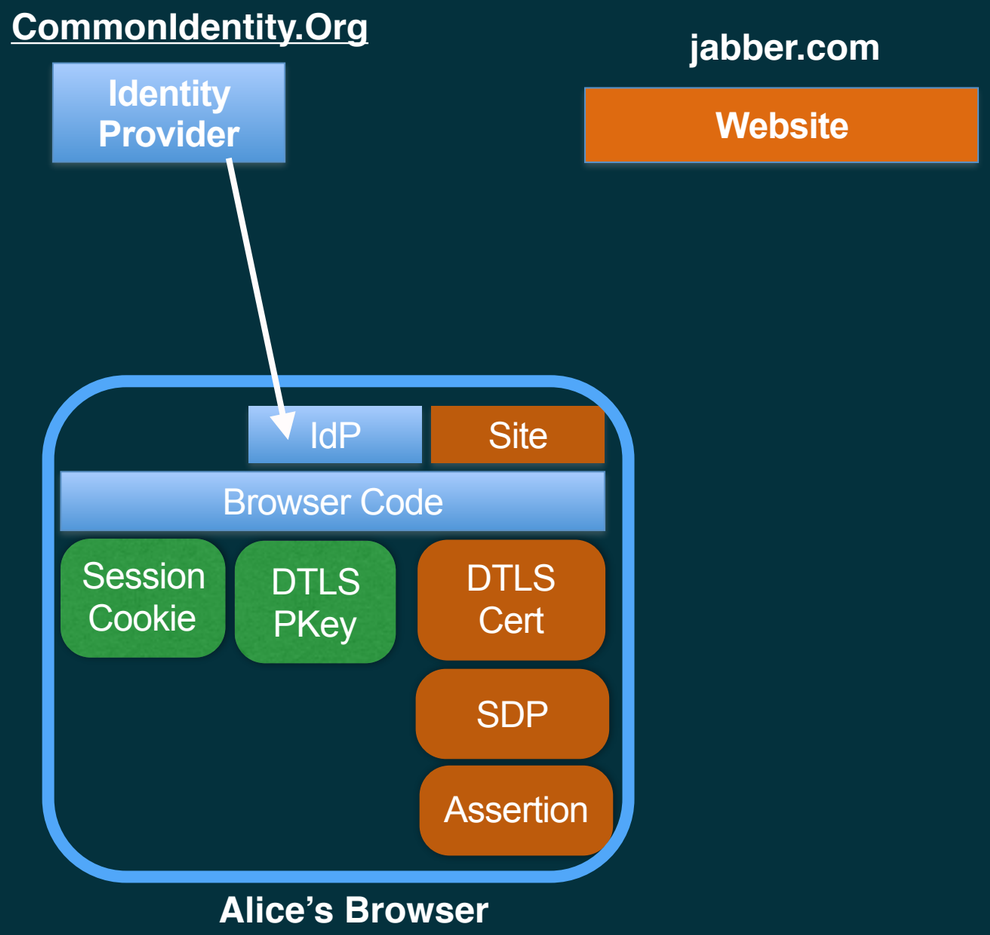
```
idp.generateAssertion = function( contents, origin, userNameHint ) {
    var p = new Promise(
        function( resolve, reject ) {
            var assertResult = {};
            assertResult.assertion = contents;
            assertResult.idp = {};
            assertResult.idp.domain = origin;
            assertResult.idp.protocol = "v1";
            resolve( assertResult );
        } );
    return p;
}
```

IdP sends request and cookie to Identity Provider



1. Authorizes to Identity Provider with cookie

Identity Provider returns signed assertion



1. The IdP JS gets the assertion
2. Other sort of IdP models might have ad the IdP JS just generated the assertion locally instead of talking to Identity Provider

IdP gives assertion to browser which adds to SDP

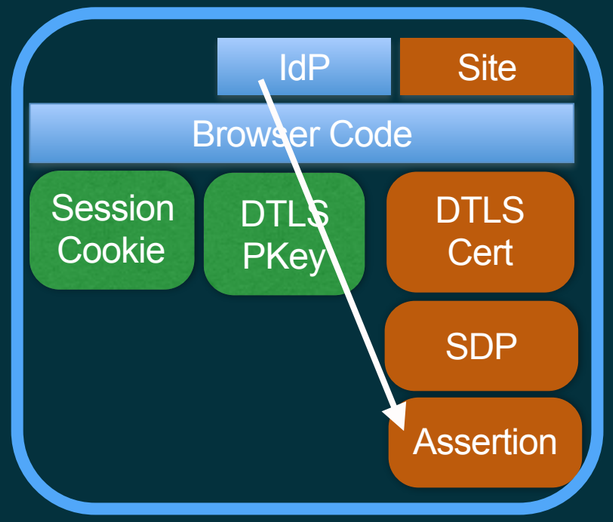
CommonIdentity.Org

Identity Provider

jabber.com

Website

1. Get puts in "a=identity" line of SDP



Alice's Browser

The code:

```
idp.generateAssertion = function( contents, origin, userNameHint ) {
    var p = new Promise(
        function( resolve, reject ) {
            var assertResult = {};
            assertResult.assertion = contents;
            assertResult.idp = {};
            assertResult.idp.domain = origin;
            assertResult.idp.protocol = "v1";
            resolve( assertResult );
        } );
    return p;
}
```

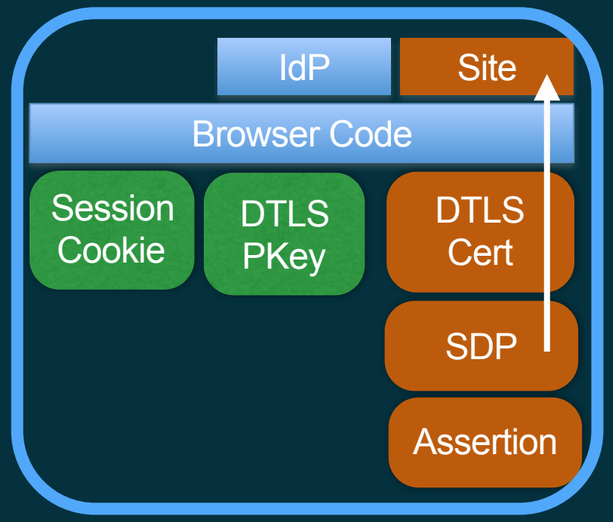
Browser computes SDP (with assertion) for Site

CommonIdentity.Org

Identity Provider

jabber.com

Website

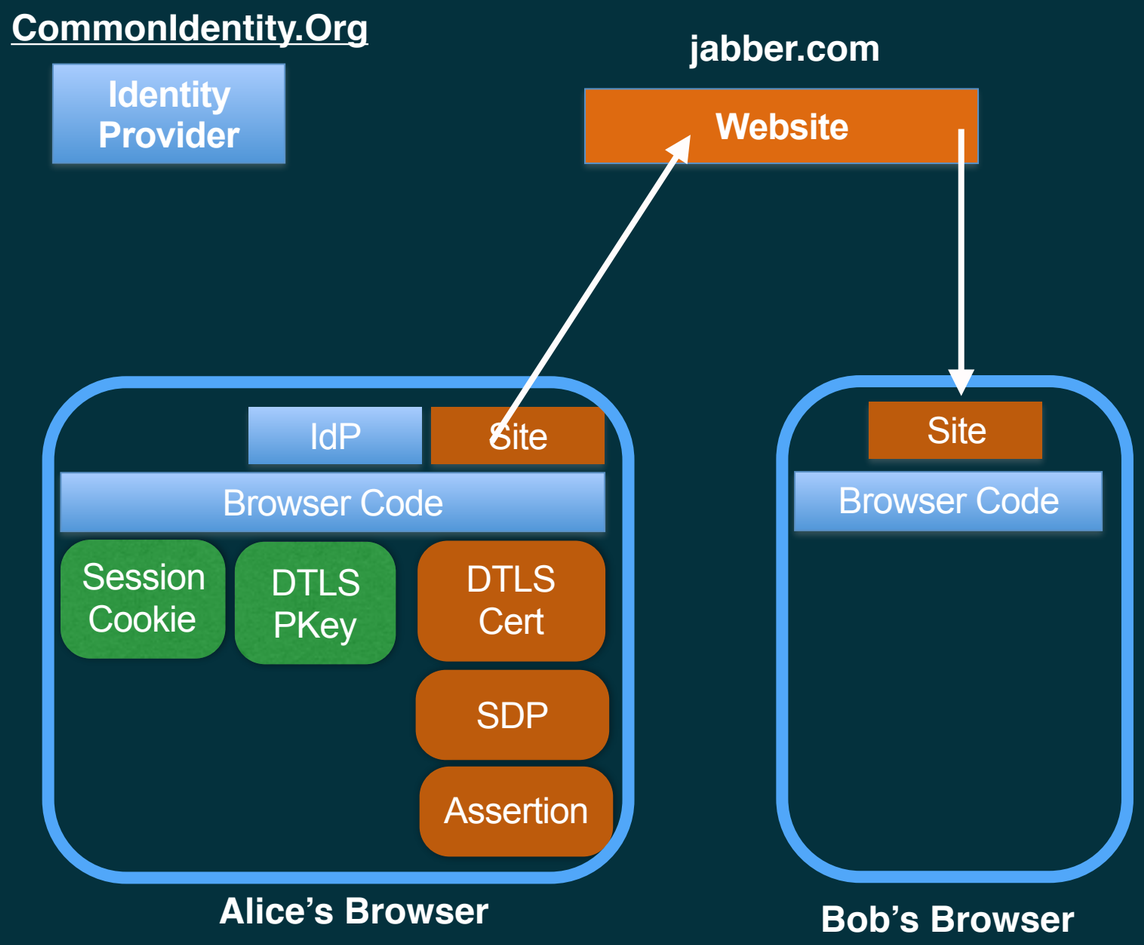


Alice's Browser

The SDP:

```
o=SDPARTA-40.0.3 6433281818588249631 0 IN IP4 0.0.0.0
s=-
t=0 0
a=sendrecv
a=fingerprint:sha-256 B1:3F:...:14:39
a=group:BUNDLE sdparta_0
a=identity:eyJhc3NlcnRpb24 .... joidjEifX0=
m=application 59974 DTLS/SCTP 5000
c=IN IP4 192.168.1.1
a=candidate:0 1 UDP 2130379007 192.168.1.1 60227 typ host
a=sendrecv
a=ice-pwd:82c30c0eb0f0af59ccb2e2a2ad1ac346
a=ice-ufrag:414c78bd
a=mid:sdparta_0
a=sctpmap:5000 webrtc-datachannel 256
a=setup:actpass
```

SDP sent to Website which sends to Bob's browser



The SDP:

```
o=SDPARTA-40.0.3 6433281818588249631 0 IN IP4 0.0.0.0
s=-
t=0 0
a=sendrecv
a=fingerprint:sha-256 B1:3F:...:14:39
a=group:BUNDLE sdparta_0
a=identity:eyJhc3 .... 4iOiJo
a=msid-semantic:WMS *
a=identity:eyJhc3NlcnRpb24 .... joidjEifX0=
m=application 59974 DTLS/SCTP 5000
c=IN IP4 192.168.1.1
a=candidate:0 1 UDP 2130379007 192.168.1.1 60227 typ host
a=sendrecv
a=ice-pwd:82c30c0eb0f0af59ccb2e2a2ad1ac346
a=ice-ufrag:414c78bd
a=mid:sdparta_0
a=sctpmap:5000 webrtc-datachannel 256
a=setup:actpass
a=ssrc:777462401 cname:{a815ff20-…-47ae3d040c3d}
```

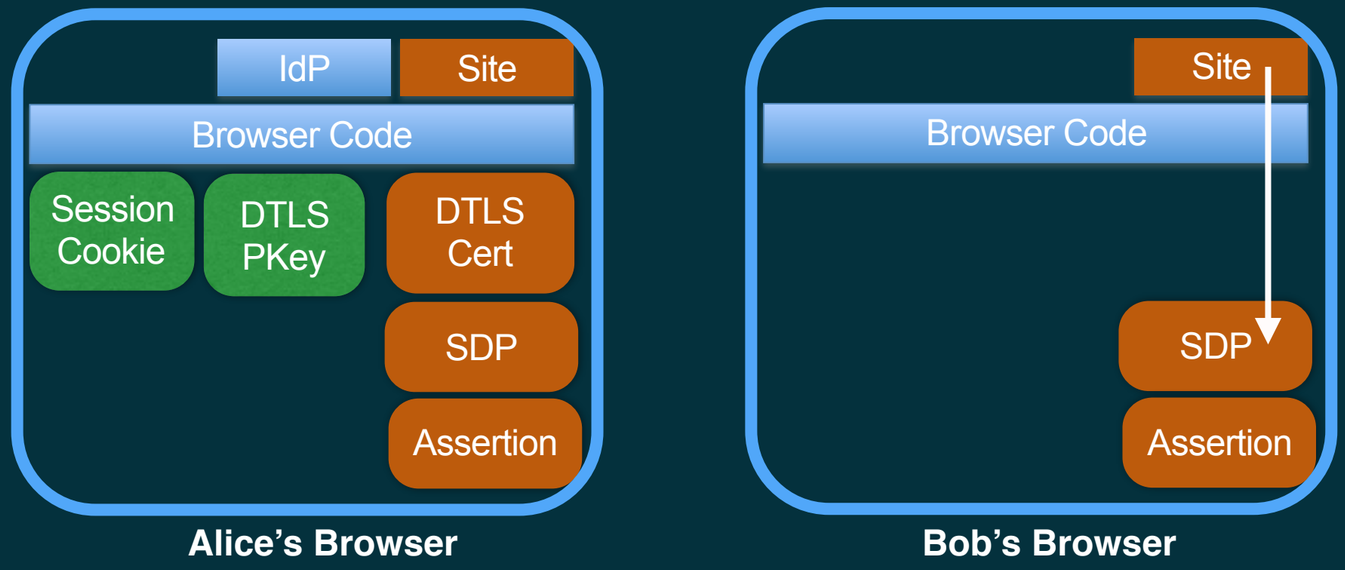
Site passes SDP to browser which extracts assertion

CommonIdentity.Org

Identity Provider

jabber.com

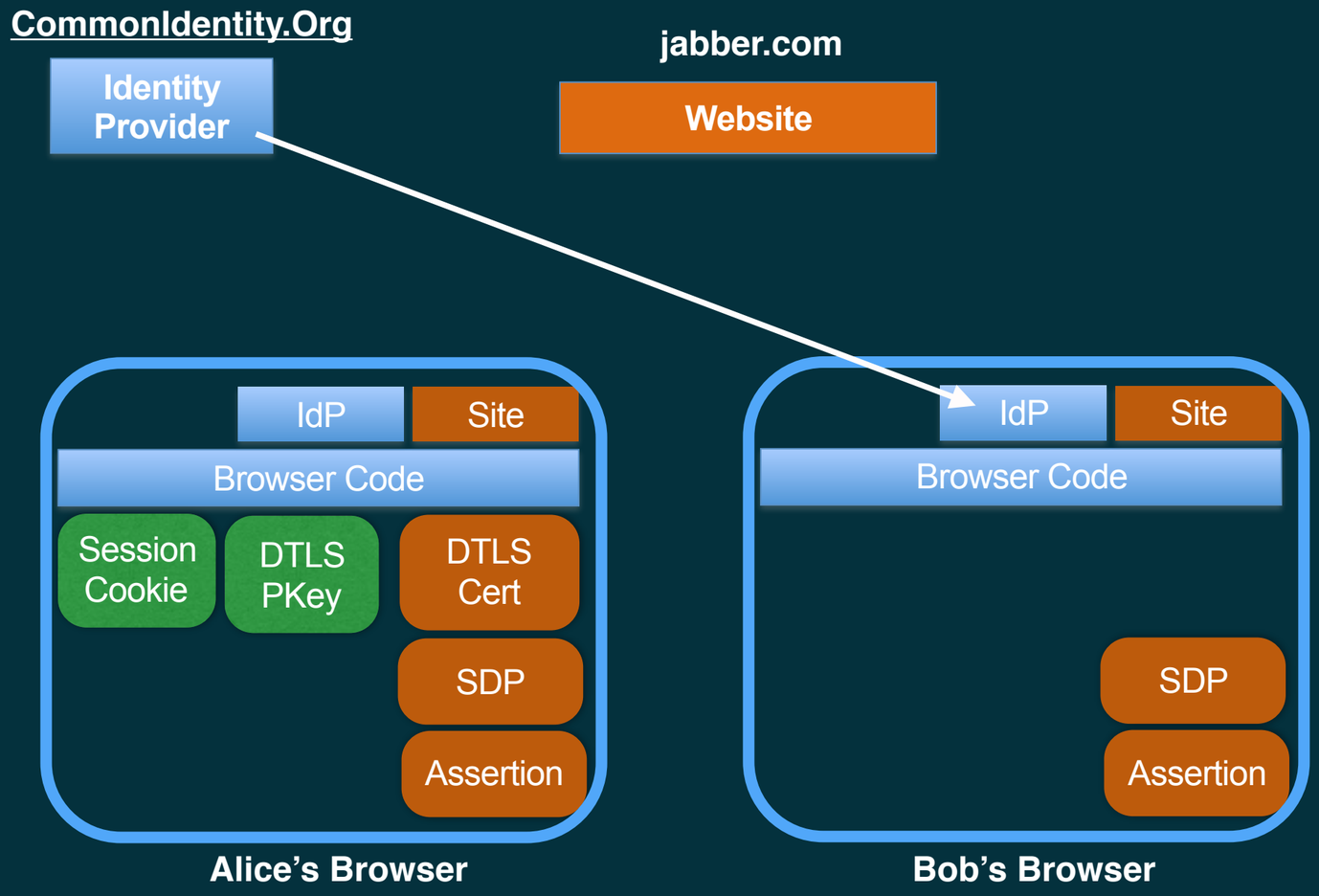
Website



The code:

```
pc.setRemoteDescription(offer);
```

Browser looks at assertion and extracts Identity Provider then loads IdP



1. IdP registers with the browser as on Alice side

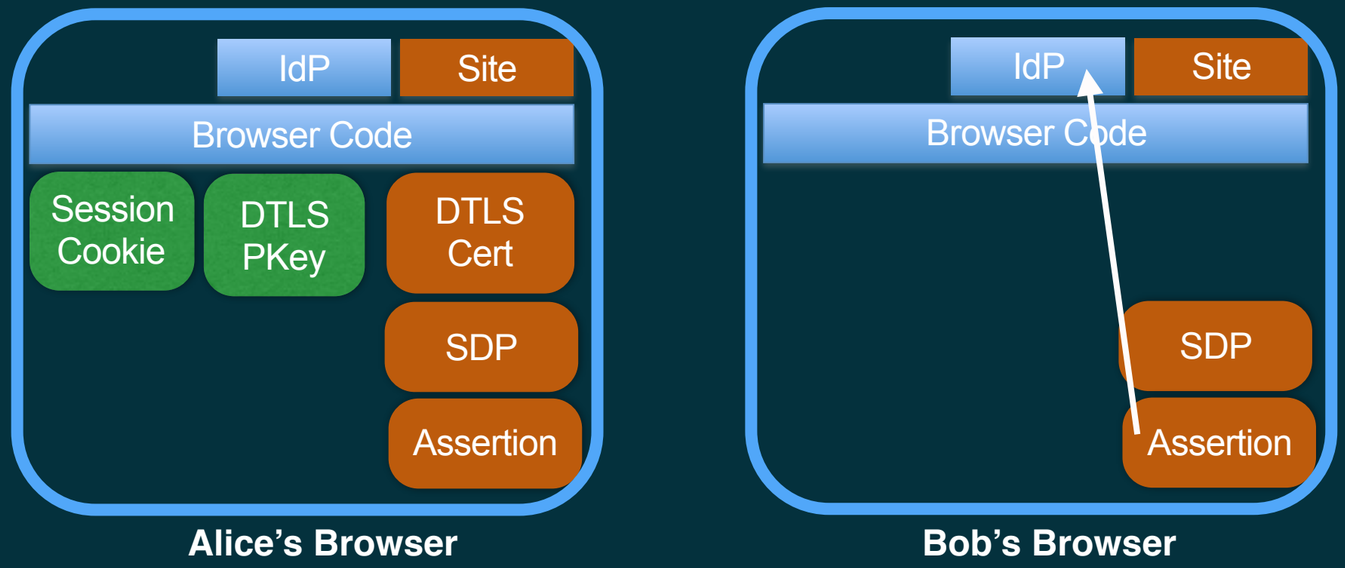
Browser passes assertion to IdP and asks the IdP to validate and provide fingerprint

CommonIdentity.Org

Identity Provider

jabber.com

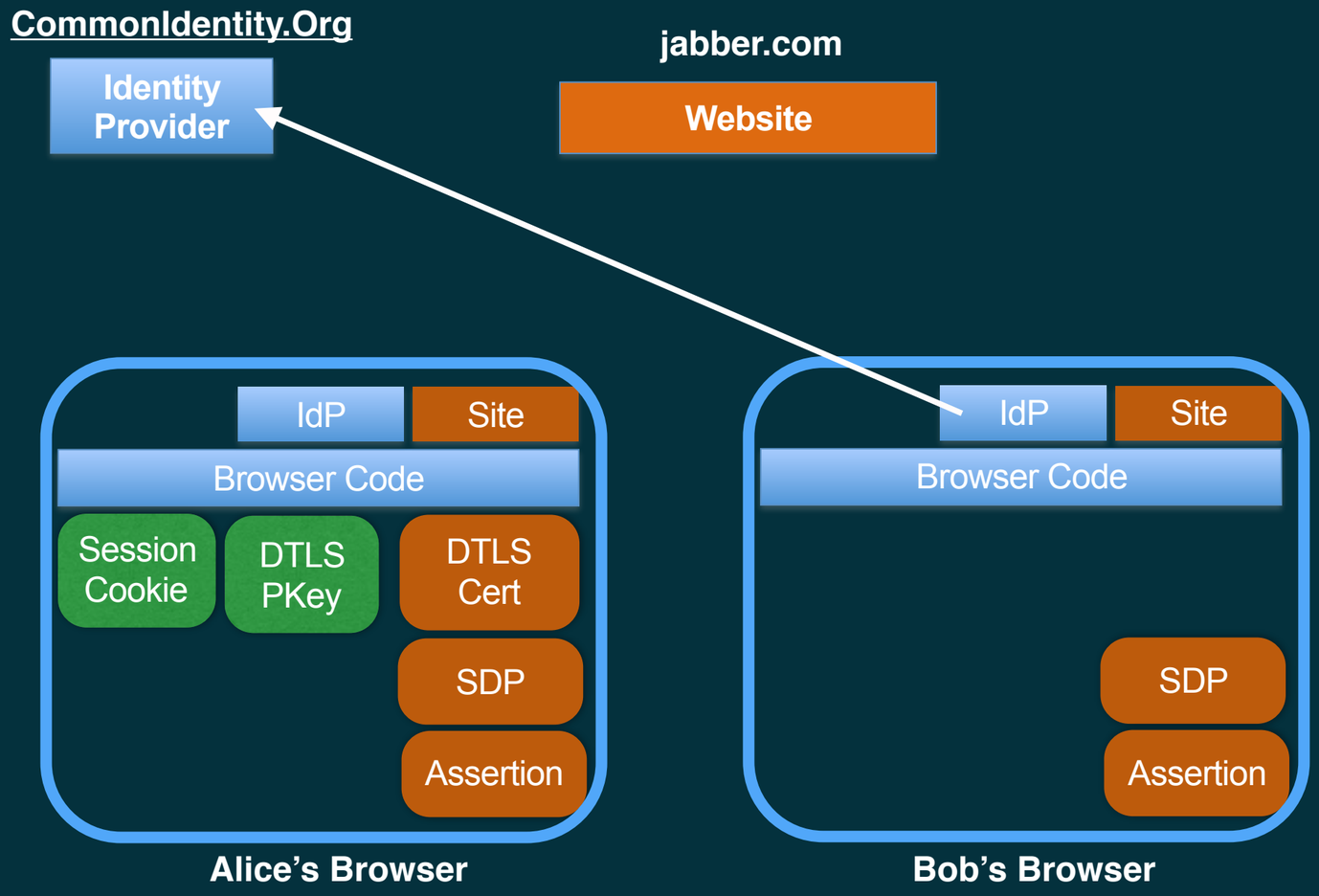
Website



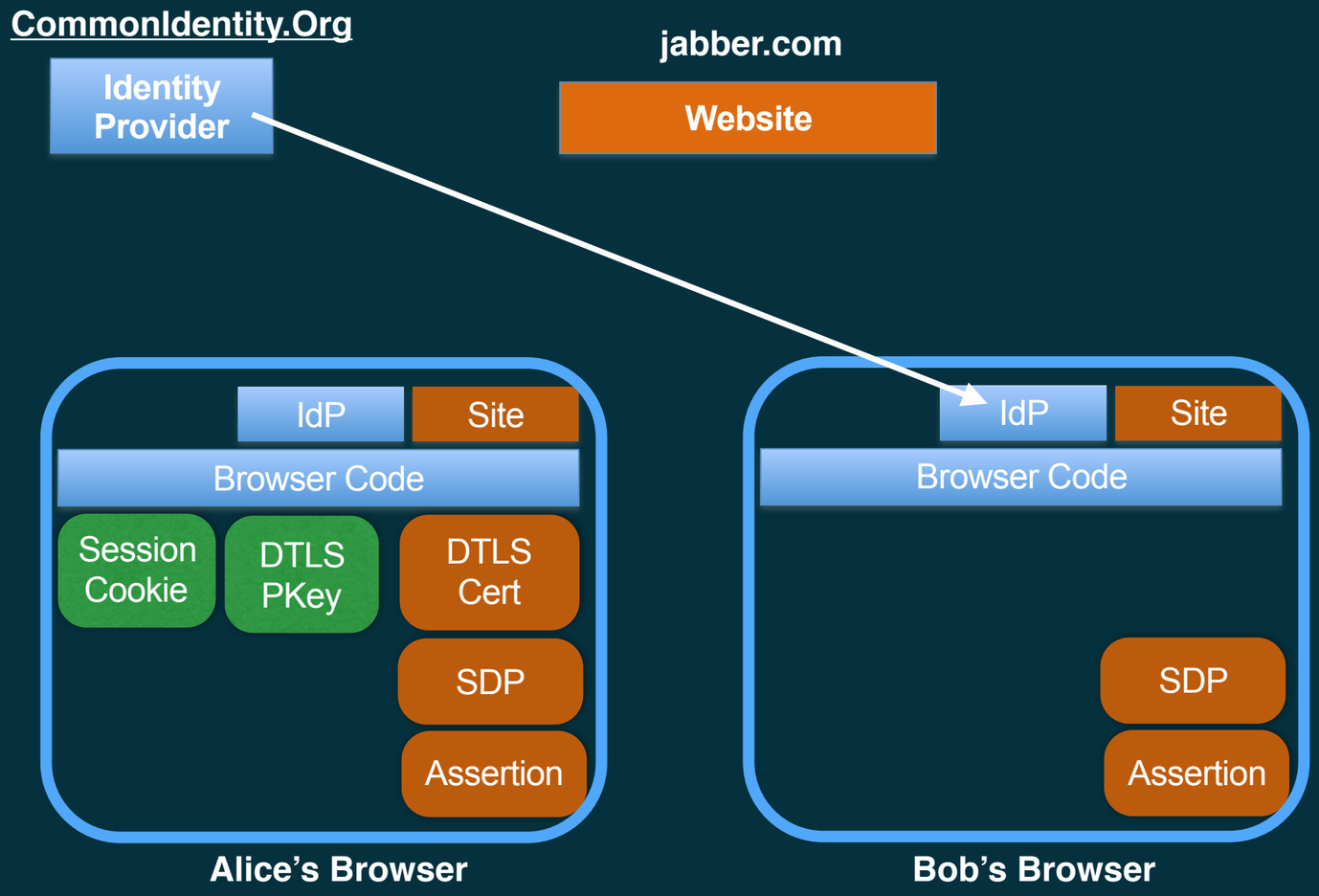
The code:

```
idp.validateAssertion = function( assertion, origin )
     var p = new Promise(
          function( resolve, reject ) {
               …
          } );
    return p;
}
```

IdP passes to Identity Provider to check signature



Identity Provider says it is OK



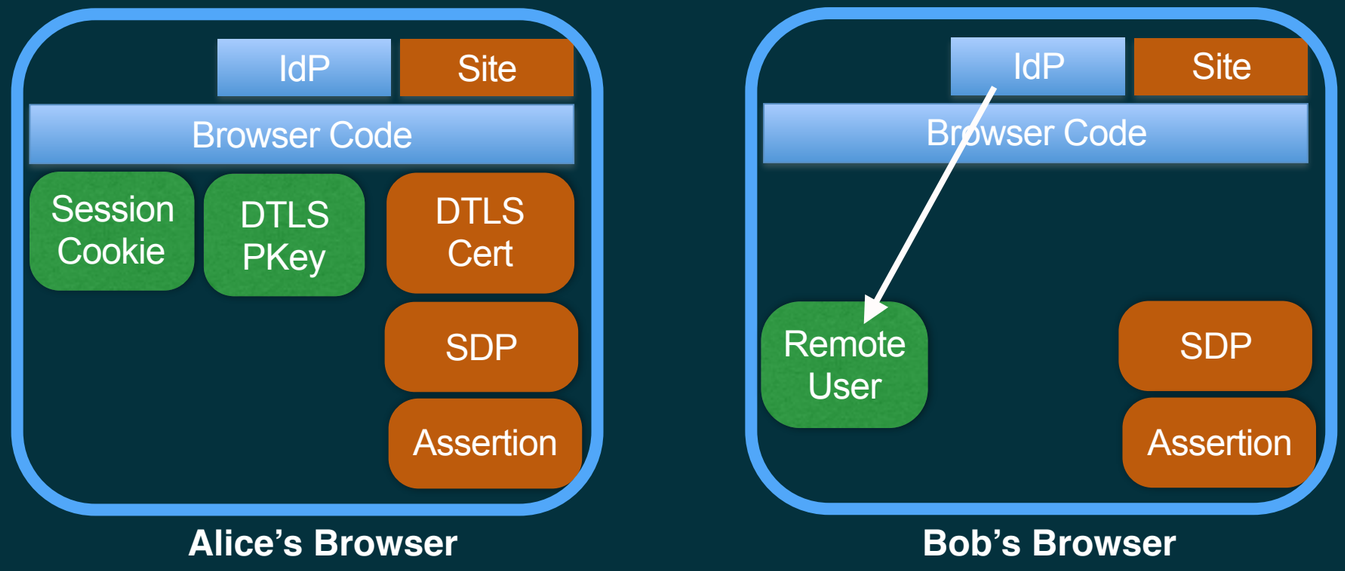
IdP extracts fingerprint and tells browser that the assertion is valid and user name it is from

CommonIdentity.Org

Identity Provider

jabber.com

Website



The code:

```
idp.validateAssertion = function( assertion, origin )
     var p = new Promise(
          function( resolve, reject ) {
               var validationResult= {};
               validationResult.identity = "fluffy";
               validationResult.contents = … ;
               resolve( validationResult );
          } );
     return p;
}
```

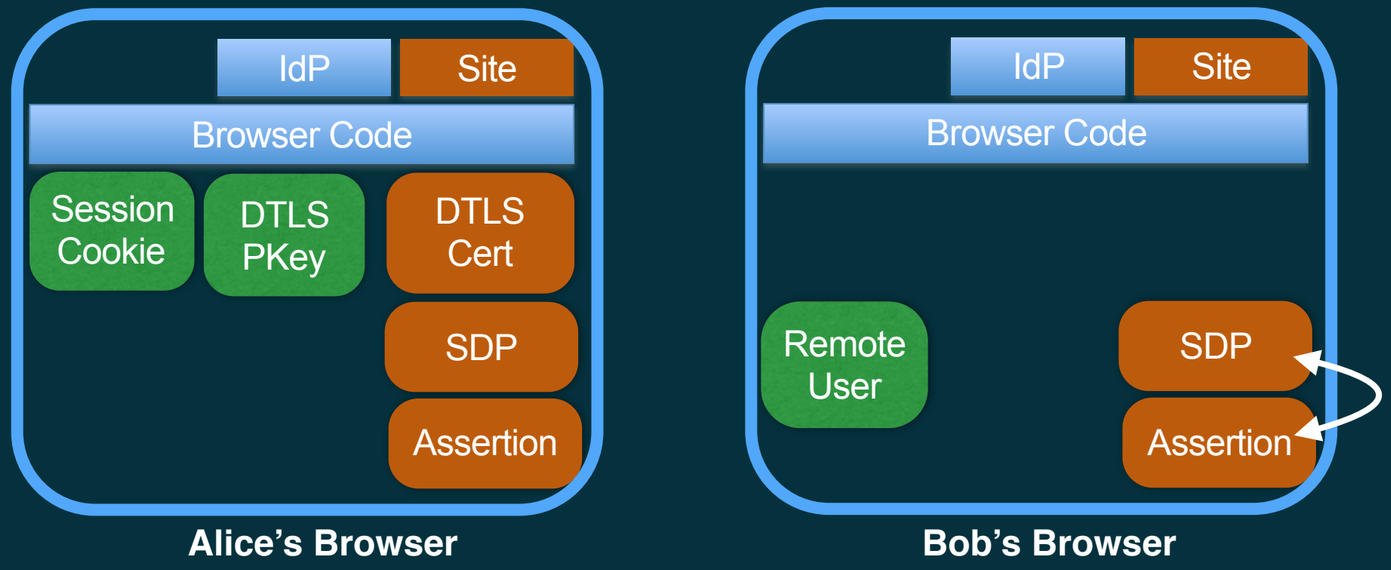
Browser checks the fingerprint in assertion matches fingerprint in SDP

CommonIdentity.Org

Identity Provider

jabber.com

Website



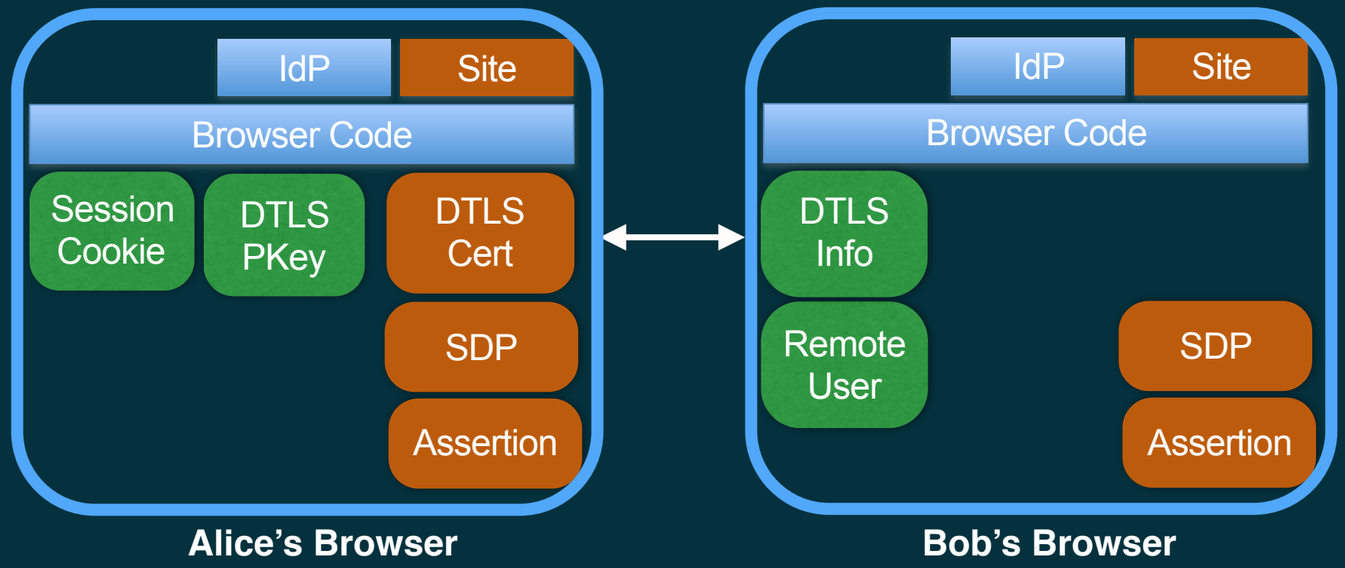
Browser sets up DTLS connection other browser

CommonIdentity.Org

Identity Provider

jabber.com

Website



Media Keying - DTLS

- DTLS is simply the same TLS used for HTTPS adapted for UDP
- DTLS handshake is used to form the session keying material for the SRTP media encryption
- Used with self signed certificates. Each certificate has a fingerprint which is bound to a user identity in a way described later in this presentation

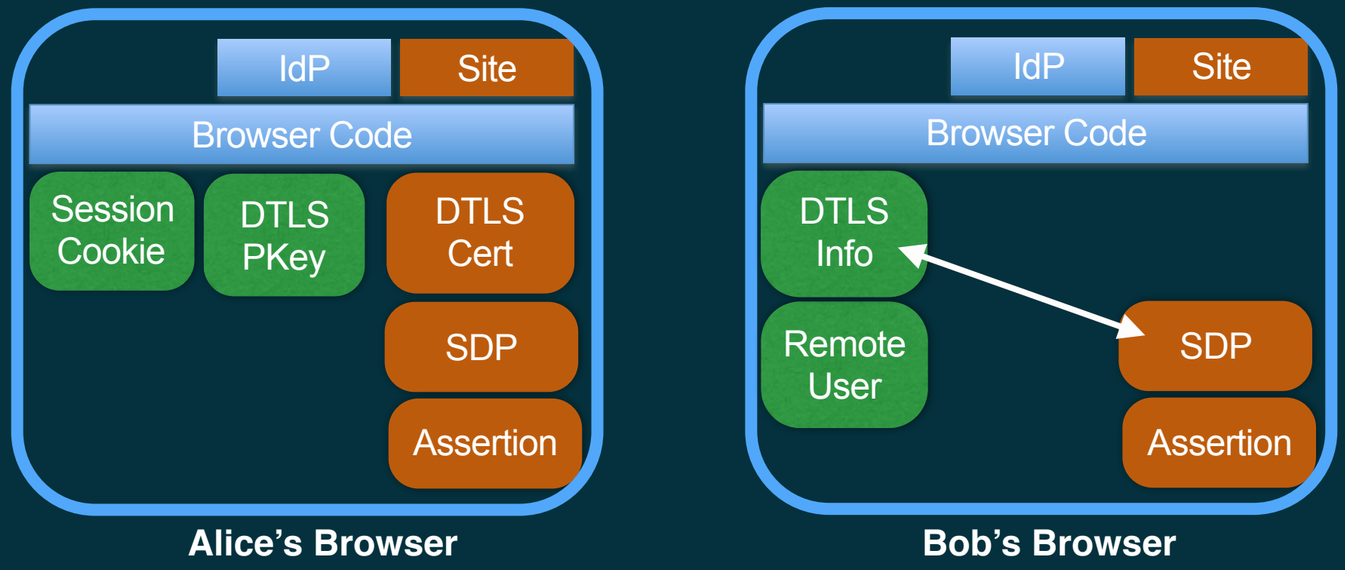
Bob’s browser authenticates remote DTLS cert by checking that it matches fingerprint in SDP

CommonIdentity.Org

Identity Provider

jabber.com

Website



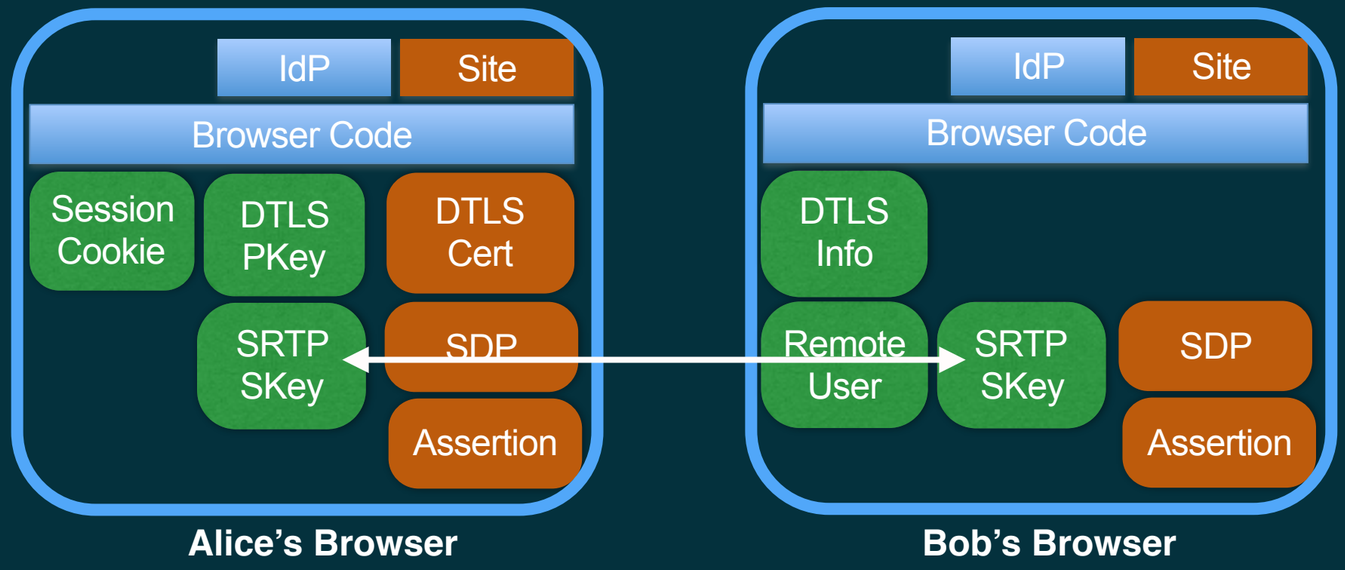
Use DTLS to form the SRTP key

CommonIdentity.Org

Identity Provider

jabber.com

Website



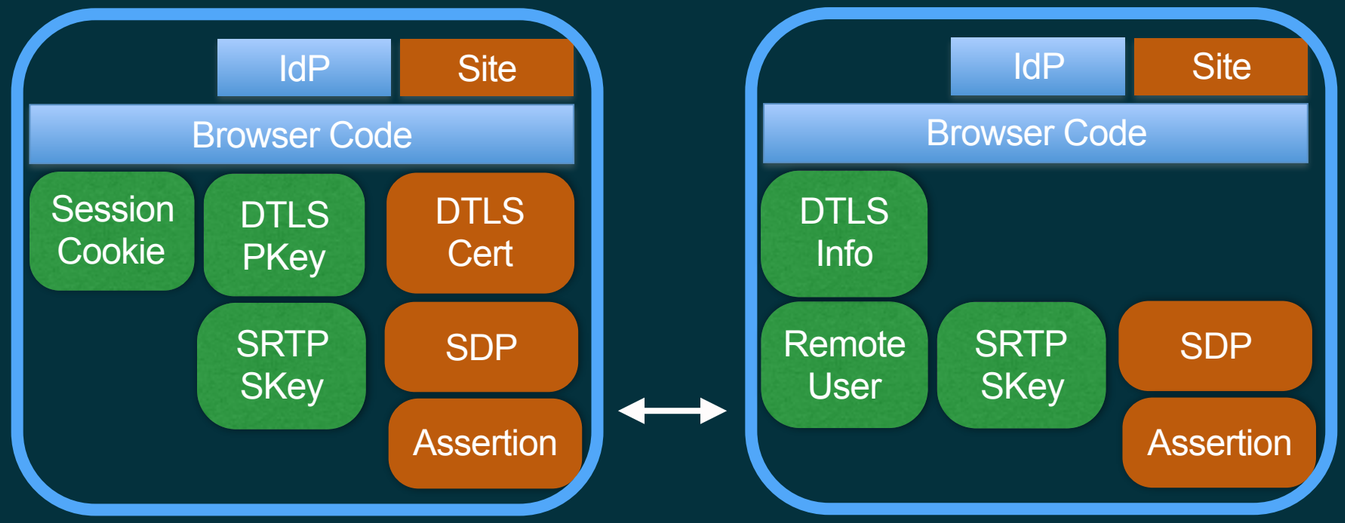
Send media encrypted with SRTP

CommonIdentity.Org

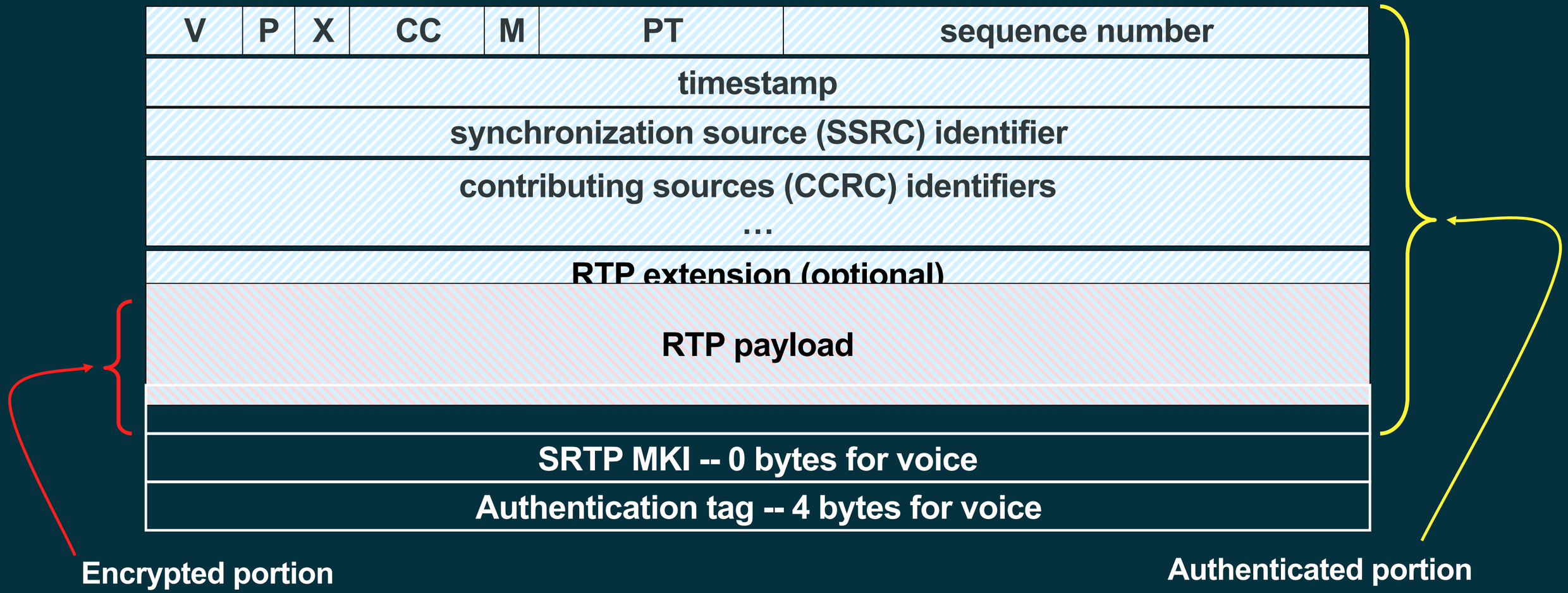
Identity Provider

jabber.com

Website



SRTP:
Media Encryption and Authentication



Browser UI can display remote username and identity provider in part of UI that site can not change

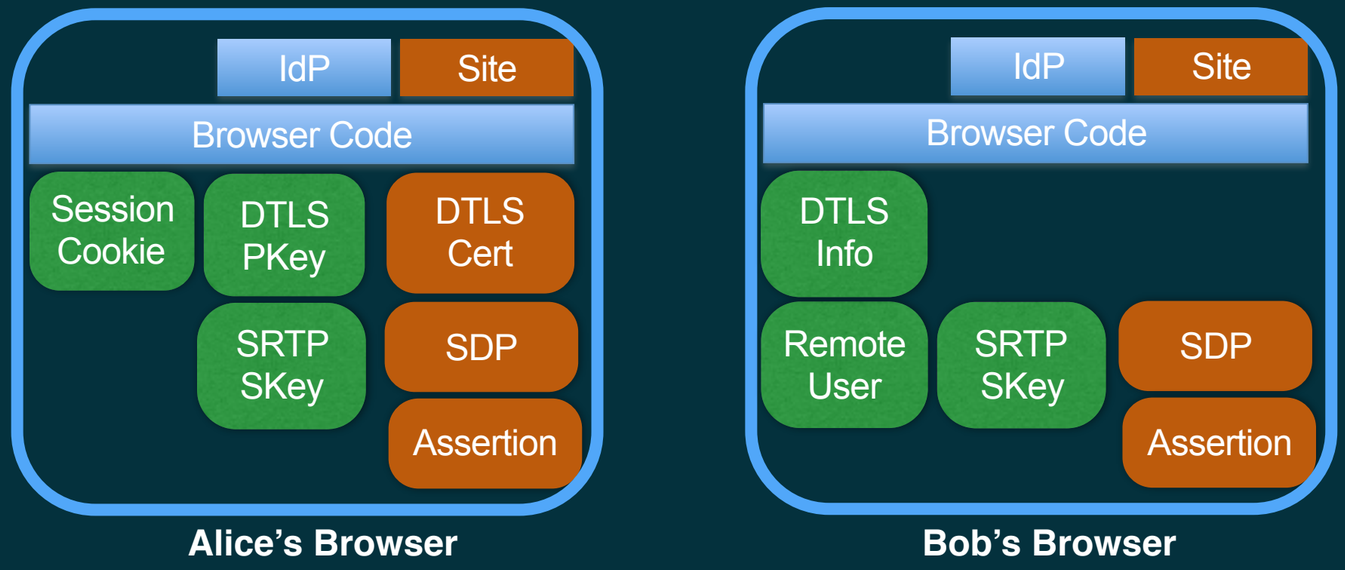
CommonIdentity.Org

Identity Provider

jabber.com

Website

1. Browser display "fluffy@CommonIdentity.org"



Roughly the same happens in opposite direction

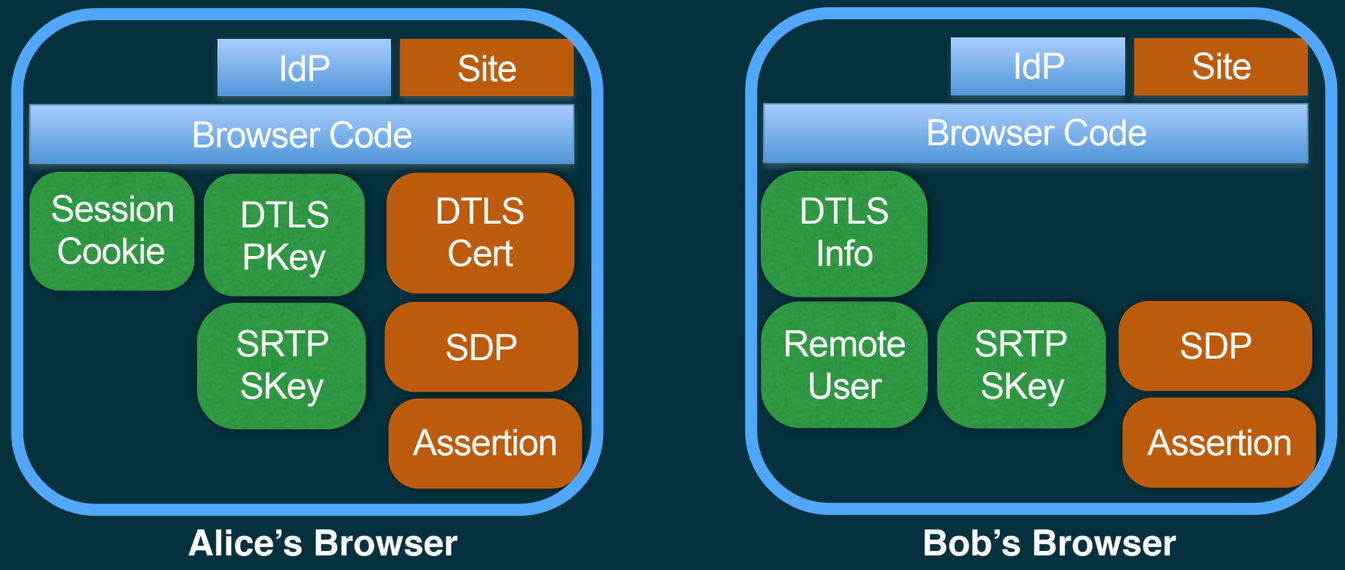
CommonIdentity.Org

Identity Provider

jabber.com

Website

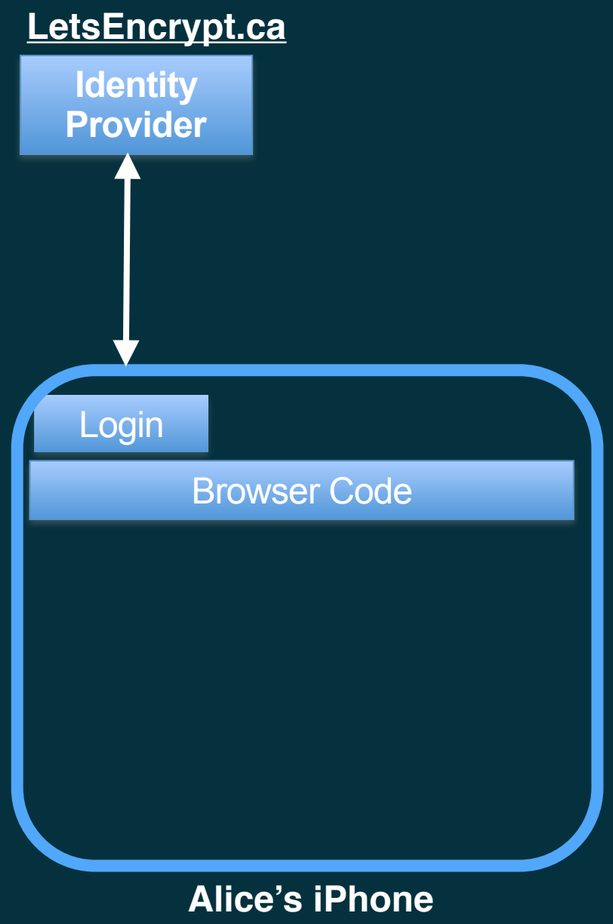
1. A similar flows happens in opposite direction to allow Alice to authenticated media is from Bob



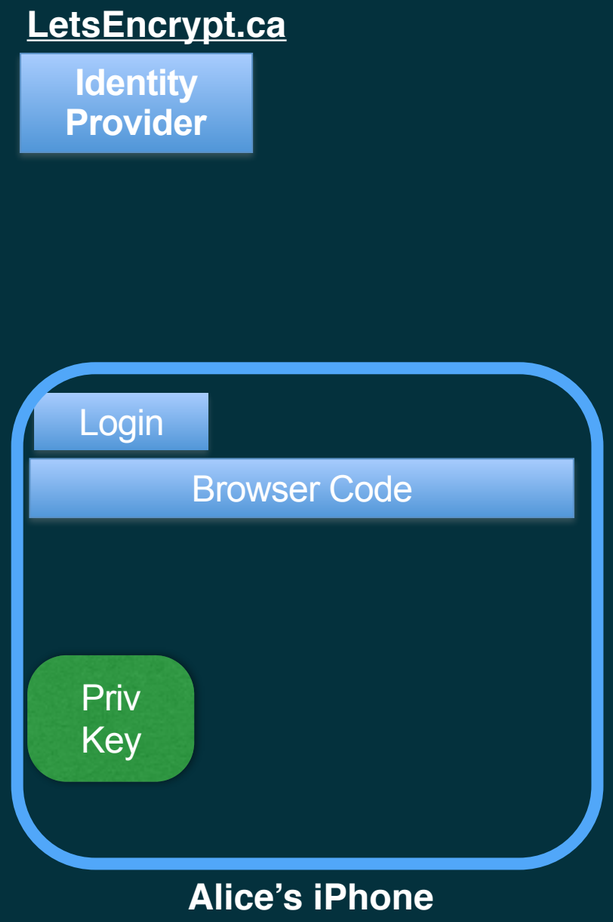
Something a little more complicated

Certificate based IdP

User sets up account for +1 408 421 9990 on LetsEncrypt.ca

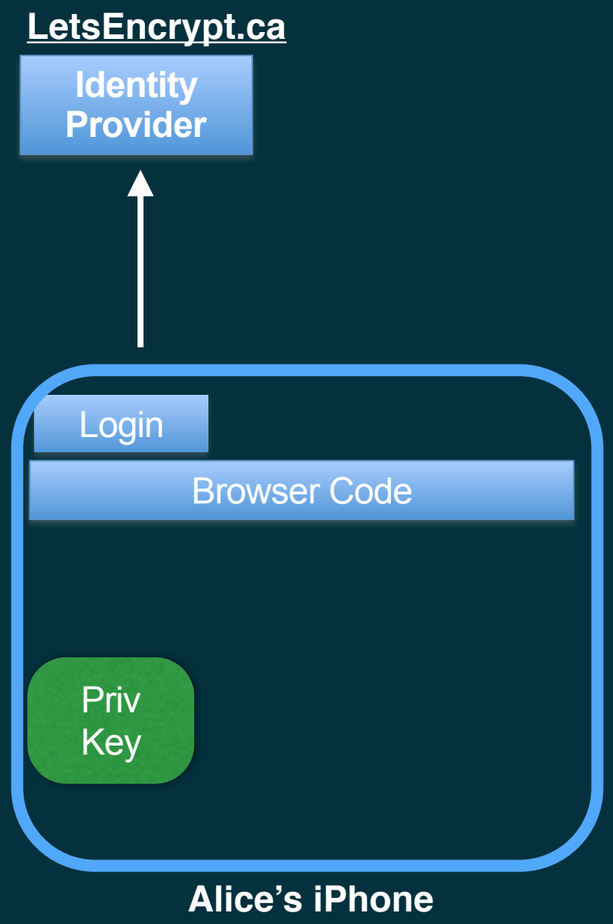


Browser generates public/private key pair

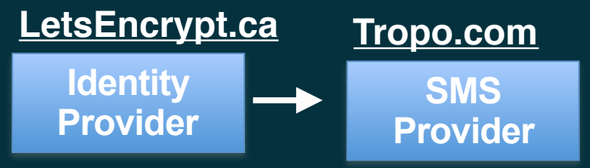


1. Use WebCrypto API to do this
2. private key is marked as non exportable so the JS can never get it
3. private key is stored in indexDB so it persists over sessions

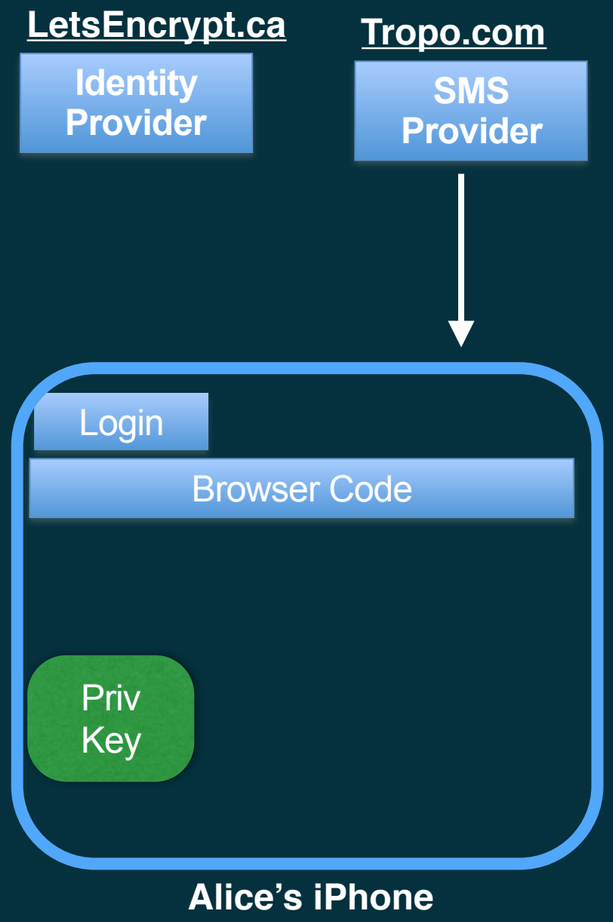
Browser creates Certificate Signing Request for cert with tel:+1-408-421-9990 and sends to LetsEncrypt



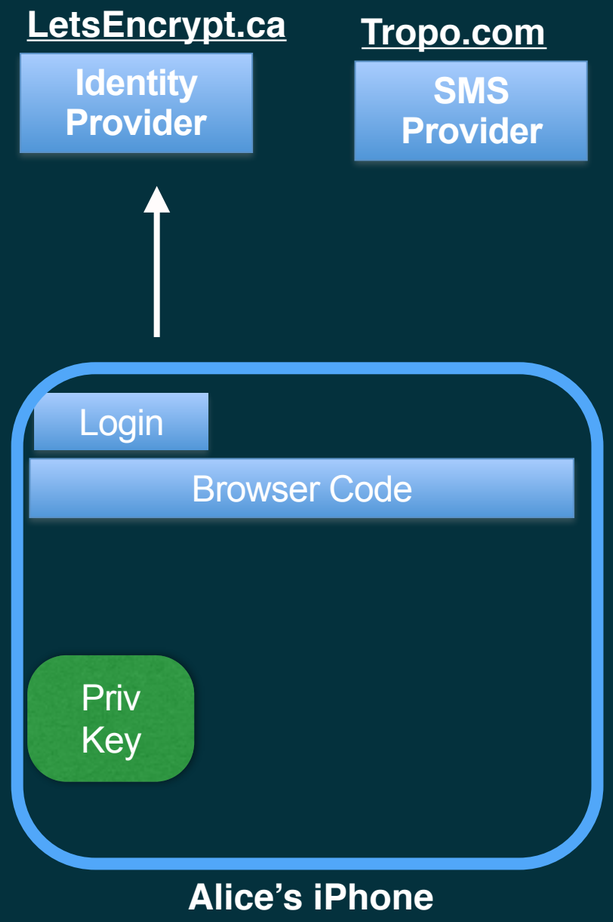
LetsEncrypt tells Tropo to send a SMS to that number with random 8 digit code



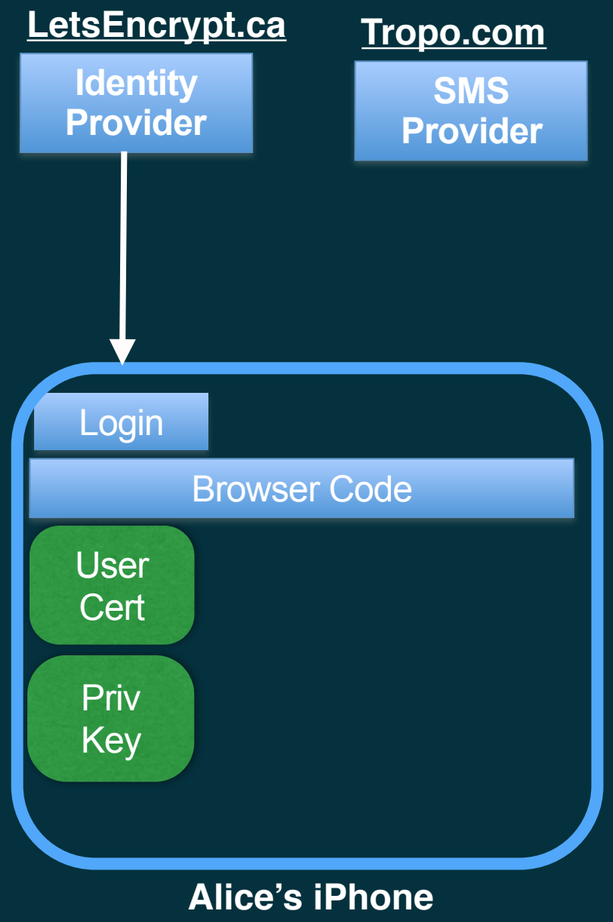
Tropo sends SMS to +1 408 421 9990



Phone recognizes SMS as for LetsEncrypt and provides code to LetsEncrypt (or user enters it in browser screen)



LetsEncrypt issues the certificate and passed back to Alice's phone



1. User only has to do this once - or if cert expires
2. If Alice has multiple phones, this needs to be repeated because the private key for this cert stays on this phone.

Bob goes to call +1 408 421 9990

LetsEncrypt.ca

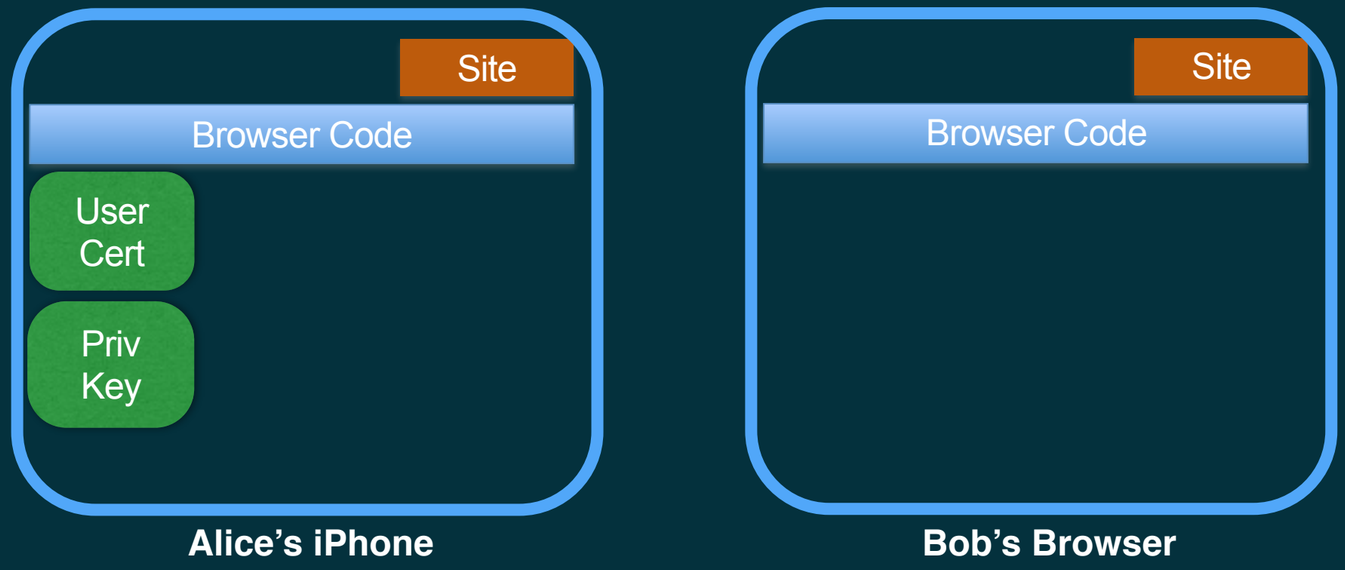
Identity Provider

Tropo.com

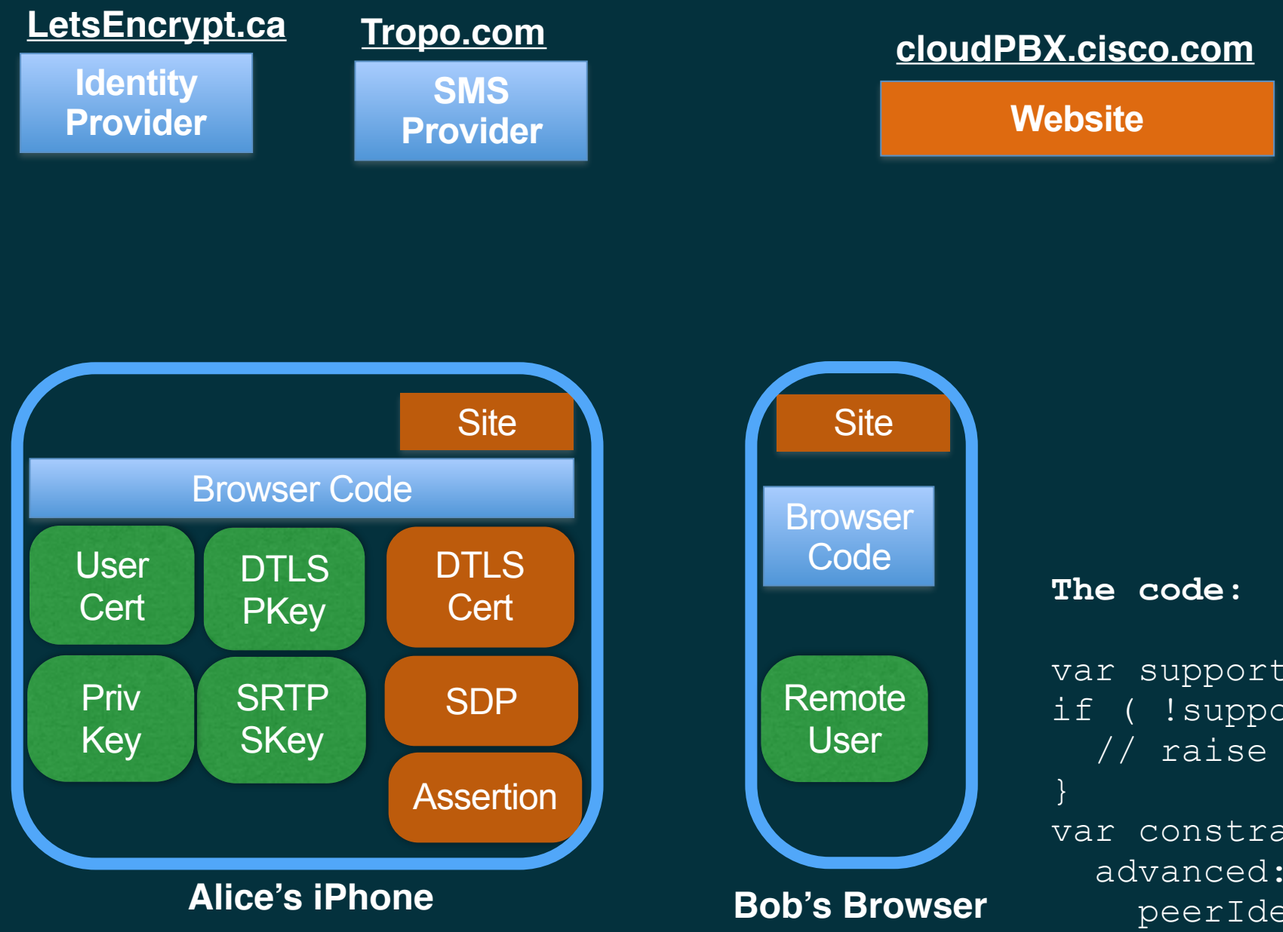
SMS Provider

cloudPBX.cisco.com

Website



Browser ask permissions to send video to that number

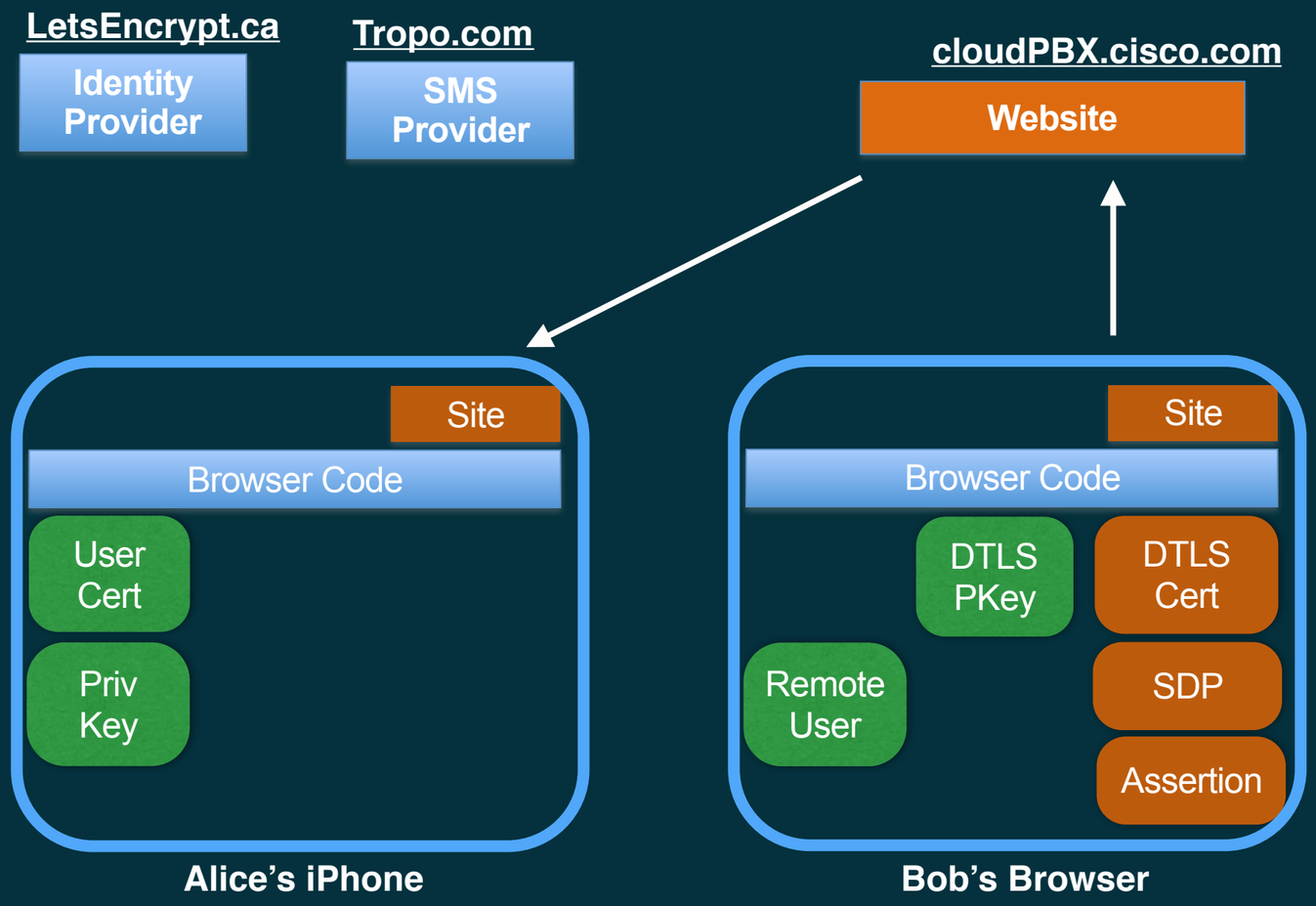


1. You can find out more about isolated video streams at http://w3c.github.io/webrtc-pc/#isolated-media-streams
2. Browser permission prompt will include the identity we want to send video to
3. JavaScript will not be able to access media
4. If media is rendered on "local" canvas, it is tainted so that Javascript can not capture it

The code:

```
var supports = navigator.mediaDevices.getSupportedConstraints();
if ( !supports["peerIdentity"] ) {
  // raise error
}
var constraints = {
  advanced: [{
    peerIdentity: "tel:+14084219990"
  }]
};
var promise = navigator.mediaDevices.getUserMedia( constraints );
```

SDP Offer send from Bob to Alice



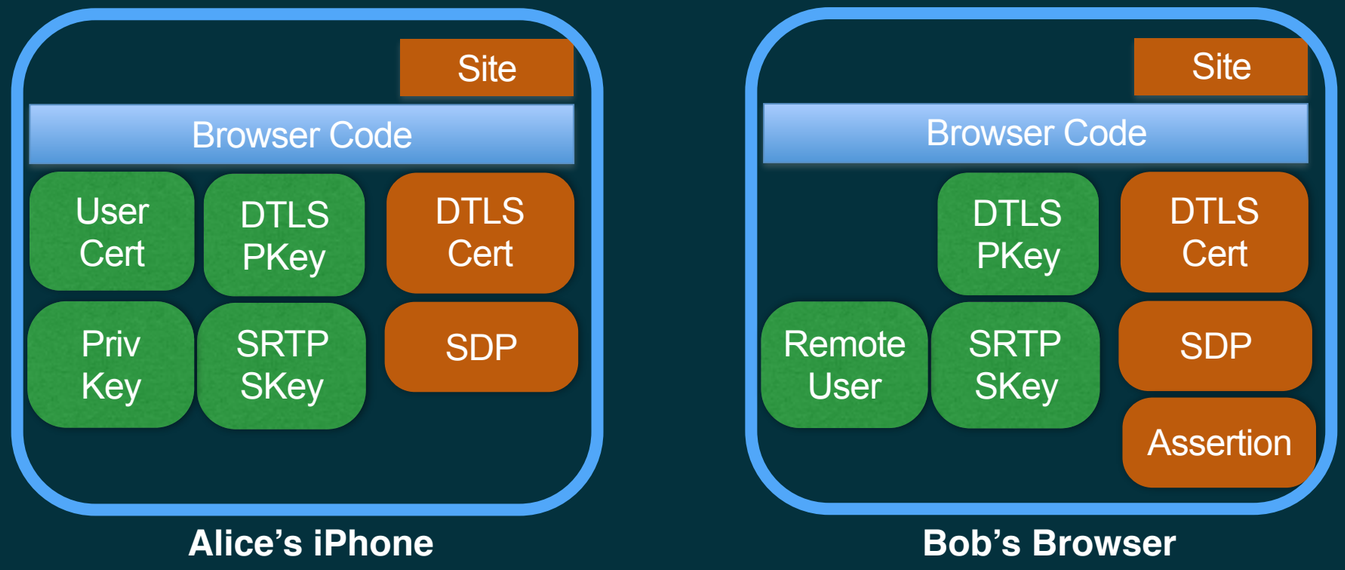
Alice creates SDP Answer with fingerprint

LetsEncrypt.ca
Identity Provider

Tropo.com
SMS Provider

cloudPBX.cisco.com
Website

1. Site creates peerConnection
2. browser creates DTLS certs and key
3. browser creates fingerprint for SDP



LetsEncrypt is used as IdP

LetsEncrypt.ca

Identity Provider

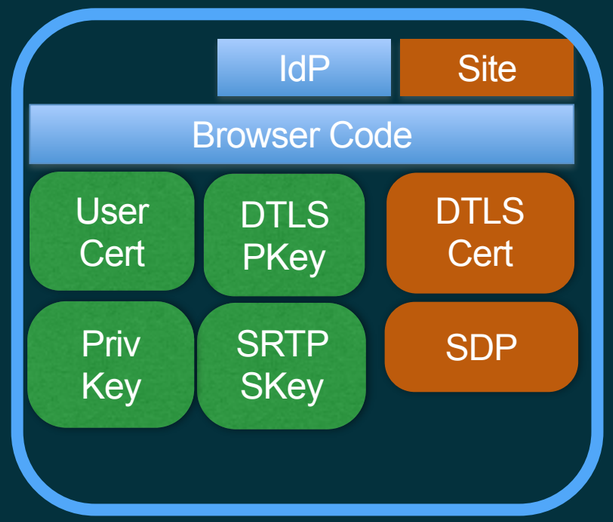
Tropo.com

SMS Provider

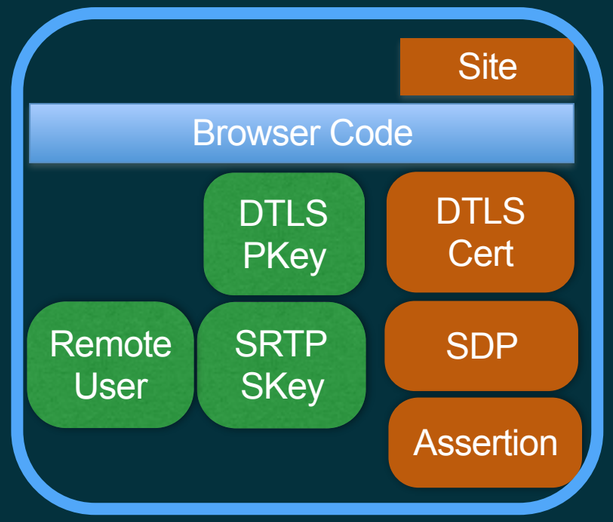
cloudPBX.cisco.com

Website

1. The IdP JS code can be loaded once and cached for long time



Alice's iPhone



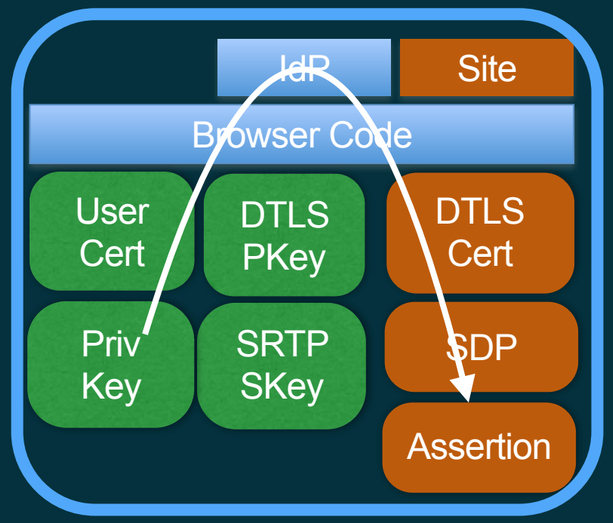
Bob's Browser

The IdP code signs assertion by using the Cert

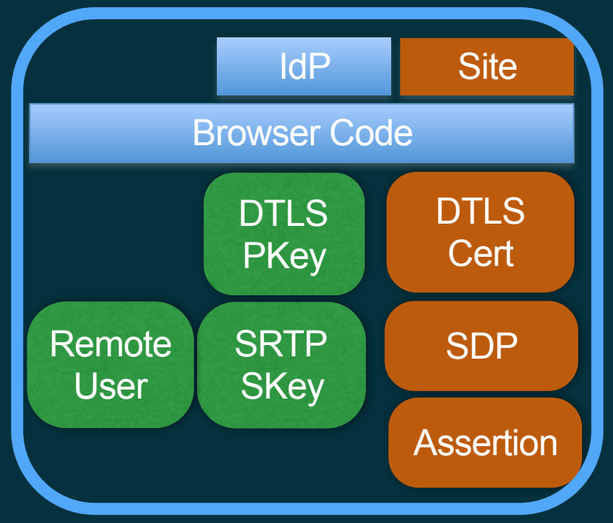
LetsEncrypt.ca
Identity Provider

Tropo.com
SMS Provider

cloudPBX.cisco.com
Website



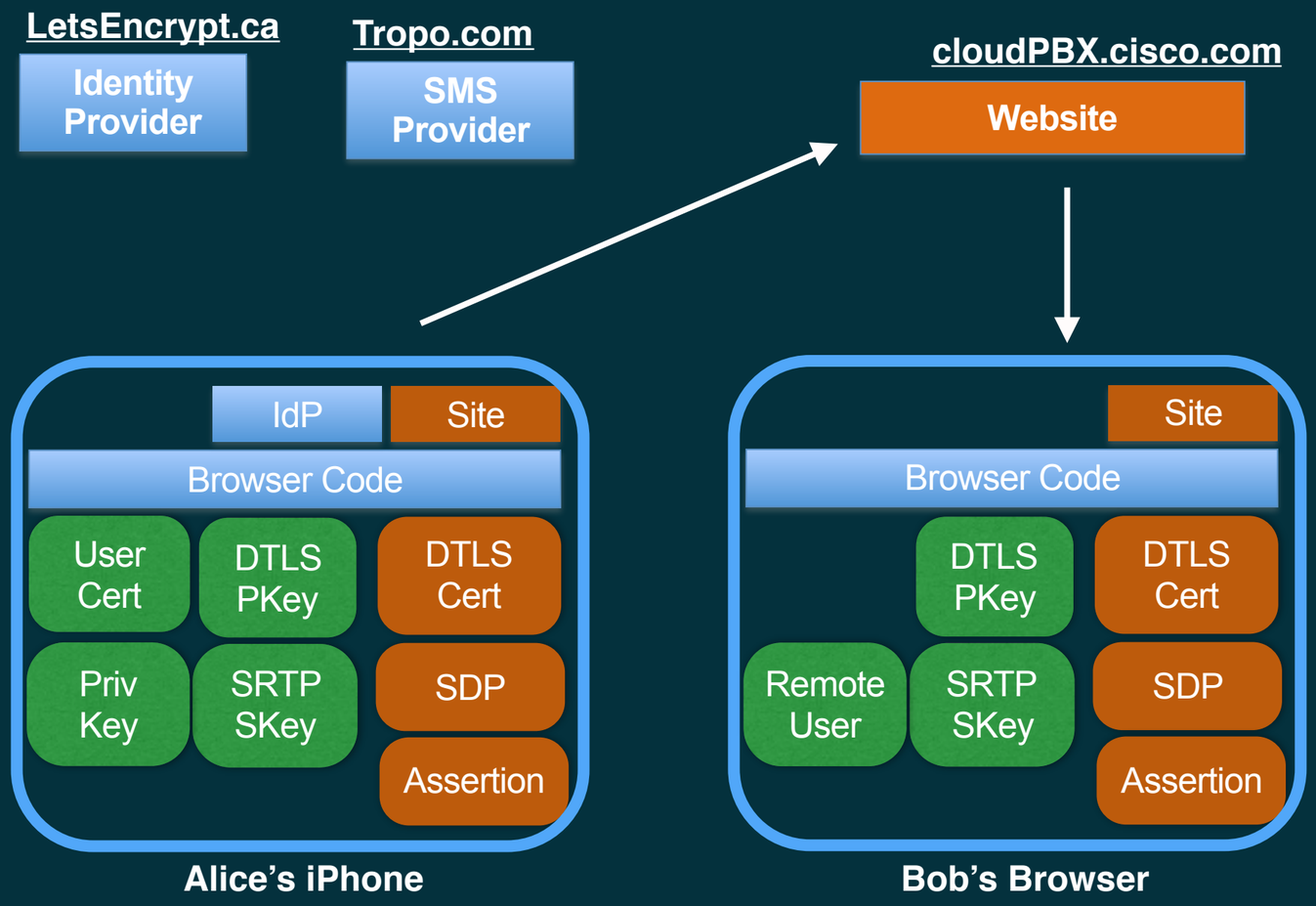
Alice's iPhone



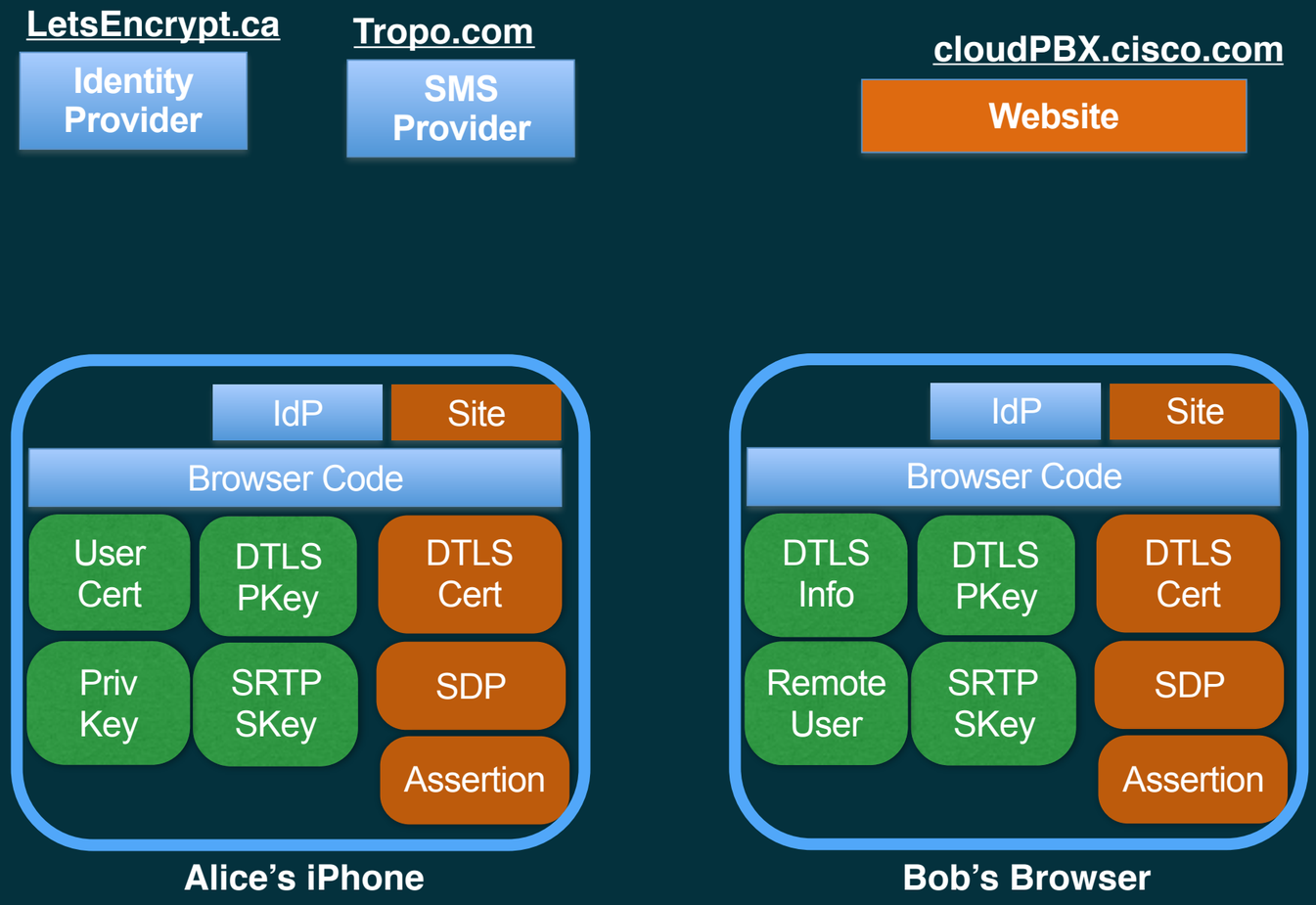
Bob's Browser

1. Site does normal creation of PeerConnection creating the DTLS cert and key
2. Site creates an answer which forms fingerprint in SDP and asks IdP to create an assertion
3. IdP code does not need to make any REST calls to LetsEncypt. It simply uses the user cert and private key to sign the assertion
4. The user name is effectively +1-408-421-9990@LetsEncyrpt.ca from an IdP point of view

SDP Answer is sent to Bob

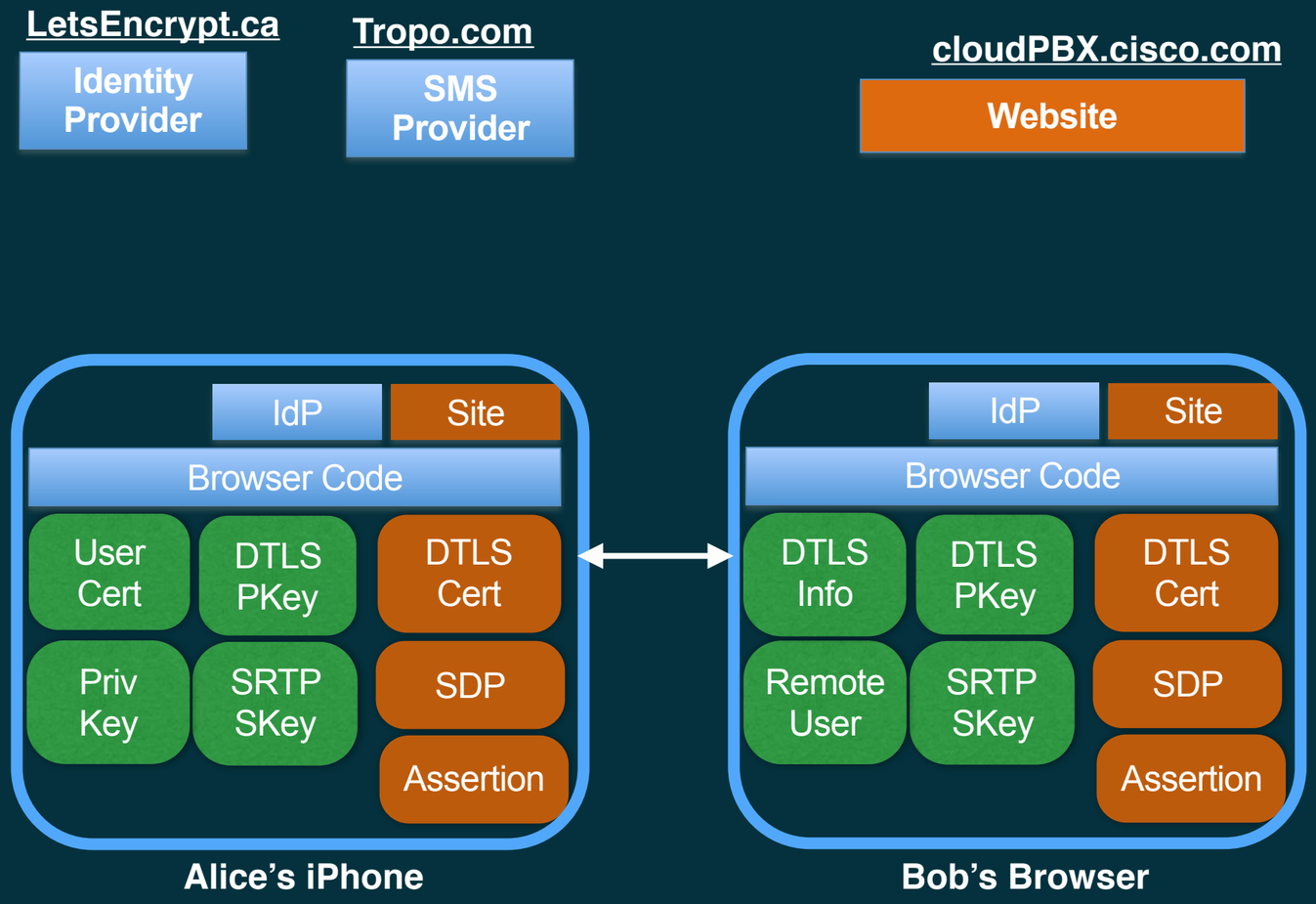


Bob’s browser loads the IdP and validates the assertion



1. The IdP code can be cached using offline style. Bob’s browser does not even need access to the public internet at this stage
2. The assertion is checked by validating that the assertion was signed by a cert issued by LetsEncrypt.ca (using LetsEncrypt’s root cert) AND that the tel URL name in the cert matches the claimed name for the call
3. Bob’s browser knows the calls if from "+1-408-421-9990 @ LetsEncrypt.ca"

Bob’s browser check media is from right person



1. Bob’s browser knows the calls if from "+1-408-421-9990 @ LetsEncrypt.ca"
2. The browser has "LetsEncypt" on a root trust list for phone numbers so it drops the @ part when displaying it
3. The browser chrome displays the call as from "+1 408 421 9990" or if it has access to the local address book replaces it with the name of that user if the number is in the address book

Java Script does not have access to media

LetsEncrypt.ca

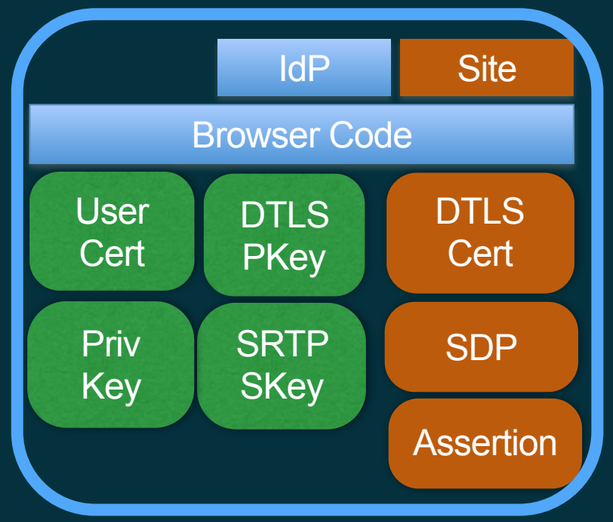
Identity Provider

Tropo.com

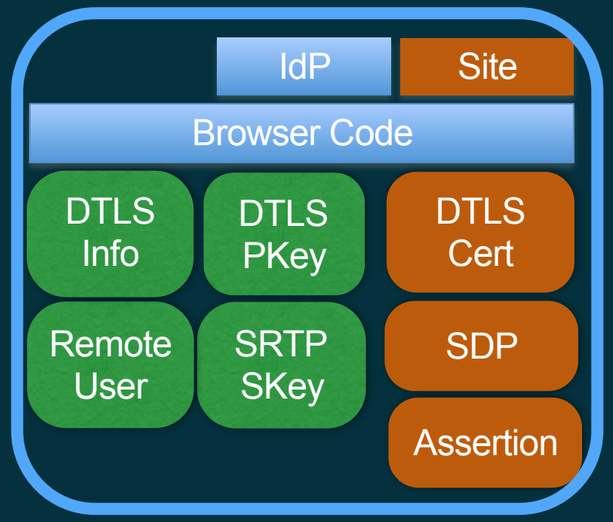
SMS Provider

cloudPBX.cisco.com

Website



Alice's iPhone



Bob's Browser

1. The DTLS connection is negotiated with an ALPN set to c-webrtc (instead of the normal webrtc)
2. This tells the other browser to taint the media stream and not allow Javascript access to it

What the user gets

- Strong encrypted media where you know who the media if from and to
- Federated identity
- Assurance that media was not intercepted by JavaScript of Cloud PBX
- You do need to trust you browser, operating system, LetsEncrypt, and whoever can intercept SMS routed to you
- Certificate transparency can allow you detect bad behavior by whoever can intercepts SMS routed to you

Thank you.