

RTCWEB Architecture Open Issues

Interim Meeting; February 2012

Eric Rescorla

`ekr@rtfm.com`

Overview

- Security architecture document adopted after Taipei
 - `draft-ietf-rtcweb-security-arch-00`
- General agreement on a lot of issues
- Purpose of the next 30 min
 - Survey the open issues
 - Resolve any that are easy

Issue: Mixed Content

- Consent is granted by origin
- What about active mixed content?
 - `https://www.example.com/` loads script from `http://www.example.com`
 - What are the `PeerConnection` permissions
- Current draft says: treat page as the HTTP origin
 - Browser security experts: **“NOOOOOO!!!!!!!!!!”**

How Browsers Handle Active Mixed Content Now

Browser	Action
Chrome	Allow with warning – (soon to be block)
Firefox	Big warning dialog
IE	Block
Safari	Accept

Proposed Resolution

- MUST treat HTTP and HTTPS origins as separate [uncontroversial]
- SHOULD * either:
 - Forbid all active mixed content [better, but out of scope]
 - Remove RTCWEB permissions for origins with mixed content
- Comments?

*Should this be a MUST?

Issue: Consent Freshness/Keepalives

- Problem: How to verify continuing consent?
 - Need some sort of keepalive
 - ICE keepalives are STUN Binding Indications (one-way)
- Proposal: use STUN Binding Requests instead
 - MUST check no less often than every 30s
- Comments?

Issue: Media Security Requirements

- DTLS/DTLS-SRTP provides the best security
 - Can detect MITM with fingerprint checks (though inconvenient)
 - Strong authentication when used with third-party IdP
- Demand for SDES, RTP, or both
 - Mostly in terms of interop with legacy systems w/o media gatewaying
 - Concerns about bid-down attacks, UI confusion, etc.

Interaction with Server-based features

- End-to-end security requires denying the server direct media access
 - Via MediaStream APIs [See Randell Jesup's slides]
 - This precludes many fancy video applications
- Need for two security models
 - Browser-to-browser secure.* Limited effects but media protected from JS
 - Javascript-visible.* Powerful effects but need to trust JS
 - This should be under control of the JavaScript but with a UI indicator
- End-to-end crypto still adds value in server-visible model
 - Protection against programmer error (excess logging, XSS, etc.)
 - Malicious activity more readily detectable

Interop Deployment Questions

- Everyone supports RTP
 - But obviously security is... bad
- Most *current* implementations support SDES
 - Unclear (at least to me) how many deployments support it
- Decision proposal:
 - Need RTP if not much SDES deployment
 - If a lot of SDES deployment, not much need for RTP

Communications Security: Implementation Requirements (Proposed)

- MUST implement DTLS-SRTP (for media) and DTLS (for data)
- MAY implement RTP(?) and SDES(?)
- Security MUST be default state
 - Implementations MUST offer DTLS and/or DTLS-SRTP for every channel
 - MUST accept DTLS and/or DTLS-SRTP whenever offered
 - MUST not do unencrypted data channel

RTCWEB Generic Identity Service

Interim Meeting; February 2012

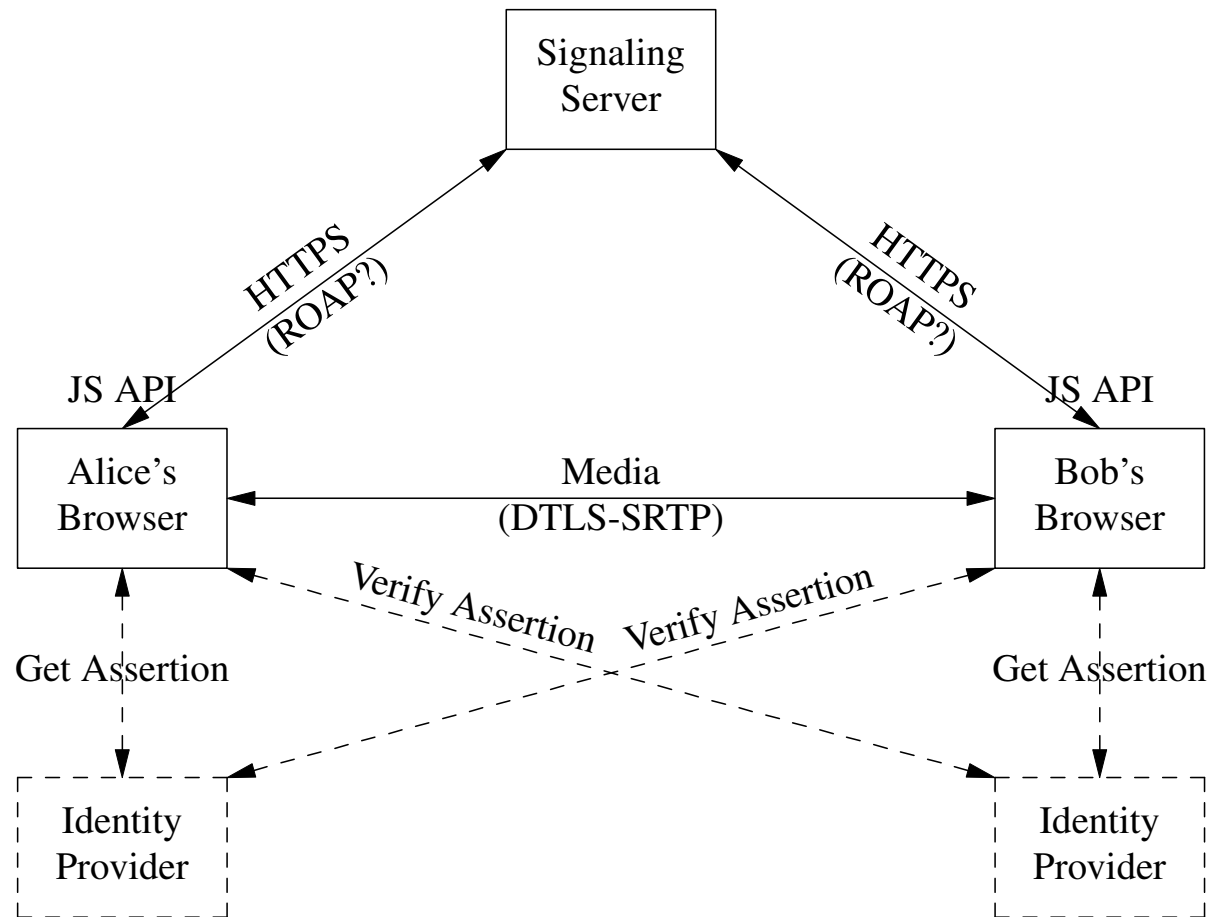
Eric Rescorla

ekr@rtfm.com

What are we trying to accomplish?

- Allow Alice and Bob to have a secure call
 - Authenticated with their identity providers
 - On any site
 - * Even untrusted/partially trusted ones
- Advantages
 - Use one identity on any calling site
 - Security against active attack by calling site
 - Support for federated cases

Topology



Terminology

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.

Types of IdP

Authoritative: Attests for identities within their own namespace

- Often multiple Authoritatives IdPs exist with different scopes
- Examples: DNSSEC, RFC 4474, Facebook Connect (for the Facebook ID)

Third-party: Attests for identities in a name-space they don't control

- Often multiple Third-Party IdPs share the same space
- Can attest to real-world identities
- Examples: SSL/TLS certificates, the State of California (driver's licenses)

Authoritative vs. Third-Party IdPs: Trust Relationship

- No need to explicitly trust authoritative IdPs
 - `ekr@example.com` is whoever `example.com` says it is
 - The problem is authenticating `example.com`
- Third-party IdPs need to be explicitly trusted
 - Example: how do I know GoDaddy is a legitimate CA?
 - Answer: the browser manufacturer vetted them
 - They are allowed to attest to *any* domain name

User Relationships with IdPs

- Authenticating Party
 - Has some account with the IdP
 - May have established their identity
 - * Especially for third-party IdPs
 - Can authenticate to the IdP in the future (e.g., with a password)
- Relying party
 - Doesn't have any account relationship with the IdP*
 - Must be able to verify the IdP's identity
 - Needs to trust third-party IdPs

*Note: privacy issues.

Web-based IdP Systems

- Facebook Connect
- Google login
- OAuth
- OpenID
- BrowserID

Web-based IdP Objectives: User Perspective

- Single-sign on
 - No need to make a new account for each service
 - Don't need to remember lots of passwords
- Privacy
 - Avoid creating a super-cookie
 - * Only authenticate to sites I have approved
 - * Control exposure of my personal information

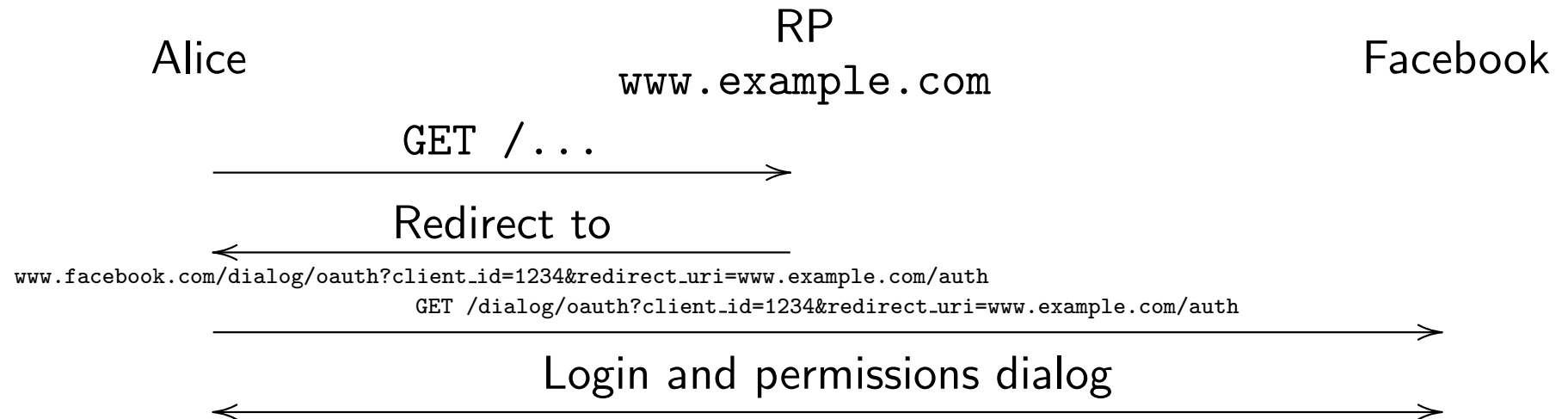
Web-based IdP Objectives: Site Perspective

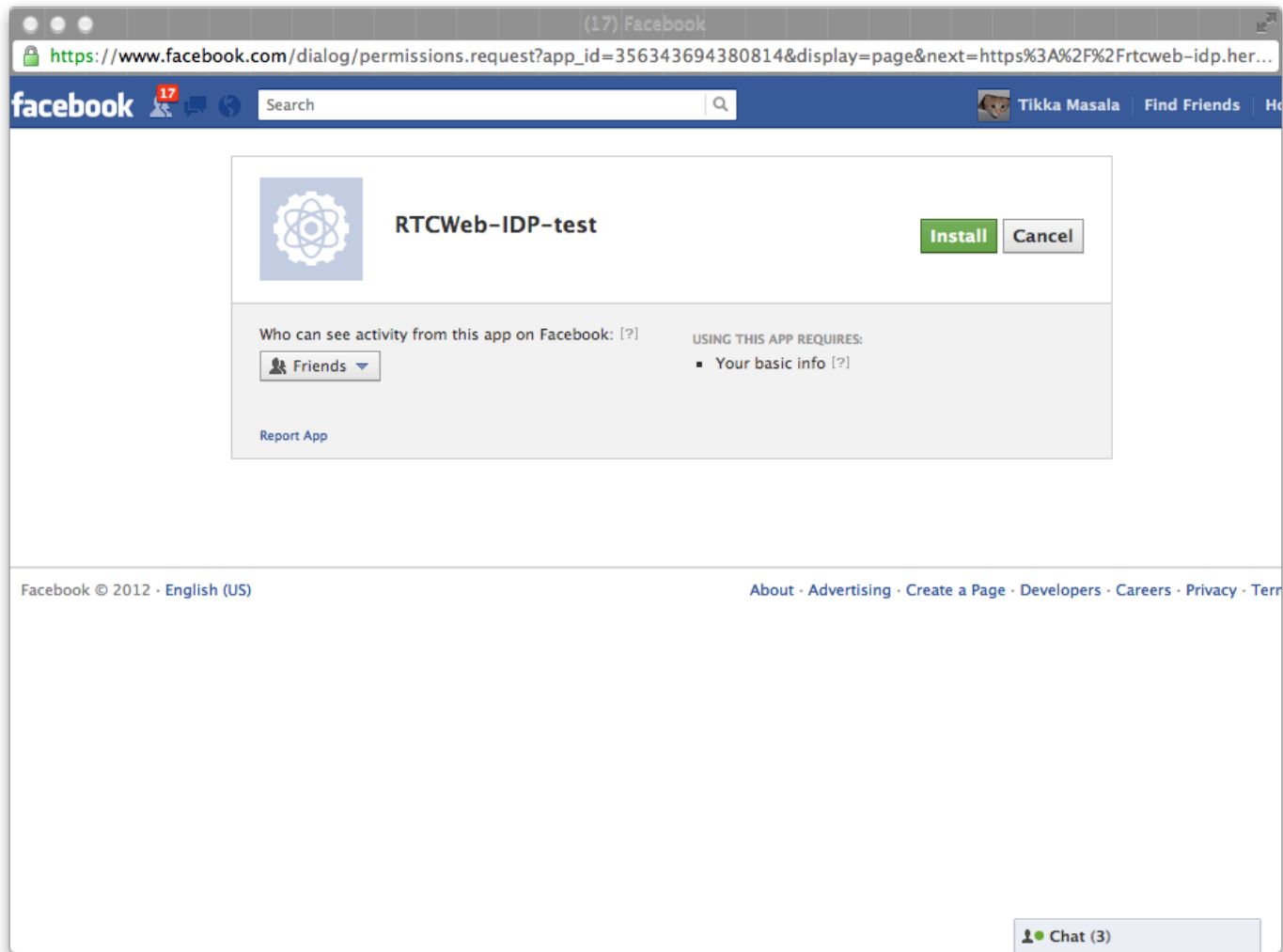
- Low friction
 - Avoid the need for account creation
 - ... the source of a lot of user rolloff
- Leverage existing user information
 - E.g., information you've stored in your FB account

Example: Facebook Connect (sorta OAuth)

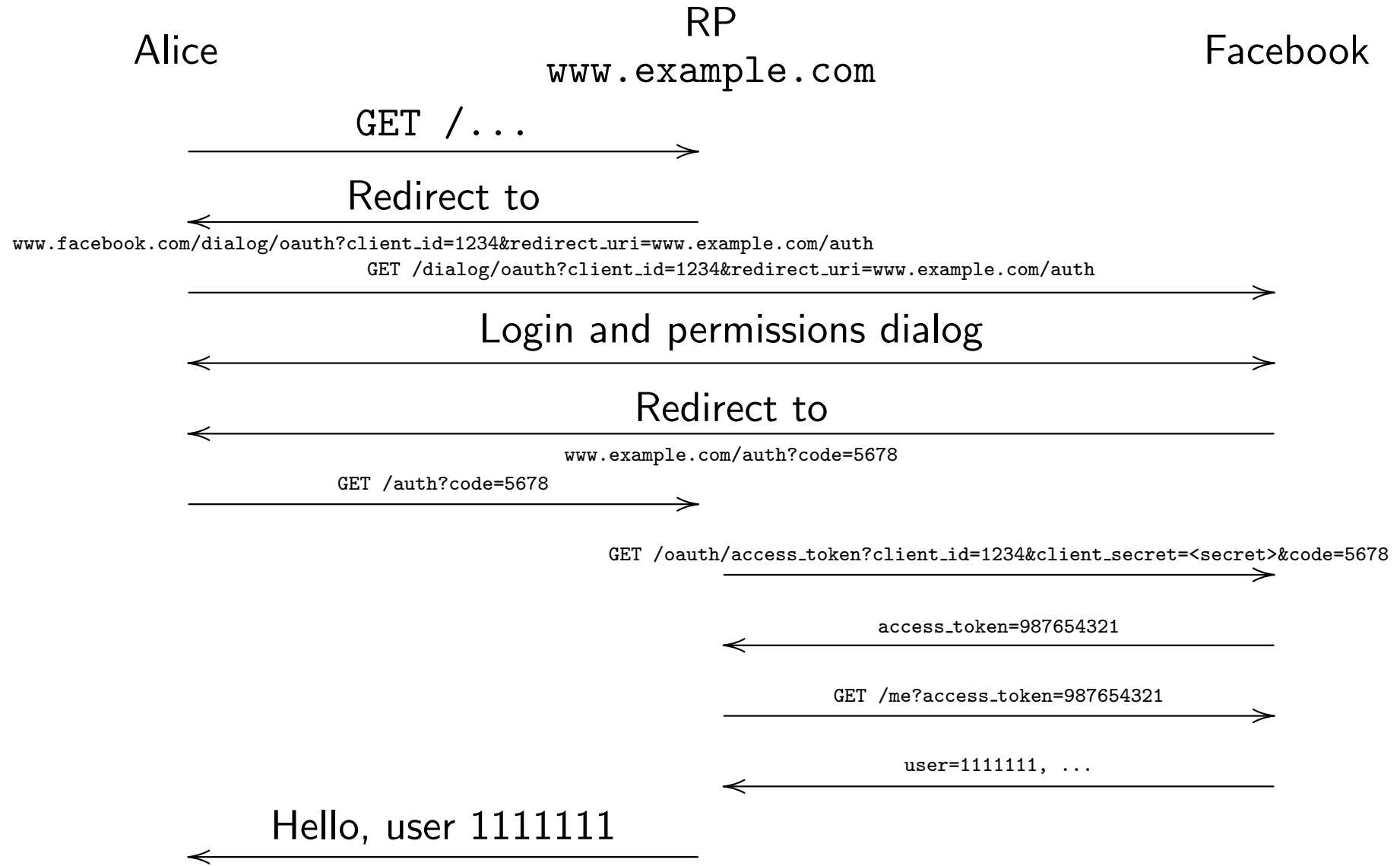
- AP is a user with a Facebook account
 - They may or may not be logged in at the moment
 - (Where *logged in* == cookies)
- RP is a Web server
 - Idea is to bootstrap Facebook authentication
 - ... rather than have your own account system
 - RP registers with Facebook and gets an application key
 - * Facebook wants to control authentication experience

Facebook Connect Call Flow (not logged in) 1

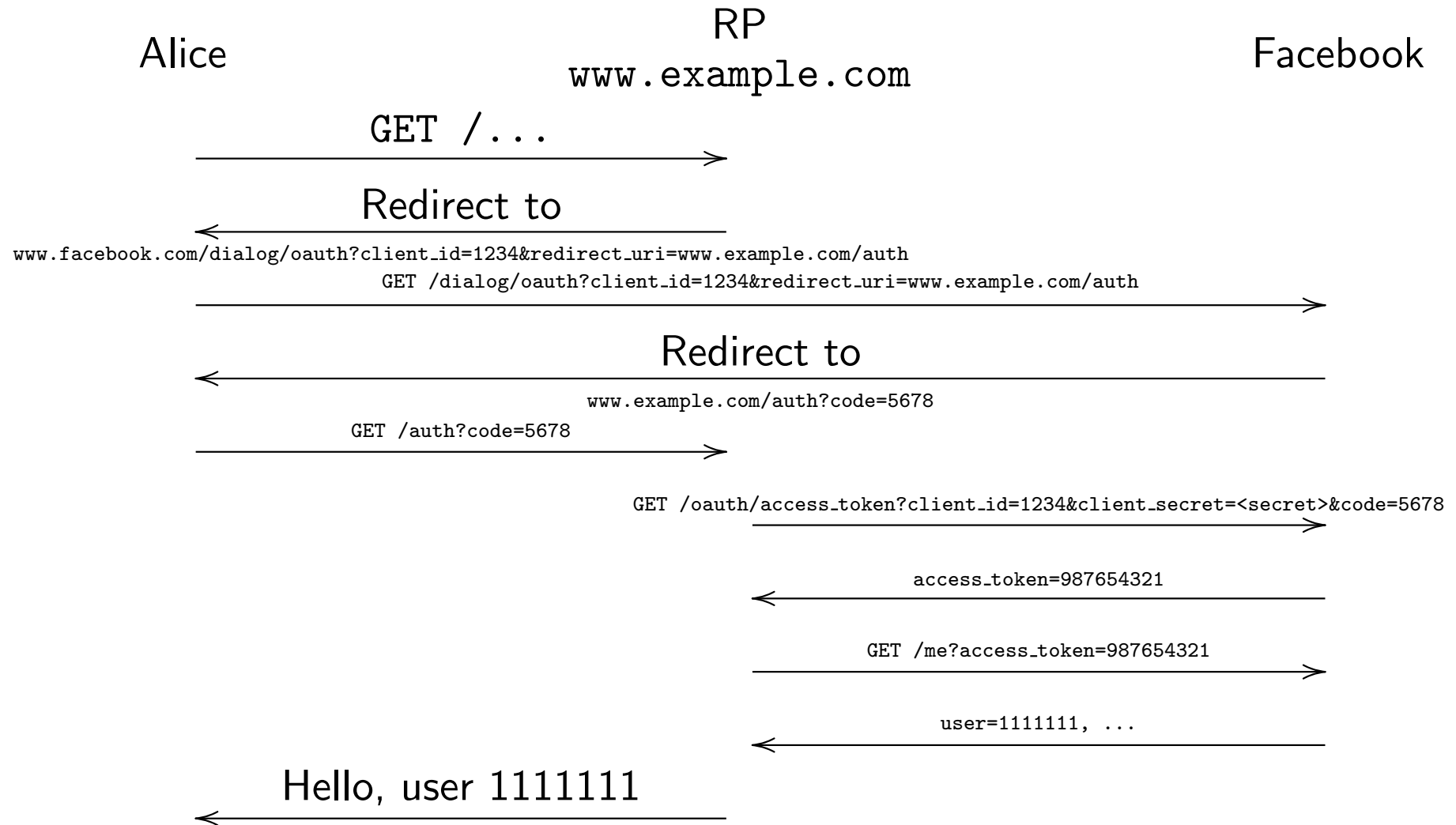




Facebook Connect Call Flow (not logged in) 2



Facebook Connect Call Flow (logged in)



Facebook Connect Privacy Features

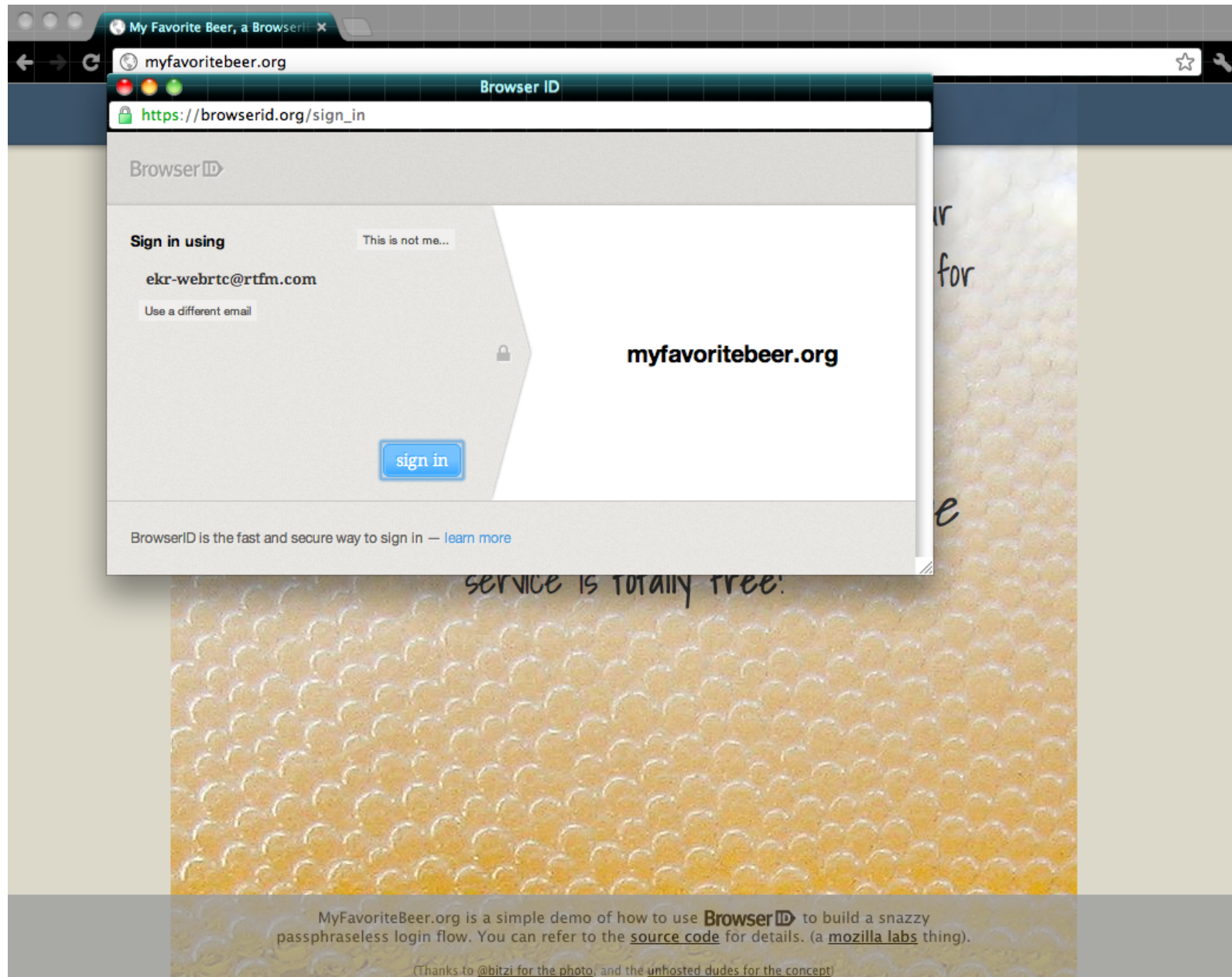
- RP needs to register with Facebook
- User approves policy separately for each RP
 - Including which user information to share
- Facebook learns about every authentication transaction
 - Including user/RP pair

Example: BrowserID

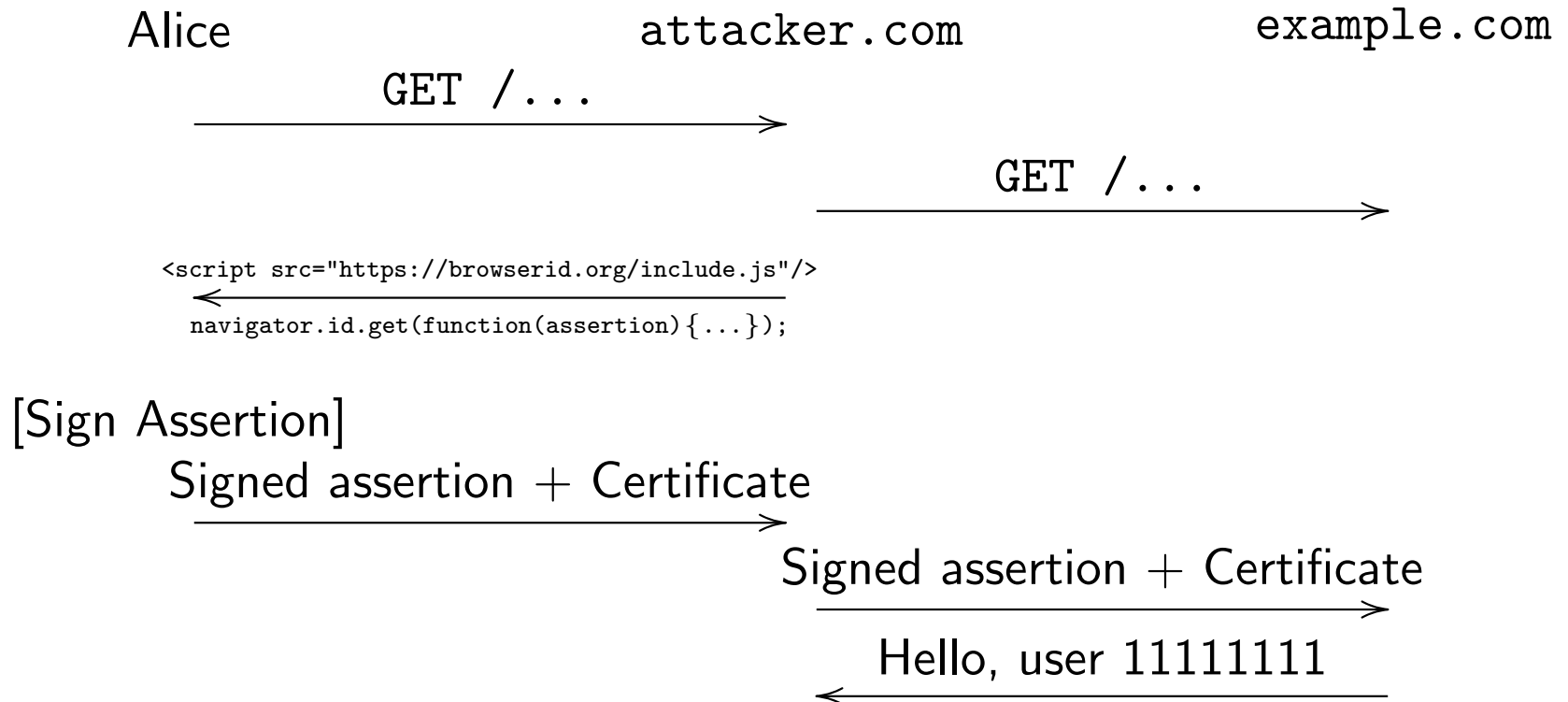
- Effectively client-side certificates
 - But user not exposed to certificates
- Why this example?
 - Easy to understand
 - Familiar-looking technology
 - Less need to wrap your head around redirects, etc.

BrowserID (no key pair)

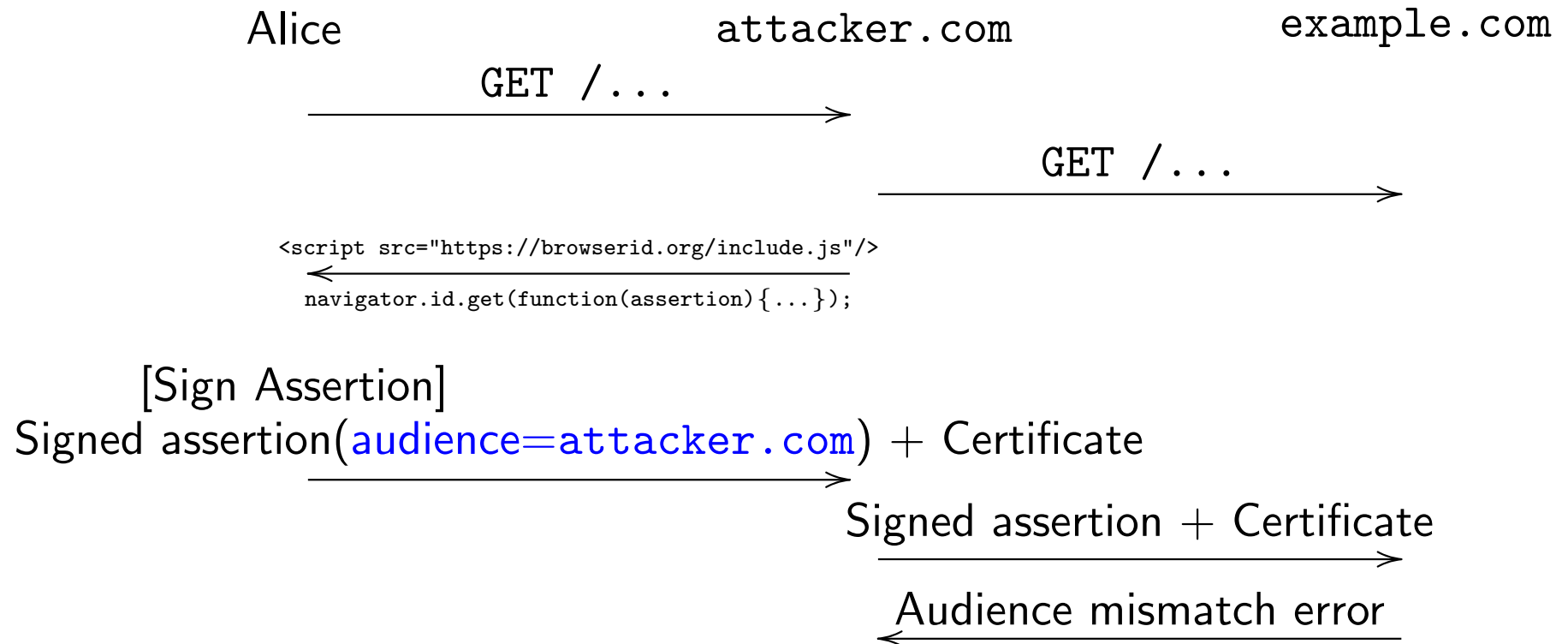




BrowserID: Why no MITM Attacks?



BrowserID: Audience Parameter



Preventing assertion forwarding

- BrowserID assertions are scoped to origin (audience parameter)
 - RPs check that the origin in the assertion matches their domain
 - This prevents assertion forwarding
- Why does this work?
 - BrowserID JS is part of the TCB
 - Browser enforces origin of requests from the calling site
 - RP transitively trusts origin/audience because it trusts BrowserID.org

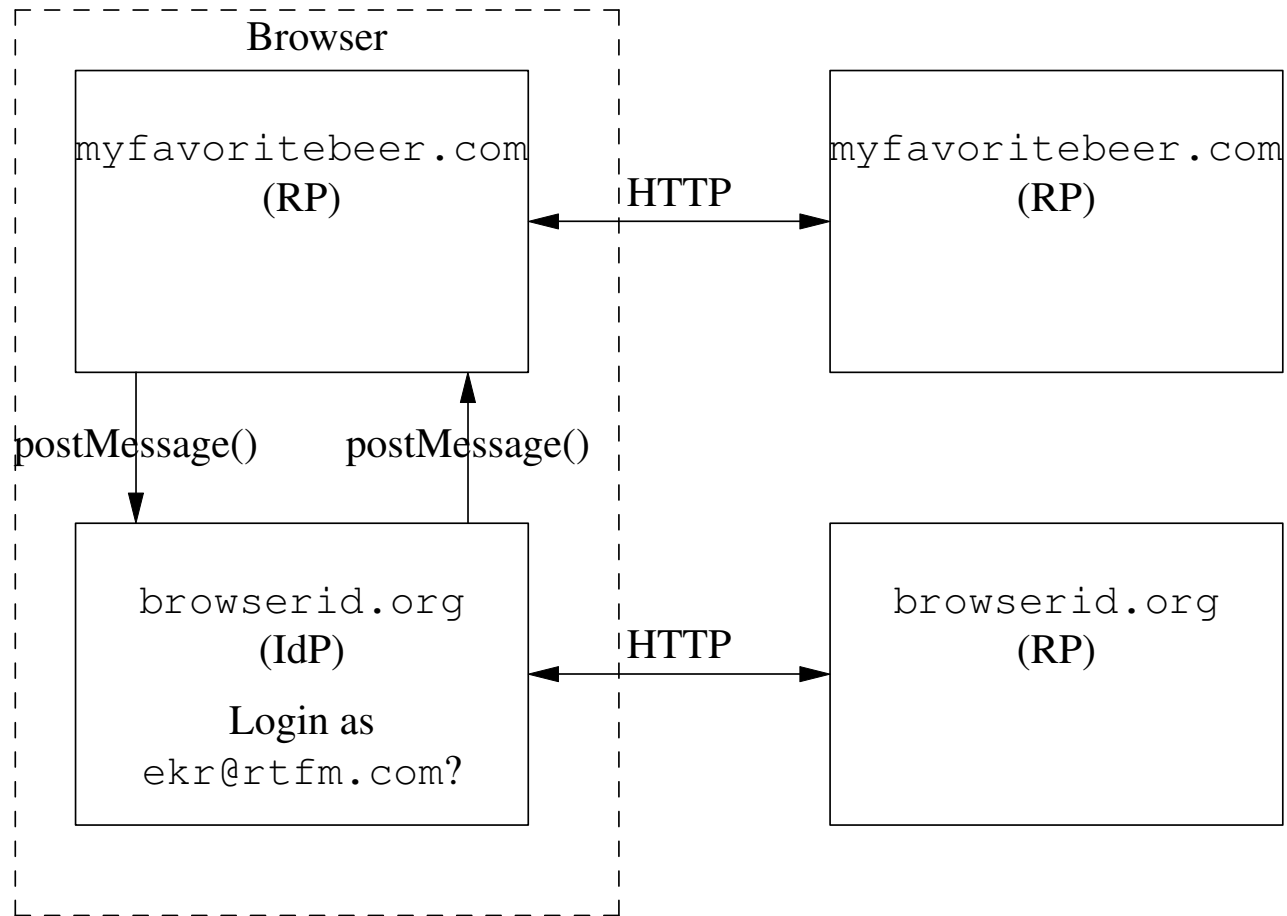
Browser-ID Privacy Features

- Client generates a key pair
 - Idp signs a binding between key pair and user ID
- Client generates assertions based on key pair
 - Sends along certificate
- RP fetches IdP public key
 - This need only happen once
- IdP never learns where you are visiting
 - No relationship between RP and IdP

Example: BrowserID (existing key pair)



BrowserID Security Architecture



One browser, multiple security contexts

- Browser security data scoped by *origin*
 - browserid.org window and myfavoritebeer.org window are isolated
 - Each runs their own JS independently
- Security guarantees
 - Origin A can't touch origin B's data
 - Origin A can't see what origin B is displaying
 - Communication is by `postMessage()` (or navigation hack)

PostMessage: Sender

```
otherWindow.postMessage(message, targetOrigin);
```

otherWindow: the window to send the message to

message: the message to send

targetOrigin: the expected origin of the other window

Why do we need `targetOrigin`?

- Malicious pages can navigate other windows
 - This creates a race condition
- RP creates the new window to IdP with `w = createWindow()`
- Attacker navigates `w` to his own site
- RP does `w.postMessage(secret, ...)`
- Attacker gets the secret
- `targetOrigin` stops this

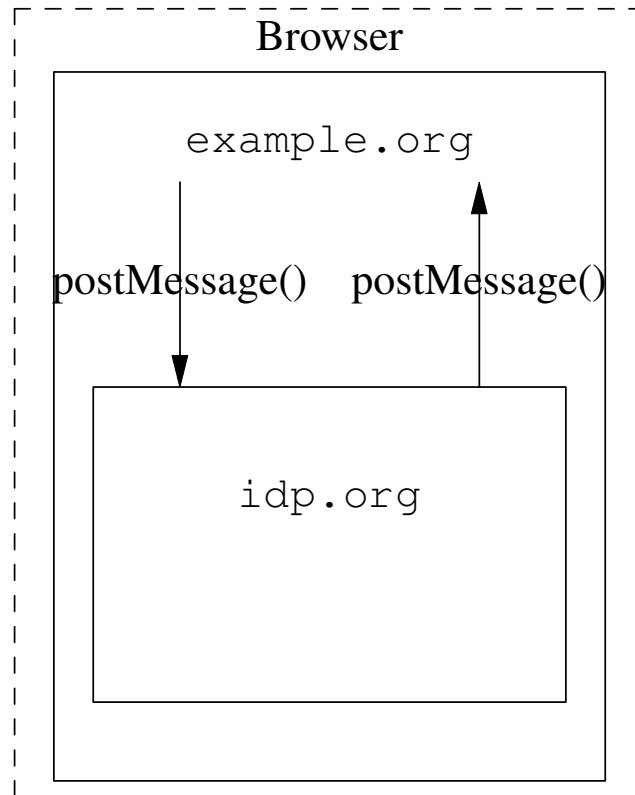
PostMessage: receiver

```
window.addEventListener('message',  
                           function(event) {  
                               ...  
                           });
```

- Event properties:
 - data: the message passed by the sender
 - origin: the sender's origin
 - source: the sender's window
- Important: origin value can be trusted
 - Enforced by the browser
 - May not be the current origin of source, however

IFRAMEs

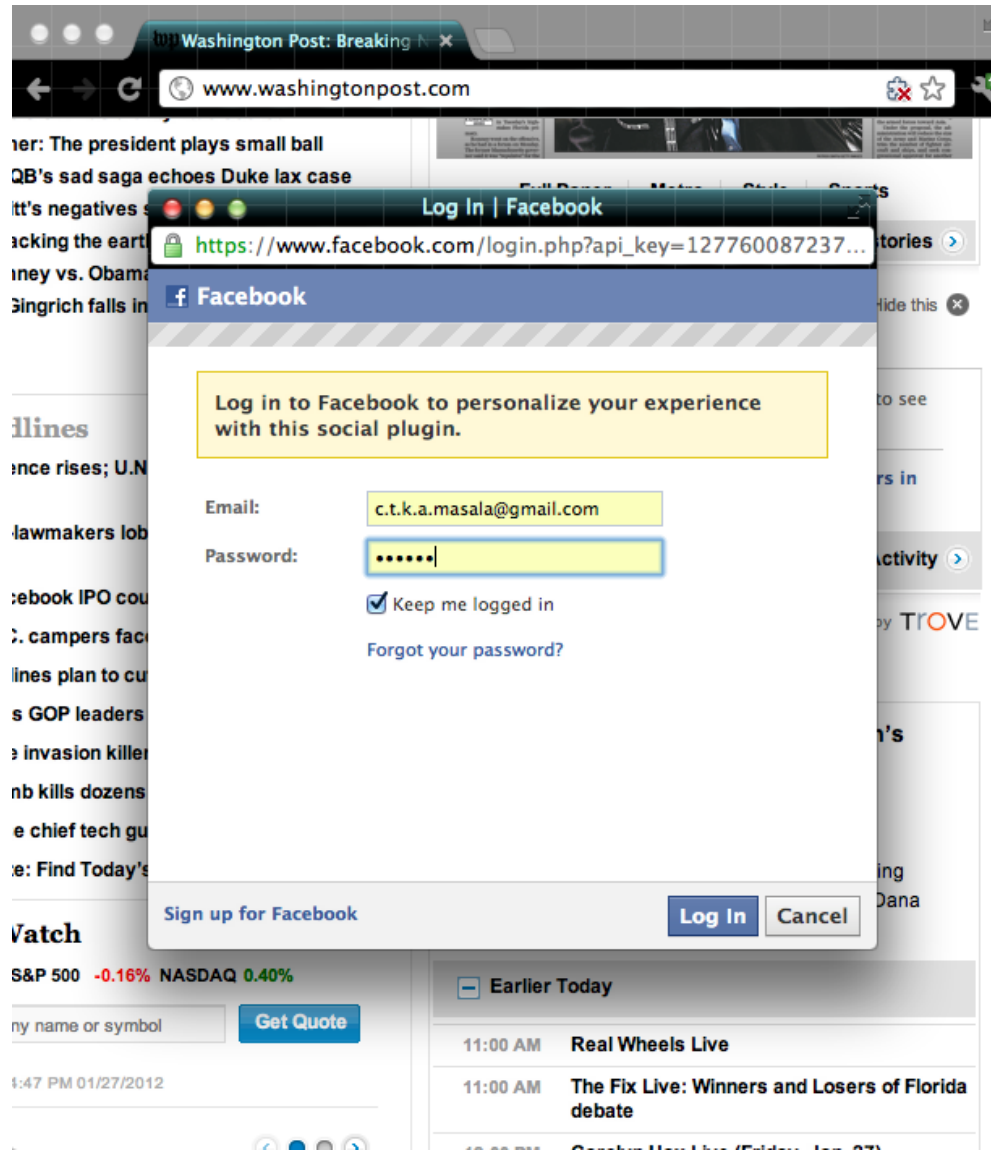
- What if I don't want another window to open?
 - Solution: IFRAMEs



IFRAME Security Properties

- Isolated from the main page
 - More or less the same rules as a separate window
- Can be easily navigated by the main page
- Can be invisible (both good and bad)

Logins generally done in separate windows



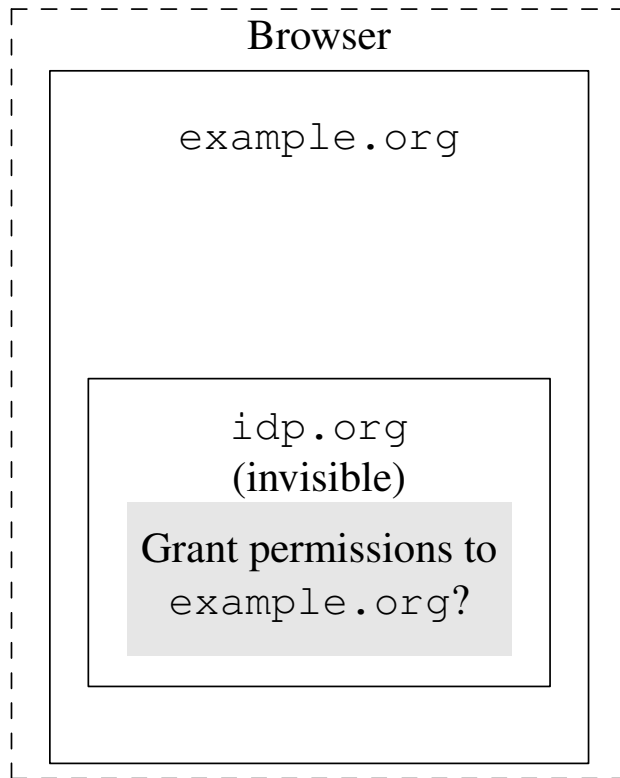
Why aren't logins done in IFRAMEs?

- Scenario: you are on `example.org`
 - `example.org` wants to log you in with `idp.org`
- Both Facebook Connect and BrowserID use a separate window
- Why?
 - IdP is soliciting the user's password
 - User needs to know they are using the right IdP
 - A separate window means they can examine the URL bar
 - Also concerns about clickjacking/redressing
- Other option is to navigate the entire page to an interstitial page

How Clickjacking Works

- Attacker embeds the victim site's page in an IFRAME
 - IFRAME is in front but marked transparent
 - The attacker's page shows through
- Attacker gets the victim to click on "his" page
 - Really the victim site's page
- Victim has just taken action on the victim site

IFRAMEs, Clickjacking, and Permissions Grants



Real frame hierarchy



What the user sees

Preventing Framing

- IdP policy is to have the login page be top-level
 - Good RPs comply with this policy
 - But we're concerned about malicious RPs
- IdPs use “framebusting” JavaScript to prevent being framed
 - This is harder than it sounds
 - ... but standard procedure

IFRAMEs don't have to be visible

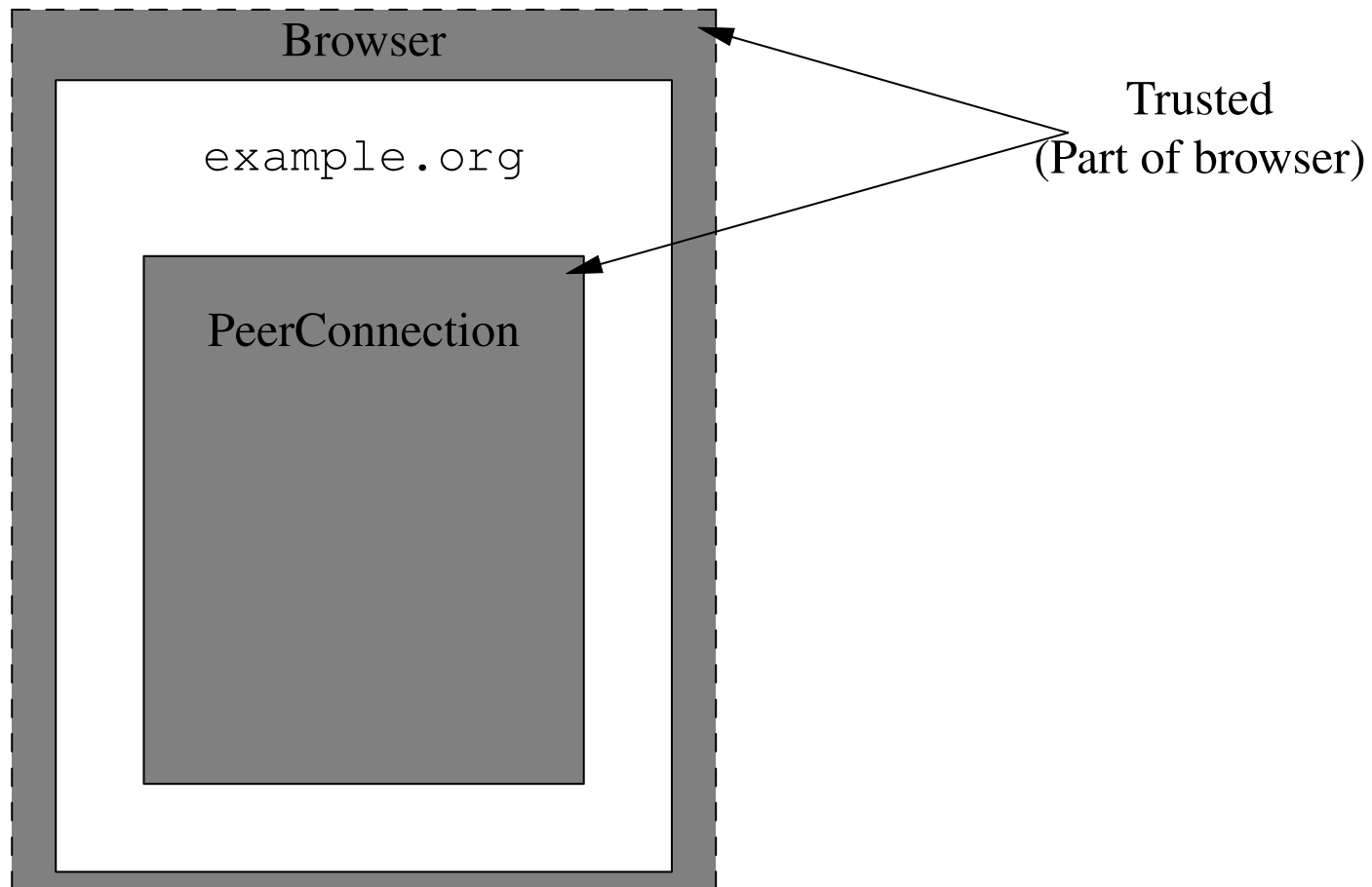
```
idp = document.createElement('IFRAME');  
$(idp).hide();
```

- This takes up no space on the screen
 - It's just JS from the IFRAME source running on the page
 - Can still `postMessage()` to and from it
- Invisible IFRAMEs are a very important tool

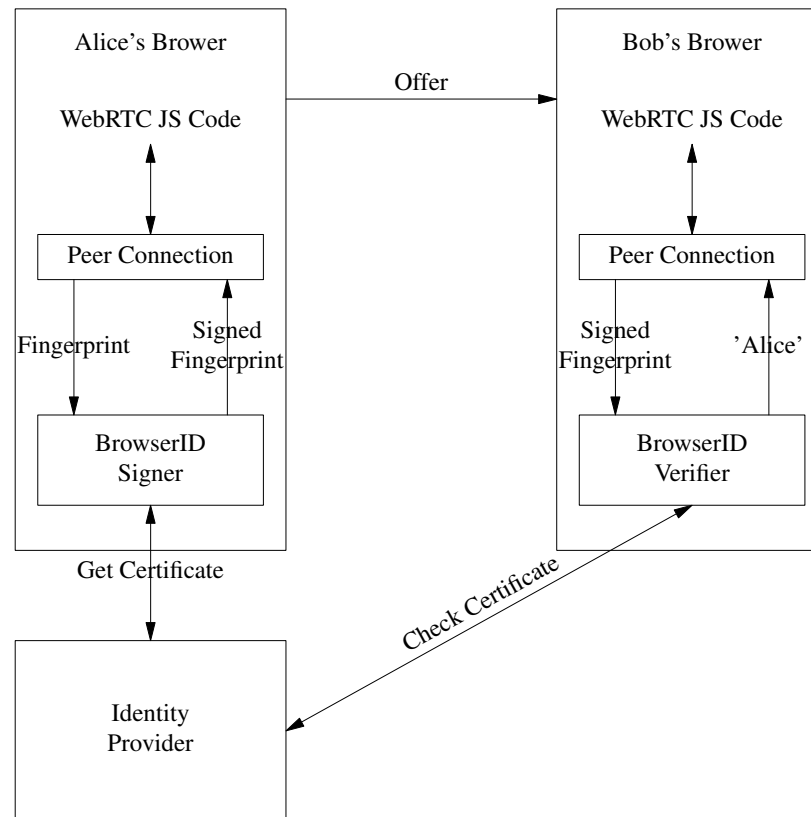
What are we trying to accomplish?

- Repurpose existing identity infrastructure for user-to-user authentication
- Requirements/objectives
 - Use existing accounts
 - Minimal (preferably no) changes to IdP
 - Easy to support at calling site
 - * Better if no change
 - Generic support in browser
 - * Single downward interface between PeerConnection object and IdP
 - * Should be able to support new IdPs/protocols without changing browser

Reminder: Trust Architecture



Example IdP Interaction: BrowserID



Example ROAP Offer with BrowserID

```
{
  "messageType": "OFFER",
  "callerSessionId": "13456789ABCDEF",
  "seq": 1
  "sdp": ""
  ...
  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
  }
}
```

Example JSEP TransportInfo with Facebook Connect (Or any private identity service)

```
{
  "pwd": "asd88fgpdd777uzjYhagZg",
  "ufrag": "8hhy",
  "fingerprint": {
    "algorithm": "sha-1",
    "value": "4AADB9B13F82183B540212DF3E5D496B19E57CAB",
  },
  "candidates": [
    ...
  ],
  "identity": {
    "idp": {
      "domain": "example.org"
      "protocol": "bogus"
    },
    "assertion": "{\"identity\":\"bob@example.org\",
                  \"contents\":\"abcdefghijklmnoprstuvwxyz\",
                  \"signature\":\"010203040506\"}"
  }
}
```

* Assumption here is that we have changed JSEP to emit transport-infos

But we want it to be generic...

- This means defined interfaces
- ... that work for any IdP

What needs to be defined

- Information from the signaling message that is authenticated [IETF]
 - Minimally: DTLS-SRTP fingerprint
 - Generic carrier for identity assertion
 - Depends on signaling protocol
- Interface from PeerConnection to the IdP [IETF]
 - A specific set of messages to exchange
 - Sent via `postMessage()` or `WebIntents`
- JavaScript calling interfaces to PeerConnection [W3C]
 - Specify the IdP
 - Interrogate the connection identity information

What needs to be tied to user identity?

- Only data which is verifiably bound is trustworthy
 - Need to assume attacker has modified anything else
- Initial analysis (depends on protocol)
 - Fingerprint (MUST)
 - ICE candidates
 - Media parameters

Security Properties of ICE Candidates

- Effect of modifying ICE candidates
 - Advertise candidates to route media through attacker
 - * Makes a MITM attack easier
 - * Mostly irrelevant if DTLS keying used
 - Route to /dev/null (DoS)
 - * Silly if you are in signaling path!
- Signaling service can affect ICE candidates anyway
 - Provide a malicious TURN server
 - Return blackhole server reflexive addresses
 - This drives data through signaling service

Security Properties of Media Parameters

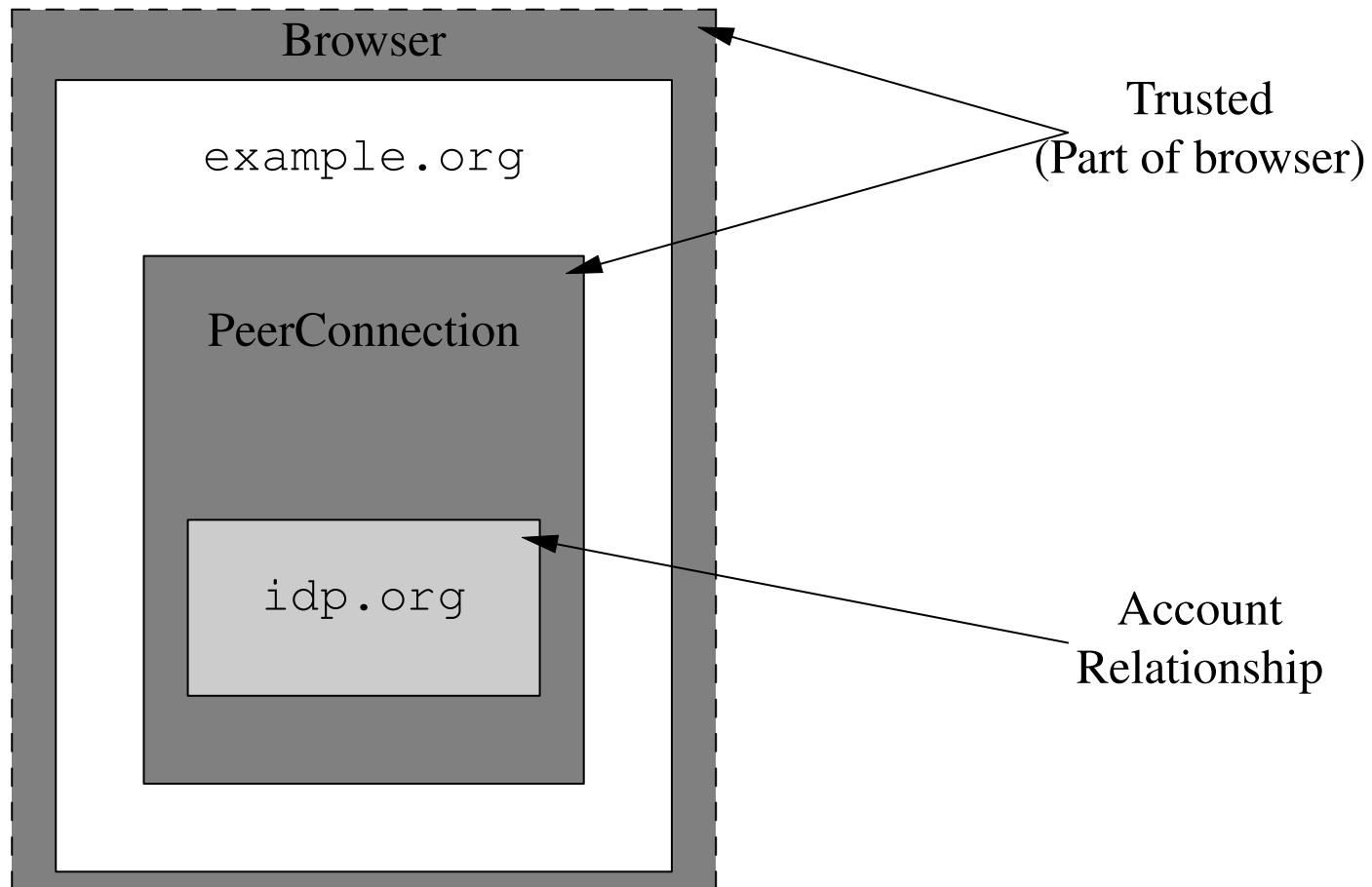
- Which media flows
 - Calling service has control of this anyway
 - But the UI needs to show what is being used
 - * For consent reasons
- Which codecs
 - Calling service can influence these
 - Might be nice to secure them
 - But too limiting
 - SRTP should provide security regardless of codec selection

Generic Structure for Identity Assertions

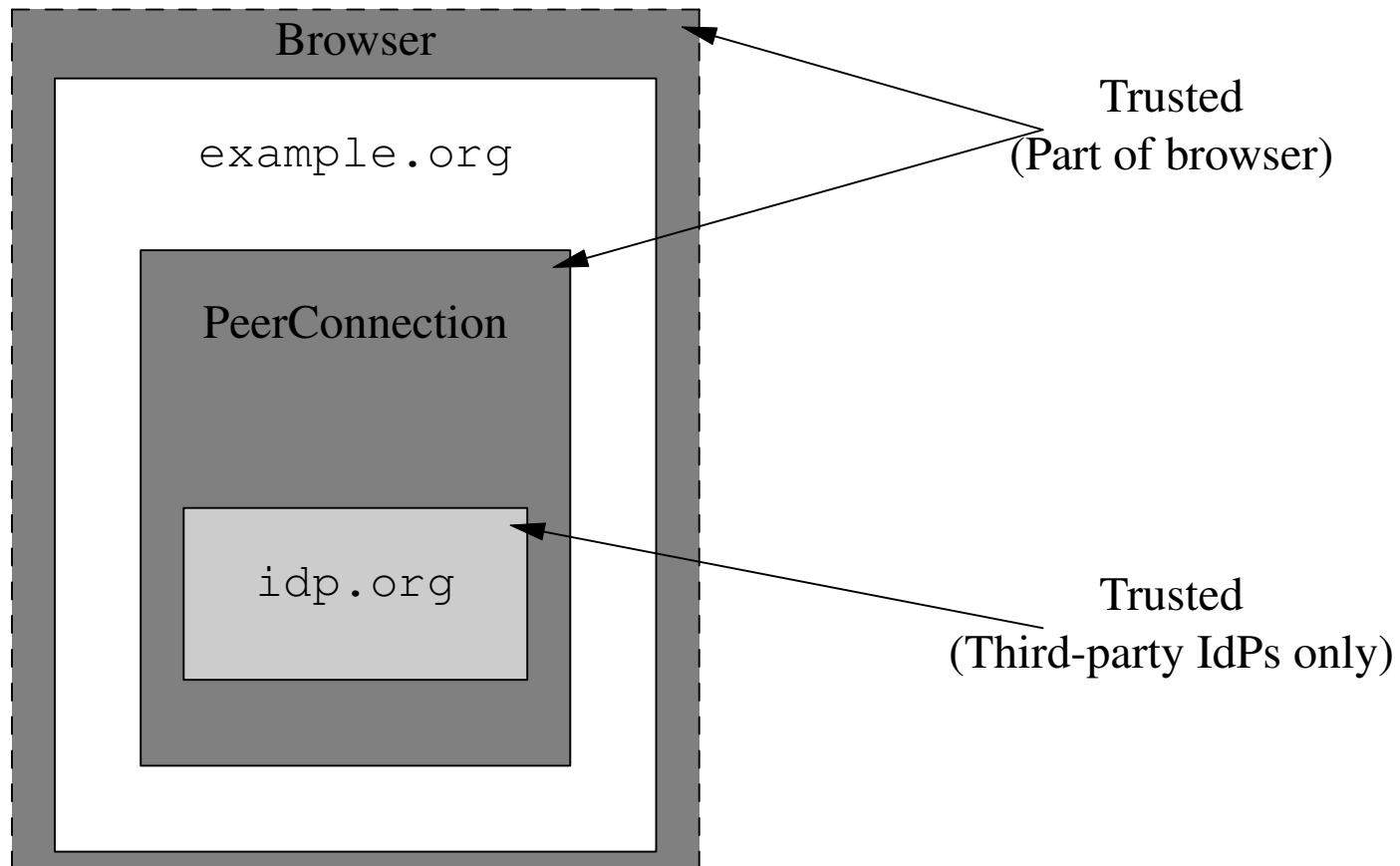
```
"identity":{  
    "idp":{      // Standardized  
        "domain":"idp.example.org", // Identity domain  
        "method":"default"          // Domain-specific method  
    },  
    "assertion": "...",              // IdP-specific  
}
```

Basic Architecture

IdP Trust Architecture: Authenticating Party



IdP Trust Architecture: Relying Party



Generic Downward Interface (Implemented by PeerConnection)

- Instantiate “IdP Proxy” with JS from IdP
 - Probably invisible IFRAME
 - Maybe a WebIntent (more later)
- Send (standardized) messages to IdP proxy via `postMessage()`
 - “SIGN” to get assertion
 - “VERIFY” to verify assertion
- IdP proxy responds
 - “SUCCESS” with answer
 - “ERROR” with error

Where is the IdP JS fetched from?

- Deterministically constructed from IdP domain name and method
`https://<idp-domain>/.well-known/idp-proxy/<protocol>`
- Why in /.well-known?
 - Trust-relationship derives from control of the domain
 - Must not be possible for non-administrative users of domain to impersonate IdP

How does PeerConnection know IdP domain?

- Authenticating Party
 - IdP domain configured into browser
 - * User “logs into” browser via UI
 - * WebIntents again
 - Specified by the calling site
 - * “Authenticate this call with Facebook connect”
 - * Need a new API point for this
- Relying party
 - Carried in the generic part of the identity assertion

Generic Message Structure

```
{  
  "type": "...",      // "SIGN","VERIFY","SUCCESS", ...  
  "id": "1",          // used for correlation  
}
```

Incoming Message Checks (IdP Proxy)

- Messages MUST come from `rtcweb:///...`
- This prevents ordinary JS from instantiating IdP proxy
 - Remember, it's just an IFRAME
 - But you can't set your origin to arbitrary values
- Messages MUST come from parent window
 - Prevents confusion about which proxy

Incoming Message Checks (PeerConnection)

- Messages MUST come from IdP origin domain
 - Prevents navigation by attackers in other windows
- Messages MUST come from IdP proxy window
 - Prevents confusion about which proxy

Signature process

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\"}"
  }
}
```

Verification Process

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"
  }
}
```

Meaning of Successful Verification

- IdP has verified assertion
 - Identity is given in “identity”
 - “name” is the actual identity (RFC822 format)
 - “displayname” is a human-readable string
- Contents is the original message the AP passed in

Processing Successful Verifications

- Authoritative IdPs
 - RHS of `identity.name` matches IdP domain
 - No more checks needed
- Third-party IdPs
 - RHS of `identity.name` does not match IdP domain
 - IdP MUST be trusted by policy
- These checks performed by PeerConnection

How do I stand up a new IdP?

1. Get some users (the hard part)
2. Implement handlers for SIGN and VERIFY messages
 - Probably < 100 lines of JS
3. Put the right JS at `/.well-known/idp-proxy`
4. Profit

Integrated IdP Support

- Things work fine with no browser-side IdP support
- But specialized support is nice too
 - “Sign-in to browser” in Chrome
 - BrowserID in Firefox
 - Better UI/performance properties
- Still specify IdP by URL
 - IdP JS detects that the browser has built-in support
 - Calls go directly to the browser code

Questions?