The Web SSO Standard OpenID Connect

In-Depth Formal Security Analysis and Security Guidelines

Daniel Fett, Ralf Küsters, Guido Schmitz

Institute of Information Security
University of Stuttgart

Universität Stuttgart

Our Contributions

OpenID Connect 1.0 with Discovery and Dynamic Registration:

- Developed formal model of the standard
  - Based on most comprehensive model of the web to date (extension of S&P 2014).
- Formalized central security properties
  - Authentication
  - Authorization
  - Session Integrity
- Proved security for (fixed) standard (see security guidelines)

Paper to appear at CSF 2017

All details: TR available at https://sec.informatik.uni-stuttgart.de

- Many flaws and attacks in web applications
- Increasing complexity of web sites & systems
- Interaction of different components

DNS Severs	Web Servers	Browsers	...

Formal methods required to …

- precisely specify security properties
- carry out security proofs

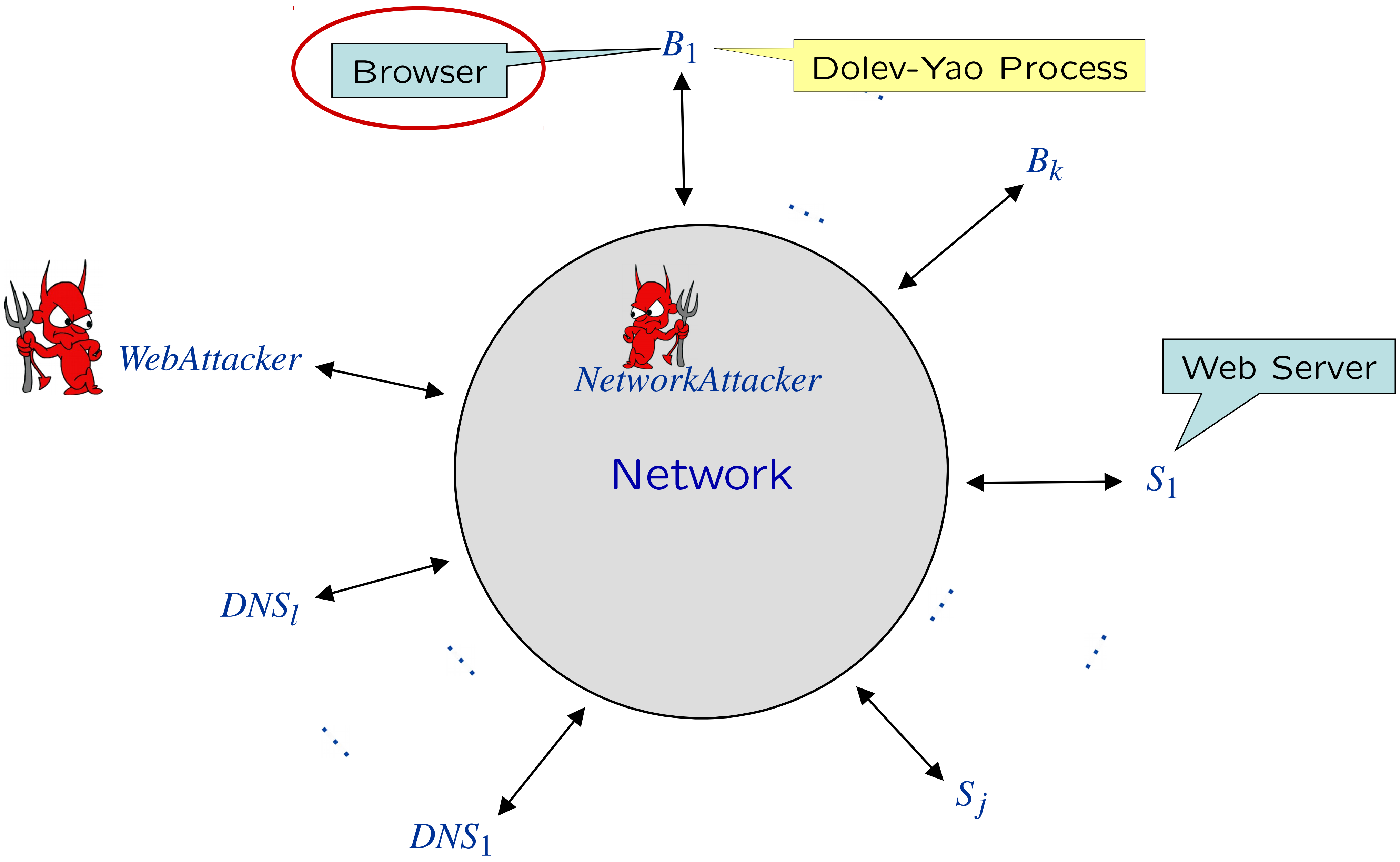
Sources

Specifications for the web are spread across many sources with mutual dependencies:

- Standards and RFCs
  - HTTP/1.1 and HTTP/2 Standards
  - W3C HTML5
  - W3C Web Storage
  - WHATWG Fetch
  - W3C Cross-Origin Resource Sharing
  - RFCs (6265, 6797, 6454, 2616, …)
- Browser implementations
  - Google Chrome
  - Mozilla Firefox
  - …

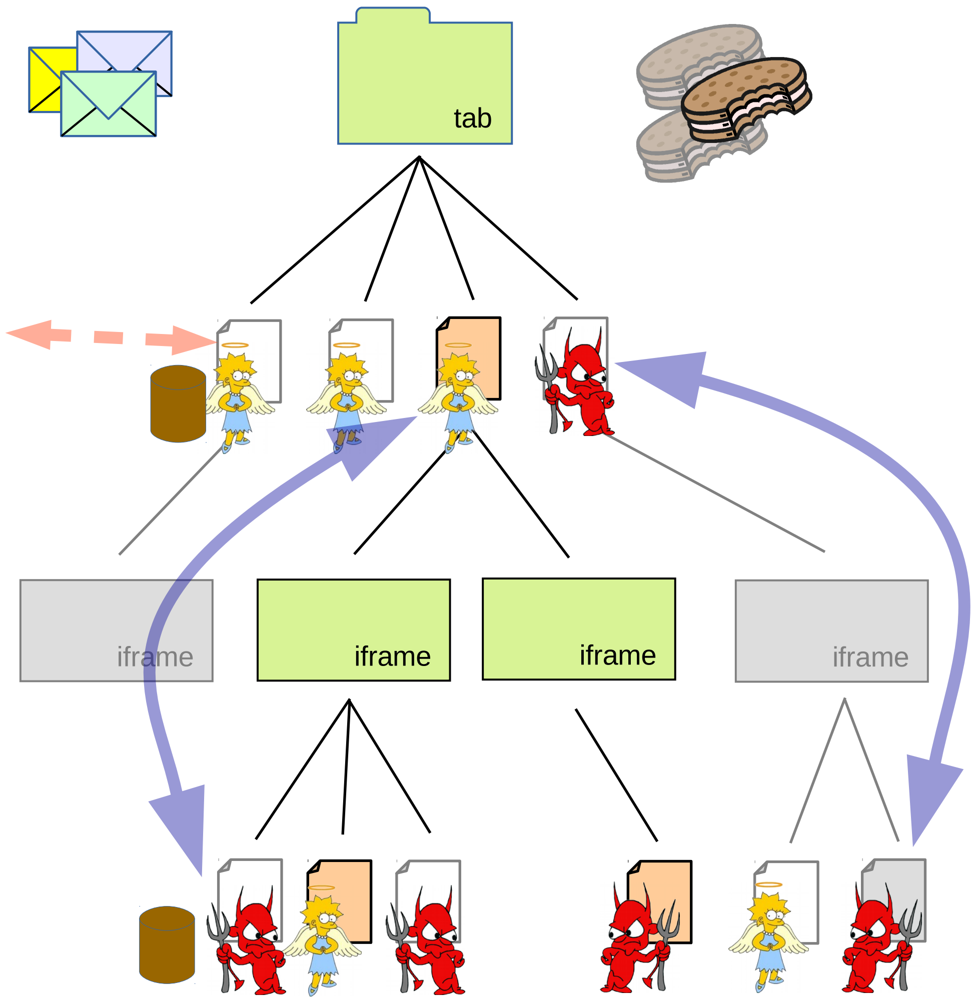
⇒ Model provides coherent view of core aspects of the web

Web Model



Browser
B_1
Dolev-Yao Process
B_k
WebAttacker
NetworkAttacker
Network
Web Server
S_1
DNS_l
DNS_1
S_j

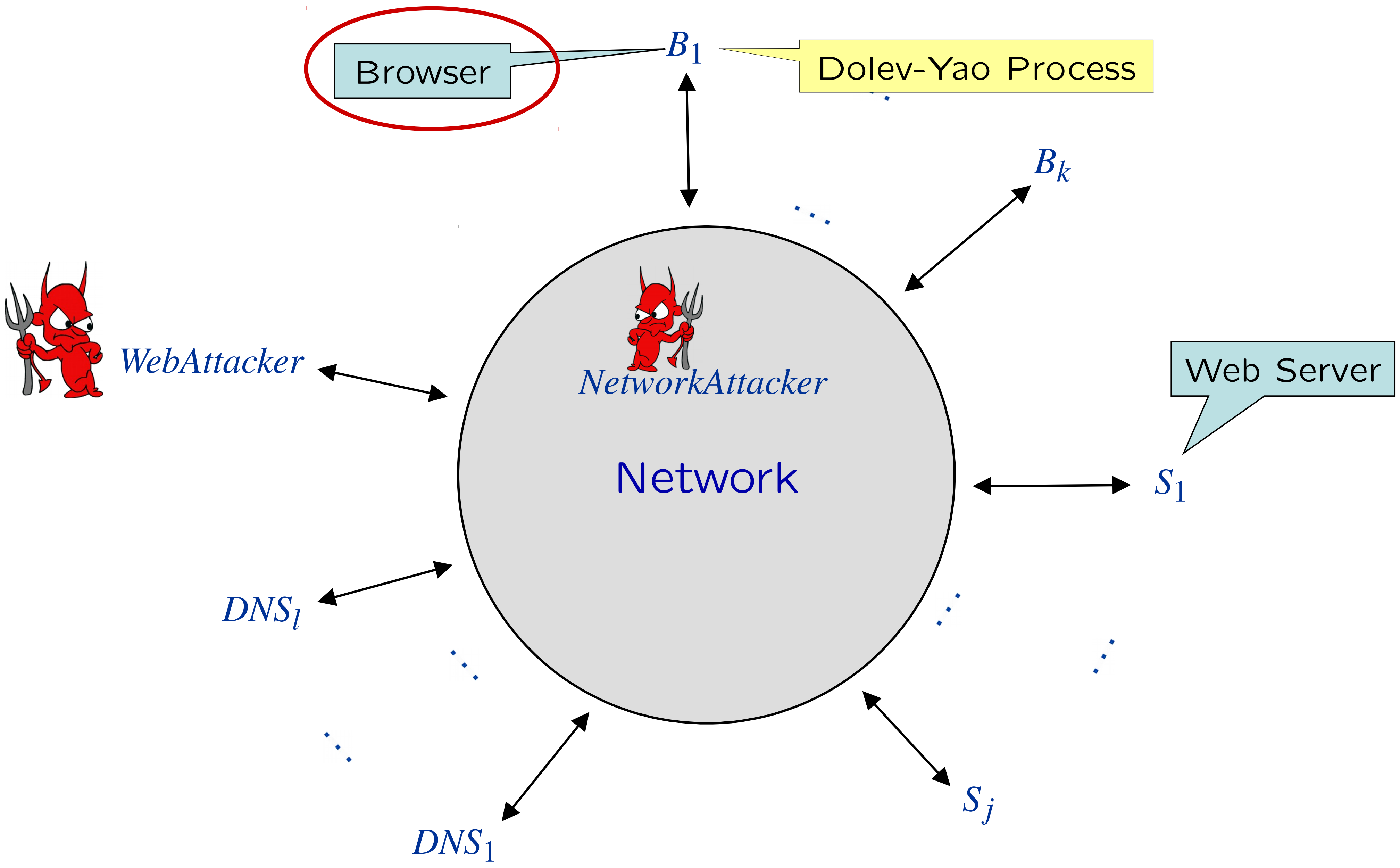
Web Browser Model



Including ...

- DNS, HTTP, HTTPS
- window & document structure
- scripts
- attacker scripts
- web storage & cookies
- web messaging & XHR
- message headers Origin: https://example.com
- redirections
- security policies
- dynamic corruption
- ...

Web Model



Browser
B_1
Dolev-Yao Process
B_k
WebAttacker
NetworkAttacker
Network
Web Server
S_1
DNS_l
DNS_1
S_j

Limitations

- No language details
- No user interface details
- No byte-level attacks (e.g, buffer overflows)
- Abstract view on cryptography and TLS

Previous Work

[SP 2014, ESORICS 2015, CCS 2015, CCS 2016]

- Formal analysis of Mozilla's BrowserID
  Main design goal: privacy
  - Found severe attacks
  - Proposed fixes for authentication and proved security
  - Privacy broken beyond repair
- Designed our own new SSO system: SPRESSO
  Provably provides strong authentication and privacy properties.
- Analysis of OAuth 2.0
  - Found attacks
  - Proposed fixes and proved security

Our Contributions

OpenID Connect 1.0 with Discovery and Dynamic Registration:

- Developed formal model of the standard
  - Based on most comprehensive model of the web to date (extension of S&P 2014).
- Formalized central security properties
  - Authentication
  - Authorization
  - Session Integrity
- Proved security for (fixed) standard (see security guidelines)

Paper to appear at CSF 2017

All details: TR available at https://sec.informatik.uni-stuttgart.de

- OpenID Connect 1.0 (OIDC): Web SSO
  - Goal: identity provider (IdP) assures relying party (RP) of user's identity
  - Based on the OAuth 2.0 authorization framework (RFC 6749)
  - Introduces ID token as one-time proof of user's identity
  - Several modes, options, and extensions:
    - 3 different modes of interaction
    - discovery
    - dynamic registration
    - …

Why Another Security Analysis?

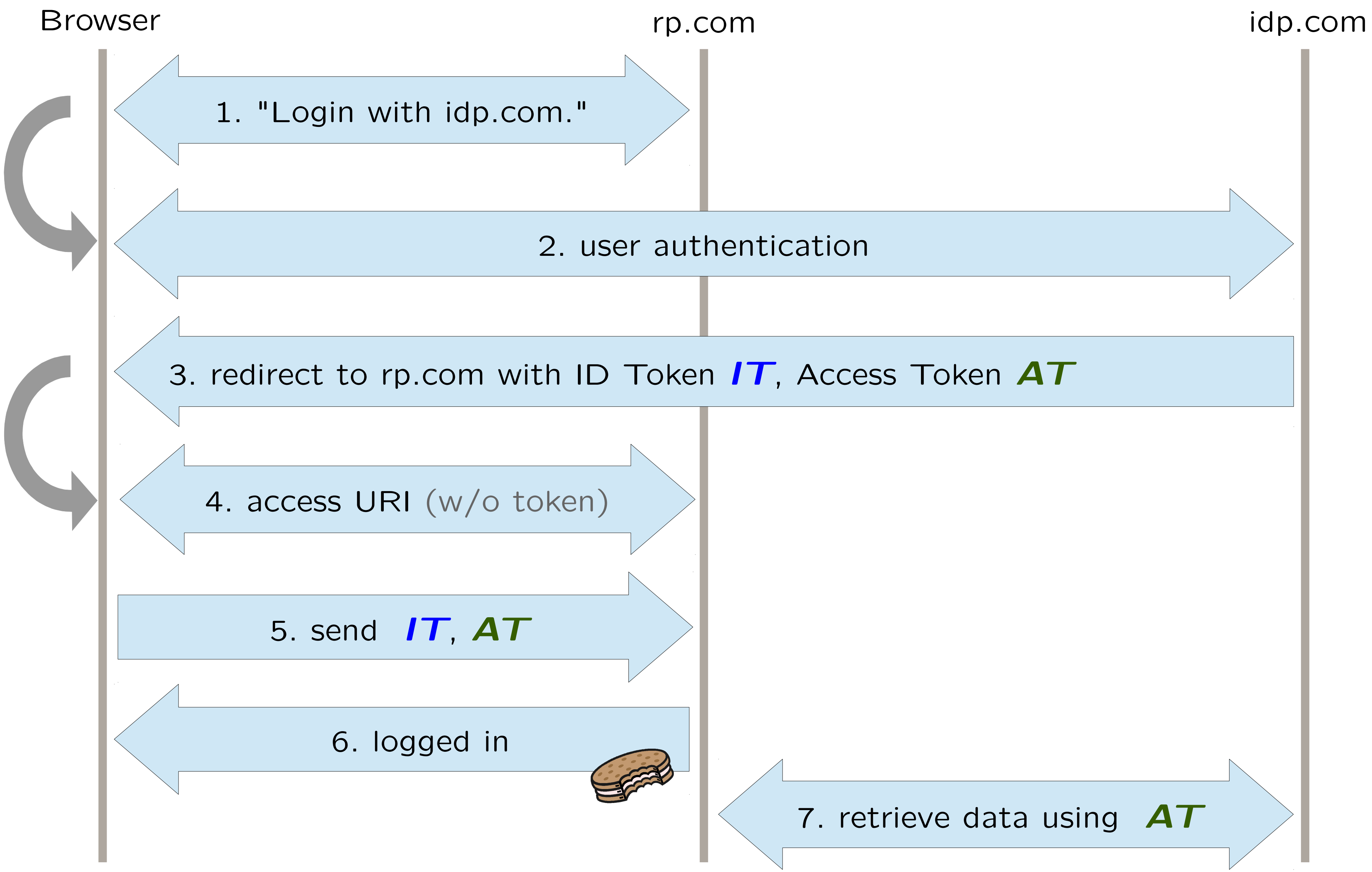
- Subtle differences between OAuth 2.0 and OIDC 1.0
  - OAuth is not for authentication (although proprietary extensions exist)
  - New hybrid mode
  - Discovery and dynamic registration
  - ID Token
- (Also: more modular proof)

ID Token

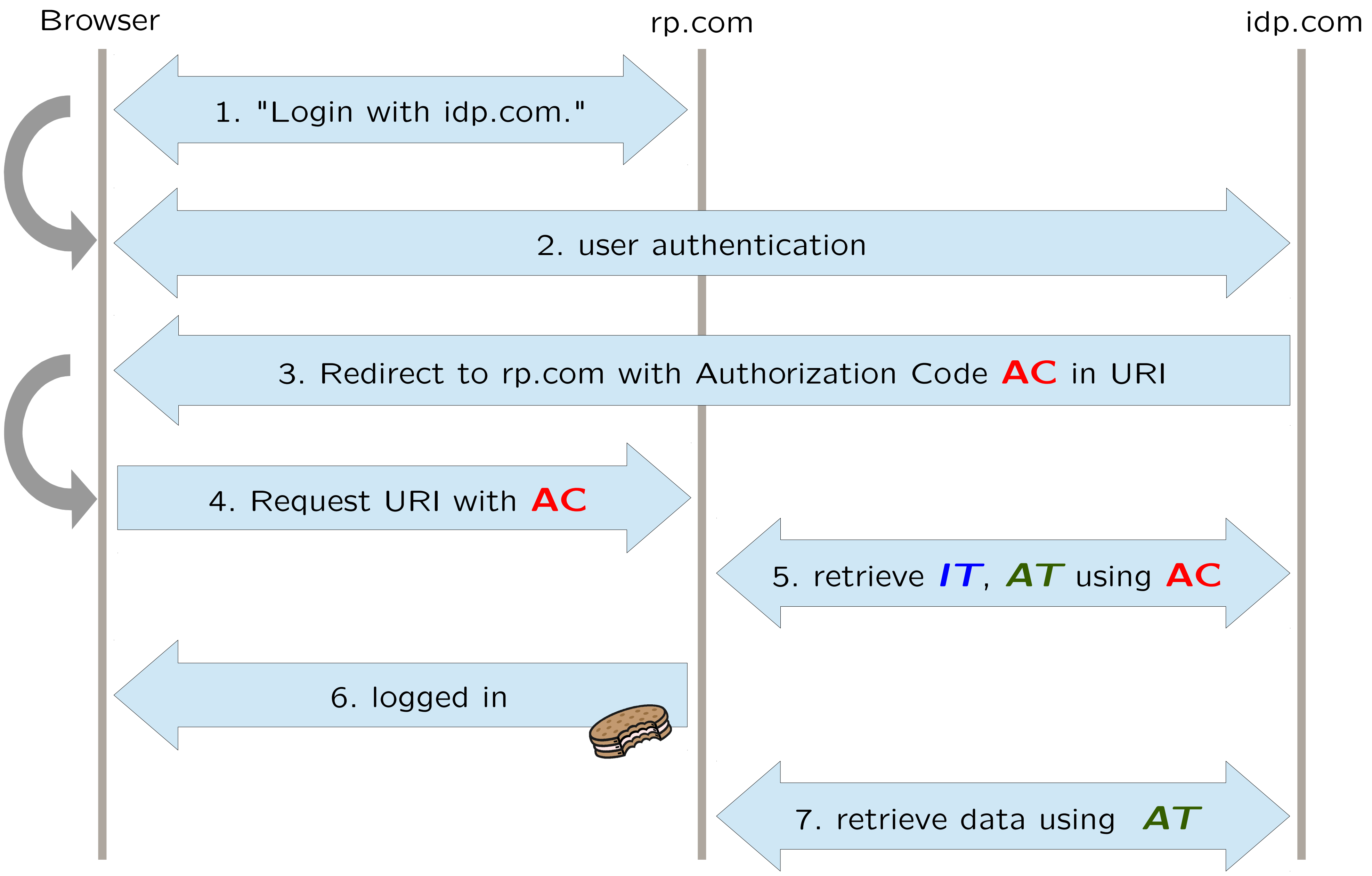
- One-time proof of the user's identity
- Issued by IdP, consumed by RP.
- Signed by IdP
- Contains following claims:
  - user identifier (unique at respective IdP)
  - issuer (IdP)
  - audience (RP)
  - …

unique global user identifier

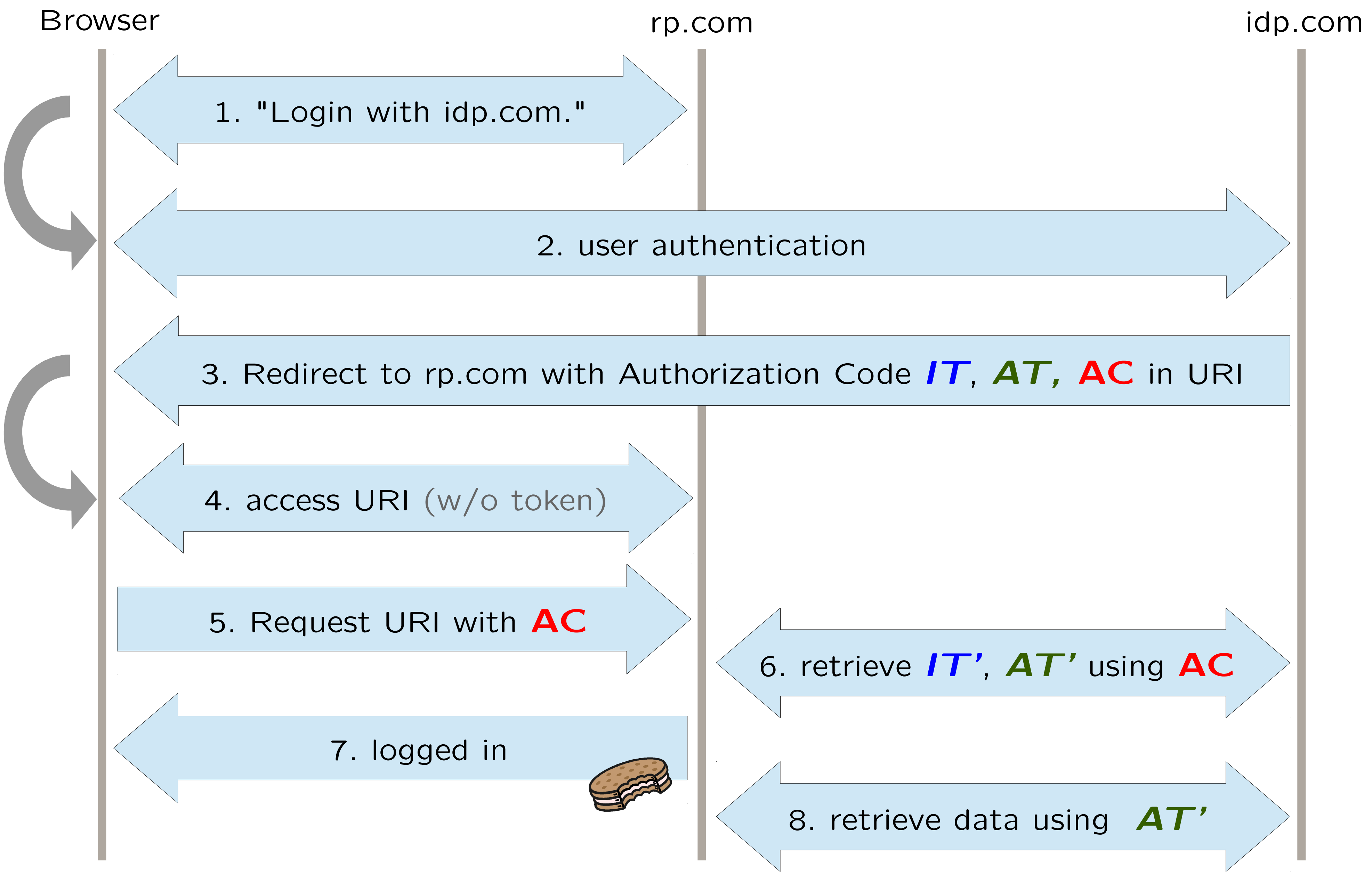
Implicit Mode



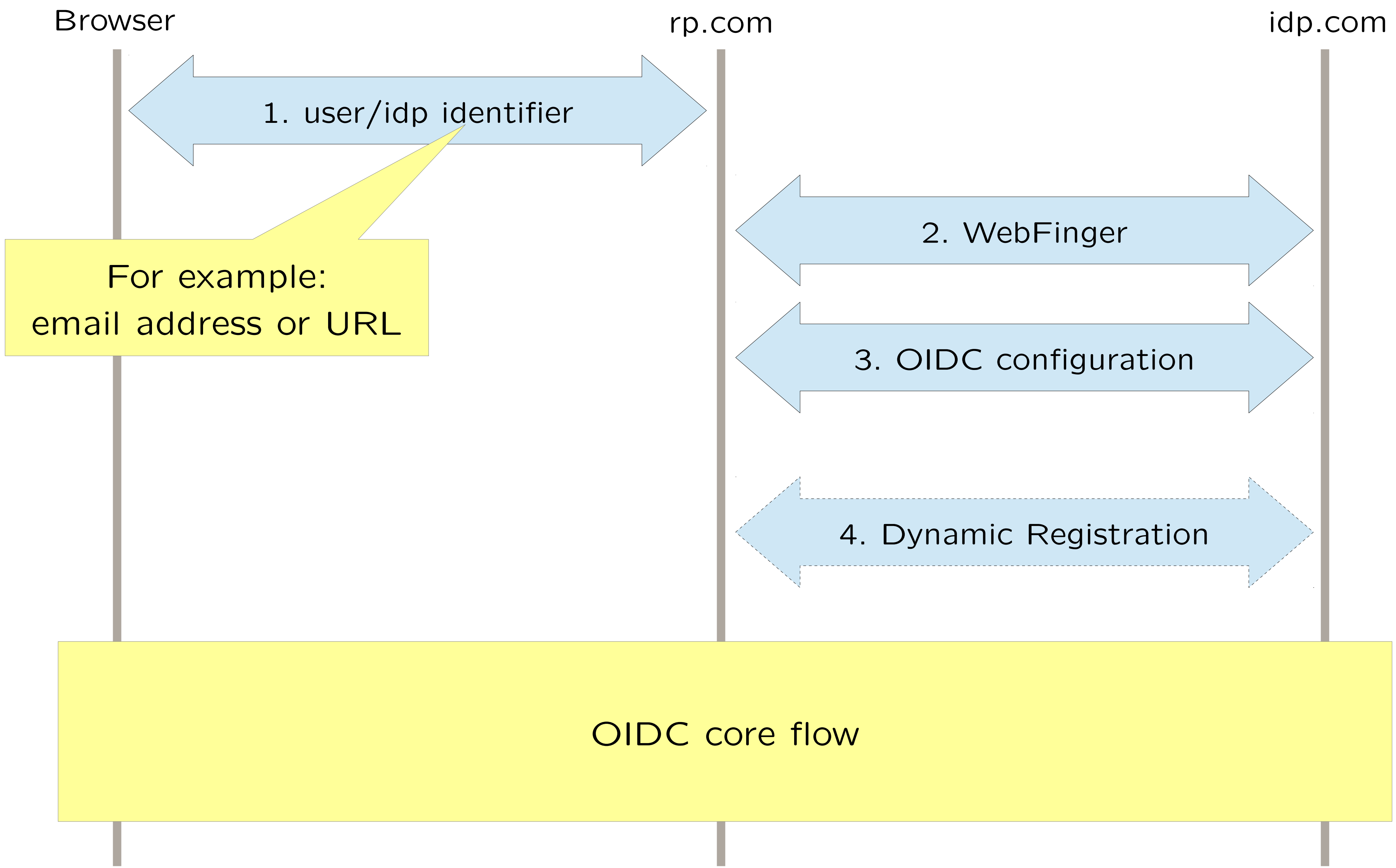
Authorization Code Mode



Hybrid Mode


Browser
rp.com
idp.com
1. "Login with idp.com."
2. user authentication
3. Redirect to rp.com with Authorization Code IT, AT, AC in URI
4. access URI (w/o token)
5. Request URI with AC
6. retrieve IT', AT' using AC
7. logged in
8. retrieve data using AT'

Discovery and Dynamic Registration



Our Contributions

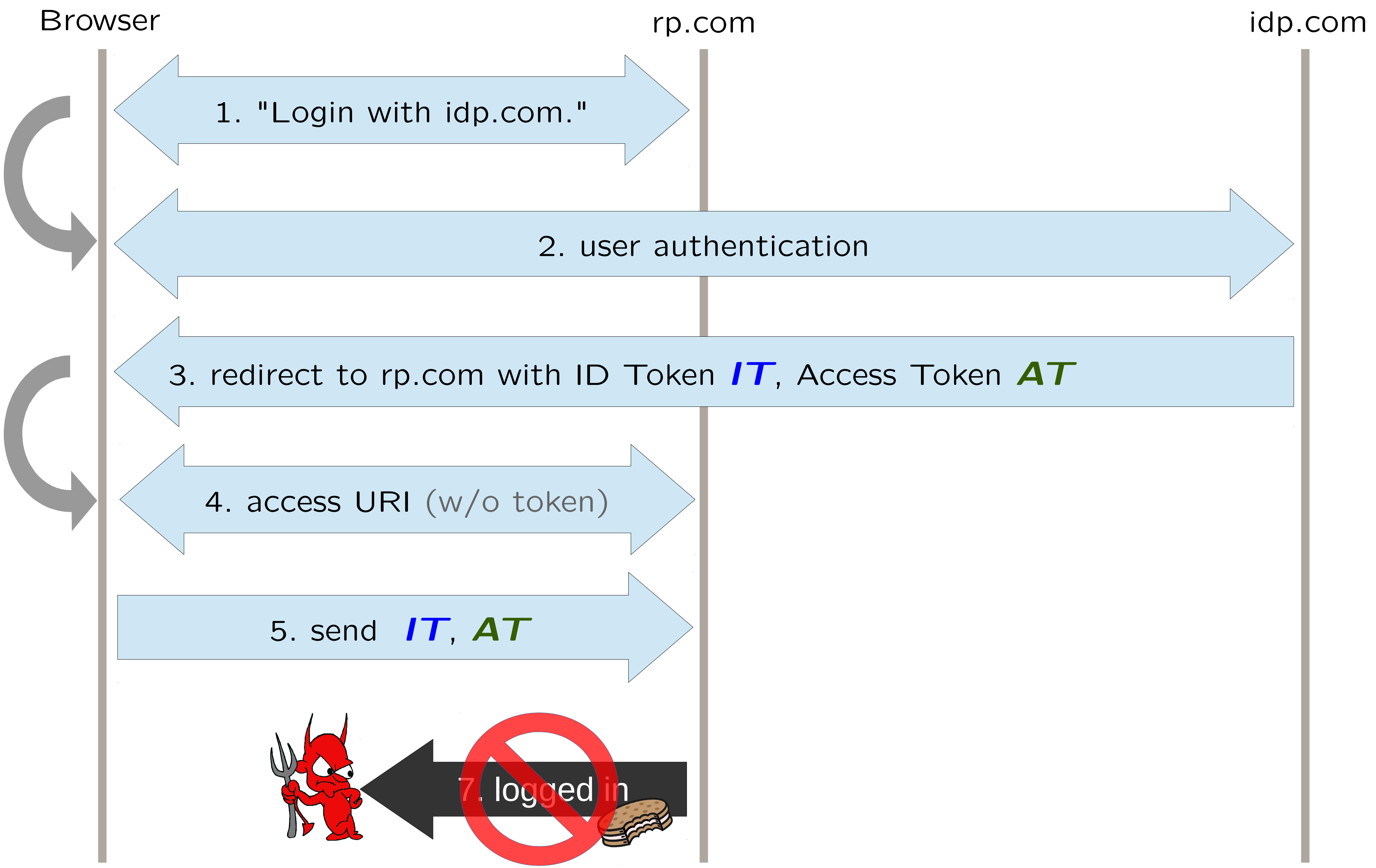
OpenID Connect 1.0 with Discovery and Dynamic Registration:

- Developed formal model of the standard
  - Based on most comprehensive model of the web to date (extension of S&P 2014).
- Formalized central security properties
  - Authentication
  - Authorization
  - Session Integrity
- Proved security for (fixed) standard (see security guidelines)

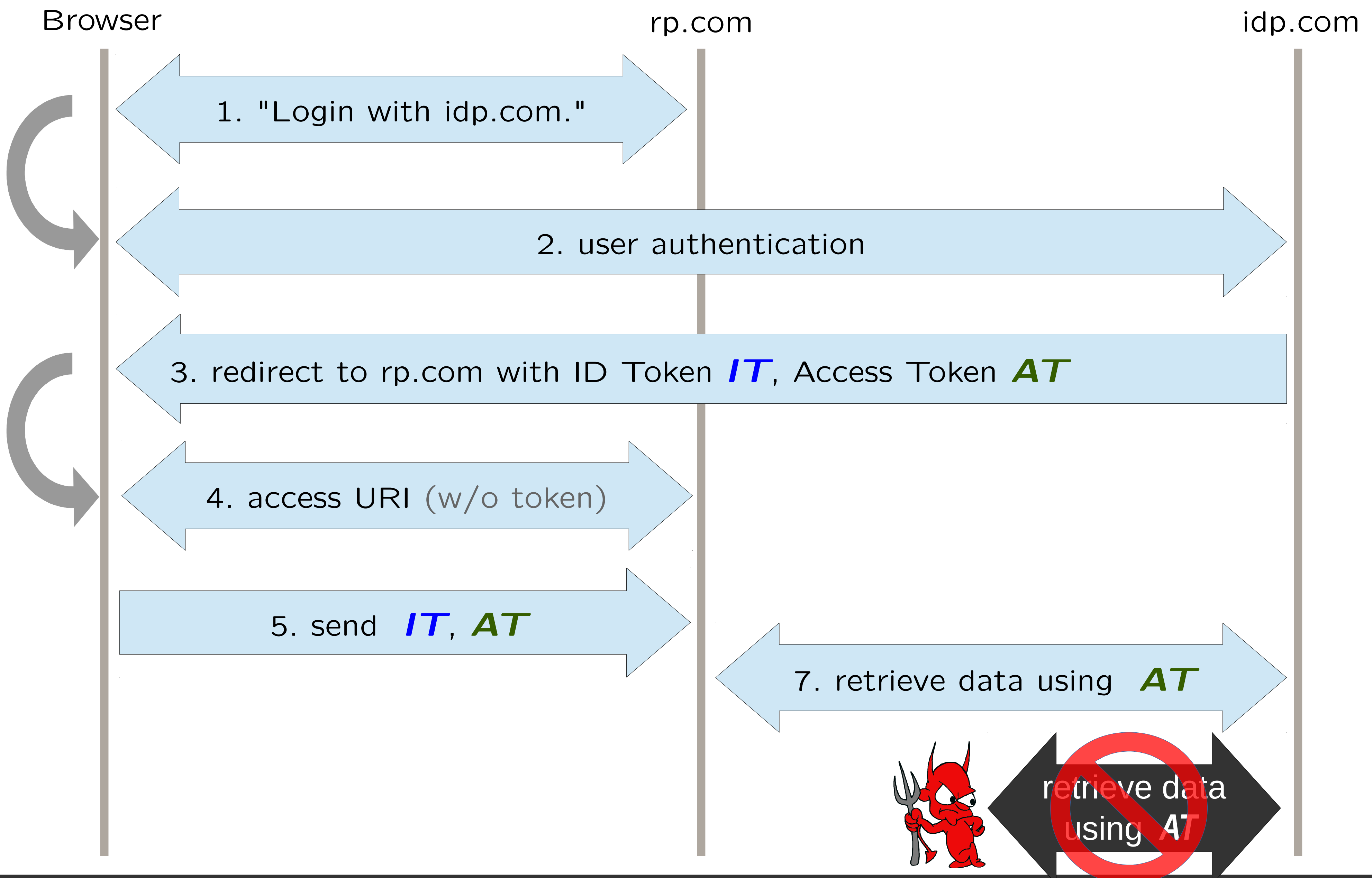
Paper to appear at CSF 2017

All details: TR available at https://sec.informatik.uni-stuttgart.de

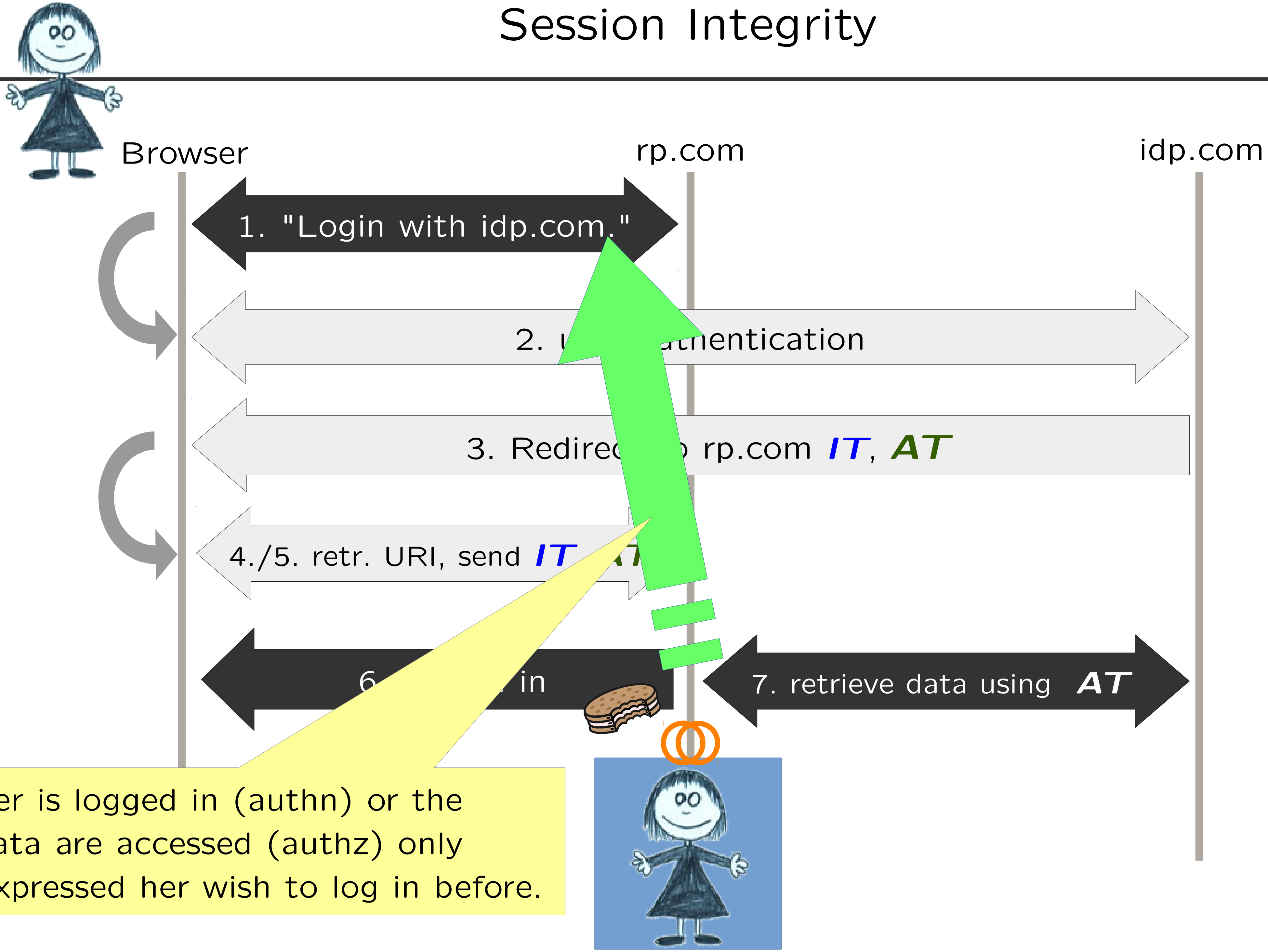
Authentication Property



Authorization Property



Session Integrity



The user is logged in (authn) or the user's data are accessed (authz) only if the user expressed her wish to log in before.

Our Contributions

OpenID Connect 1.0 with Discovery and Dynamic Registration:

- Developed formal model of the standard
  - Based on most comprehensive model of the web to date (extension of S&P 2014).
- Formalized central security properties
  - Authentication
  - Authorization
  - Session Integrity
- Proved security for (fixed) standard (see security guidelines)

Paper to appear at CSF 2017

All details: TR available at https://sec.informatik.uni-stuttgart.de

Formal Model of OIDC

- Formal description of OIDC RP and OIDC IdP (with discovery and dynamic registration)
- Implements best practices and follows security guidelines
- Unbounded number of users (browsers), RPs, and IdPs
- Network attacker for authentication and authorization properties
- (Unbounded number of) web attackers for session integrity properties
- Browsers, RPs, and IdPs can become corrupted

Example: RP Checks an ID Token

Algorithm 20 Relying Party R^r: Check id token.

function CHECK_ID_TOKEN($sessionId$, id_token, s') → **Check id token validity and create service session.**
  let $session := s'.\texttt{sessions}[sessionId]$ → Retrieve session data.
  let $identity := session[\texttt{identity}]$
  let $issuer := s'.\texttt{issuerCache}[identity]$ → Retrieve issuer.
  let $oidcConfig := s'.\texttt{oidcConfigCache}[issuer]$ → Retrieve OIDC configuration for that issuer.
  let $credentials := s'.\texttt{clientCredentialsCache}[issuer]$ → Retrieve OIDC credentials for issuer.
  let $jwks := s'.\texttt{jwksCache}[issuer]$ → Retrieve signing keys for issuer.
  let $data := \mathsf{extractmsg}(id\_token)$ → Extract contents of signed id token.
  if $data[\texttt{iss}] \not\equiv issuer$ **then**
    stop → Check the issuer.
  if $data[\texttt{aud}] \not\equiv credentials[\texttt{client\_id}]$ **then**
    stop → Check the audience against own client id.
  if $\mathsf{checksig}(id\_token, jwks) \not\equiv \top$ **then**
    stop → Check the signature of the id token.
  if $nonce \in session \wedge data[\texttt{nonce}] \not\equiv session[\texttt{nonce}]$ **then**
    stop → If a nonce was used, check its value.
  let $s'.\texttt{sessions}[sessionId][\texttt{loggedInAs}] := \langle issuer, data[\texttt{sub}]\rangle$ → User is now logged in. Store user identity and issuer.
  let $s'.\texttt{sessions}[sessionId][\texttt{serviceSessionId}] := \nu_4$ → Choose a new service session id.
  let $request := session[\texttt{redirectEpRequest}]$ → Retrieve stored meta data of the request from the browser to the redir. endpoint in order to respond to it now. The request's meta data was stored in PROCESS_HTTPS_REQUEST (Algorithm 17).
  let $headers := [\texttt{ReferrerPolicy:origin}]$
  let $headers[\texttt{Set-Cookie}] := [\texttt{serviceSessionId:}\langle \nu_4, \top, \top, \top\rangle]$ → Create a cookie containing the service session id.
  let $m' := \mathsf{enc_s}(\langle \texttt{HTTPResp}, request[\texttt{message}].\texttt{nonce}, 200, headers, \texttt{ok}\rangle, request[\texttt{key}])$ → Respond to browser's request to the redirection endpoint.
  stop $\langle\langle request[\texttt{sender}], request[\texttt{receiver}], m'\rangle\rangle$, s'

Security Guidelines

- Mix-Up attack mitigation (send issuer identifier along with tokens)
- Fresh nonce for state (each time a new flow is stated)
- Referrer Policy to avoid code, token, and state leakage via Referer header
- Explicit user intention tracking (i.e., RP stores user's choice of IdP)
- HTTP redirects with 303 (not 307)
- No open redirectors
- CSRF protection
- No third-party resources on endpoints
- TLS everywhere
- Sessions follow best practices, separate sessions before and after login

Our Contributions

OpenID Connect 1.0 with Discovery and Dynamic Registration:

- Developed formal model of the standard
  - Based on most comprehensive model of the web to date (extension of S&P 2014).
- Formalized central security properties
  - Authentication
  - Authorization
  - Session Integrity
- Proved security for (fixed) standard (see security guidelines)

Paper to appear at CSF 2017

All details: TR available at https://sec.informatik.uni-stuttgart.de

Theorem: OIDC fulfills security properties

- Authentication
- Authorization
- Session Integrity

Proof:

- Secondary security properties
  - discovery and dynamic registration are sane
  - id tokens and state do not leak
- Proof of theorem for each property separately by contradiction

Our Contributions

OpenID Connect 1.0 with Discovery and Dynamic Registration:

- Developed formal model of the standard
  - Based on most comprehensive model of the web to date (extension of S&P 2014).
- Formalized central security properties
  - Authentication
  - Authorization
  - Session Integrity
- Proved security for (fixed) standard (see security guidelines)

Thank you!

Paper to appear at CSF 2017

All details: TR available at https://sec.informatik.uni-stuttgart.de