UNIVERSITÄT SIEGEN

Security of Online Social Networks

Interfaces

Lehrstuhl IT-Sicherheitsmanagment

Universität Siegen

May 3, 2012

Recapitulation

- Graph Model
  - formal
  - data representation
- Login Procedures
- Authentification
  - Web ID
  - Open ID
- TLS

Overview Lesson 04

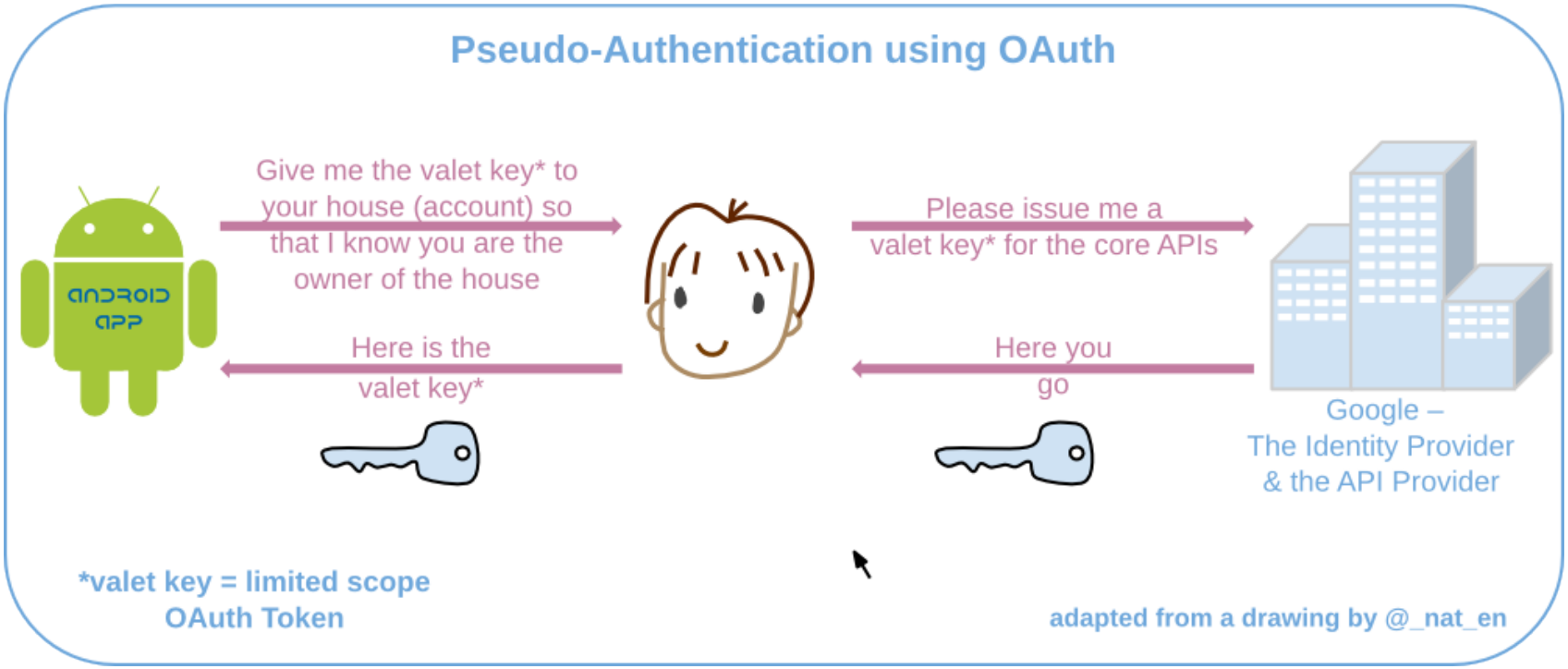
OAuth

Concluding Authentication

Open Graph

Open Social

OAuth vs. OpenID



[http://en.wikipedia.org/wiki/File:OpenIDvs.Pseudo-AuthenticationusingOAuth.svg]

OAuth

OAuth Overview

resource owner (User)

client (Consumer)

server (Service Provider)

client credentials

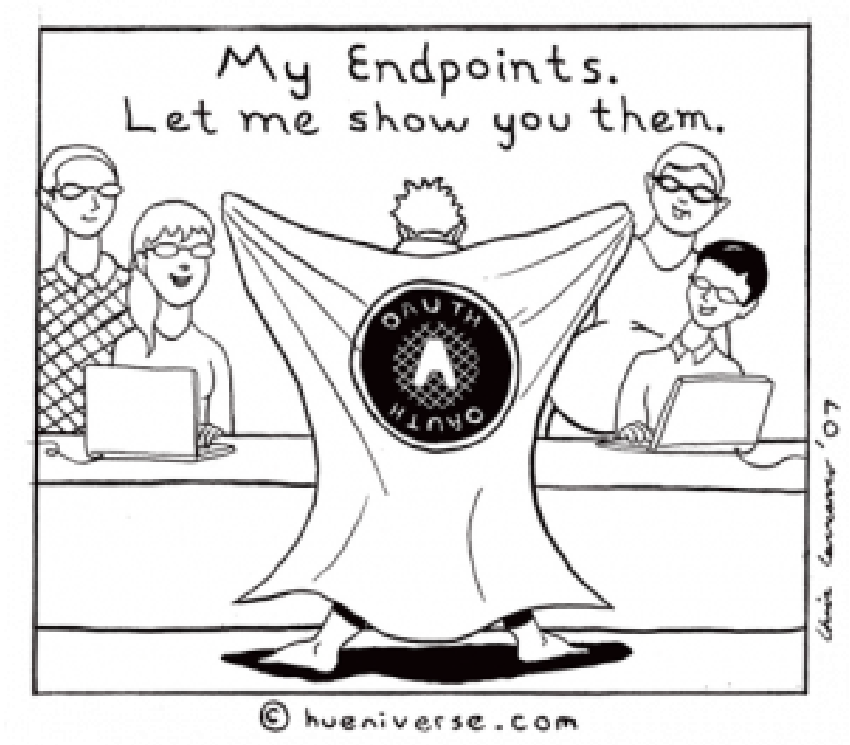
temporary credentials

token credentials

Objectives:

- Redirection-Based Authentication
- partial authorisation to (web)-resource
- no password disclosure to client

OAuth



(source: http://hueniverse.com/oauth/)

State 2012

- OAuth 1.0 RFC 5849 [1] Session Fixation Attack
- OAuth 2.0 Facebook deployed, Microsoft, Google experimental

OAuth Phases

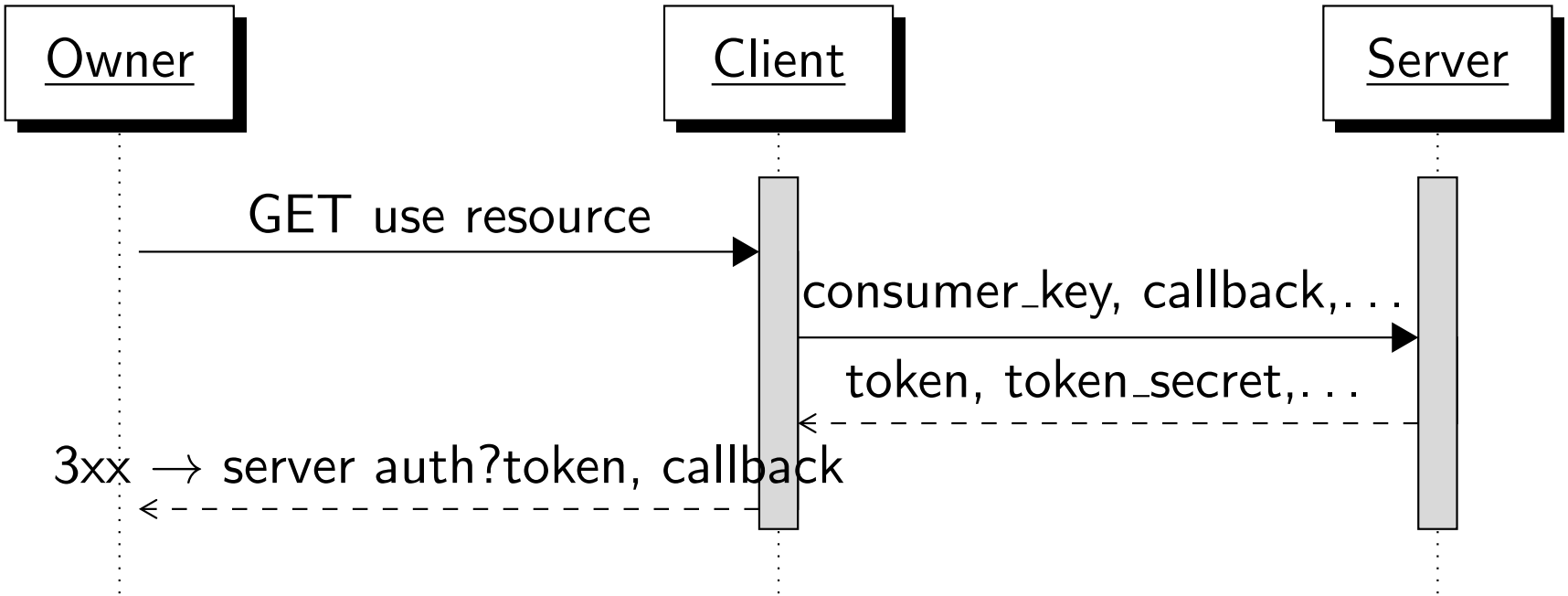
Preliminaries:

0. (Client Credentials)

Server Communication Endpoints:

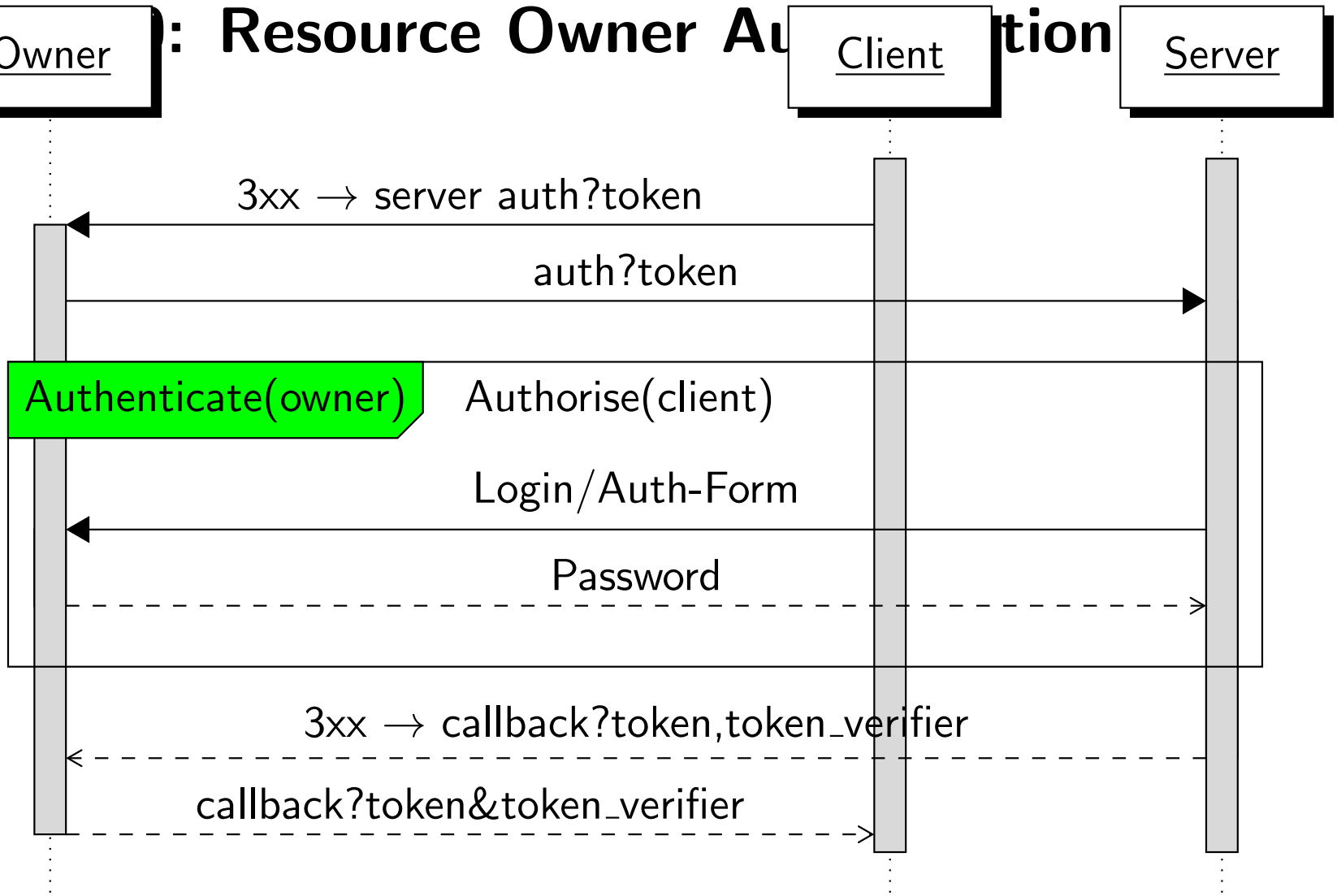
1. Temporary Credential Request
2. Resource Owner Authorisation
3. Token Request

OAuth 1.0: Temporary Credential Request



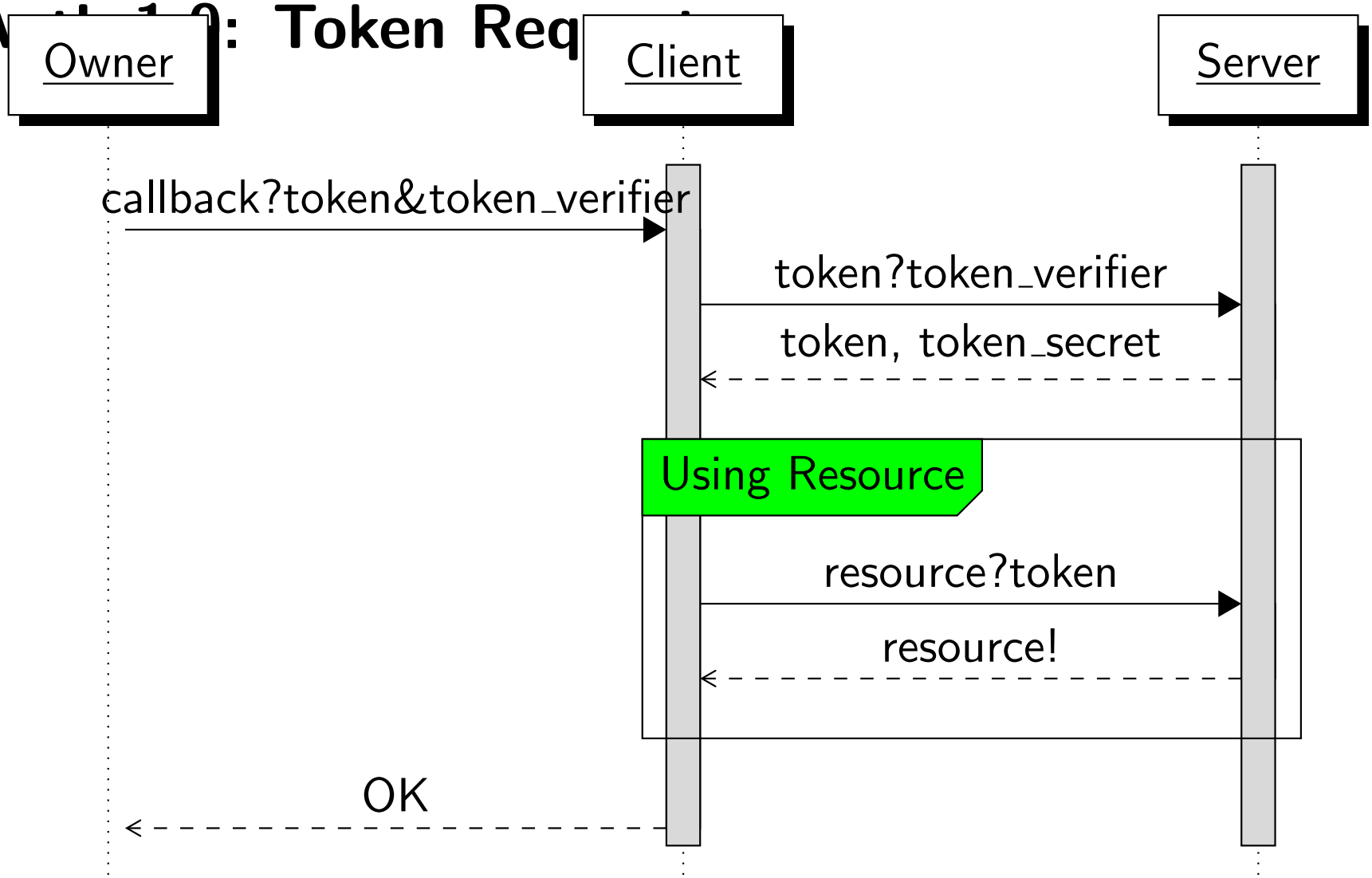
OAuth 1.0: Resource Owner Authorisation

OA : Resource Owner Au tion



OAuth 1.0: Token Request

OA[illegible]: Token Req[illegible]


Owner
Client
Server
callback?token&token_verifier
token?token_verifier
token, token_secret
Using Resource
resource?token
resource!
OK

OAuth 1.0: Session Fixation

- Attacker uses (honest) client to get temp. credential
- Attacker does not follow authorisation redirect
- Attacker tricks resource owner to click redirect
- Owner authorises honest client at server
- Attacker uses saved temp. credential to request token
- Attacker uses token to access resource

(source: `http://oauth.net/advisories/2009-1/`)

OAuth 2.0

Why update?

- OAuth 1.0 too complex
- Scalability issues
- Incompatible to existing Auth. Schemes

(source http://hueniverse.com/2010/05/introducing-oauth-2-0/)

State:

- Almost stable IETF draft v2.22

(see http://tools.ietf.org/html/draft-ietf-oauth-v2-22)

OAuth 2.0: Delta

- Role Separation: Authorization Server
- 6 Different Protocol Flows:
  - User-Agent,
  - Web-Server,
  - Device,
  - Username-Password,
  - Client Credentials,
  - Assertion (eg. SAML)
- Bearer Tokens
- Short-Lived Tokens/Long-Lived authorizations

(source `http://hueniverse.com/2010/05/introducing-oauth-2-0/`)

OAuth 2.0: Flow

(source: OAuth v2.22 (draft)

Concluding Authentication

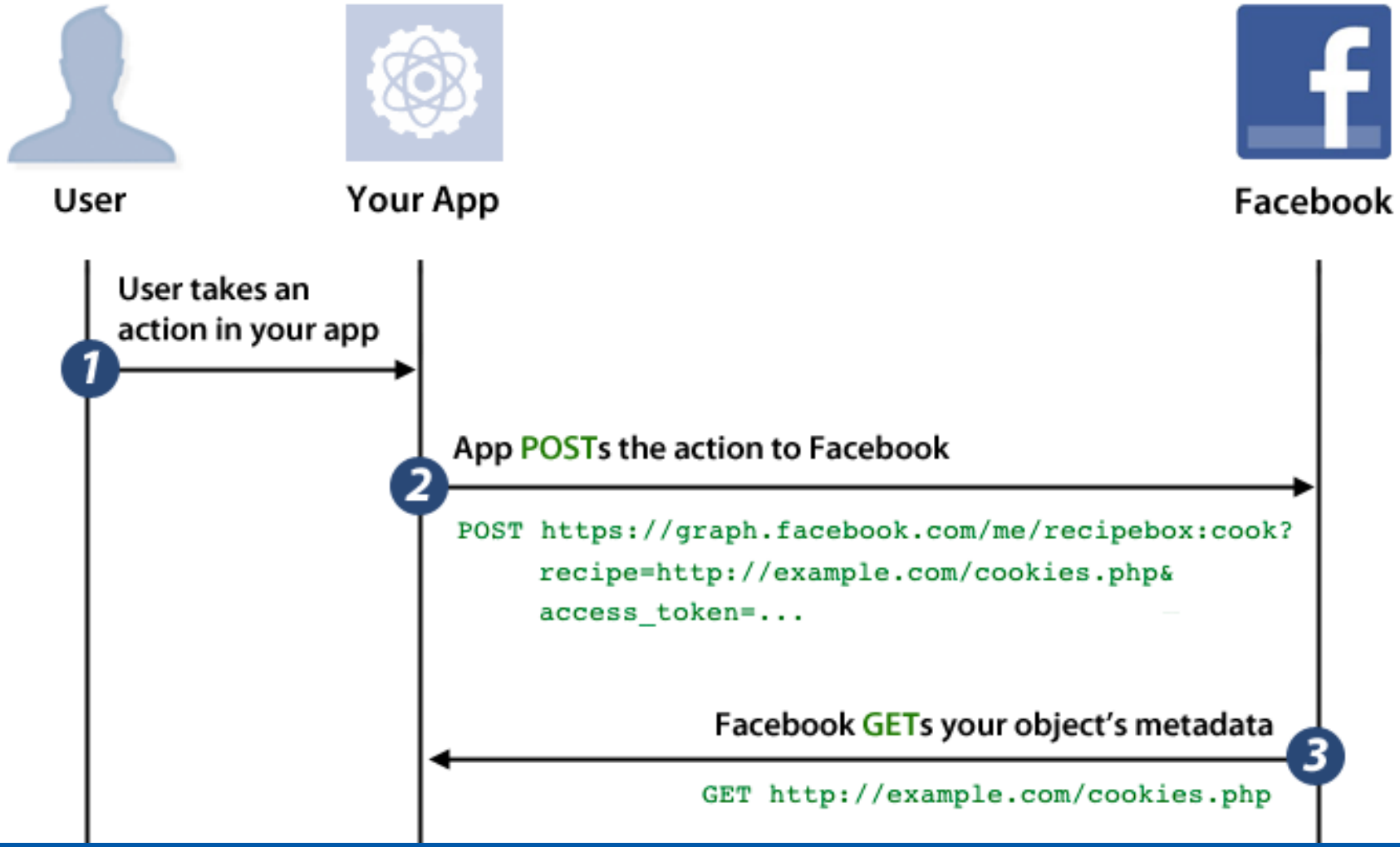
WebID vs. OpenID vs OAuth

	WebID	**OpenID**	**OAuth**
ID Provider	"self"	self/3rd	3rd
Authentication	local key	passwd/assertion	various
"Channel"	TLS	assertion	token

Open Graph

- facebook social graph representation
- `https://developers.facebook.com/docs/opengraph/`
- `http://ogp.me/`

OG mechanics



Open Social

Overview

- set of API
- Community product, no “owner” (OpenSocial Foundation)
  - Contribution Licensing Agreement
  - Non-Assert Agreement
- Standardisation process: “consensus and running code”
  - Gadgets, Container, Social Server
- current version 2.0.1

Gadget web-based software component

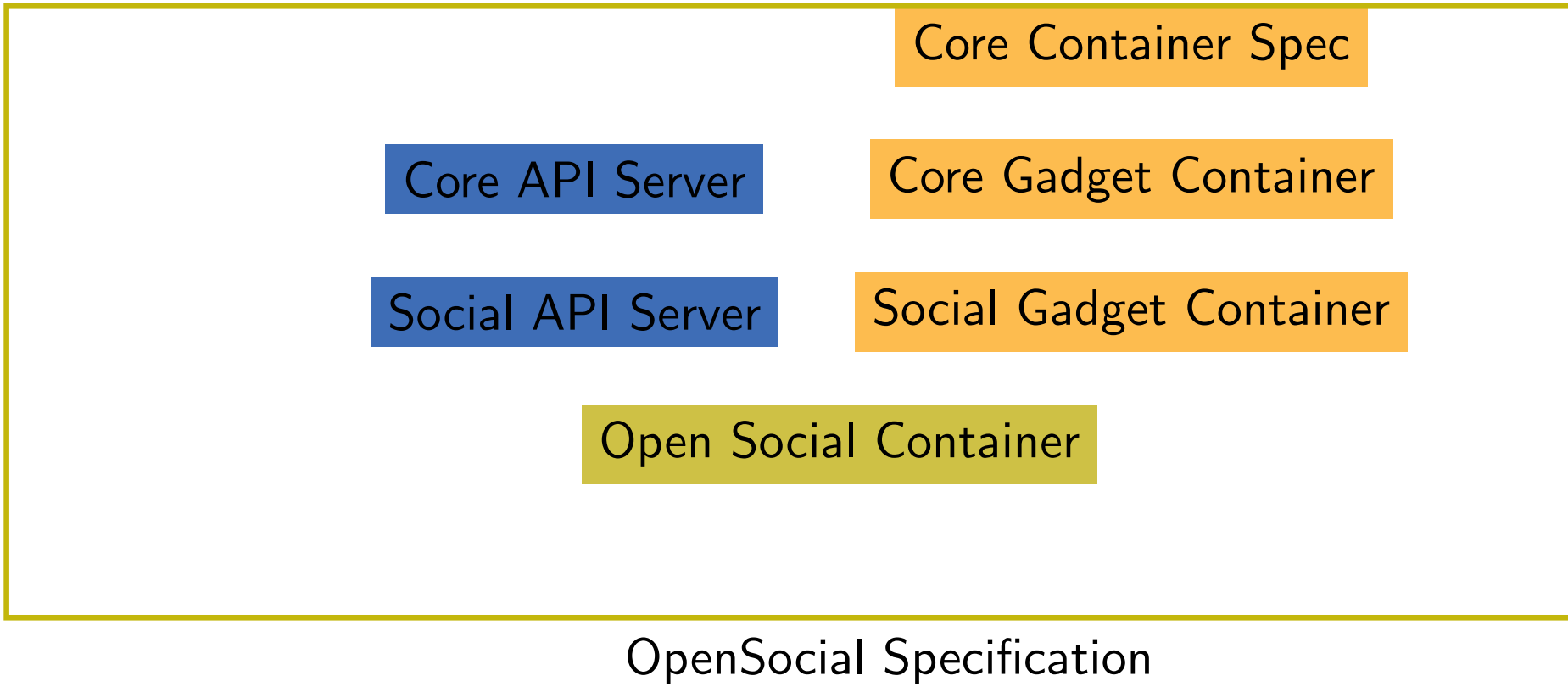
Container context of a gadget (e.g. web-page)

User “viewer of gadget at runtime”

Social API Server

OpenSocial Container

OpenSocial Specification Dependencies



Core API Server Spec

- Protocols: REST or RPC
- Security: OAuth (Access Tokens)
- Content Upload
- Common Parameters: Request-ID, Auth-Token, Content-Type, Return-Object, Invalidation-Key-List, HTTP-Status-Code
- Request Parameters: Updated-Since, Encoding Format
- Discovery
- Services
- Concurrency Control: HTTP/AtomPup (MAY)

REST Protocol

Request:

```
GET /api/people/@me/@self?fields=name HTTP/1.1
Host: api.example.org
Authorization: hh5s93j4hdidpola
Content-Type: application/json
```

Response:

```
HTTP/1.x HTTP-Status-Code
[ "Content-Type: " Content-Type ]
[ REST-Response-Payload ]
```

Social API Server Spec

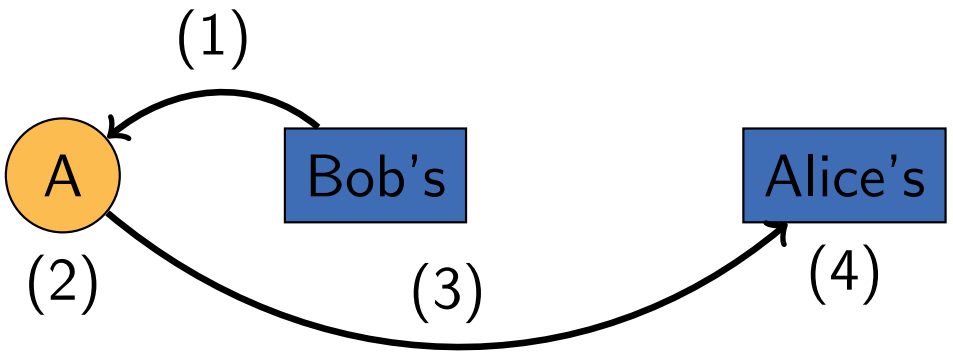
- OpenSocial Social API Server Specification 2.0.1 [2]
- People, Groups, Activity Streams, AppData, Albums, MediaItems, Messages

REST Create Relationship

```
REST-HTTP-Method        = "POST"
REST-URI-Fragment       = "/people/" User-Id "/" Group-Id
REST-Query-Parameters   = null
REST-Request-Payload    = Person
```

```
POST /rest/people/@self/@friends HTTP/1.1
HOST api.example.org
Authorization: hh5s93j4hdidpola
Content Type: application/xml
<entry xmlns="http://ns.opensocial.org/2008/opensocial">
  <id>example.org:34KJDCSKJN2HHF0DW20394</id>
</entry>
```

Reflective Create Relationship (WebID+REST)



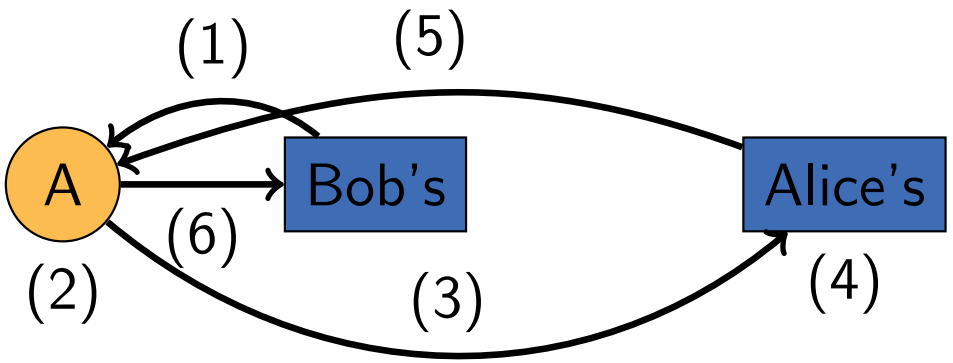
(1) Bob's Page (for Alice)

(2) click 'know him!'

(3)
```
POST /alice/rest/people/@alice#me/@knows HTTP/1.1
HOST alice.info
Authorization: alice.info/alice#me
Content Type: application/xml
[...]
<foaf:Person>
  <foaf:homepage rdf:resource="http://bob.org/bob#me"/>
  <foaf:mbox_sha1sum>8a75535cfeb076f13del68aa113e91abaeb7340</foaf:mbox_sha1sum>
</foaf:Person>
```

(4) append "knows Person bob"

Reflective Create Relationship (WebID+REST)



(1) Bob's Page (for Alice)

(2) click 'know him!'

(3)
```
POST /alice/rest/people/@alice#me/@knows HTTP/1.1
HOST alice.info
Authorization: alice.info/alice#me
Content Type: application/xml
[...]
<foaf:Person>
  <foaf:homepage rdf:resource="http://bob.org/bob#me"/>
  <foaf:mbox_sha1sum>8a75535cfeb076f13del68aa113e91abaeb7340</foaf:mbox_sha1sum>
</foaf:Person>
```

(4) append "knows Person bob"

(5) Redirect back to Bob's
Return-Object: Person Bob

(6) Request Bob's Page

OpenSocial with WebID

- Authorisation?
- Identities in URLS?

RPC create friendship (WebID)

```
POST /bob/rpc HTTP/1.1
Host: bob.org
Authorization: <auth token>
Content-Type: application/json
{
  "method" : "people.create",
  "id" : "createFriend"
  "params: {
    "userId" : "@alice.info/alice#me",
    "groupId" : "@knows",
    "person" : {
      "id" : "@bob.org/bob#me"
} } }
```

Sending a message

```
REST-HTTP-Method        = "POST"
REST-URI-Fragment       = "/messages/" User-Id
                                       "/@self/@outbox"
REST-Query-Parameters   = null
REST-Request-Payload    = Message
```

HTTP-Status Codes

400 BAD REQUEST invalid syntax

401 UNAUTHORIZED missing OAuth Credentials/no access

403 FORBIDDEN insufficient context rights

404 NOT FOUND resource missing

405 METHOD NOT ALLOWED response with Allow header

409 CONFLICT response with details

500 INTERNAL SERVER ERROR generic

501 NOT IMPLEMENTED optional feature missing

Conclusion Protocols

- Authentication
  - towards Identity Provider
  - towards 3rd Party
- Data Exchange Formats
- WebID needs additional API

Literatur I

 *The OAuth 1.0 Protocol*, IETF Informational RFC 5849, April 2010.

 *OpenSocial Social API Server Specification 2.0.1*, OpenSocial Foundation Std., Rev. 2.0.1. [Online]. Available: http://opensocial-resources.googlecode.com/svn/spec/2.0.1/Social-API-Server.xml