Cookies, sessions and third-party authentication

Claudia Hauff
cse1500-ewi@tudelft.nl

TU Delft

Cookies, sessions and third-party authentication

Claudia Hauff
cse1500-ewi@tudelft.nl

TU Delft

Learning goals

- **Decide** for a given usage scenario whether cookies or sessions are suitable
- **Explain and implement** cookie usage
- **Explain and implement** session usage
- **Implement** third-party authentication

Introduction to cookies and sessions

Recall: HTTP

- HTTP is **stateless**
- HTTP requests contain **all necessary information**
- Web servers do not need to keep track of the requests received
- Advantage: simple server architecture
- Disadvantage: clients have to resend **the same information** in every request

We do a lot of things on the web requiring a known state

- bol.com keeps your shopping cart full, even when you leave the website
- statcounter.com (a popular user tracking toolkit) can exclude a particular visitor from being tracked
- Games can be continued when revisiting them
- …

Not the stateless web is the norm, but the stateful one.
Cookies and sessions are today's major technologies to achieve a stateful web.

Cookies cannot …

- **Execute** programs
- Access information from a **user's hard drive**
- Generate spam
- Be **read by arbitrary parties**
  - Only the server setting the cookie can access it
  - But: beware of **third-party cookies**

Cookies = server's short term memory

Cookies and sessions are ways to **introduce state** on top of the stateless HTTP.

Cookie: a short amount of text **(key/value)** **sent by the server** and **stored by the client** for some amount of time.

Minimum client storage requirements (`RFC6265`)

- Store at least **4096 bytes** per cookie
- Store at least **50** cookies per domain
- Store at least **3000** cookies total.

They have been around since the early 1990s.

bol.com bv (NL) | https://www.bol.com/nl/

Search

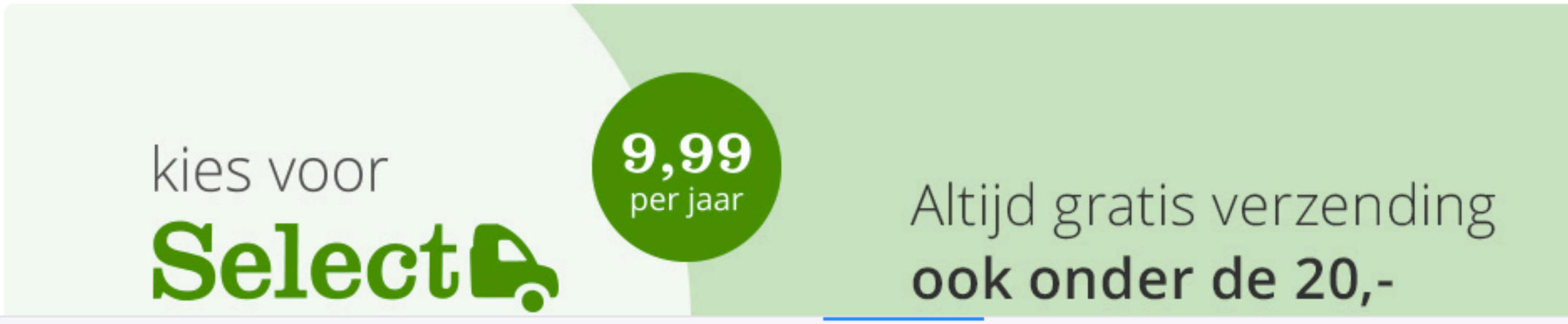
bol.com

Waar ben je naar op zoek? Alles

Categorieën | Cadeaus & Inspiratie | Aanbiedingen | Verkopen | Zakelijk | Cadeaubon | Bestelstatus | Kla

Demo time 4

Bespaar continu | Nieuwe collectie tassen | Dieren Deals | Alles voor je zaak | Select



Inspector | Console | Debugger | { } Style Editor | Performance | Memory | Network | Storage

Filter URLs | All | HTML | CSS | JS | XHR | Fonts | Images | Media | WS | Other

Status	Method	F...	D...	Cause	Type	Transferred	Size	0 ms	40.96 s
301	GET	/		ww...document	html	36.49 KB	150.03 KB	→ 19 ms	
200	GET	/nl/		ww...document	html	41.02 KB	150.03 KB	→ 65 ms	
200	GET	27945-...		ww...img	jpeg	49.24 KB	48.71 KB	→ 57 ms	
200	GET	180911_...		ww...img	jpeg	43.53 KB	43 KB	→ 67 ms	
200	GET	bulkale...		ww...img	jpeg	27.63 KB	27.11 KB	→ 70 ms	
200	GET	mediah...		s.s... stylesheet	css	54.48 KB	378.68 KB	→ 340 ms	
200	GET	overrid...		s.s... stylesheet	css	2.86 KB	7.57 KB	→ 308 ms	
200	GET	180821...		s.s... img	jpeg	54.73 KB	54.13 KB	→ 369 ms	
200	GET	92000...		s.s... img	jpeg	4.51 KB	3.91 KB	→ 483 ms	
200	GET	billie_ve...		s.s... img	png	31.13 KB	30.54 KB	→ 500 ms	
200	GET	357636...		s.s... script	js	59.14 KB	173.03 KB	→ 200 ms	
200	GET	960578...		s.s... script	js	64.50 KB	224.46 KB	→ 193 ms	

Headers | Cookies | Params | Res

Filter cookies

Response cookies

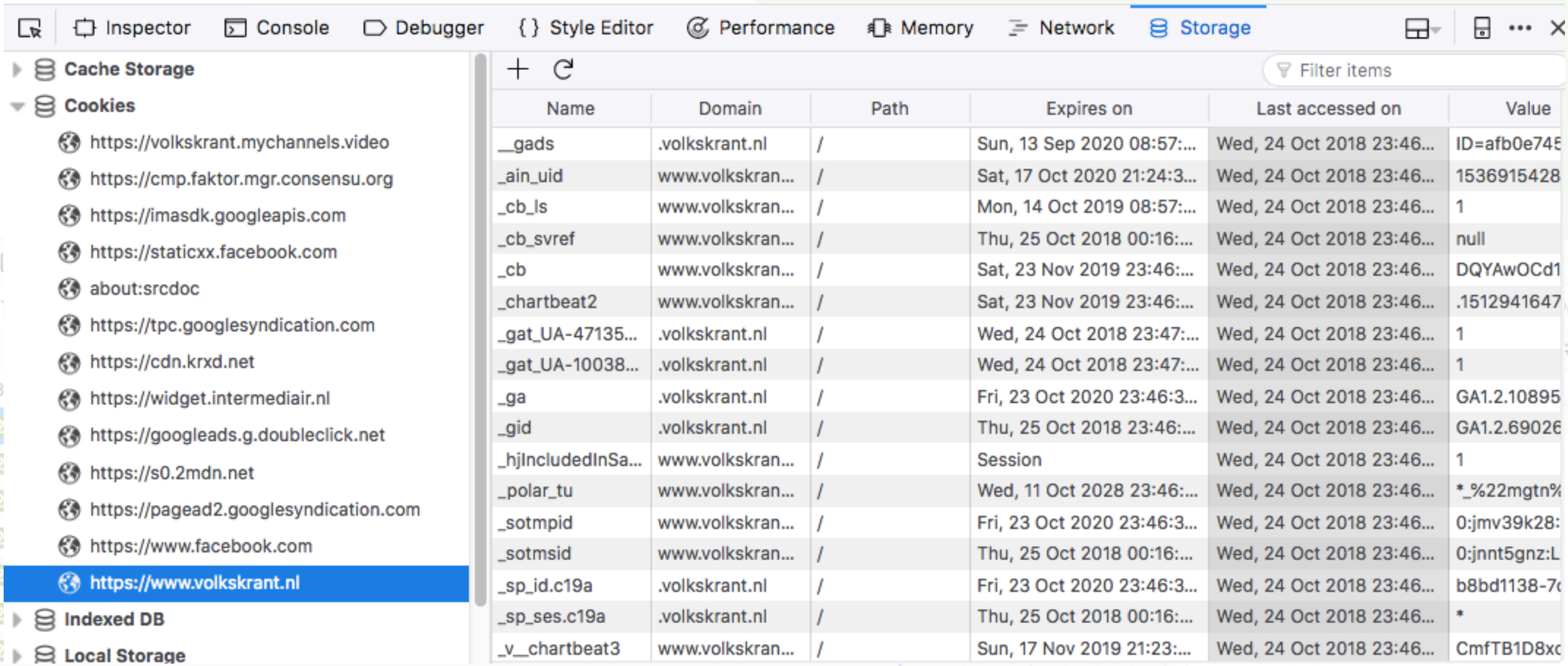
- abt_aid_session:
  - expires: 2018-09-18T18:03:05.000Z
  - path: /
  - secure: true
  - value: true
- BUI:
  - domain: .bol.com
  - expires: 2021-09-17T17:33:05.000Z
  - path: /
  - secure: true
  - value: ygo0gs0d8clbwz8fkw2hjxhylvyd

Request cookies

- __gads: ID=e5db5a191b6604ed:T=153349
- _ga: GA1.2.905755118.1531860555
- _gid: GA1.2.713913342.1537290939

101 requests | 3.71 MB / 1.40 MB transferred | Finish: 2.07 min | DOMContentLoaded: 755 ms | load: 1.40 s

Storage Inspector dev tool



Cookie security

- **Visible** to clients
- Clients can **delete/disallow** them
- Clients can **alter** them
  - Line of attack: **servers** should not send sensitive information in cookies
- **Sessions** are preferable to cookies
  - Sessions make use of cookies
  - Cookie contains only the session identifier

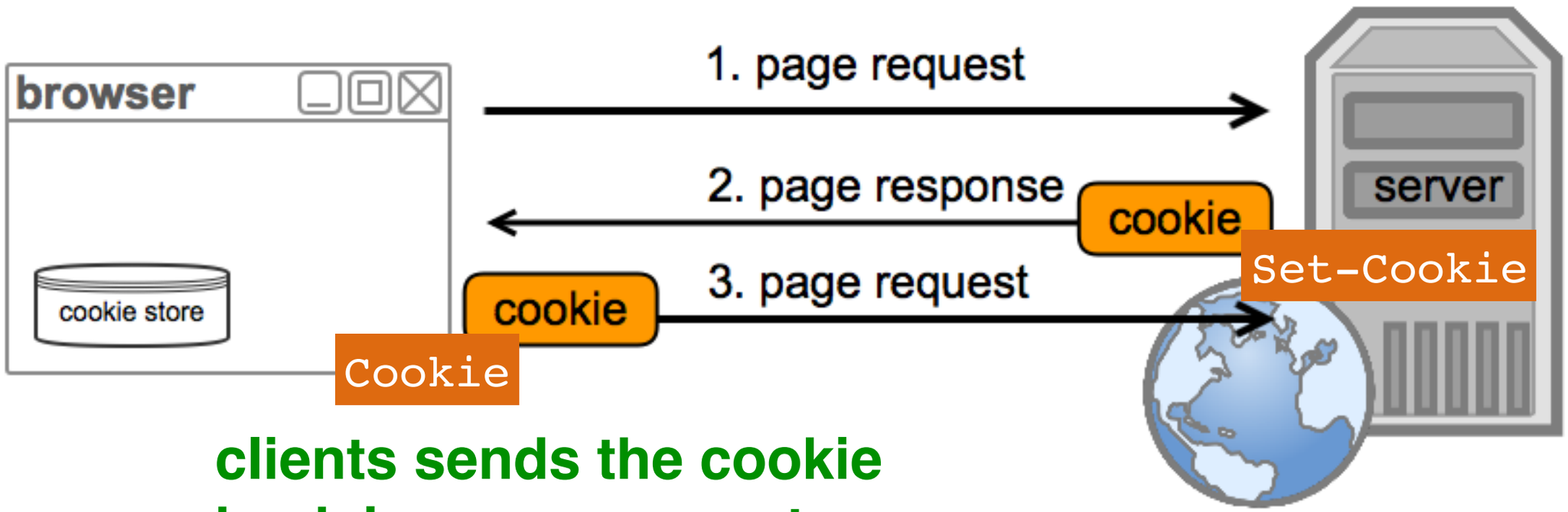
A word of warning: RFC6265

"This document defines the HTTP **`Cookie`** and **`Set-Cookie`** header fields. These header fields can be used by HTTP servers **to store state** (called cookies) at HTTP user agents, letting the servers maintain a **stateful session** over the mostly stateless HTTP protocol.

Although cookies have many historical infelicities that degrade their security and privacy, the `Cookie` and `Set-Cookie` header fields are widely used on the Internet. "

Cookie flow

server sends a cookie once;
resends when key/value changes



clients sends the cookie
back in every request

- Encoded in the **HTTP header**
- Web frameworks offer designated methods
- Cookies are **bound** to a **site domain name** (security feature)

Cookies in more detail

Transient vs. persistent cookies

Not the tab or window!

- **Transient** (or session) cookies:
  - Exist in memory only; deleted when the browser is closed
  - Cookies are transient if **no expiration date** is defined
- **Persistent** cookies:
  - Remain intact after the browser is closed
  - Have a **maximum age**
  - Are **send back to the server** as long as they are **valid**

Cookie fields

- **`cookie-name=cookie-value`** the only **required** field
- **`Expires`** (expiration date) or **`Max-Age`** (*s* until it expires)
- **`Domain`** the cookie is associated with; cookies can only be assigned to the **same domain** the server is running on
- **`Path`** the cookie is applied to (automatic wildcarding):
  `/` matches all pages, `/todos` all pages within `todos`, etc.
- Possible flags:
  - **`Secure`**
  - **`httpOnly`**
  - **`Signed`**

Making cookies more robust

`Secure` flag via HTTP: the cookie will **not** be sent

- **`Secure`** cookies:
  - Ensures that cookies are sent via HTTPS
- **`HttpOnly`** cookies:
  - Cookies are **not accessible** to non-HTTP entities (e.g. client-side **JavaScript**)
  - Minimises the threat of cookie theft
  - Should always be applied to session management cookies
- **`Signed`** cookies (appended `HMAC[value]`):

Hash Message Authentication Code

  - Server can check whether the **value** has been **tampered** with by the client

```
s%3Amonster.TdcGYBnkcvJsd0%2FNcE2L%2Bb8M55geOuAQt48mDZ6RpoU
```

Cookie field `Domain`

- **Origin**: **request domain** of the cookie (a cookie is always applicable to its origin server)
  `GET http://www.my_site.nl/todos` → `www.my_site.nl`
  - Port or scheme can differ, the received cookie is also applicable to `https://www.my_site.nl:3005`
- **Domain attribute**: a cookie's `Domain` attribute has to cover the origin domain
  - If not set, a cookie is only applicable to its origin domain (a cookie from `www.my_site.nl` is not applicable to `my_site.nl`)
  - If set, a cookie is applicable to the domain listed in the attribute and all its **subdomains**

```
GET http://www.my_site.nl/todos
Set-Cookie: name=value; Path=/; Domain=my_site.nl
```

applicable to `www.my_site.nl` `todos.my_site.nl`
`serverA.admin.todos.my_site.nl`

```
npm install cookie-parser
```

Cookies in express

```
1  var express = require("express");
2  var http = require("http");
3  var credentials = require('./credentials.js');
4  var cookies = require("cookie-parser");
5
6  var app = express();
7  app.use(cookies(credentials.cookieSecret));
8  http.createServer(app).listen(port);
9
```

`cookie-parser` middleware

```
10 app.get("/sendMeCookies", function (req, res) {
11     console.log("Handing out cookies");
12     res.cookie("chocolate", "kruemel");
13     res.cookie("signed_choco", "monster", { signed: true});
14     res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18     console.log("++++ unsigned ++++");
19     console.log(req.cookies);
20     console.log("++++ signed ++++");
21     console.log(req.signedCookies);
22     res.send();
23 });
```

creating cookies

signing a cookie

reading cookies

Cookies in express

```
npm install cookie-parser
```

```
module.exports = {
    cookieSecret: 'abc'
};
```

```
1  var express = require("express");
2  var http = require("http");
3  var credentials = require('./credentials.js');
4  var cookies = require("cookie-parser");
5
6  var app = express();
7  app.use(cookies(credentials.cookieSecret));
8  http.createServer(app).listen(port);
9
10 app.get("/sendMeCookies", function (req, res) {
11     console.log(“Handing out cookies");
12     res.cookie("chocolate", "kruemel");
13     res.cookie("signed_choco", "monster", { signed: true});
14     res.send();
15 });
16
17 app.get("/listAllCookies", function (req, res) {
18     console.log("++++ unsigned ++++");
19     console.log(req.cookies);
20     console.log("++++ signed ++++");
21     console.log(req.signedCookies);
22     res.send();
23 });
```

Accessing and deleting cookies in express

- **Accessing the value** of a particular key/value pair:
  `var val = req.signedCookies.signed_choco;`

cookie key

- **Deleting** a cookie:
  `res.clearCookie('chocolate');`

delete in the **response**!

```
res.clearCookie = function clearCookie(name, options) {
  var opts = merge({ expires: new Date(1), path: '/' }, options);

  return this.cookie(name, '', opts);
};
```

A more pessimistic view on cookies

Evercookie

"`evercookie` is a javascript API available that produces **extremely persistent cookies** in a browser. Its goal is to **identify a client** even **after they've removed standard cookies** […]
`evercookie` accomplishes this by storing the cookie data in **several types of storage mechanisms** that are available on the local browser.
Additionally, if `evercookie` has found the user has removed any of the types of cookies in question, it **recreates** them using each mechanism available."

Source: https://github.com/samyk/evercookie

Evercookie

Browser Storage Mechanisms

Client browsers must support as many of the following storage mechanisms as possible in order for Evercookie to be effective.

- Standard HTTP Cookies
- Flash Local Shared Objects
- Silverlight Isolated Storage
- CSS History Knocking
- Storing cookies in HTTP ETags (Backend server required)
- Storing cookies in Web cache (Backend server required)
- HTTP Strict Transport Security (HSTS) Pinning (works in Incognito mode)
- window.name caching
- Internet Explorer userData storage
- HTML5 Session Storage
- HTML5 Local Storage
- HTML5 Global Storage
- HTML5 Database Storage via SQLite
- HTML5 Canvas - Cookie values stored in RGB data of auto-generated, force-cached PNG images (Backend server required)
- HTML5 IndexedDB
- Java JNLP PersistenceService
- Java exploit CVE-2013-0422 - Attempts to escape the applet sandbox and write cookie data directly to the user's hard drive.

Source: https://github.com/samyk/evercookie

First-party cookies are cookies that belong to the same domain that is shown in the browser's address bar.

Third-party cookies

Set-Cookie: x.org

first party

Web server x.org

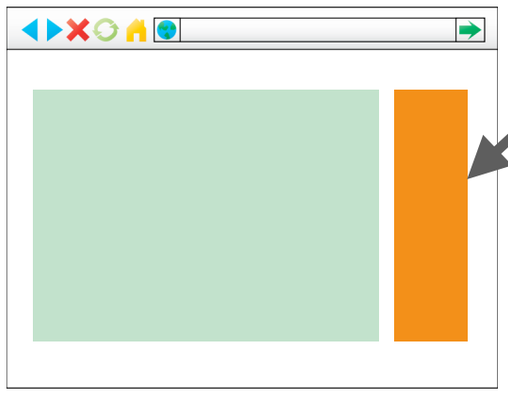
Set-Cookie: ads.agency.com

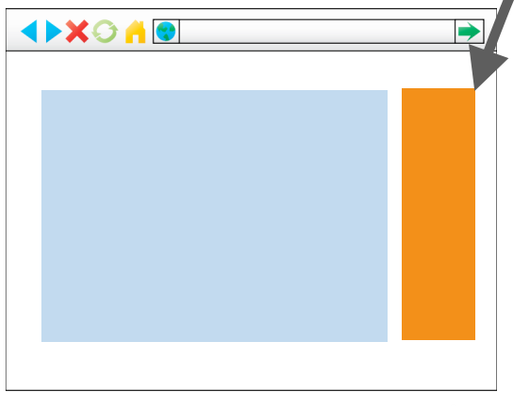
third party

Global Ad Agency
ads.agency.com

served by the same ad agency

Set-Cookie: ads.agency.com





user visits three different websites

based on the cookies a complete user profile can be created

Set-Cookie: ads.agency.com

Client-side cookies

Cookies in JavaScript

- Not always necessary to receive cookies from a server
- Cookies can be **set in the browser**
- Standard use case: remember form input

```
//set TWO(!) cookies
document.cookie = "name1=value1";
document.cookie = "name2=value2; expires=Fri, 24-Jan-2019 12:45:00 GMT";

//delete a cookie by RESETTING the expiration date
document.cookie = "name2=value2; expires=Fri, 24-Jan-1970 12:45:00 GMT";
```

Reading cookies in JavaScript

- `document.cookie["name1"]` does **not work**
- **String** returned by `document.cookie` needs to be parsed

```
visits=123; whenMin=21
```

```
var cookiesArray = document.cookie.split('; ');
var cookies=[];

for(var i=0; i < cookiesArray.length; i++) {
    var cookie = cookiesArray[i].split("=");
    cookies[cookie[0]]=cookie[1];
}
```

Sessions

Introduction (once again)

- **Scenario**: users interact with a web application for a short amount of time (a session)
- **Goals**:
  - Track the user without relying (too much) on cookies
  - Allow large amounts of data to be stored
- **Problem**: without cookies the server cannot tell clients apart
- **Solution**: **hybrid approach** between cookies and server-side saved data

Sessions in one slide

- Cookies are used to store a **single ID** on the **client**
- **Remaining user information** is stored **server-side**

Establishing a session

1. **Client** requests a first page from the server
2. Server creates **unique session ID** and initiates the storage of the session data for that client
3. Server sends back a page with a **cookie** containing the session ID
4. From now on, the client sends **page requests together with the cookie**
5. Server can use the **ID to personalise** the response
6. A **session ends** when no further requests with that session ID come in (timeout)

Sessions in Express with memory stores

- Easy to set up in Express
- Same drawback as any in-memory storage: not **persistent** across machine failure
- Middleware component **`express-session`** https://github.com/expressjs/session
- Most common use case is **authentication**

Sessions in Express with memory stores

```
npm install cookie-parser
npm install express-session
```

```
var express = require("express");
var http = require("http");
var credentials = require("./credentials");
var cookies = require("cookie-parser");
var sessions = require("express-session");

var app = express();
app.use(cookies(credentials.cookieSecret));
app.use(sessions(credentials.cookieSecret));
http.createServer(app).listen(3001);

app.get("/countMe", function (req, res) {
    var session = req.session;
    if (session.views) {
        session.views++;
        res.send("You have been here " +
              session.views + " times (last visit: " + session.lastVisit + ")");
        session.lastVisit = new Date().toLocaleDateString();
    }
    else {
        session.views = 1;
        session.lastVisit = new Date().toLocaleDateString();
        res.send("This is your first visit!");
    }
});
```

cookie & session setup

session object available on **`req`** object only

session exists!

session does not yet exist

Third-party authentication

Quora

A place to share knowledge and better understand the world.

Continue with Google

Continue with Facebook

Continue With Email. By signing up you indicate that you agree to Quora's Terms of Service and acknowledge Quora's Privacy Policy.

Login

Email

.....

Forgot Password?

Login

You are now logged out of this browser, but are still logged in with other browsers.
Logout of all browsers.

New: Hindi, Indonesian, and Portuguese

What password?

- **Weakest link** in an authenticated application is the **user's password**
- **Application-based decision**
  - Does the application need authentication?
  - Are cookies/sessions enough?
  - If authentication is needed, should third-party authentication be used?

Third-party authentication

- Authenticating users through popular social Web services (Twitter, Facebook, Google, etc.)
- **Easy** to develop for popular platforms
- **Trusted** social Web platforms **provide authentication**, no need to store passwords or employ particular security measures
- **However**: some users may not use social Web platforms or do not like to hand over their data

OAuth 2.0 Authorization Framework

"The OAuth 2.0 authorization framework enables a **third-party application** to obtain **limited access** to an HTTP service, either **on behalf** of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf."

Source: https://tools.ietf.org/html/rfc6749

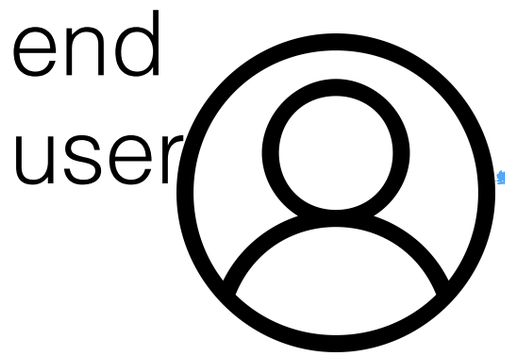
OAuth 2.0 roles

- **Resource owner**: entity that grants access to a protected resource
- **Resource server**: server hosting protected resources, capable of accepting and responding to protected resource requests using **access tokens**

a string denoting a specific scope, lifetime and other access attributes

- **Client**: application making protected resource requests on behalf of the resource owner **and with her authorisation**
- **Authorization server**: server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

OAuth 2.0 roles exemplified



printing service

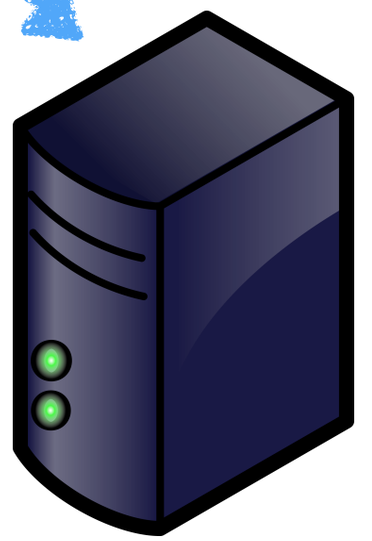
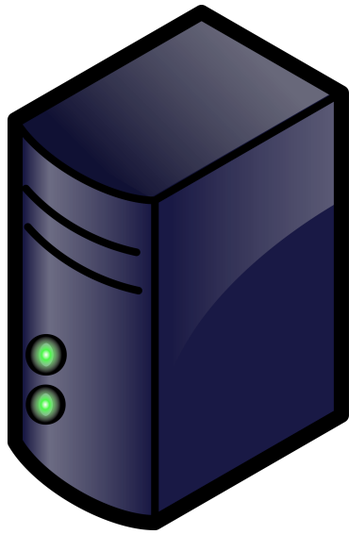


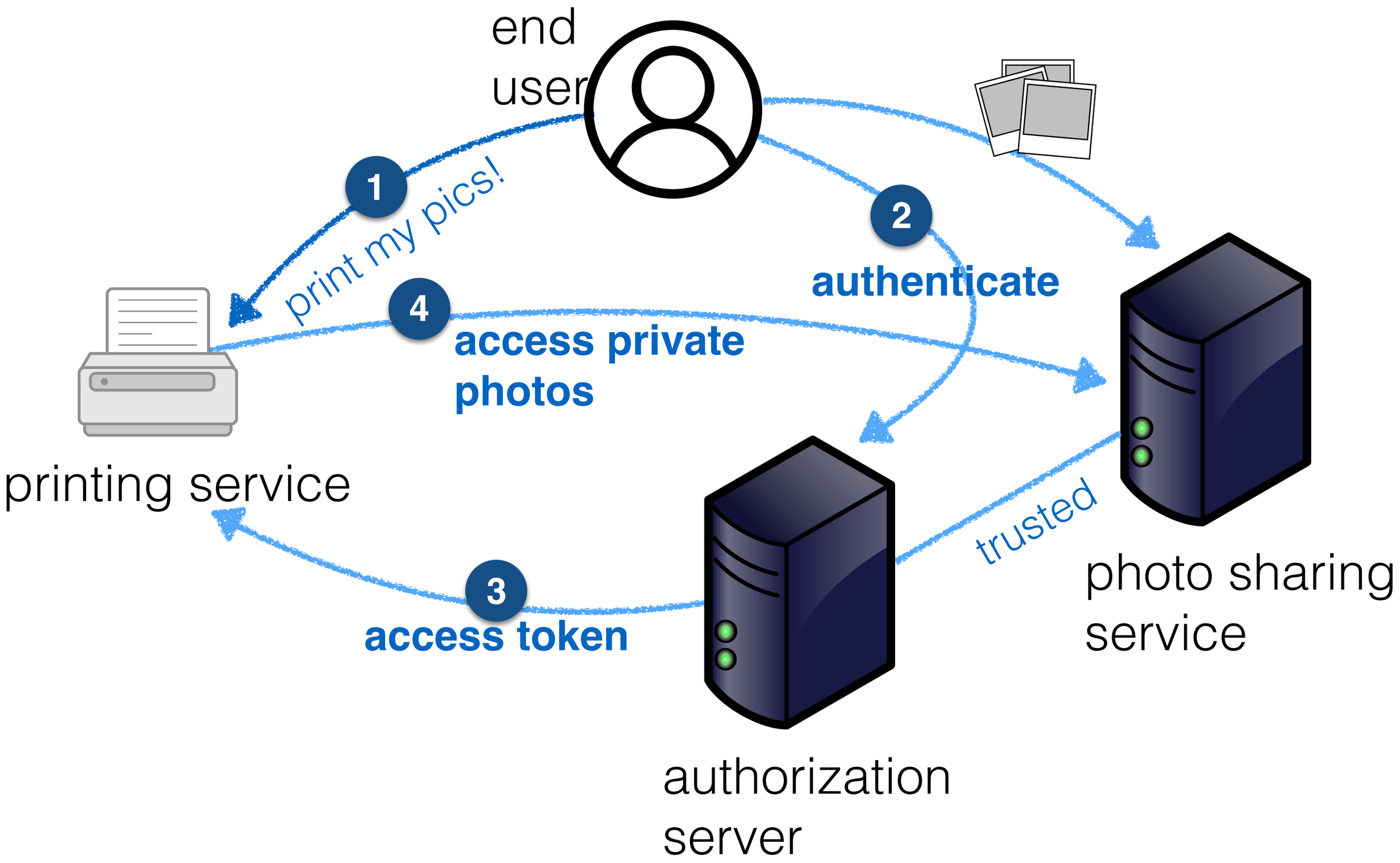
photo sharing service

Scenario: user wants to grant permission to a printing service to print its private photos stored at a photo-sharing service **without giving away her username/password**.

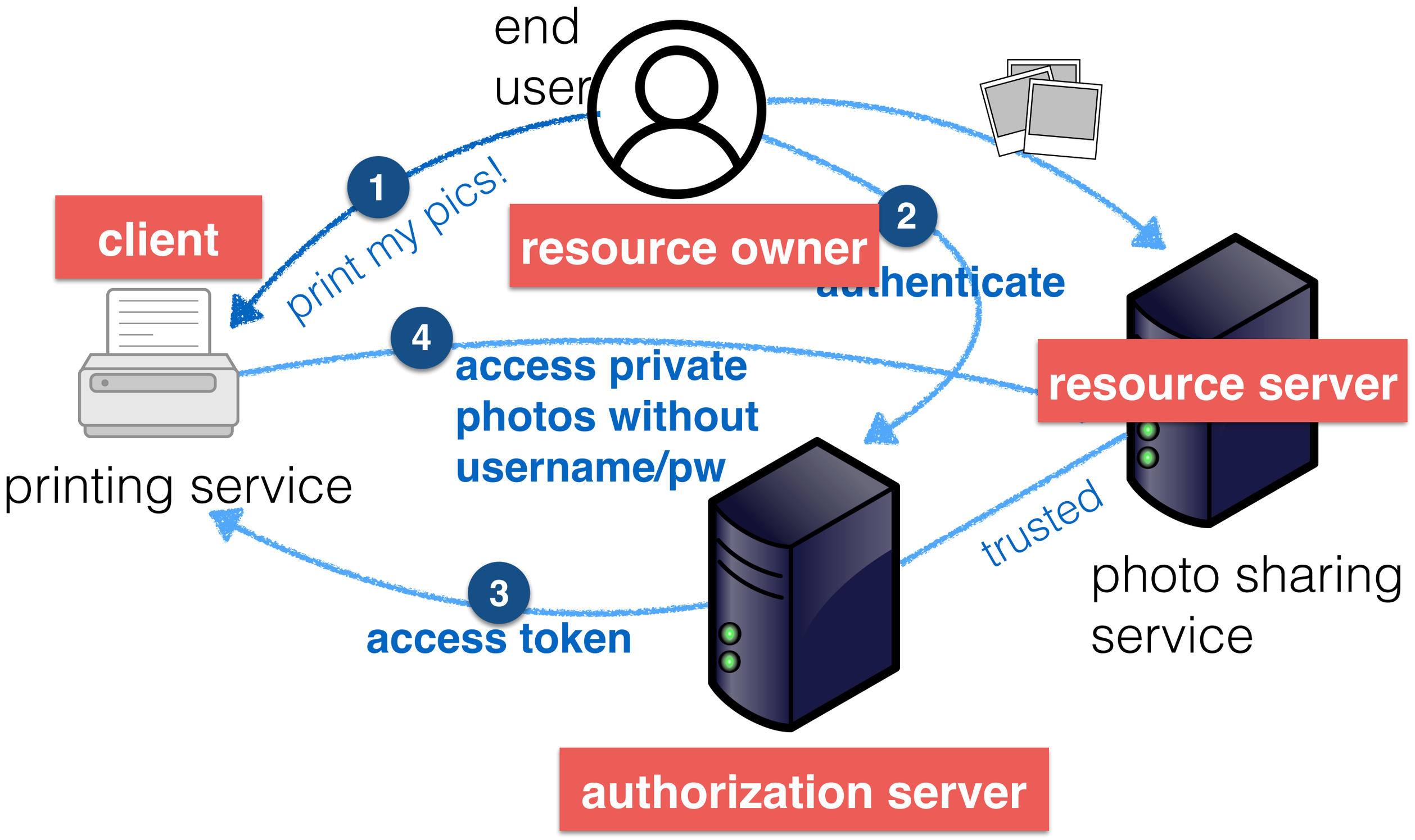


authorization server

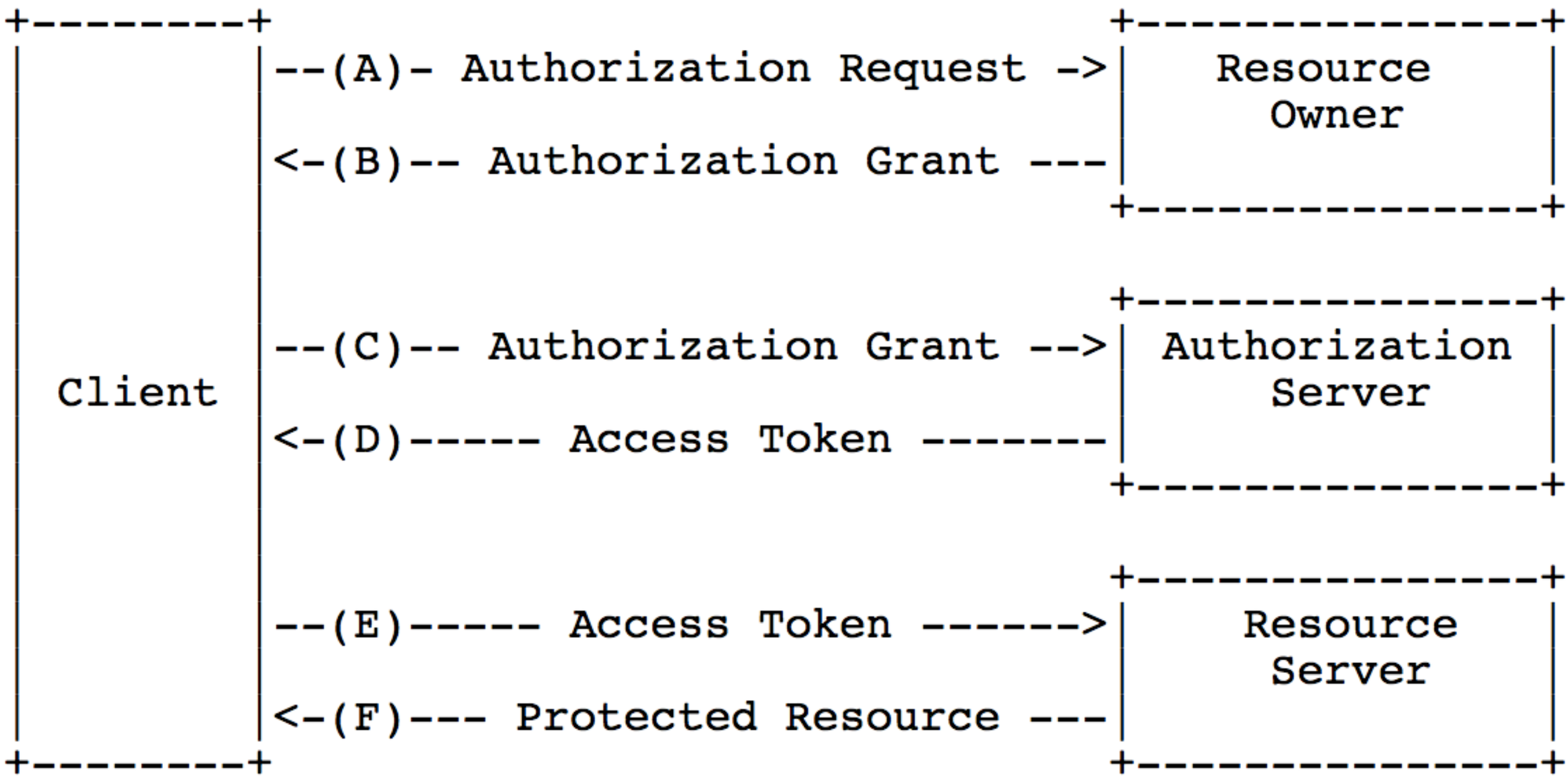
OAuth 2.0 roles exemplified



OAuth 2.0 roles exemplified



Abstract protocol flow



Source: https://tools.ietf.org/html/rfc6749

Third-party authentication with Express

- Express can make use of **`passport`**, one of the most popular **authentication middleware components**
  - 300+ authentication **strategies**
  - Supports OpenID and OAuth
- **`Passport`** hides a lot of the protocol's complexity

```
$ npm install passport
```

```
$ npm install passport-twitter
```

Installing a strategy

http://passportjs.org/

- Work on **Assignment 3.**
- Book yourself an assessment timeslot (one per group).
- **Prepare for the midterm. It takes place next week.**

flickr@courtneyrian