“3rd party” authorization to facilitate collaboration between services

As a NYTimes reader, I want to know what articles my Facebook friends are reading, so that I can find articles I might be interested in

NYTimes needs to be able to access your data on Facebook, but

You don’t want give the NYTimes your Facebook password

Instead, you authorize NYTimes to just access specific Facebook data (with a token)

Reminder: Never trust the client

The user has total control over their browser

- Can bypass any “protections” you built into app

Or could access your API end points directly

- Your JS isn’t needed to make HTTP requests

*Thus, any validation, authentication and authorization **must** be performed on a/the server (or with its assistance)*

Authentication vs. authorization

Authentication (authn)

Are you who you say you are?

Do you have some kind of secret that proves your identity?

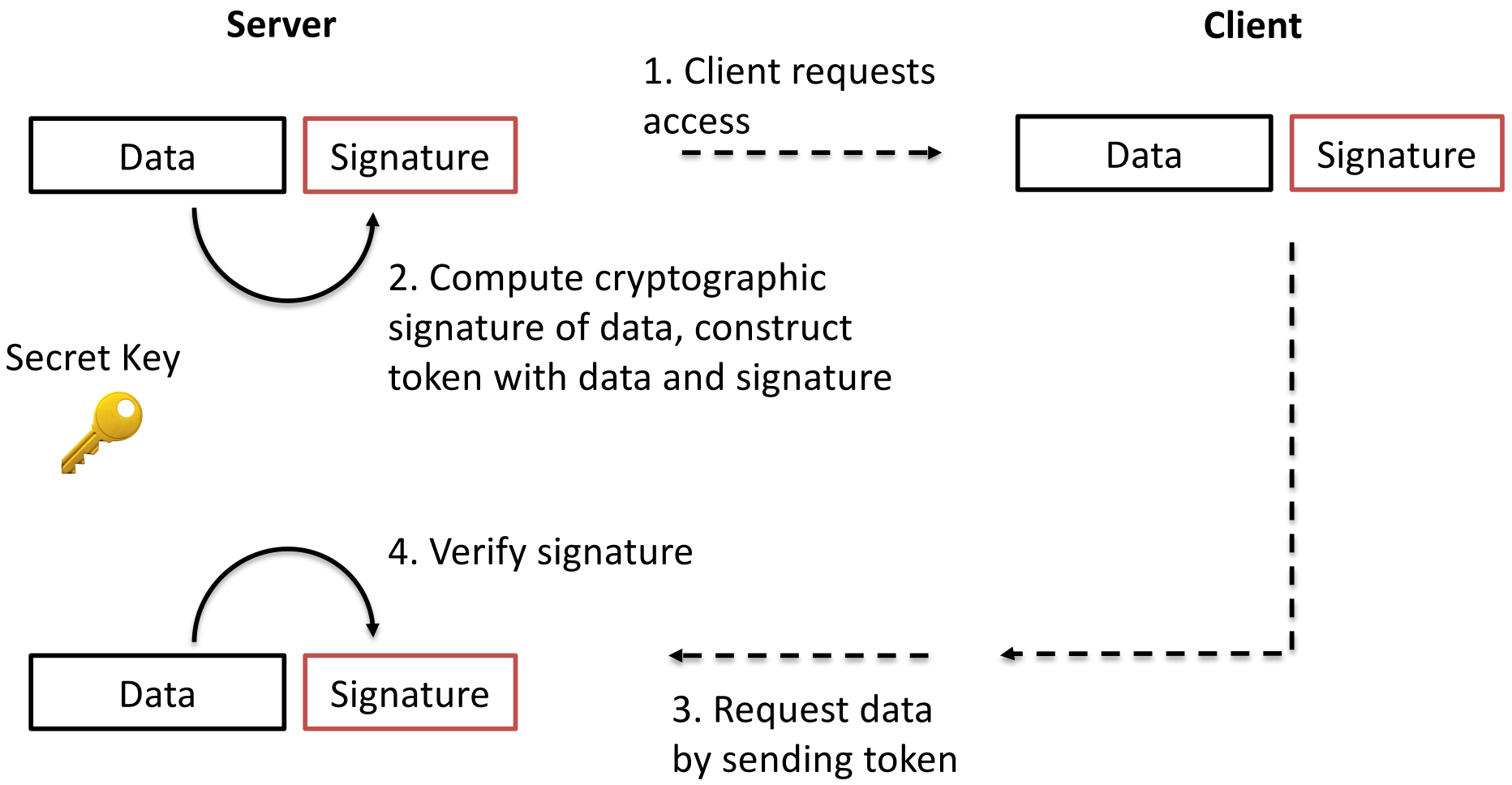
Authorization (authz):

Are you allowed to take that action?

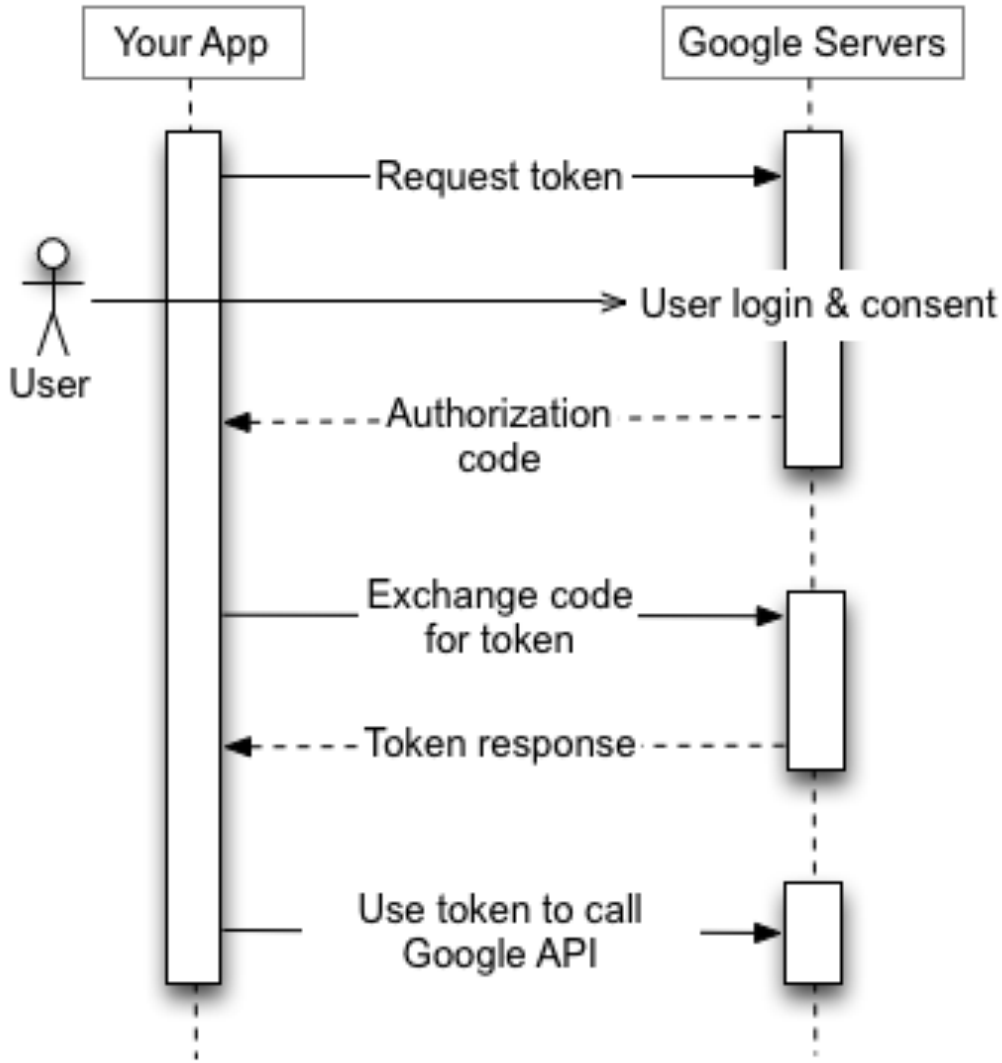
Does the application record you has having that privilege, or do you have some kind token granting that privilege?

Building blocks: Tamper evident tokens

Using cryptographic methods sign (and optionally encrypt) token data

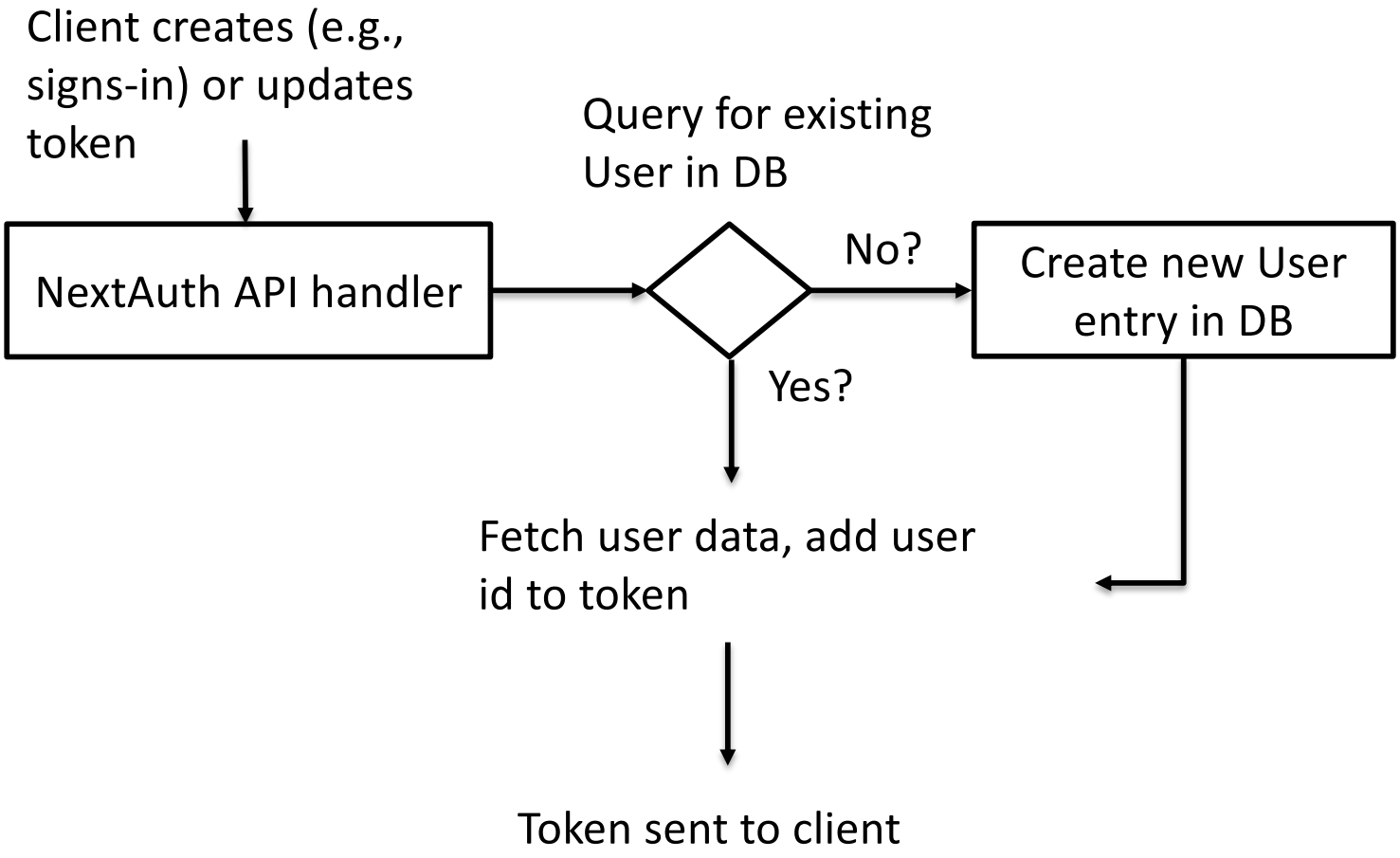


Our 3rd party authentication workflow



https://developers.google.com/identity/protocols/oauth2#webserver

Creating users as part of authentication



Managing statelessness: Cookies

- Observation: *HTTP is stateless*
- Early Web (pre-1994) didn't have a good way to guide a user "through" a flow of pages…
  - IP addresses are shared
  - Query parameters hard to cache, makes URLs private information
- Quickly superseded by *cookies*

  Set by server, sent by browser on every request

  Since client-side, must be tamper evident

  Remember: Never trust the client!

Preventing eavesdropping with SSL

Since we use the token/cookie to prove identity we need to keep it secret

Attacker could eavesdrop on communication between browser and server to intercept credentials (and impersonate user)

SSL (HTTPS) encrypts communication between browser and server (using public-private key encryption)

What SSL does and does not do

Prevents eavesdropping on traffic between browser and server

Assure browser that the server is legitimate (for some value of legitimate)

✗ Validate identity of user

✗ Protect data *after* it reaches the server

✗ Ensure server doesn't have other vulnerabilities

✗ Protect browser from malicious server

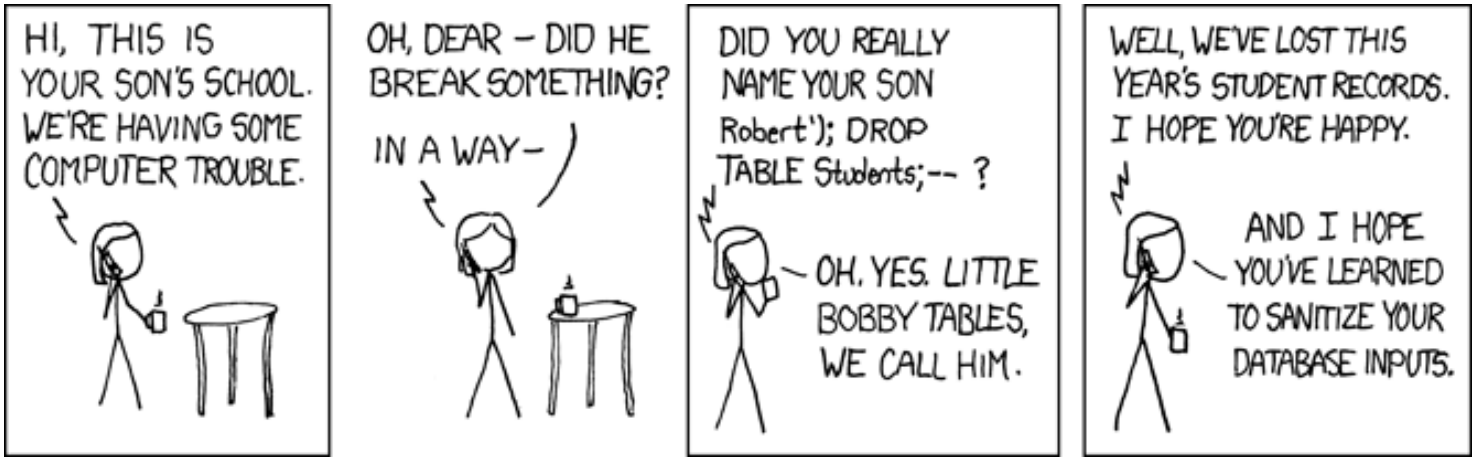
Securing our applications

There are many potential vulnerabilities

- Eavesdropping
- (SQL) injection
- Man-in-the-Middle/Session hijacking
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)

And much more…

Example: SQL injection



User supplied input

```
knex.raw(`SELECT * FROM Article WHERE id = ${id}`);
```

```
SELECT * FROM Article WHERE id = 1; DROP table Article; --
```

```
Knex('Article').where('id', id); // Knex automatically sanitizes
```

https://xkcd.com/327/

Securing our applications

There are many potential vulnerabilities

- Eavesdropping
- (SQL) injection
- Man-in-the-Middle/Session hijacking
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)

And much more…