ECS 162
Web Programming

Announcements

- Last chance on regrade requests for the midterm. We will not consider any submitted after midnight Thursday. Submit on Gradescope.

- If you did poorly on Flashcard 1, but you hand in a complete working app for Flashcard 2, I will discount your Flashcard 1 grade. So it should be worth your time to get it working.

Comic version of login

HTTPS

- Notice the Browser says she will encrypt the HTTP request, making him an HTTPS request, when he heads off to Google.
- What actually gets in encrypted? It still has to be addressed to Google, otherwise how would it get there.
- But the whole URL contains lots of information that does not have to be public.

What does HTTPS hide?

- Everything after the domain name. So

```
https://accounts.google.com/o/oauth2/v2/auth?response_type=code&redirect_uri=http://server162.site:30057/auth/redirect&scope=profile&client_id=[actual cliend id]
```

looks to intermediate places on the internet like

```
https://accounts.google.com
```

with everything else encrypted.

Really odd part

- HTTPS vs HTTP is a property of the Server, not the Browser. So how is the Browser doing the encryption?

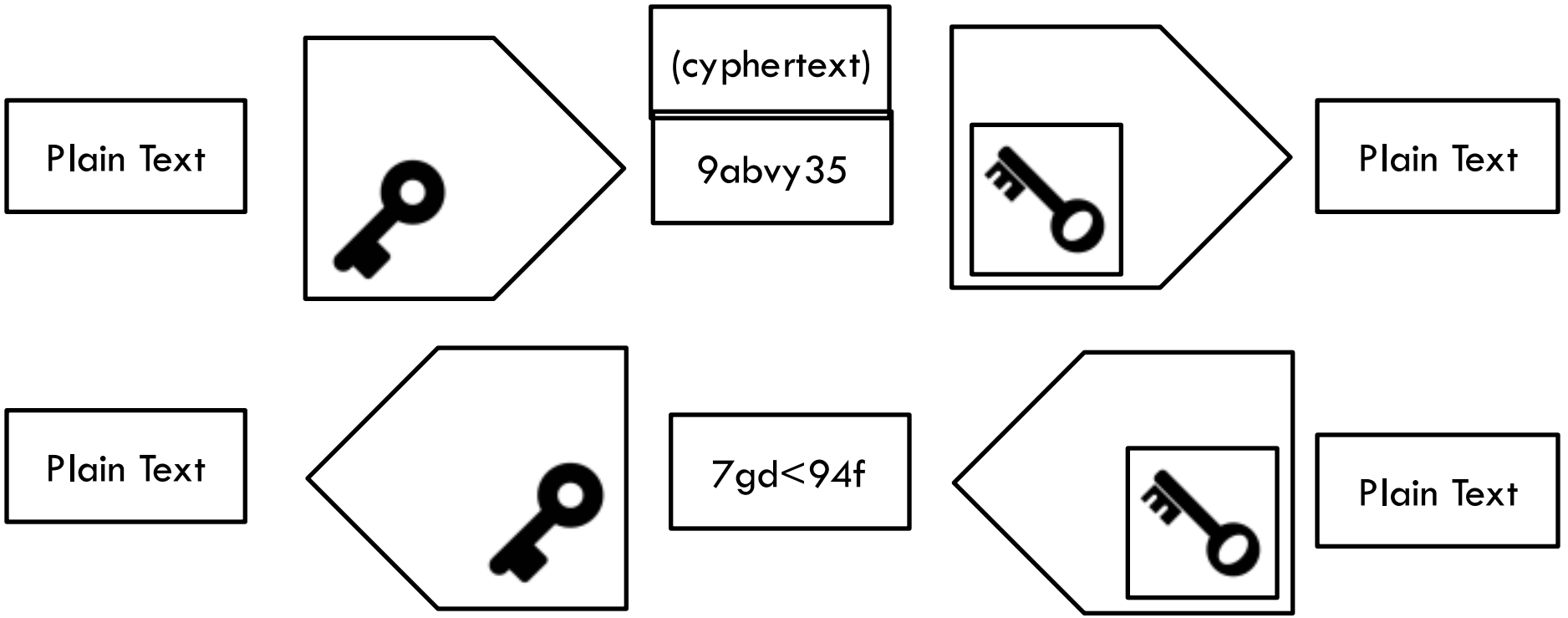
Really odd part

- HTTPS vs HTTP is a property of the Server, not the Browser. So how is the Browser doing the encryption?

  Uses Public Key Encryption.

- Fundamental trick of secure internet communication (TLS/SSL, transport layer security).
- Encryption with two separate keys (each of which is really a big, pseudo-random number).

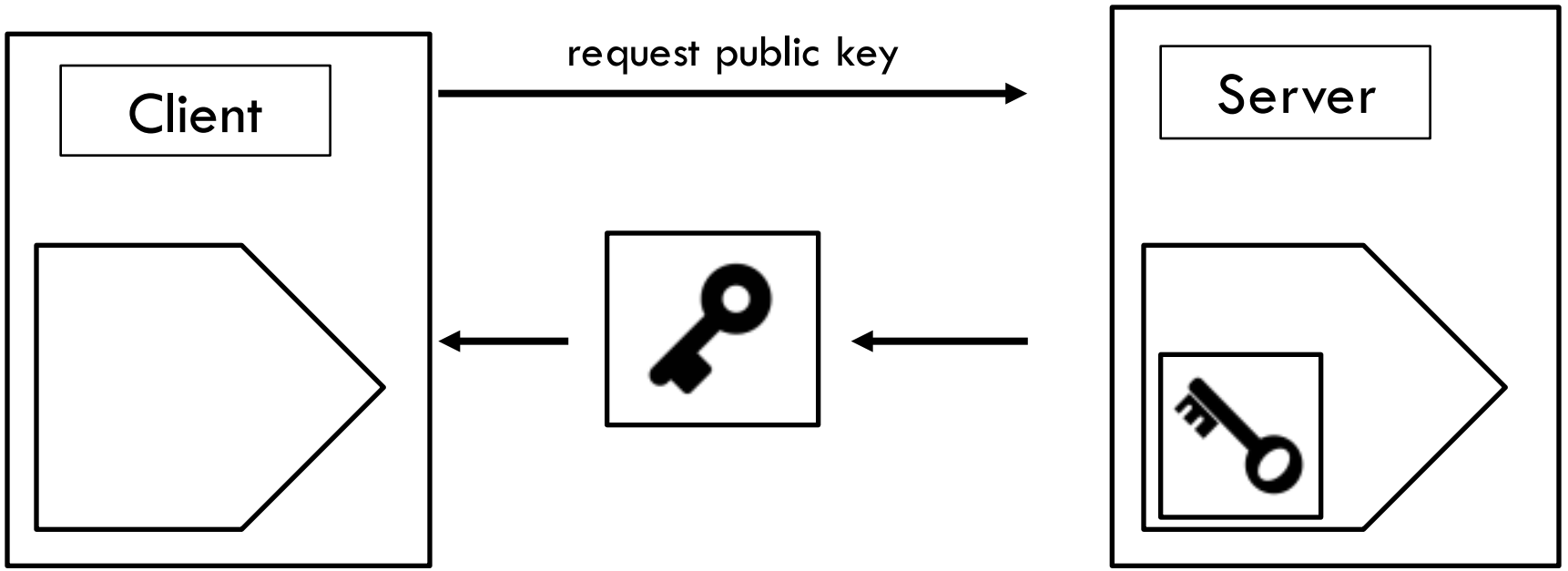
Asymmetric Encryption



- Functions using the keys encrypt or decrypt
- Green decrypts anything encrypted by purple, and visa versa.

TLS with public keys

- Server hands out encryption key to whoever wants it; this is the public key.



- Now Client (or anyone) can encrypt data sent to Server, and can decrypt data it receives.

Man in the middle attack

- What does the stick-up the bird suffered correspond to in real life?

- "Packet sniffers" or "packet capture" are programs that look at passing HTTP requests/responses and other TCP/IP packets. These are widely used legitimately, but can be malicious.
- Is the government reading your email? Probably not; more likely they will get data from the big companies if they want it.

What can a man-in-the-middle do?

- With an HTTP request (unencrypted), a router can see all of it and change any part of it.
- Our Server sends this redirect URL in an unencrypted HTTP response:

https://accounts.google.com/o/oauth2/v2/auth?response_type=code&redirect_uri=http://server162.site:30057/auth/redirect&scope=profile&client_id=[actual client id]

- What might be a problem?

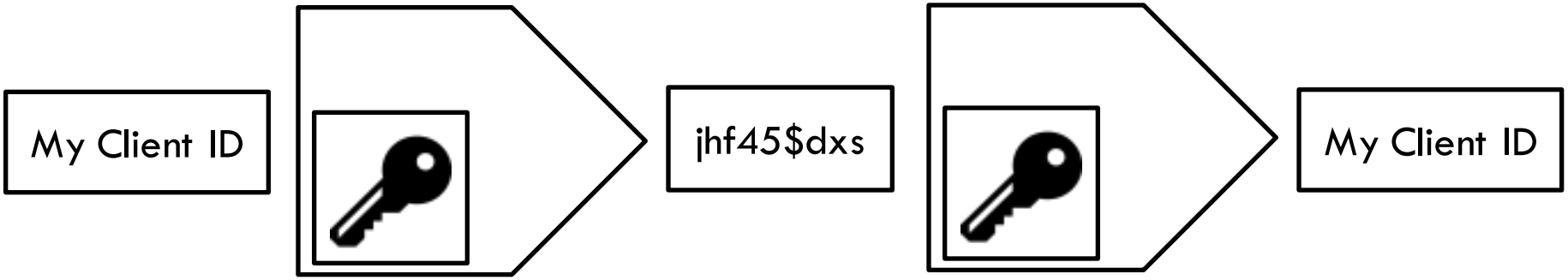
The URL for the request to Google

https://accounts.google.com/o/oauth2/v2/auth?response_type=code&redirect_uri=http://server162.site:30057/auth/redirect&scope=profile&client_id=[actual client id]

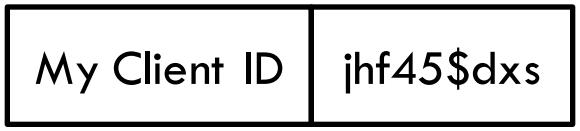
- We don't want someone using our Client ID.
- While it would be nice to hide the redirect address, it's going to get an HTTP request later in the process, so we can't.
- How does Oauth2 handle "public" Client IDs?

Recall Digital Signature

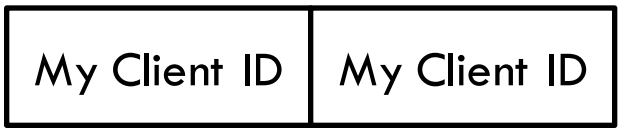
- Uses a secret symmetic key.



- "Sign" by adding encrypted text to the plain text

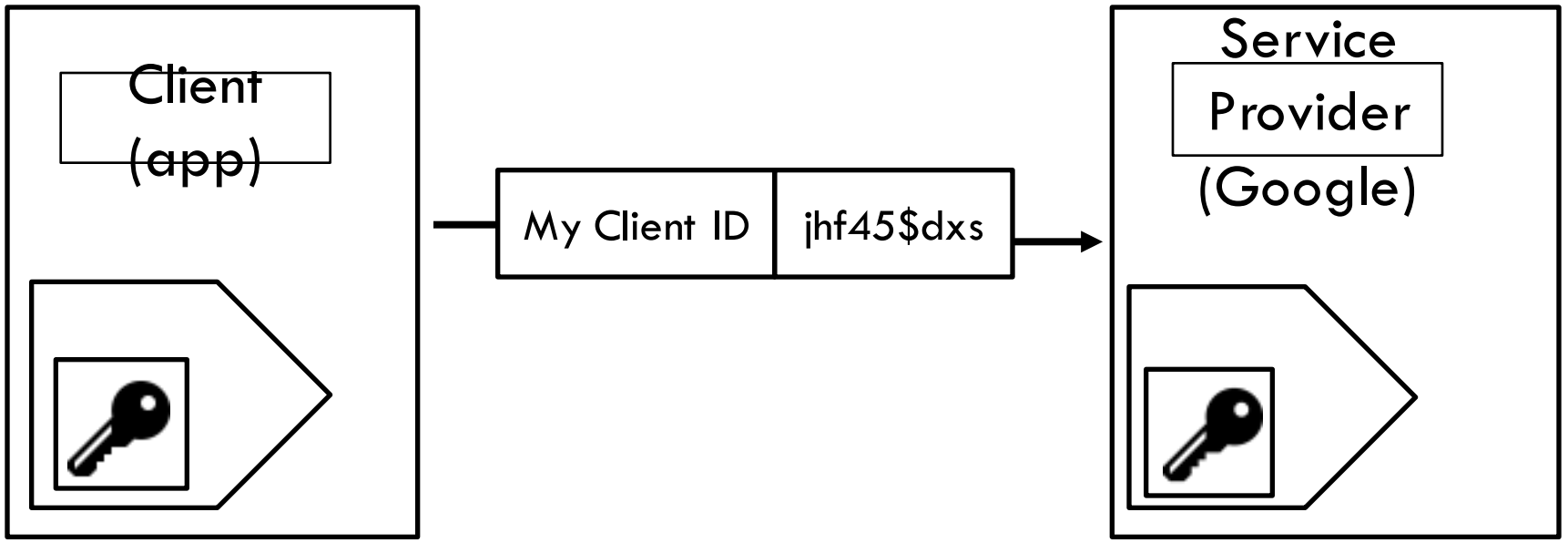


- After decryption:



Signing a request to the Server

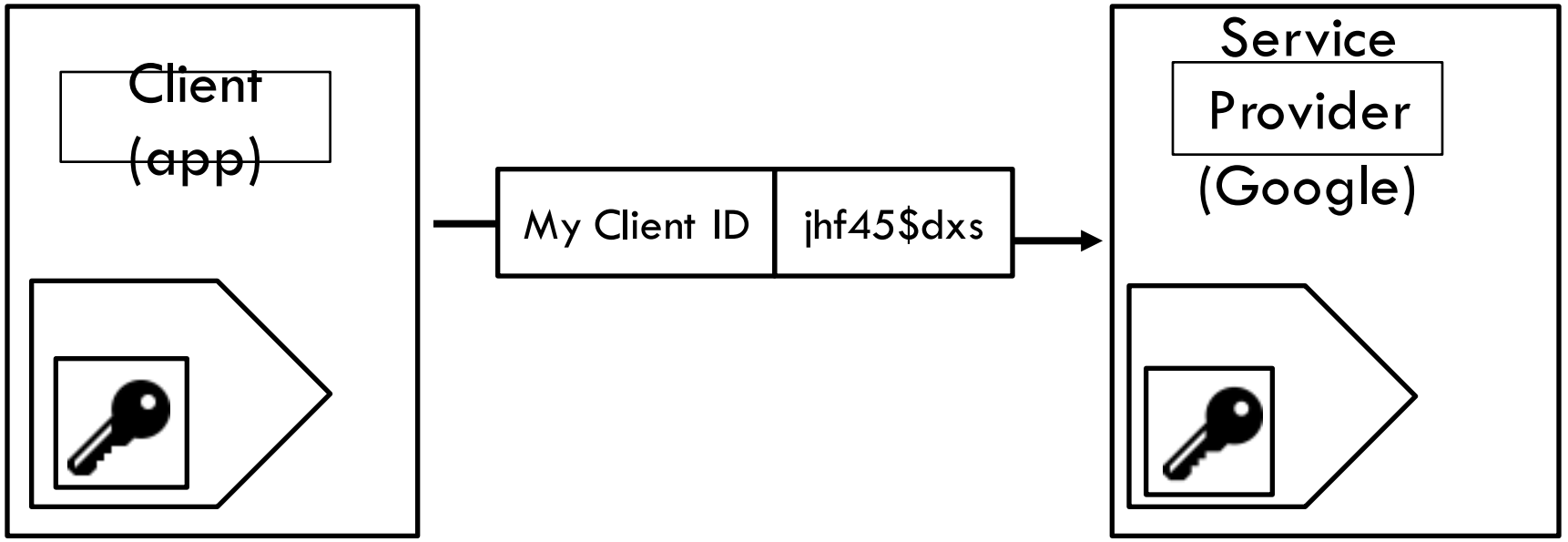
- Server thus verifies that Browser knows the symmetric secret key.



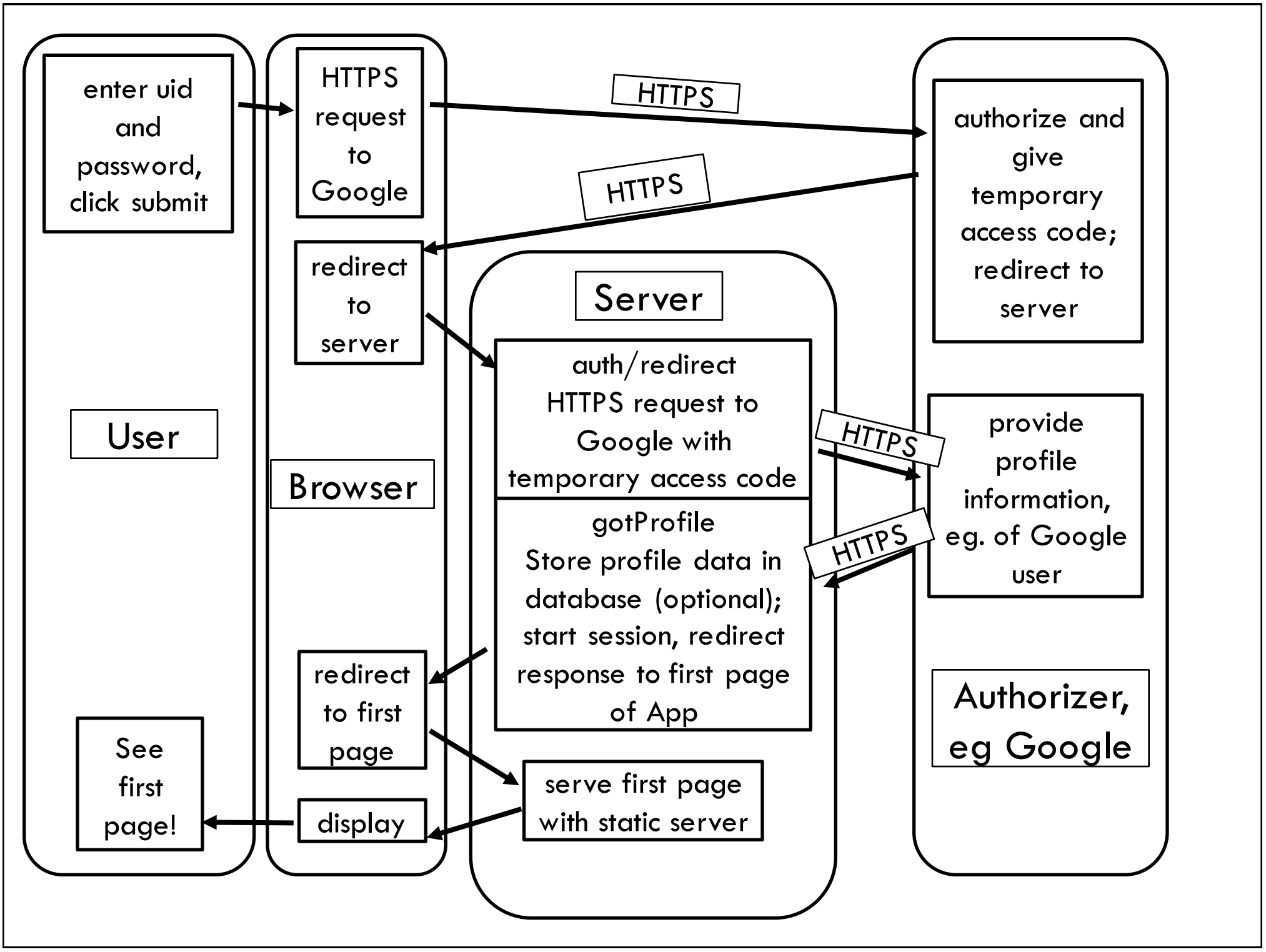
- What is the secret symmetric key?

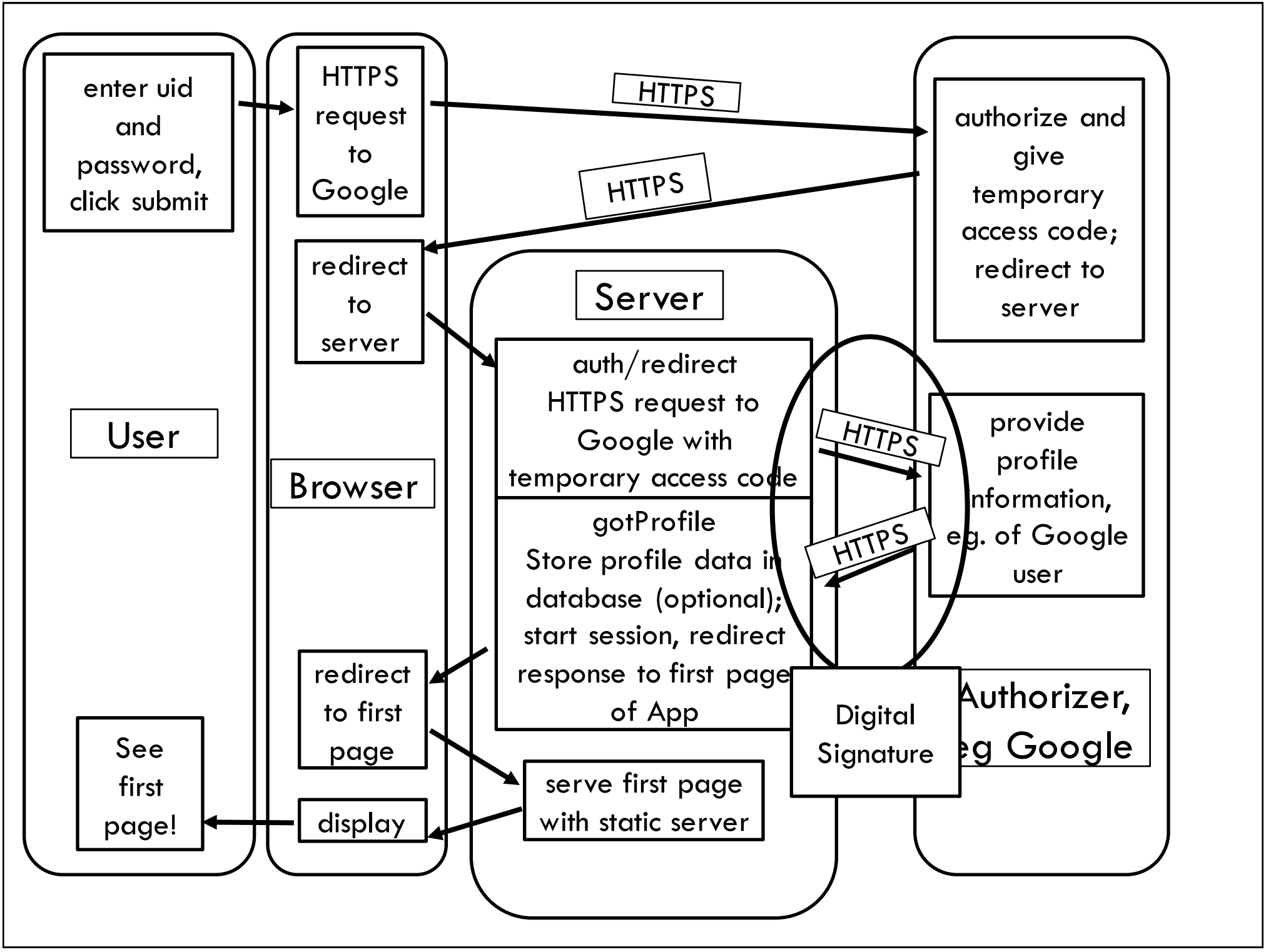
Signing a request to the Server

- Server thus verifies that Browser knows the symmetric secret key.



- What is the secret symmetric key? The client secret!

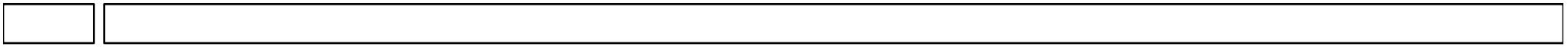

enter uid and password, click submit
HTTPS request to Google
HTTPS
authorize and give temporary access code; redirect to server
HTTPS
redirect to server
Server
auth/redirect HTTPS request to Google with temporary access code
User
Browser
HTTPS
provide profile information, eg. of Google user
gotProfile Store profile data in database (optional); start session, redirect response to first page of App
HTTPS
redirect to first page
Authorizer, eg Google
See first page!
display
serve first page with static server


enter uid and password, click submit
HTTPS request to Google
HTTPS
authorize and give temporary access code; redirect to server
HTTPS
redirect to server
Server
redirect to server
auth/redirect
HTTPS request to Google with temporary access code
User
Browser
HTTPS
provide profile information, eg. of Google user
HTTPS
gotProfile
Store profile data in database (optional); start session, redirect response to first page of App
redirect to first page
Digital Signature
Authorizer, eg Google
See first page!
display
serve first page with static server

Summary

- HTTPS protects communications with Google (or other Service Provider) from man-in-the-middle attacks.
- Client ID is passed as plain text between Server and Browser, not secured.
- Digital Signature using the Client Secret is required for Server to actually get the User's profile information from Google.
- Keep Client Secret secret! If revealed, request a new one.

Where else did we see digital signature?



Where else did we see digital signature?

- The Server signs the session cookies he distributes to Browsers during login.
- This lets him recognize his own cookies.
- What if a man-in-the-middle stole the session cookie?

Where else did we see digital signature?

- The Server signs the session cookies he distributes to Browsers during login.
- This lets him recognize his own cookies.
- What if a man-in-the-middle stole the session cookie?

He could impersonate the Browser to the Server.

- This is an obvious security flaw in our app.
- What is a good way to prevent this?

Where else did we see digital signature?

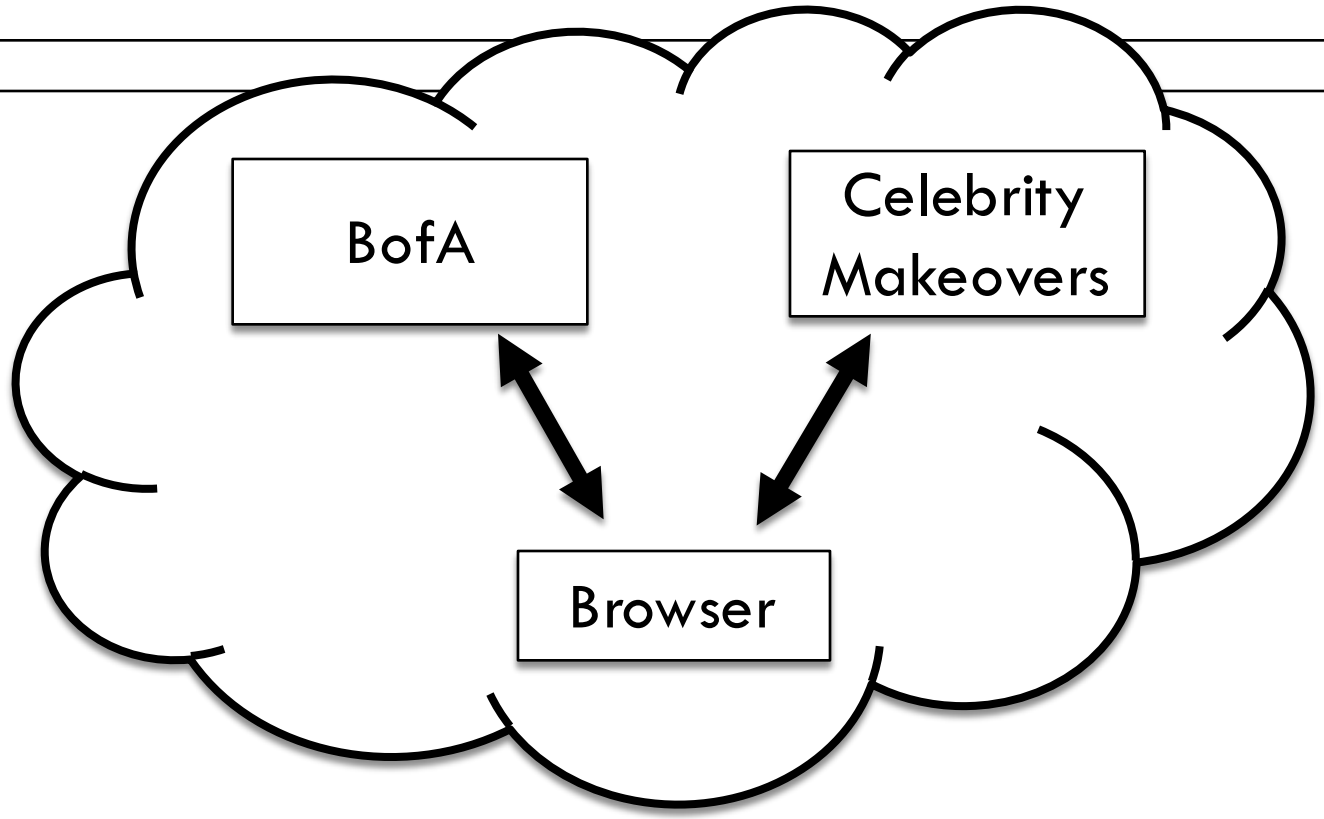
- The Server signs the session cookies he distributes to Browsers during login.
- This lets him recognize his own cookies.
- What if a man-in-the-middle stole the session cookie?

He could impersonate the Browser to the Server.

- This is an obvious security flaw in our app.
- What is a good way to prevent this?

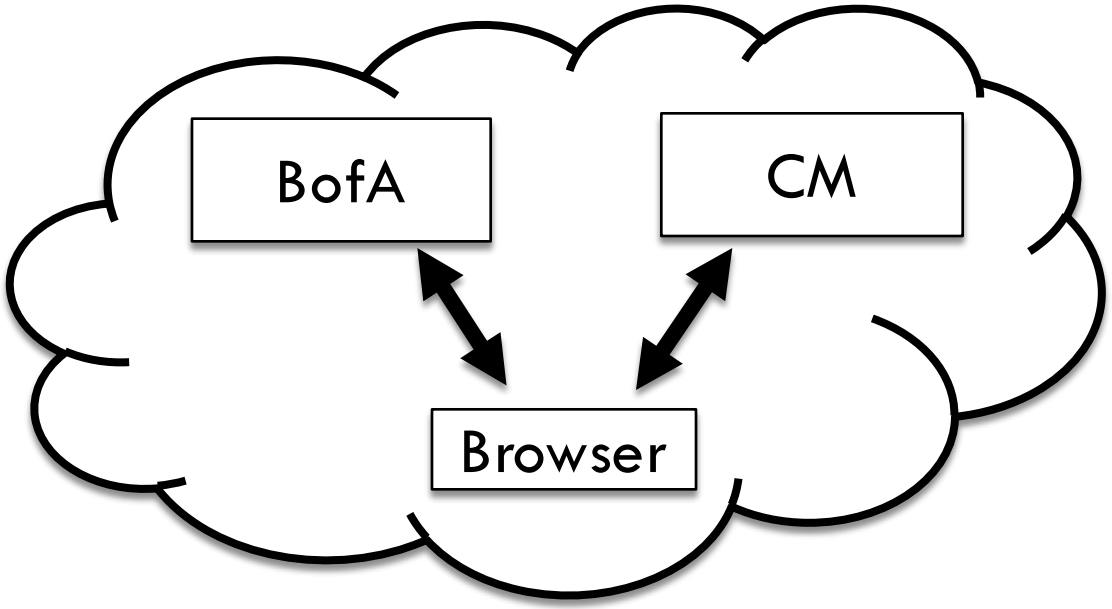
Our Server should be using HTTPS. Next year.

SOP, revisited



- Recall the Same Origin Policy usually prevents CM from sending an AJAX request to BofA. But if a hacker finds a hole in the SOP, they can send a Cross-Site Request Forgery (CSRF)

HTTPS is no defense here



- The Browser will attach the session cookie to any AJAX request, even a forged one.

- Encrypted session cookies are no defense if CSRF is possible! So what can we do?

Csurf vs CSRF

- Idea: App's Browser code puts some kind of encrypted token in the body of any legitimate HTTP request (the session cookie itself or something more complex).
- The Server checks the token in the body.
- The hacker doing the CSRF needs to know token to be able to fake a legitimate request.
- Express has a Csurf module that handles the Server end of this defense.