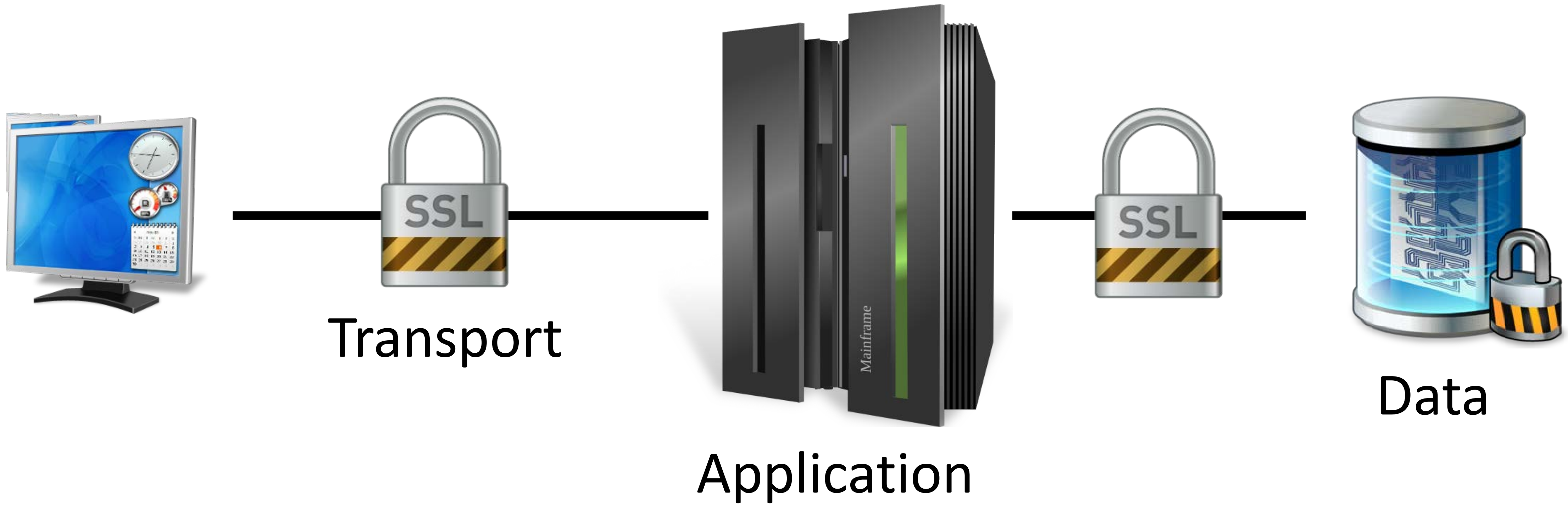
Day 31

Security Layers

- Application
  - Security requirements of an application
  - Eg. who can access what
- Transport
  - Ensures that the communication channel between a client and the application is secure
  - Eg. using TLS between the browser and the server, VPN between subnets
- Message
  - Refers to the security of the data
  - Eg. has it been changed?

Security Layers



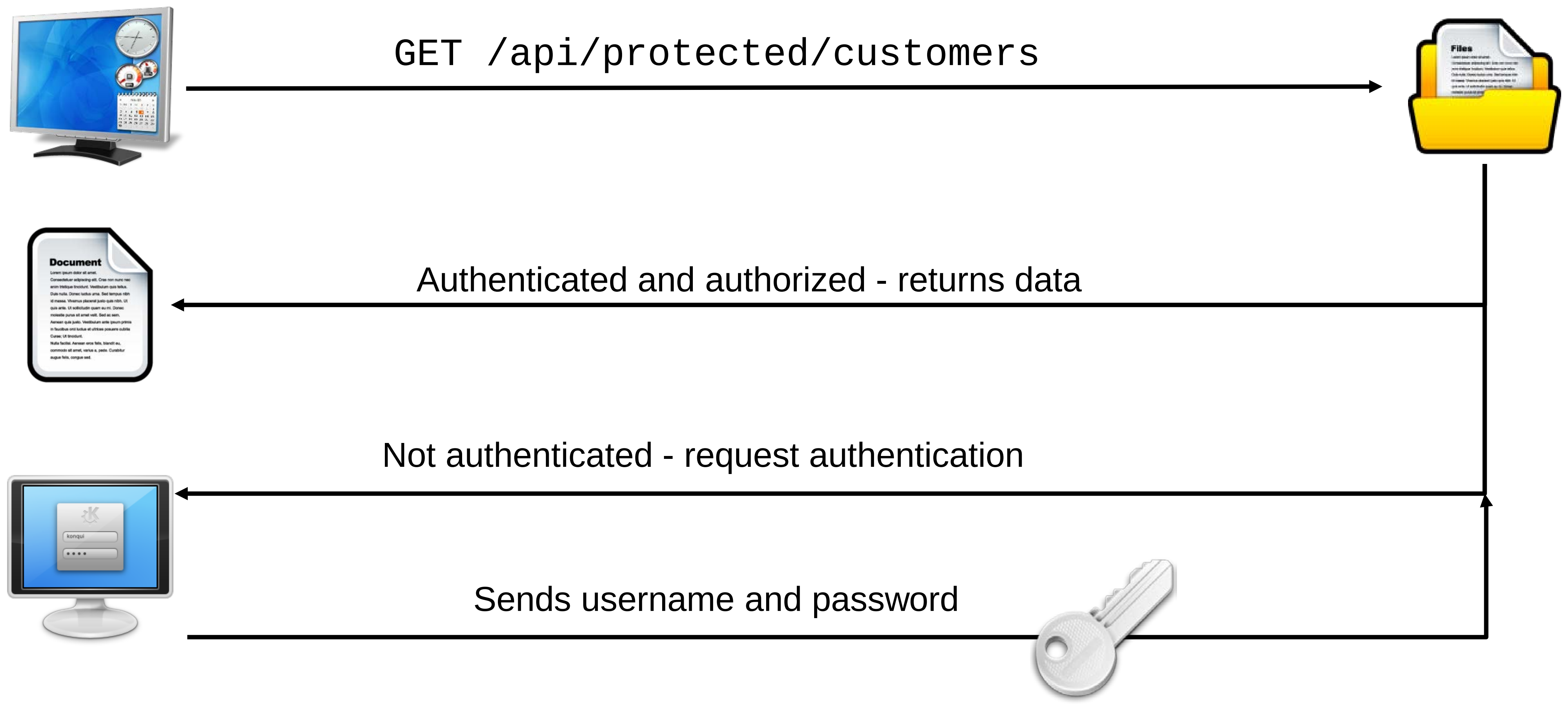
Security Attributes

- Authentication
  - Verifying the identity of a user, a server, a request, etc.
- Authorization
  - Control over who can access what
- Data confidentiality
  - Preserve the secrecy of the message
- Data integrity
  - Preserve the data against tampering
- Non-repudiation
  - Cannot renounce a action eg. a business transaction
- Auditing
  - Trail of events

Protecting Web Application

- Keep all sensitive resources under a common resource root
  - Eg. All resources rooted under `/protected` contains sensitive data
- Access to these resources must be authenticated
- Some HTTP methods require authentication when used with certain resources
  - Eg. `GET /cart` may not require authentication
  - Eg. `POST /cart` requires authentication

Accessing Protected Resources


GET /api/protected/customers
Authenticated and authorized - returns data
Not authenticated - request authentication
Sends username and password

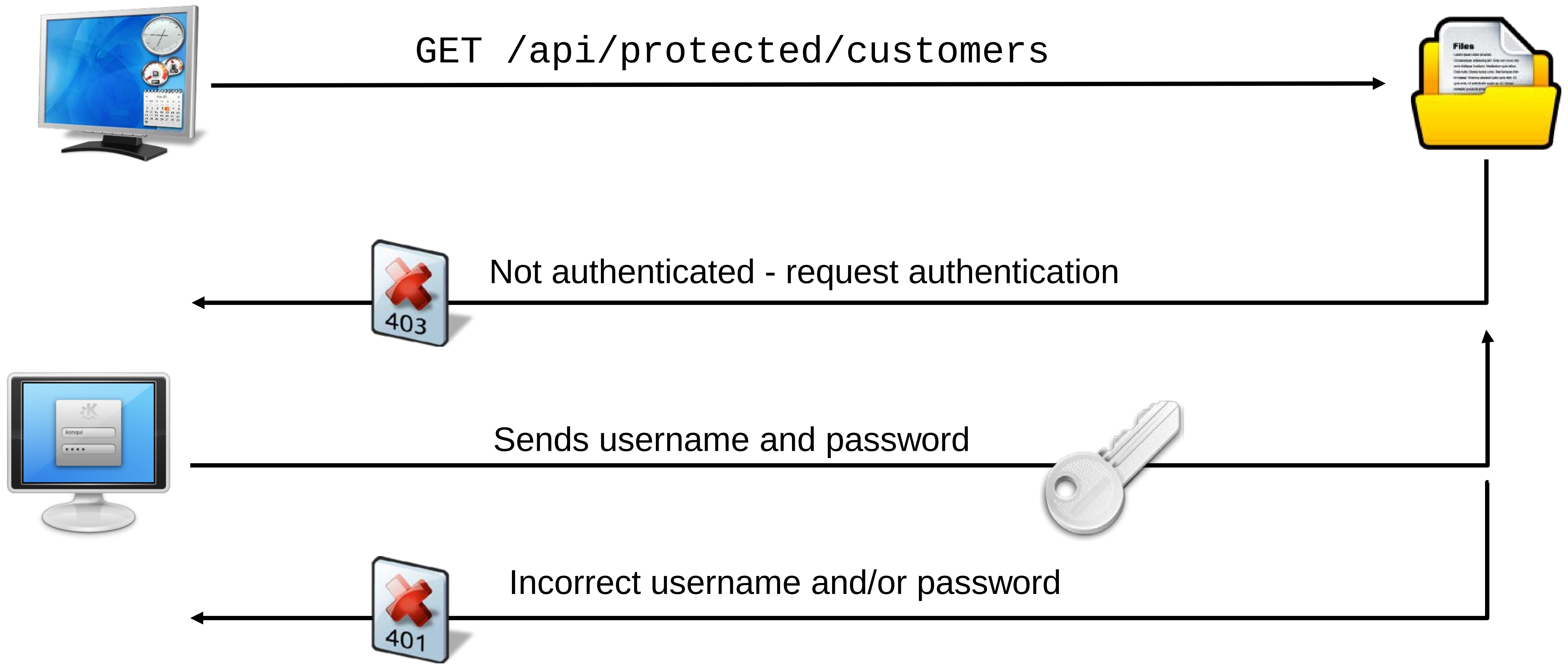
HTTP Status Code

- The following HTTP codes are used for indicating authorization status
- `401 Unauthorized` - when a clients authentication is incorrect
  - Eg. incorrect username/password, expired access token

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm=“my app”
```

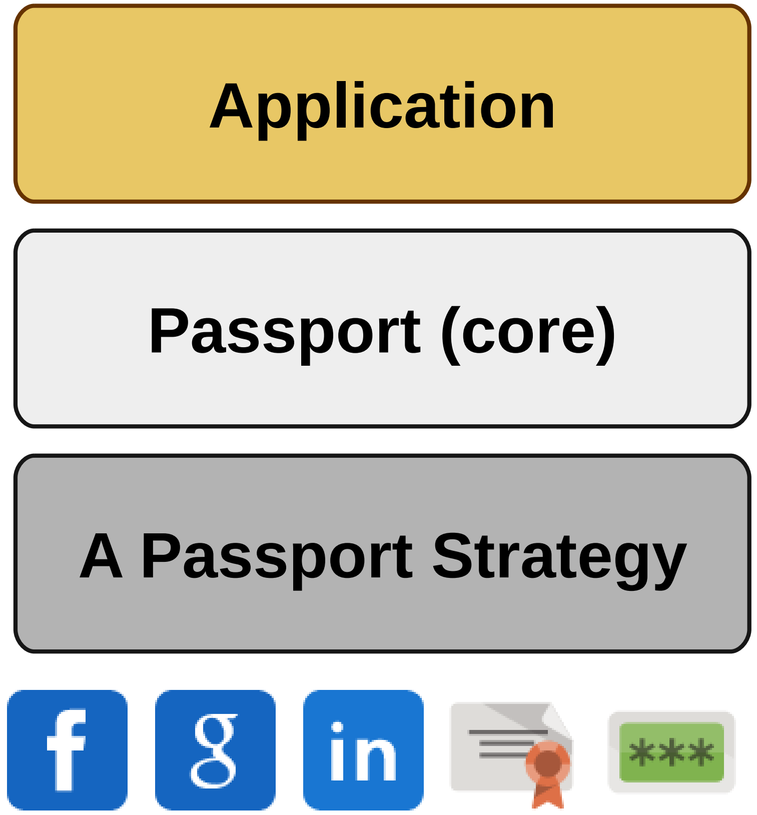
- `403 Forbidden` - when a client does not have permission to access the resource

Accessing Protected Resources

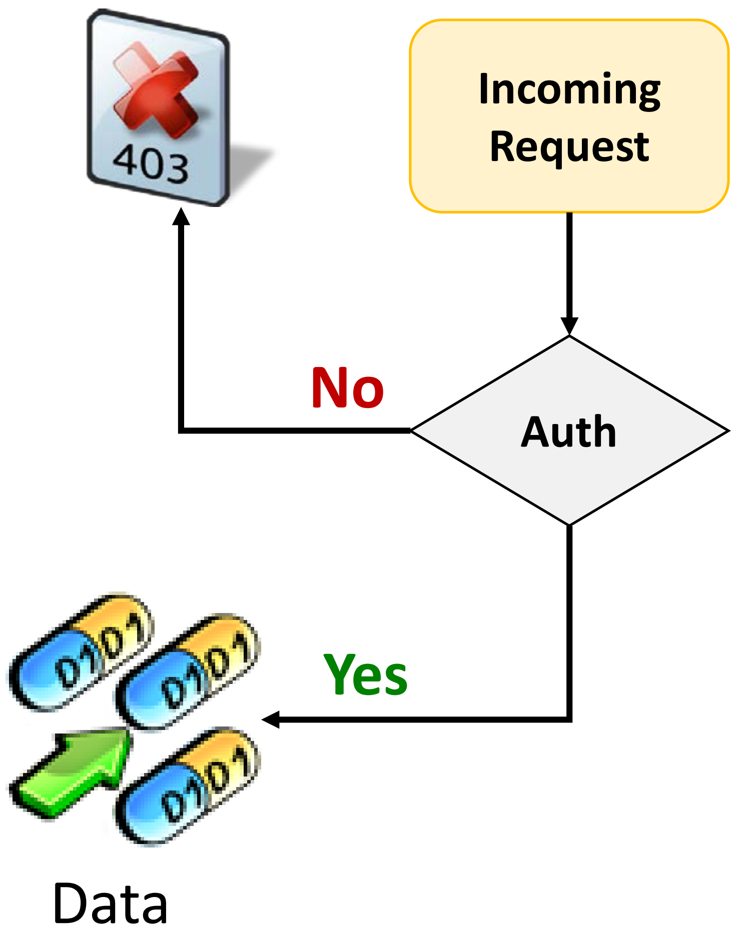

GET /api/protected/customers
Not authenticated - request authentication
403
Sends username and password
Incorrect username and/or password
401

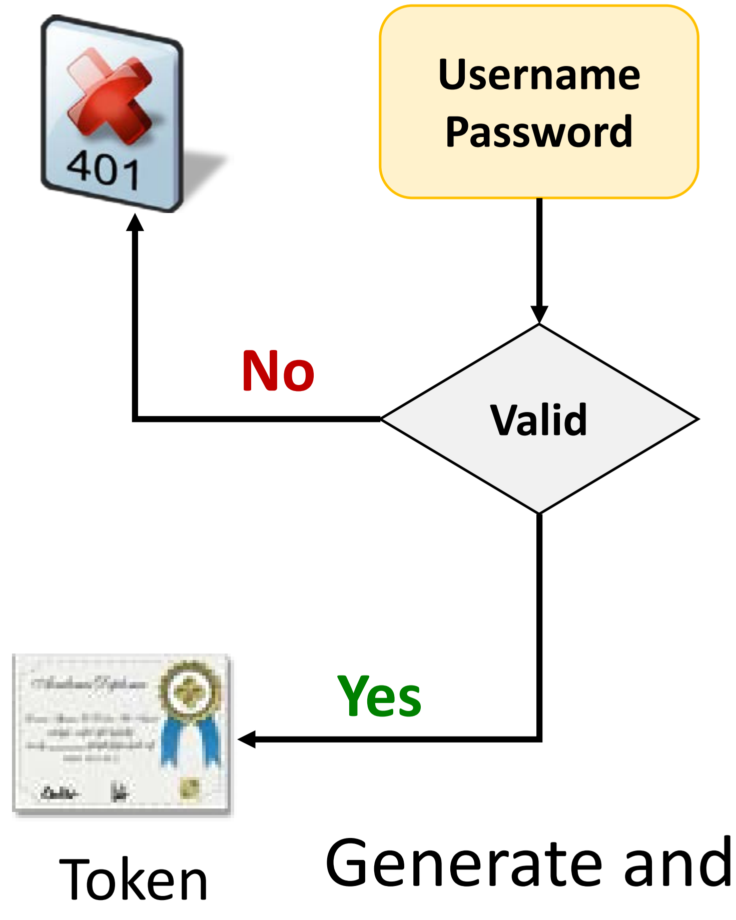
Passport

- Passport is an Express based authentication framework
- Consist of 2 parts
  - Core - provides a standard way to enforce security
  - Specific authentication mechanism for Facebook, LinkedIn, JWT, etc



Authentication and Authorization Process

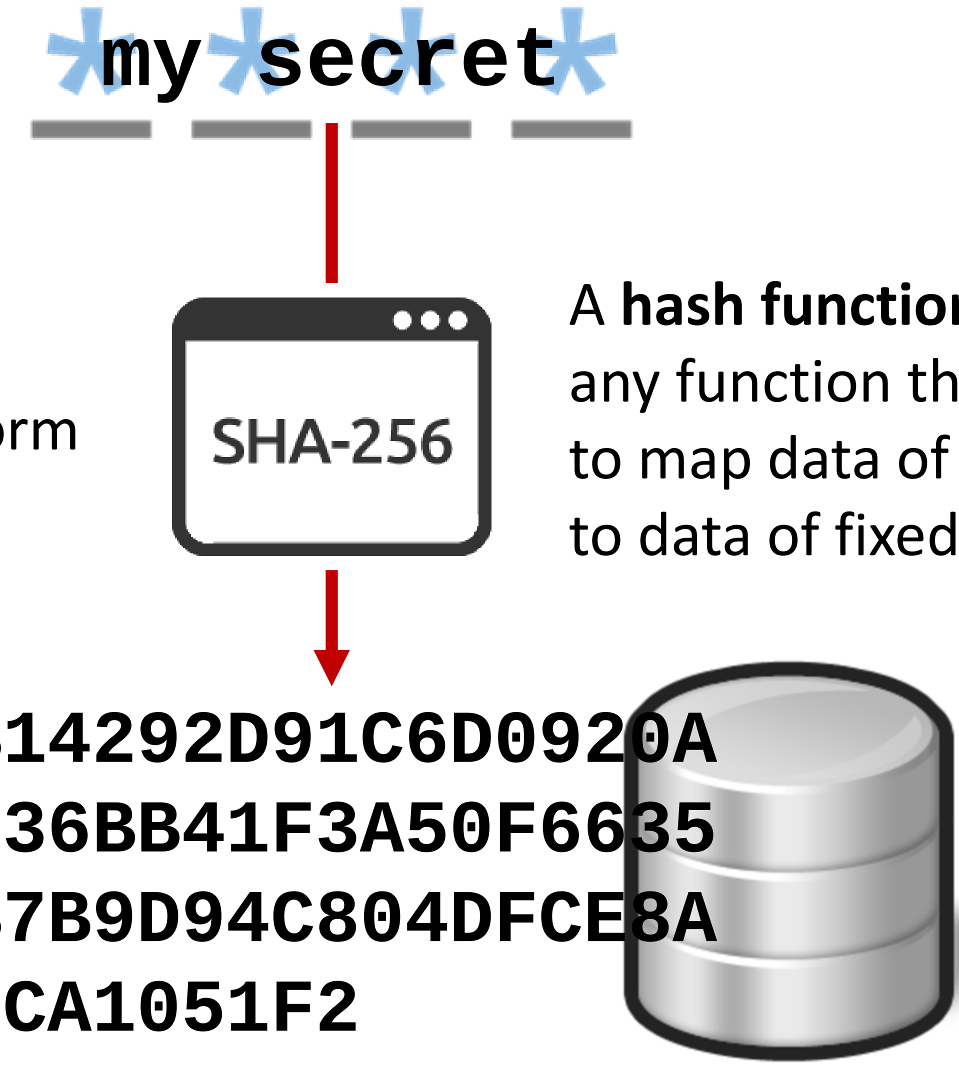




Generate and return access token

Storing Password

Computationally expensive to perform the reverse



```
BB14292D91C6D0920A
5536BB41F3A50F6635
1B7B9D94C804DFCE8A
96CA1051F2
```

A **hash function** is any function that can be used to map data of arbitrary size to data of fixed size

Only store the password hash instead of the password

Storing Password as Hash

```
insert into user (username, password)
values (‘fred’, sha2(‘yabadabadoo’, 256));
```

Hash ‘yabadabadoo’
using sha256

Authenticating Using Passport

- Client username and password to Passport
  - Info is sent POST using `x-www-form-urlencoded`
- Will check if username/password comb is correct
  - Eg. check against database
- If correct continue processing request
- Send a 403 status back to client if password is incorrect

Passport Setup

- Install passport and local-strategy
  - `passport` is the core
  - `local-strategy` allow application to authenticate the username/password. Typically this will be against a record stored in the database
  - `body-parser` will be required to parse x-www-form-urlencoded

```
npm install --save passport
npm install --save passport-local
npm install --save body-parser
```

HK: what if i were to send my form in application/json???
how do i configure the username and pw

Passport Setup - Configure Strategy

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy(
    {
      usernameField: 'email',
      passwordField: 'password',
      passReqToCallback: true
    },
    (req, username, password, done) => {
      //perform authentication
      ...
    })
);
```

Access to request object in callback

The 2 field names that hold the username/login and password

To indicated if authentication is successful

This is the function that will be performing the authentication

Request object

Passport Setup - Configuration Sequence

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

const express = require('express');
const bodyParser = require('body-parser');
const app = new express();
```

1
```
passport.use(new LocalStrategy(...));
```

Configure passport with a strategy eg. local - see previous slide

2
```
app.use(bodyParser.urlencoded({extended: true}));
```

Enable parsing form-urlencoded media type. Must set before initializing passport.

3
```
app.use(passport.initialize());
```

Initialize passport

Passport - Authenticating a Request

```
passport.use(new LocalStrategy(
   { usernameField: ‘email’, passwordField: ‘password’,
     passReqToCallbac: true },

   (req, username, password, done) => {
      pool.getConnection((err, conn) => {
         conn.query(
            ‘select * from user where email like ?
               and password like sha2(?, 256)’,
            [username, password],
            (err, result) => {
               try {
                  if (result.length) return (done(null, result[0]));
                  else return (done(null, false));
               } finally { conn.release(); }
            }
      })
   }
);
```

Compare the saved hashed password with the current password which is also hashed

Return the user record

Return false in done to indicated authentication failed

Note: omitting error checks for brevity

Passport Setup - Authenticating

Authenticate with `local` strategy

Use stateless authentication

```
app.post('/login',
    passport.authenticate('local', {session: false}),
    (req, resp) => {
        req.user
    }
);
```

This middleware gets called if authentication using the `local` strategy is successful

Returns 401 if authentication fails

```
return (done(null, false));
```

Calls the next middleware if authentication is successful. `user` is from

```
return (done(null, result[0]))
```

Passport Authentication Flow

```
passport.use(new LocalStrategy({ /* configuration */ },
  (req, username, password, done) => {
    done(null, userDetails);

    done(null, false);
  }
}

app.post('/login',
  passport.authenticate('local', {session: false}),
  (req, resp) => {
      req.user
  }
)
```

1 Authenticate calls the specify strategy to perform the authentication

2 If authentication is successful

3 Calls the next middleware

4 Returns 401 if fail authentication

401

Authenticating with Angular

Login details

```
performLogin(user, passwd) {
   HttpParams loginDetails = new HttpParams()
      .set('email', user)
      .set('password', passwd);

   HttpHeaders httpHeaders = new HttpHeaders()
      .set('Content-Type', 'application/x-www-form-urlencoded');

   this.httpClient.post('/login', loginDetails.toString(),
         { headers: httpHeaders })
      .pipe(take(1)).toPromise()
      .then(token => { ... })
      .catch(error => {
         if (401 == error.status) {
            //handle incorrect login
         }
      })
}
```

Set appropriate content type

If error, check if it is a 401 status.