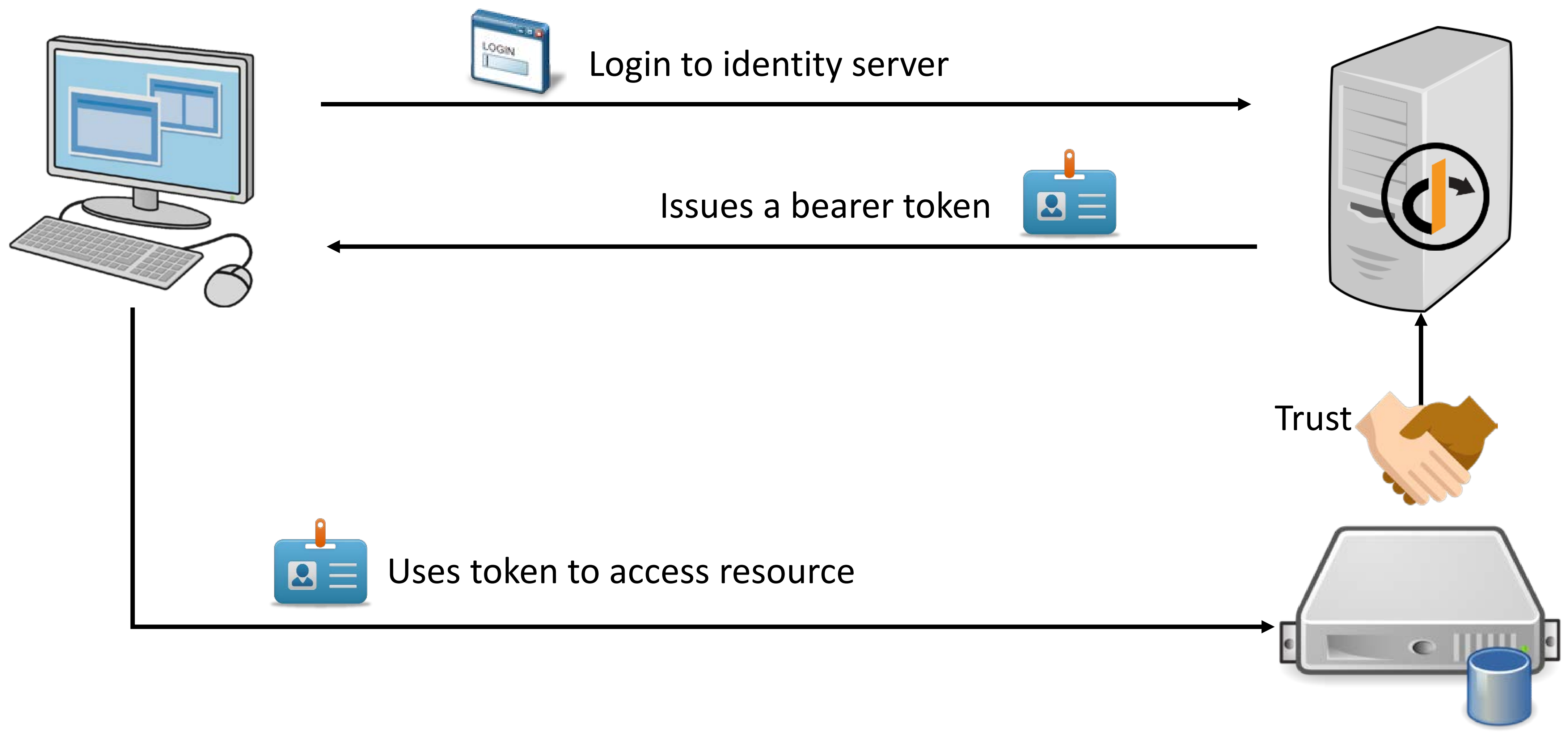
Day 32

Token Based Authentication

- The client is issued a token after successful 'login'
  - Need authenticate the client before issuing the token
- From here on, the server/resource is going to only look at the issued token
- Token encodes pertinent information about the client
  - Eg. name, email, scope the token, validity, issuer, etc
- Tokens are self contained
  - The validity of a token can be found in the token itself
  - Digitally signed to prevent tempering
- Basis for federated identity
  - Eg. NRIC is a token that is recognized and accepted universally in Singapore and overseas

Token Based Authentication



What Can Tokens Do

- Contain information about who you are, facts about you
  - Eg valid email
- The scope of your capabilities
  - Eg. Can only view details of those employees that reports to you
- Token with lifespan
  - Tokens that are only valid at some future date
  - Tokens that are invalid after a certain date
- Limited uses
  - Eg. once, 10 times

JWT - JSON Web Token

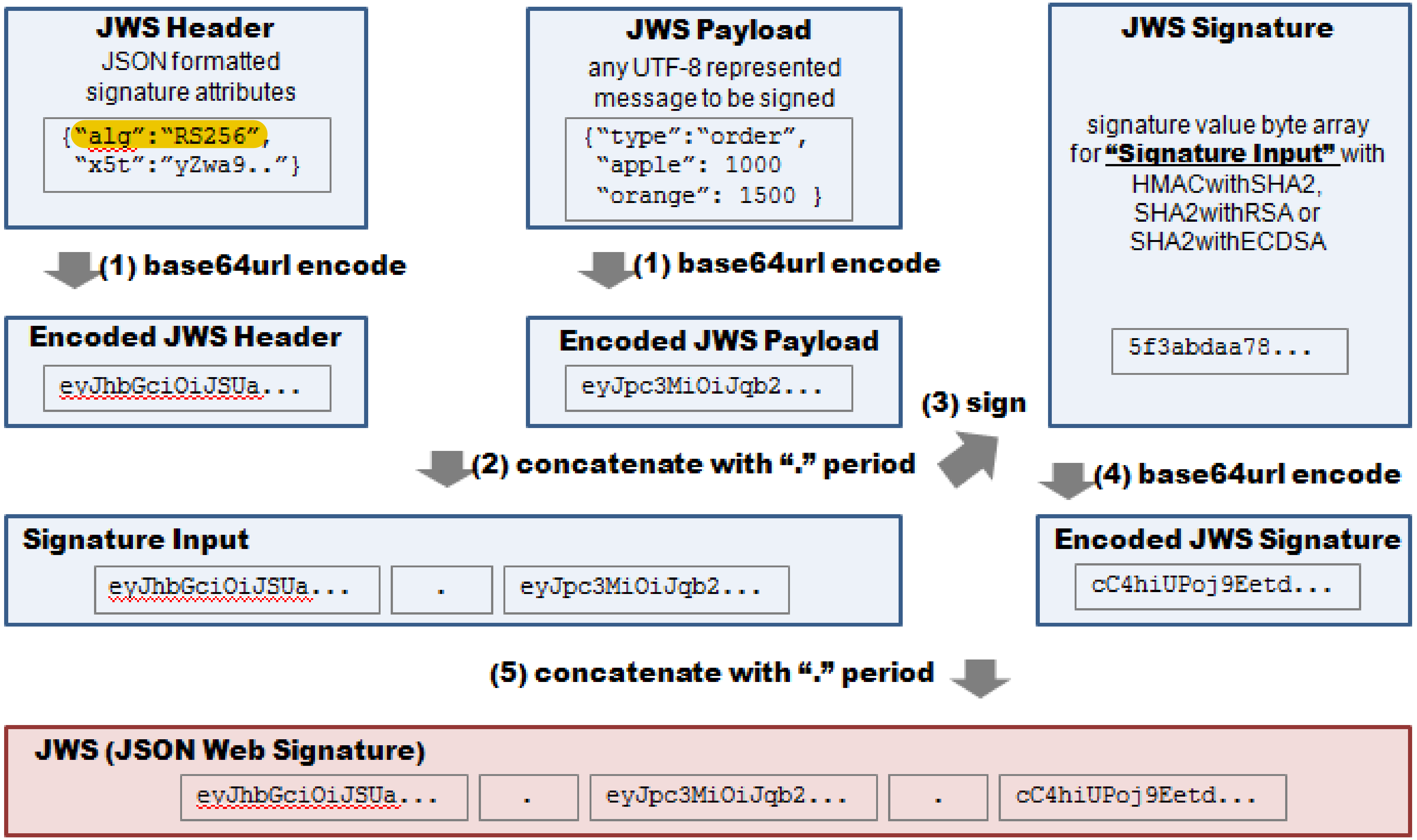


Image from https://kjur.github.io/jsjws/

Bearer Tokens - Creating

- Install `jsonwebtoken` module

```
npm install --save jsonwebtoken
```

- Create a token

```
const jwt = require('jsonwebtoken');
const currTime = (new Date()).getTime()
const token = jwt.sign({
   sub: 'fred',
   iss: 'hannabarbera',
   iat: currTime,
   exp: currTime + ~~(1000 *60 *60),~~
   data: { email: 'fred@gmail.com' }
}, 'secret');
```

should be 60*60 because it's in seconds

Shared secret. Must be the same when verifying

sub: issued to
iss: issuer
aud: audience
nbf: not before
iat: issue time
exp: expire in
data: additional claims

See https://tools.ietf.org/html/rfc7519

Bearer Tokens - Verifying

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.
eyJzdWIiOiJmcmVkIiwiaXNzIjoiaGFubmFiYXJiZXJhIiwiaWF0
IjoxNTIwNTE1MTYxOTQ0LCJleHAiOjE1MjA1MTg3NjE5NDQsImRh
dGEiOnsiZW1haWwiOiJmcmVkQGdtYWlsLmNvbSJ9fQ
.
BCc5Ry9riY_vuuMzkdFdi4iq8fGo2vO_LZpIPZmSsxY
```

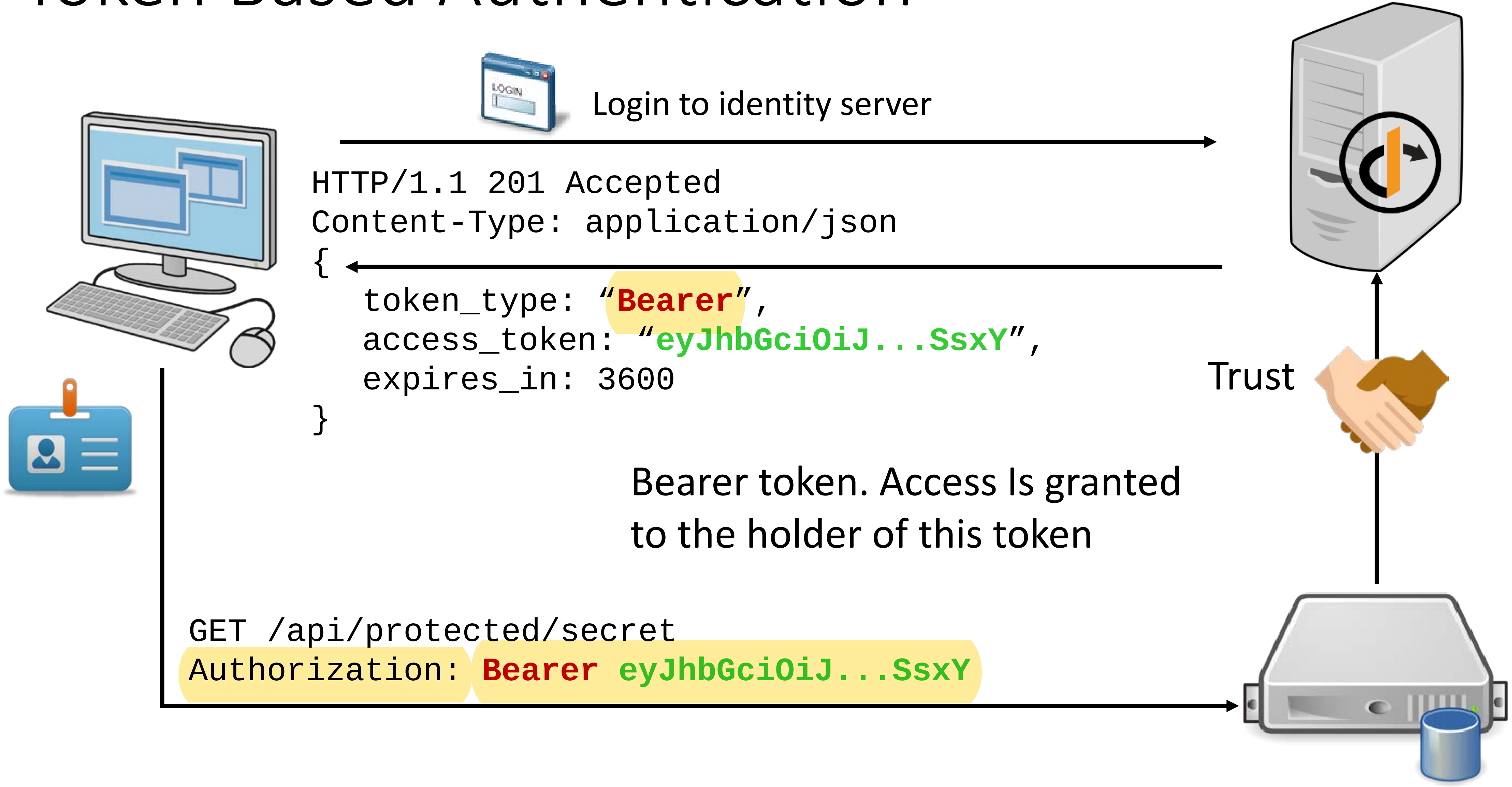
Header

Payload

Signature

```
try {
   const decoded = jwt.verify(token, ‘secret’);
} catch (e) {
   console.log(‘Token failed verification’);
}
```

Token Based Authentication



```
HTTP/1.1 201 Accepted
Content-Type: application/json
{
   token_type: "Bearer",
   access_token: "eyJhbGciOiJ...SsxY",
   expires_in: 3600
}
```

Bearer token. Access Is granted to the holder of this token

```
GET /api/protected/secret
Authorization: Bearer eyJhbGciOiJ...SsxY
```

Generating JWT with Passport

- Generate JWT token after a request has successfully authenticated with the installed strategy eg .local
- Use the user's information passed from authentication callback to generate the token
  - Get additional information if required
- Pass the generated token back as `application/json` type with the following information
  - `token_type` - the type of token, Bearer
  - `access_token` - the JWT token
  - `expires_in` - the number of seconds that the token will expire

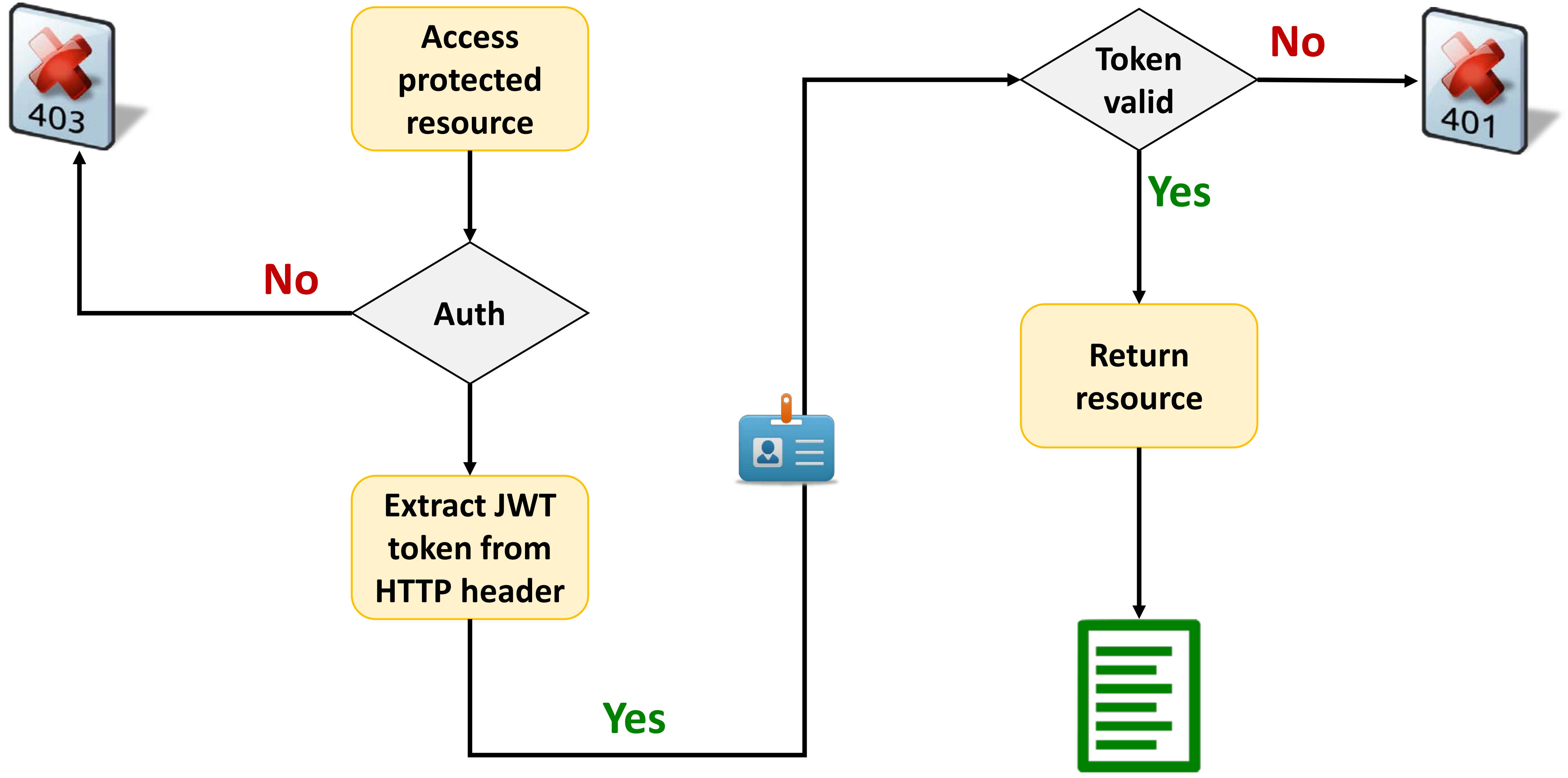
Generating JWT with Passport

```
passport.use(new LocalStrategy( { /* configuration */ },
  (req, username, password, done) => {
    const userDetails = //get from database
     //assume authentication is successful
    return (done(null, userDetails));
  }
));

app.post(‘/login’, passport.authenticate(‘local’, { session: false }),
  (req, resp) => {
    const token = jwt.sign({ sub: req.user.username, ..}, ‘secret’);
    resp.status(201).json({
      token_type: ‘Bearer’, access_token: token,
      expire_in: 1800
    });
  }
);
```

Use request details from the strategy to generate and return the token to the client

Authorizing JWT with Passport



Verifying a Request

Verify request has a JWT bearer token

```
app.get('/api/protected/customer/:cid',
  (req, resp, next) => {
     const authHeader = req.get('Authorization');
     if (!(authHeader && authHeader.startsWith('Bearer ')) {
        resp.status(403).json({error: 'Not authenticated'}); return;
     }
     const token = authHeader.substring('Bearer '.length);
     try {
        req.jwtToken = jwt.verify(token, 'secret');
        next();
     } catch (e) {
        resp.status(403).json({error: e}); return;
     }
  },
  (req, resp) = > {
     req.jwtToken
  }
)
```

Check if the request has the Authorization header and that it is a Bearer type authorization

Extract the token

Verify and decode the token. Add the decoded token to the request object so that it is available in subsequent middleware

Pass to the next middleware

The token available in the next middleware

Token Verification vs Decode

- JWT claims/payload can be accessed by decoding without verification

```
const payload = jwt.decode(token);
```

- The following show how to return both the payload and the header

```
const decoded = jwt.decode(token, { complete: true });
console.log(‘header = ‘, decoded.header);
console.log(‘payload = ‘, decoded.payload);
```

- Note that decodes only decodes (un-base64) an unencrypted token. No verification is performed on the token.
- Use this option if you only wishes to extract the claims or that you are confident that the token is valid