The Cookie crumbles,
JSON Web Tokens to the rescue !

Ayan Dave
@daveayan

Fusion Alliance

Speaker

Ayan Dave

Linkedin – http://linkedin.daveayan.com

Github – http://github.daveayan.com

Blog – http://blog.daveayan.com

Java Competency Lead at Fusion Alliance

Hiring Engineers

adave@fusionalliance.com

Next 50 ish mins …

Outline

- What is a Cookie
- How is Cookie used
- Few problems with Cookie based solution
- What is a JWT
- How is a JWT used
- Building some other use cases with JWT
- Code
- Conclusion

Building Apps with Http Cookie

IETF Specification: RFC6265

https://tools.ietf.org/html/rfc6265

Fusion Alliance

What is a Cookie?

HTTP Cookie

A piece of data generated by the server and saved to the browser

This data is served up by the browser to the server every time a request is made to the same server

The server can lookup this data on subsequent requests and use the data stored within

Structure of a Cookie

Http Cookie

Key Value Pair

Attributes

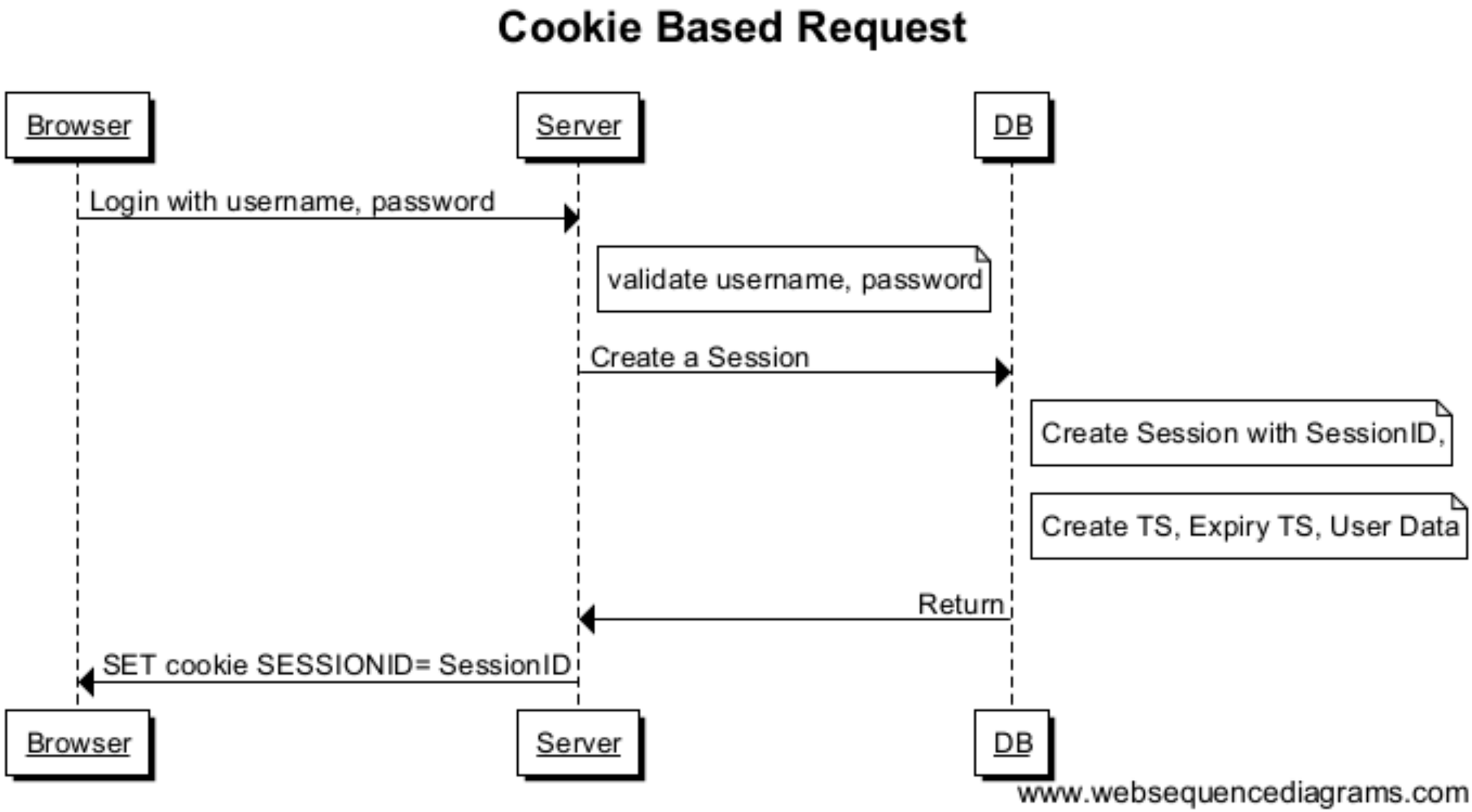
Expires	Max-Age
Domain	Path
Secure	HttpOnly

Set-Cookie and Cookie Headers

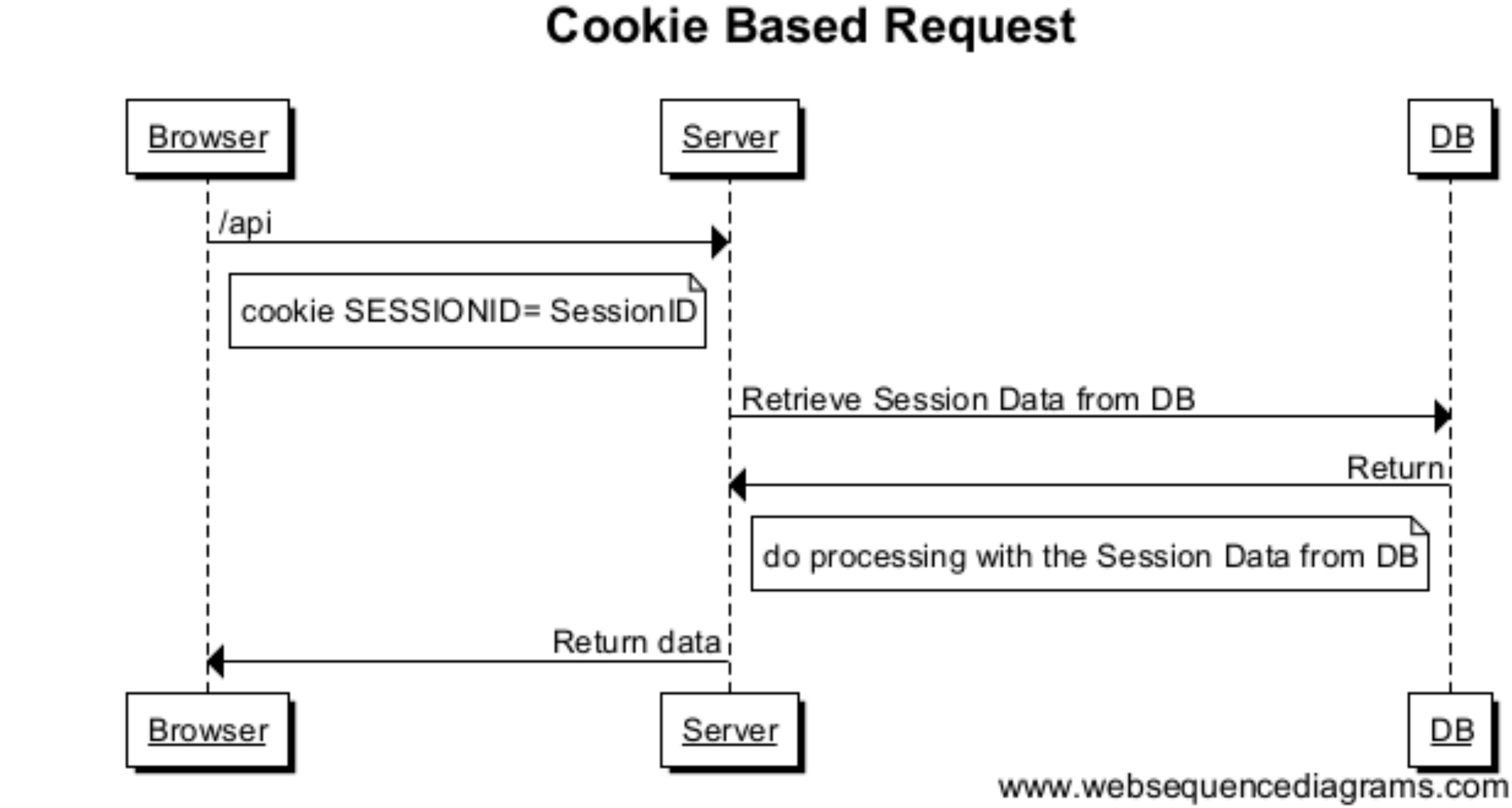
```
Name: JSESSIONID
Value: 0D68F9AD0C564FC772D5FEFB5703AFCF
Path: /account/
Domain: example.com
HttpOnly: FALSE
Expires: Session
```

How is a Cookie used by Client and Server?

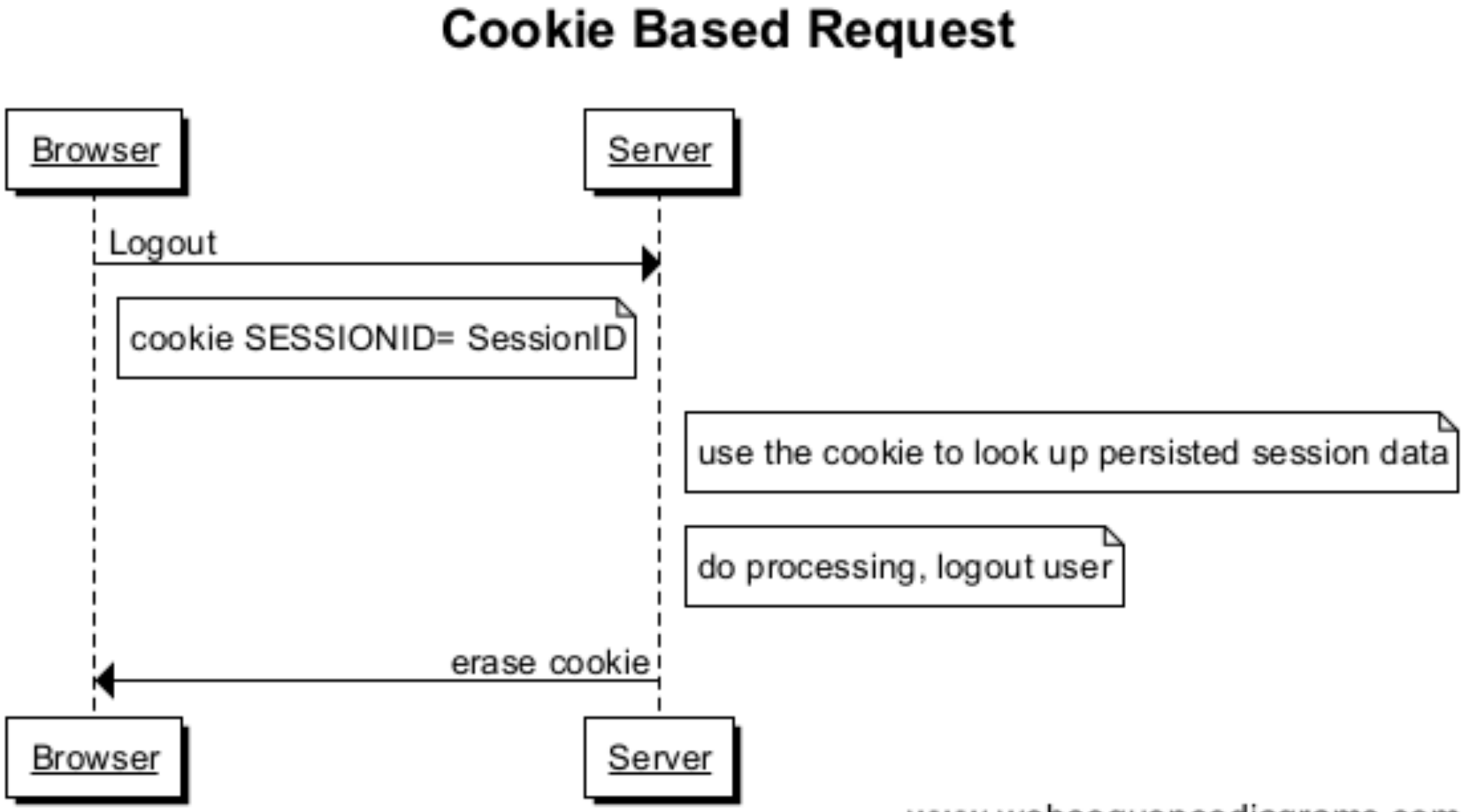
How is a Cookie Used?



How is a Cookie Used?



How is a Cookie Used?



Problems with Cookie based system

Reliance on the User Agent / Browser

Fusion Alliance

Separation of Designation from Authorization

Fusion Alliance

Reliance on DNS and other Infrastructure

Fusion Alliance

Privacy Considerations

Fusion Alliance

Build Stateless Systems

Fusion Alliance

Alternative – Use Access Tokens (we are not talking JWT yet)

What is an Access Token?

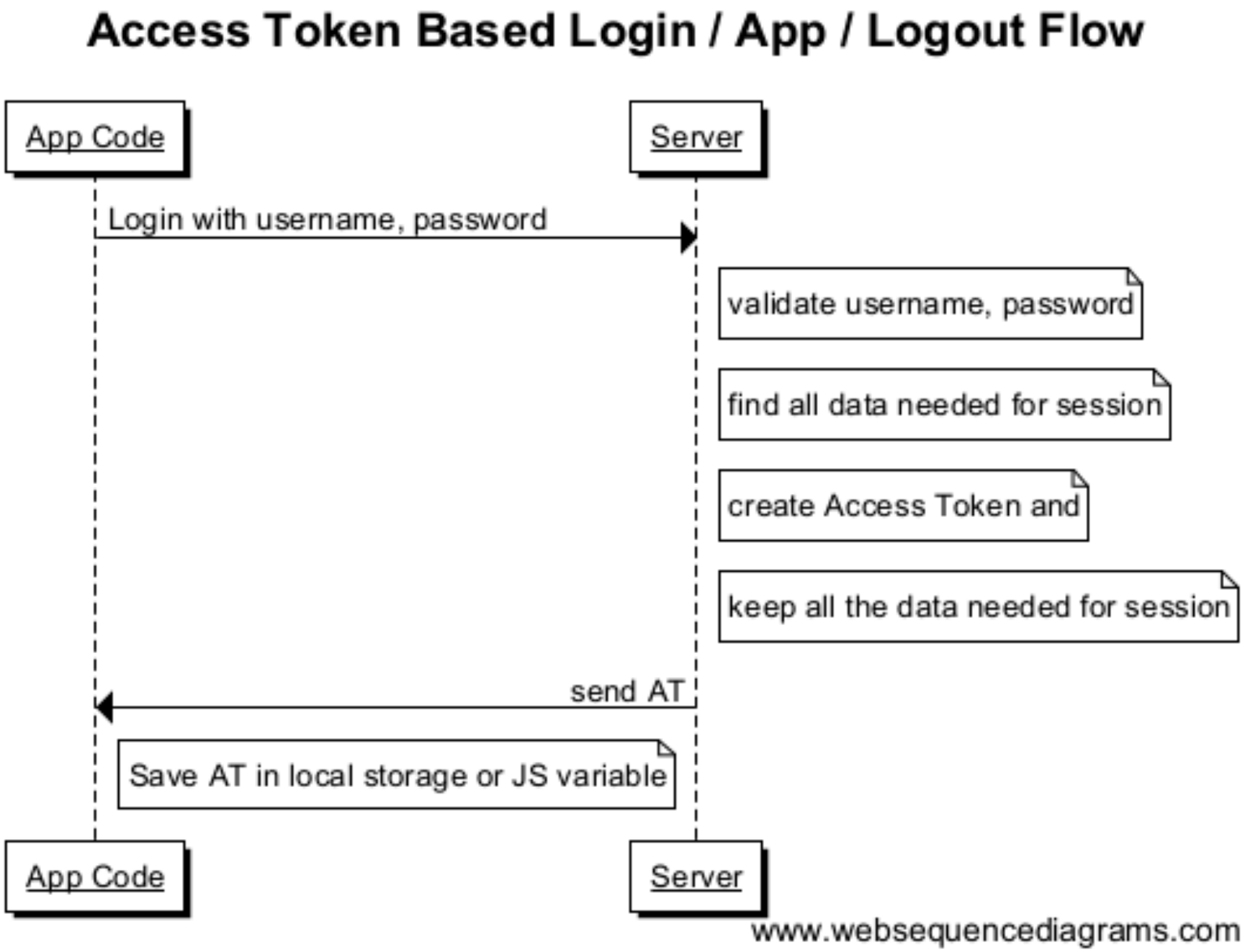
- https://tools.ietf.org/html/rfc6749#page-10

1.4. Access Token

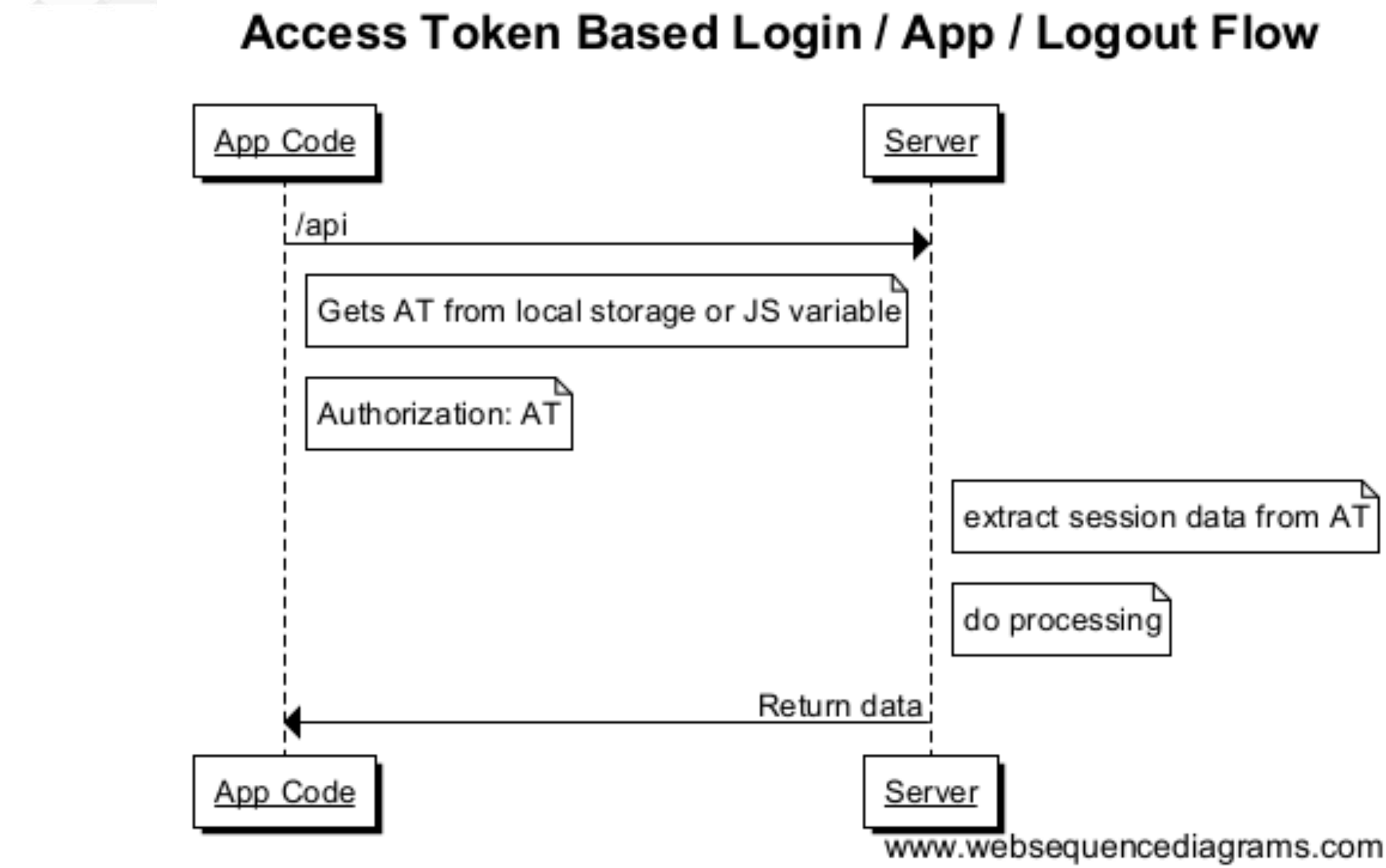
Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.

How is an Access Token used by Client and Server?

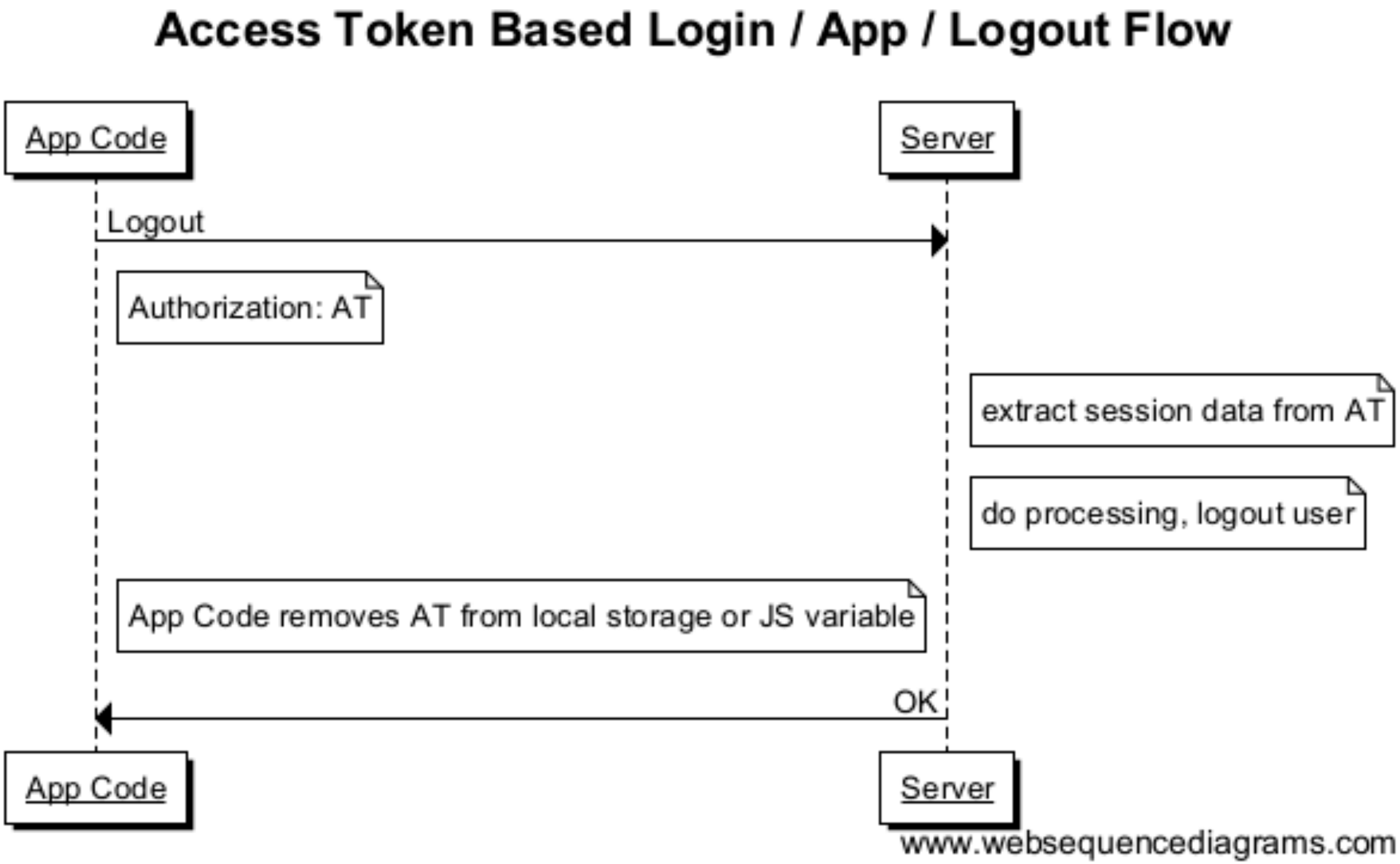
How is a Access Token used?



How is a Access Token used?



How is a Access Token used?



Access Token

Good Access Token

- Well Structured, Well Defined
- URL Safe
- Self Contained
- Compact

JSON Web Tokens

IETF Specification – RFC7519

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

Fusion Alliance

What is a JSON Web Token?

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdW
IiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lI
iwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrH
DcEfxjoYZgeFONFh7HgQ

ANATOMY OF A JWT

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) secret base64 encoded
```

Signature Verified

JWT HEADER

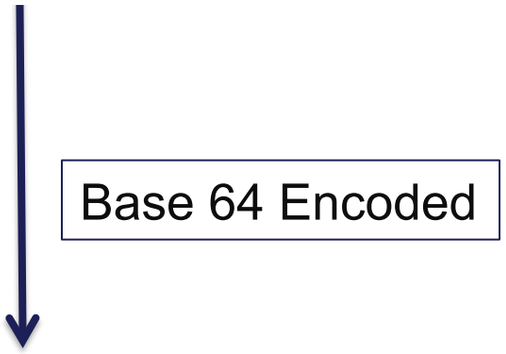
```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Base 64 Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

JWT PAYLOAD

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```




```
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiYWRtaW4iOnRydWV9
```

Generate a Signature

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Base 64 Encoded

```
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

Full Token

```
{
  "kid": "1",
  "alg": "RS256"
}
```

+

```
{
  "key": "value",
  "aud": "Service-1",
  "sub": "POST",
  "jti": "4Qj0ZF3-iLIn8TQY35Jaug",
  "iat": 1460823863,
  "iss": "Consumer-1"
}
```

↓ Base 64 Encoded

↓ Base 64 Encoded

eyJraWQiOiIxIiwiYWxnIjoiUlMyNTYifQ.eyJrZXkiOiJ2YWx1ZSIsImF1ZCI6IlNlcnZpY2UtMSIsInN1YiI6IlBPU1QiLCJqdGkiOiI0UWowWkYzLWlMSW44VFFZMzVKYXVnIiwiaWF0IjoxNDYwODIzODYzLCJpc3MiOiJDb25zdW1lci0xIn0.

+

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Base 64 Encoded →

TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

URL Safe – Base64 Encoded

Fusion Alliance

Its Signing, Not Encryption

Fusion Alliance

Its Signing, Not Encryption

- HMAC Based
- PPK using RSA
- Use JWT over TLS / SSL

alg Param Value	Digital Signature or MAC Algorithm
HS256	HMAC using SHA-256
HS384	HMAC using SHA-384
HS512	HMAC using SHA-512
RS256	RSASSA-PKCS-v1_5 using SHA-256
RS384	RSASSA-PKCS-v1_5 using SHA-384
RS512	RSASSA-PKCS-v1_5 using SHA-512
ES256	ECDSA using P-256 and SHA-256
ES384	ECDSA using P-384 and SHA-384
ES512	ECDSA using P-521 and SHA-512
PS256	RSASSA-PSS using SHA-256 and MGF1 with SHA-256
PS384	RSASSA-PSS using SHA-384 and MGF1 with SHA-384
PS512	RSASSA-PSS using SHA-512 and MGF1 with SHA-512
none	No digital signature or MAC performed

Well Structured – JWT Payload

Payload = Claims

Fusion Alliance

Well Structured Token

JWT Claims

Registered Claims

Public Claims

Private Claims

```
{
  "iss": "Consumer-1",
  "sub": "POST",
  "aud": "Service-1",
  "exp": 1462165511,
  "nbf": 1462161731,
  "iat": 1462161911,
  "jti": "8YF-xT62kEIR58OsLdfaxg",
  "key": "value"
}
```

JWT Claims – Registered Claims

Registered Claims

```
{
  "iss": "Consumer-1",
  "sub": "POST",
  "aud": "Service-1",
  "exp": 1462165511,
  "nbf": 1462161731,
  "iat": 1462161911,
  "jti": "8YF-xT62kEIR58OsLdfaxg",
  "key": "value"
}
```

- “iss” Issuer – who issued the JWT
- “sub” Subject – what is the issuer trying to do
- “aud” Audience – who is it for
- “exp” Expiration Time
- “nbf” Not Before
- “ist” Issued At
- “jti” JWT ID

JWT Claims – Public Claims

Public Claims

Registered with IANA JWT Claims registry

Examples: name, first_name, last_name, nonce

http://www.iana.org/assignments/jwt/jwt.xhtml

```
{
  "iss": "Consumer-1",
  "sub": "POST",
  "aud": "Service-1",
  "exp": 1462165511,
  "nbf": 1462161731,
  "iat": 1462161911,
  "jti": "8YF-xT62kEIR58OsLdfaxg",
  "key": "value"
}
```

JWT Claims – Private Claims

Private Claims

```
{
  "iss": "Consumer-1",
  "sub": "POST",
  "aud": "Service-1",
  "exp": 1462165511,
  "nbf": 1462161731,
  "iat": 1462161911,
  "jti": "8YF-xT62kEIR58OsLdfaxg",
  "key": "value"
}
```

- Agreed upon by the Producer and Consumer
- No Conflict with Registered or Public Claims
- Use with caution, there can be other conflicts
- Makes the token self contained

Well Structured Token

JWT Claims

```
{
  "iss": "Consumer-1",
  "sub": "POST",
  "aud": "Service-1",
  "exp": 1462165511,
  "nbf": 1462161731,
  "iat": 1462161911,
  "jti": "8YF-xT62kEIR58OsLdfaxg",
  "key": "value"
}
```

Registered Claims

“iss”	“sub”	“aud”
“aud”	“exp”	“nbf”
“nbf”	“iat”	“jti”

Public Claims

Claim Names registered in IANA “JSON Web Token Claims” registry

Private Claims

A name that producer and consumer agree to. No Conflict

Well Structured – JWT Header

JWT Header

JWT Header

typ: Type of token

alg: Type of the algorithm

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Compact Token

Fusion Alliance

Compact Token

Can be transmitted over HTTP Header, POST Body or URL Param

Keep an eye on the size though

Building Use Cases with JWT's

Login

Fusion Alliance

Login

issuer = client code

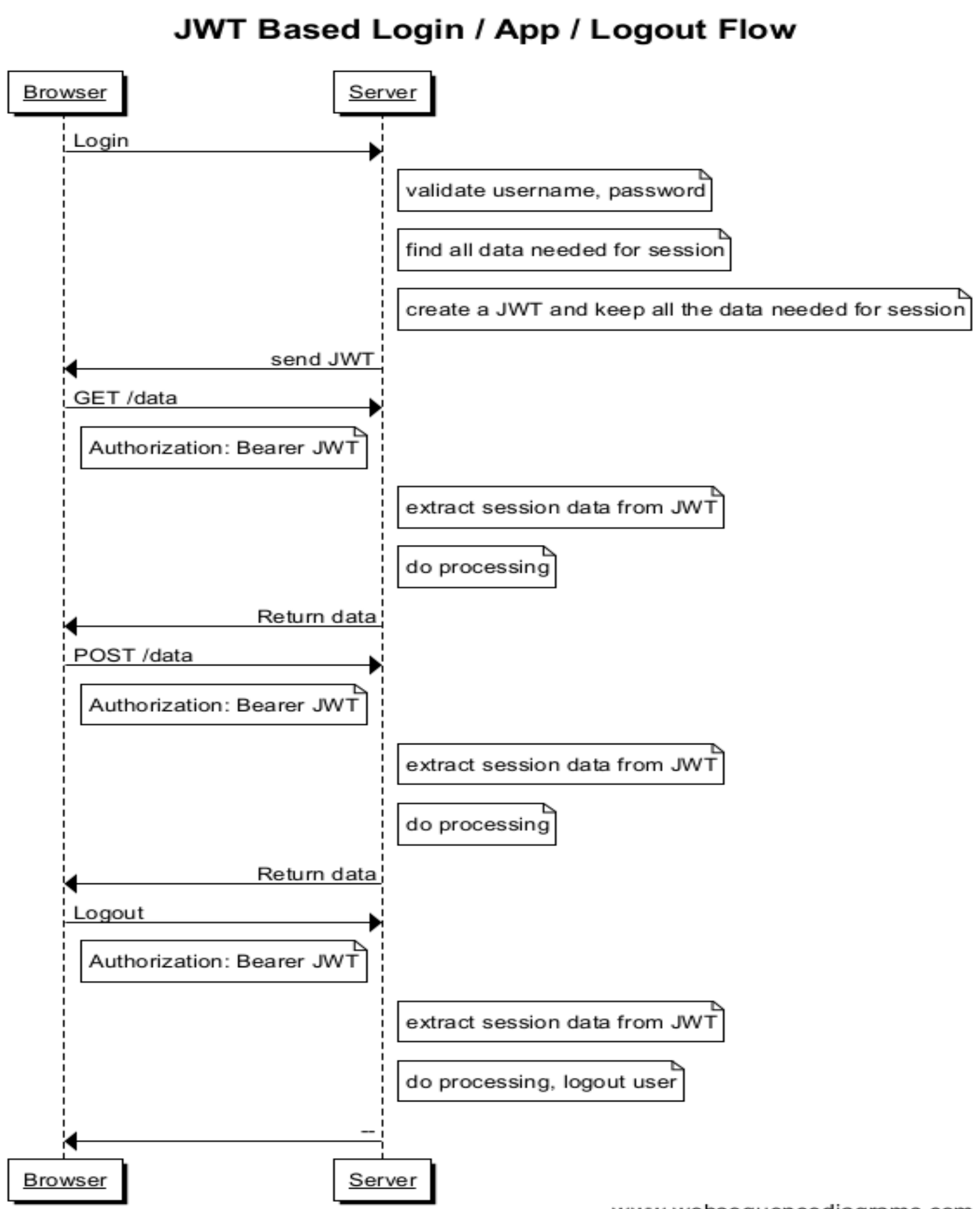
audience = server

subject = server

expiry time = 15 minutes

payload = specific session data, encrypted

Fusion Alliance



Calling REST API's

Fusion Alliance

Calling REST API's

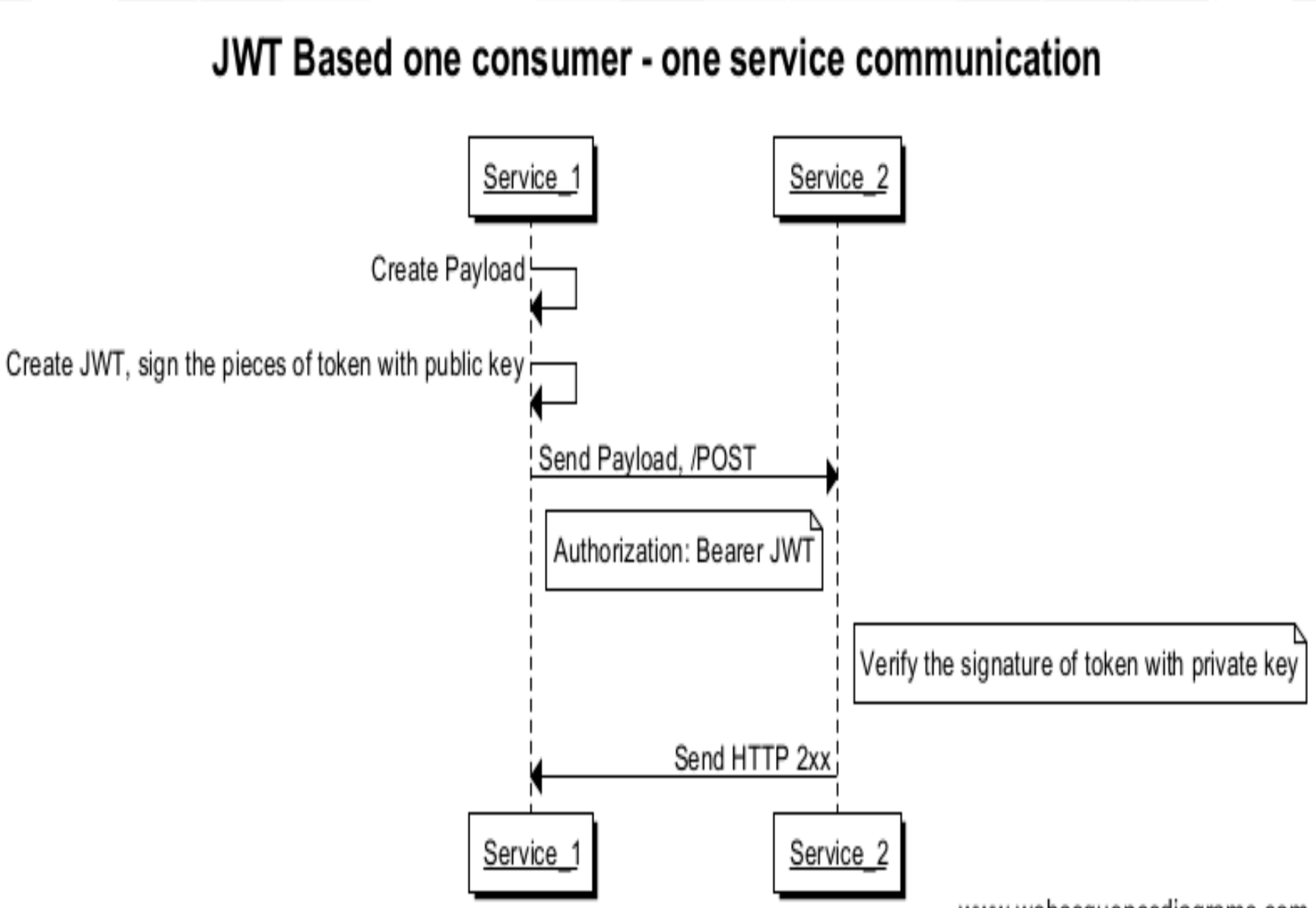
issuer = service 1

audience = service 2

subject = service 2 | methods

expiry time = 5 seconds

payload = the md5 of the post body



Federated Authentication

Fusion Alliance

Federated Authentication

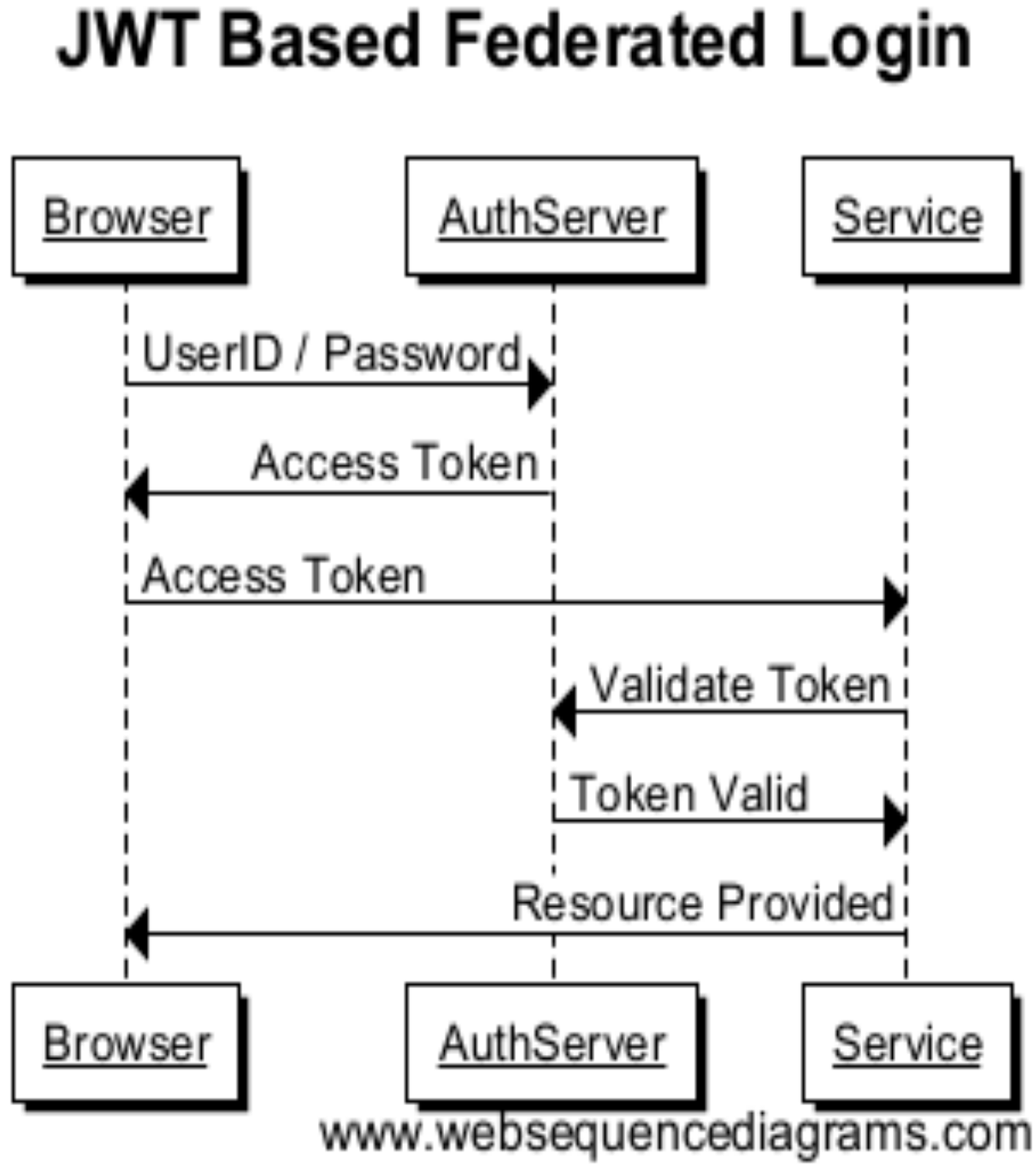
issuer = authserver

audience = service

subject = auth server

expiry time = 10 minutes

payload = more information about user, encrypted



Fusion Alliance

And many more use cases

Fusion Alliance

Code

Libraries

Java

✓ Sign	✓ HS256
✓ Verify	✓ HS384
✓ `iss` check	✓ HS512
✓ `sub` check	✓ RS256
✓ `aud` check	✓ RS384
✓ `exp` check	✓ RS512
✓ `nbf` check	✓ ES256
✓ `iat` check	✓ ES384
✓ `jti` check	✓ ES512

__b_c View Repo

```
maven: org.bitbucket.b_c / jose4j / 0.4.4
```

.NET

✓ Sign	✓ HS256
✓ Verify	✓ HS384
✓ `iss` check	✓ HS512
✓ `sub` check	✓ RS256
✓ `aud` check	✓ RS384
✓ `exp` check	✓ RS512
✓ `nbf` check	✗ ES256
✓ `iat` check	✗ ES384
✓ `jti` check	✗ ES512

Microsoft ☆ 118 View Repo

```
Install-Package
System.IdentityModel.Tokens.Jwt
```

Node.js

MINIMUM VERSION 4.2.2 ⓘ

✓ Sign	✓ HS256
✓ Verify	✓ HS384
✗ `iss` check	✓ HS512
✗ `sub` check	✓ RS256
✓ `aud` check	✓ RS384
✓ `exp` check	✓ RS512
✗ `nbf` check	✓ ES256
✗ `iat` check	✓ ES384
✗ `jti` check	✓ ES512

Auth0 ☆ 2409 View Repo

```
npm install jsonwebtoken
```

Create Token with a Secret

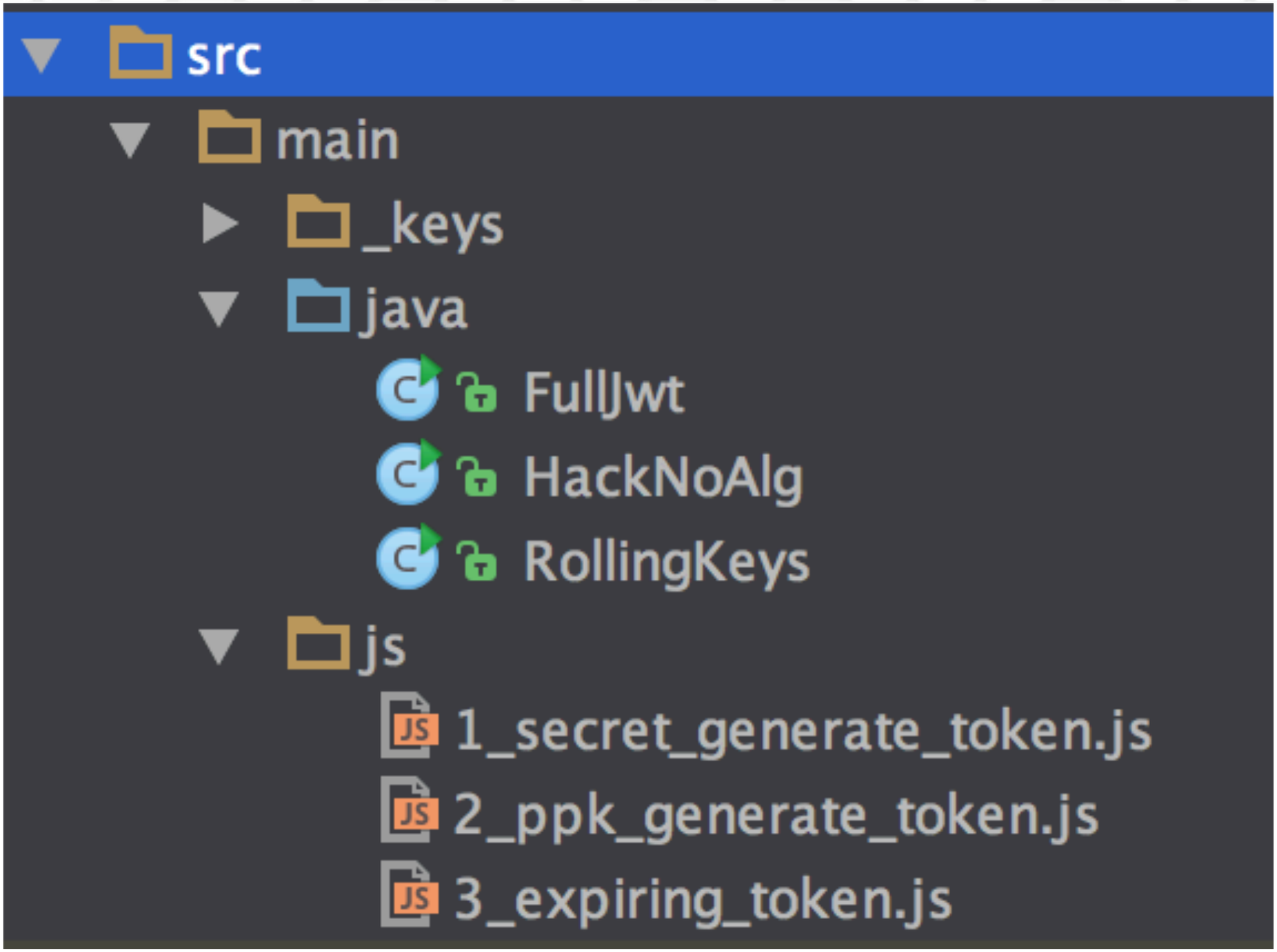
```
var jwt = require('jsonwebtoken');

var dataToEncode = {
    key1: 'value1',
    key2: 'value2'
}
console.log("\nData to Encode = ");
console.log(dataToEncode);

var token = jwt.sign(dataToEncode, 'this_is_a_secret');
console.log("\nEncoded Token = \n" + token + "\n");

var decodeResult = jwt.verify(token, "this_is_a_secret");
console.log("\nDecoded Data with the good secret =");
console.log(decodeResult);

try {
    var decodeResultWrong = jwt.verify(token, "this_is_NOT_the_secret");
    console.log("\nDecoded Data with bad secret =");
    console.log(decodeResultWrong);
} catch(error) {
    console.log("\nCannot decode result with wrong secret");
}
```

Create Token with a Secret - Output

```
Data to Encode =
{ key1: 'value1', key2: 'value2' }

Encoded Token =
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJrZXkxIjoidmFsdWUxIiwia2V5MiI6InZhbHVlMiIsImlhdCI6MTQ2MTUyNTI3Nn0.FKIHHPUTfc8gLp9tDBT7javsUzICIJXA6FLJZtOhvAE


Decoded Data with the good secret =
{ key1: 'value1', key2: 'value2', iat: 1461525276 }

Cannot decode result with wrong secret
```

More Code

https://github.com/daveayan/jwt-presentation



Information Security

Information Security

Confidentiality

Availability

Integrity

Information Security

JWT brings Integrity to the equation

Its signing, not encryption

Encrypt a token, use JWE – RFC 7516

Other Related Specs

Specifications

JWT Spec	The jose working group within ietf prepared the specification
	https://datatracker.ietf.org/wg/jose/charter/
	Javascript Object Signing and Encryption
	RFC7519

Related Specs	json web signature (jws) – rfc 7515
	json web encryption (jwe) – rfc 7516
	json web key (jwk) – rfc-7517
	json web algorithms (jwa) – rfc7518

JSON Web Encryption

JWE

JWE can be useful in certain cases

JWE encrypts the json’s of the token as well, thus bringing confidentiality into the equation

JSON Web Encryption (JWE) represents encrypted content using JSON-based data structures [RFC7159]. The JWE cryptographic mechanisms encrypt and provide integrity protection for an arbitrary sequence of octets.

Conclusion

Conclusion

Key Points

- JWT is an open standard – RFC7519
- Just a simple data structure
- The tokens are compact, self contained, well structured
- Base64 encoded for safe transfer over the wire
- Signed, not encrypted, use over TLS / SSL
- Ensures Integrity of the Payload
- https://jwt.io
- Cookies are good and have use cases

Conclusion

Ayan Dave

adave@fusionalliance.com

http://linkedin.daveayan.com

http://github.daveayan.com

http://blog.daveayan.com

https://github.com/daveayan/jwt-presentation

Questions

Enjoy rest of the conference …

Fusion Alliance