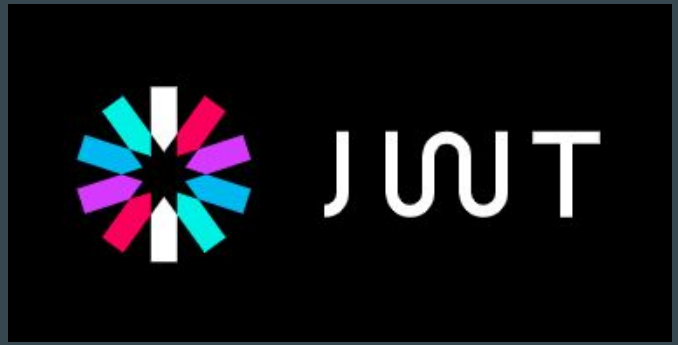
COM3123 - Lecture 12

Security & Authentication



Topics

- **Web Storage**
- **Authentication & Authorization**
- **JWT Tokens**

Web Storage

Brief History of Cookies

- Cookies were introduced by Netscape in 1994 as part of Netscape Navigator *(which later became Mozilla Firefox)*
- Netscape was looking for a way to retain a sense of state, but not on server side.
- Cookies were chosen as the technical solution.
- Simple in form and structure, cookies allow small files of info to be stored on user computer
  (ie. Last logged in, Last page viewed, track advertisements etc)

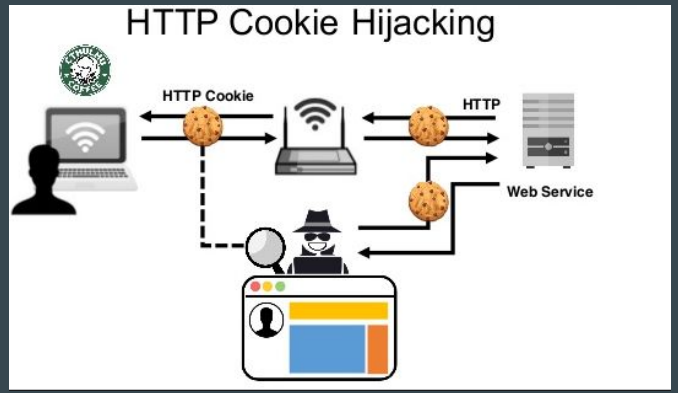
How Cookies work..

```
[sourcecode language="HTML"]
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: page_loaded=25; Expires=Wed, 09 Jun 2021 10:18:14 GMT
[/sourcecode]
```

Name	Value	Expires
page_loaded	25	Wed, 09 Jun 2021 10:18:14 GMT

- Cookies are created when a browser receives a Set-Cookie header from the web in response to a page request.
- The browser receives HTTP 200 code and Set-Cookie header and will create the cookie.
- When expire and be removed from the browser. If it's not expired, all future HTTP requests to that site will get the cookie information appended to the header of the request.

Cookie Security

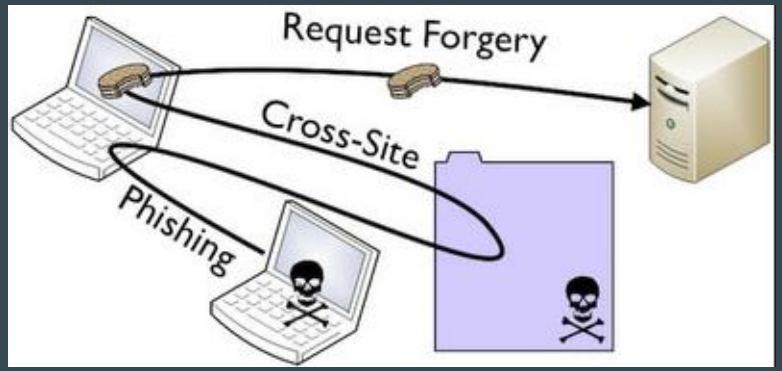


- Another unique aspect of Cookies is that they have some security-related properties that help with ensuring secure data transfer. Cookies can be marked as the following:

- Secure - The browser will only append the cookie if the request is being made over HTTPS

- Http Only -It is not accessible by JavaScript code at all! This can prevent a script injection attack (XSS)

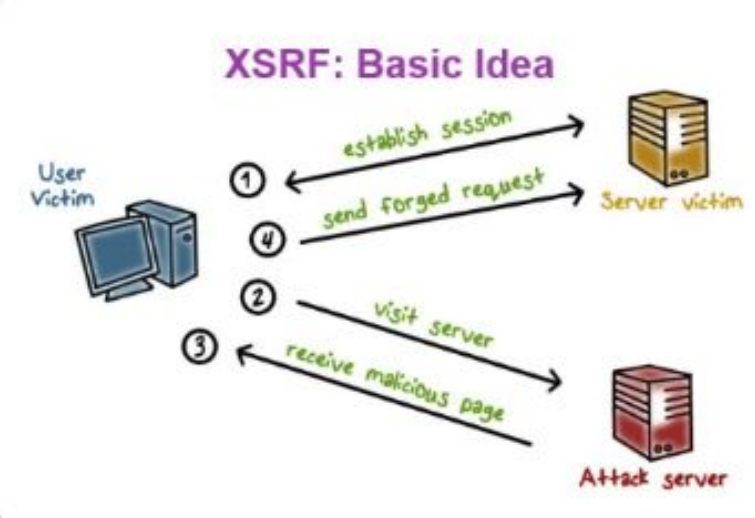
The downside of Cookies



- Cookies have long had a bad reputation with respect to potential privacy and security implications.
- They are vulnerable to a host of security issues including key attacks in Cross Site Request Forgery (CSRF), Cross Site Scripting Attacks (XSS) and Session Hijacking
- However, without them, alot of personalization of the web would not have been possible.

Disadvantages of Cookies - XSRF

Applications with Bearer tokens stored in a Cookie suffer from a vulnerability called Cross-Site Request Forgery, also known as XSRF or CSRF. Here is how it works:



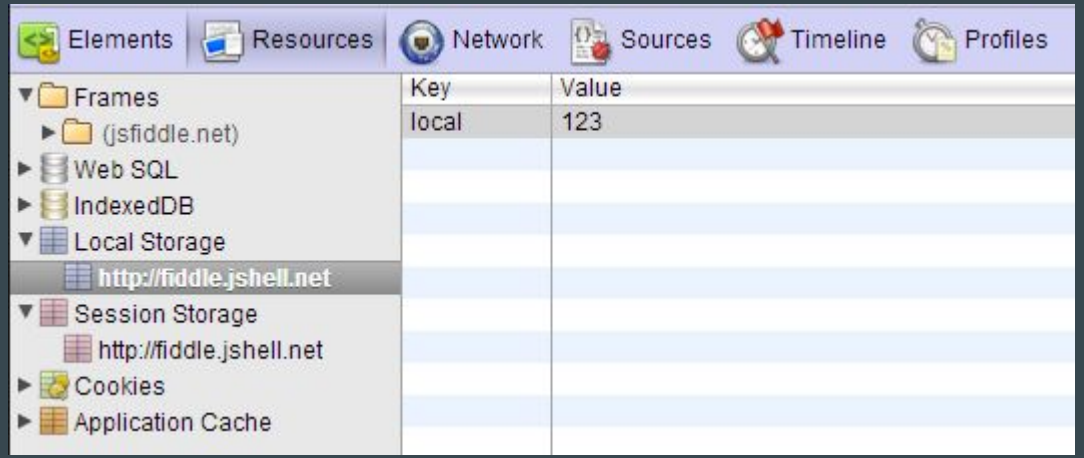
- somebody sends you a link and you click on it
- The link ends up sending an HTTP request to the site under attack containing all the cookies linked to the site
- And if you were logged into the site this means the Cookie containing our JWT bearer token will be forwarded too,(this is done automatically by the browser)
- Server receives valid JWT, no way for the server to distinguish between this attack and valid request.

HTML5 Storage

HTML5 introduced Web Storage as an alternative to Cookies. It has two flavors, local and session.

LocalStorage

- Persists the data until the user explicitly deletes the data
- Use it when you need to store some data for the long term.
- Data is available: Even after the browser window is closed and reopened



HTML5 Session Storage

SessionStorage

- Persists a storage area for the duration of the page session.
- Use it when you need to store some data temporarily
- Data is available: As long as browser is open

Web Storage Pros/Cons

Weaknesses

- Data is stored as simple string, manipulation needed to store objects of different types (ie. bool, int, obj)
- It has a default 5MB limit
- Admin can disable it
- Storage can be slow with complex data sets

Strengths

- Apps can work both online and off
- Has less overhead than cookies, no extra header data is sent with browser requests
- Provides more space than cookies, so more complex data can be kept
- Has the ability to hook into browser events, ie. offline, online

Web Storage API

The Web Storage API is simple and very easy to learn. It consists of only four methods:

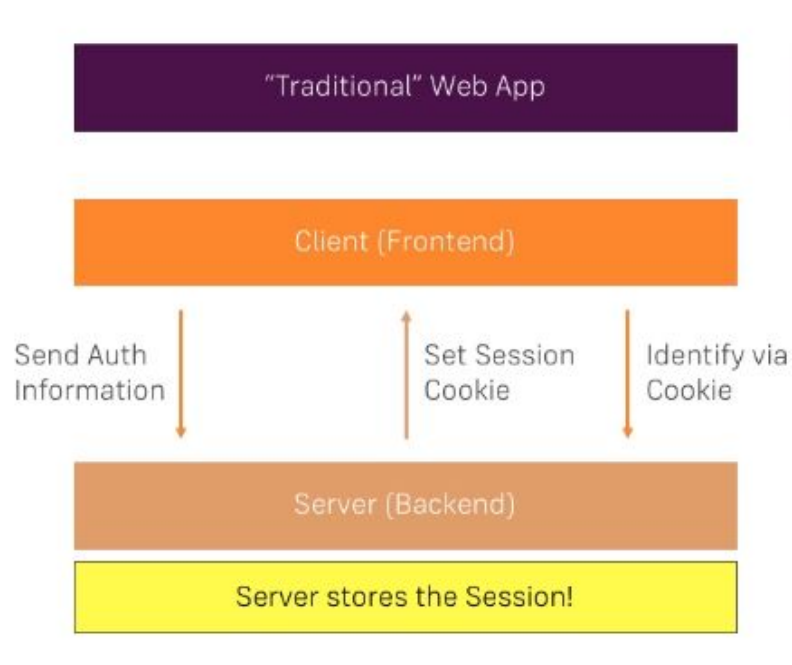
method/attribute	args	returns
setItem	String key, String value	
getItem	String key	String value
removeItem	String key	
clear		

Cookies vs Local Storage vs Session Storage

	Cookies	Local Storage	Session Storage
Capacity	4kb	10mb	5mb
Browsers	HTML4 / HTML 5	HTML 5	HTML 5
Accessible from	Any window	Any window	Same tab
Expires	Manually set	Never	On tab close
Storage Location	Browser and server	Browser only	Browser only
Sent with requests	Yes	No	No

Authentication

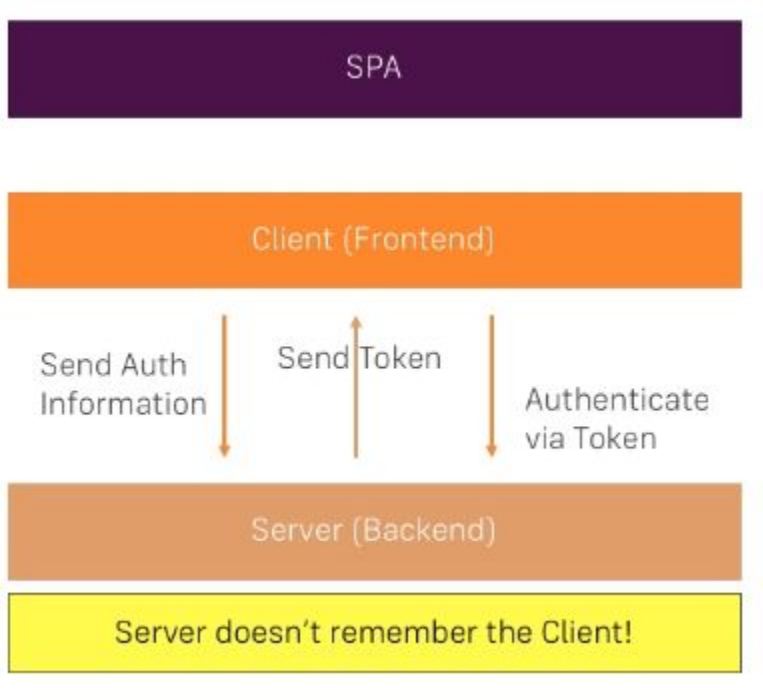
How does Authentication Work?



Traditional Web Application

- Very strong connection between the two layers.
- Server-side framework will use some type of templating engine to render the views.
- The client send credentials and the server will validate and create a session and store data on the server.
- The server is always able to remember the client because he communicating with the client all the time.

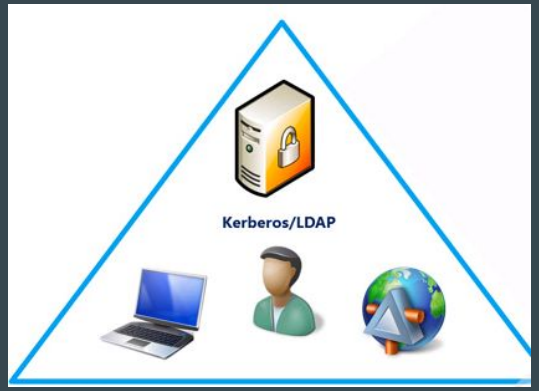
How does Authentication Work?

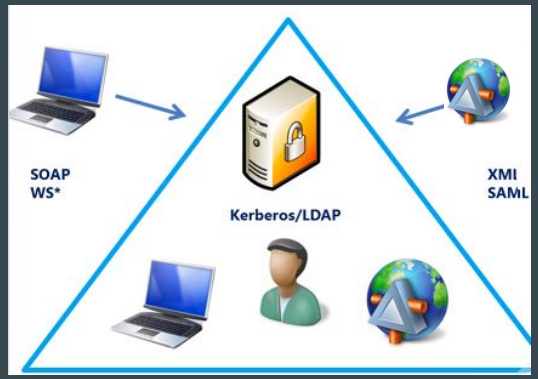


Single Page Application

- We don't have a strong connection between the two layers.
- The client might communicate to the backend from time to time, but that's not guaranteed
- We receive one page and the client Angular is responsible for changing the HTML dynamically.
- The backend might be a very generic RESTful API that can be connected by different web and mobile apps.
- The server doesn't know the client to store a session. A client token must be used to authenticate and passed on each request

Enterprise Security



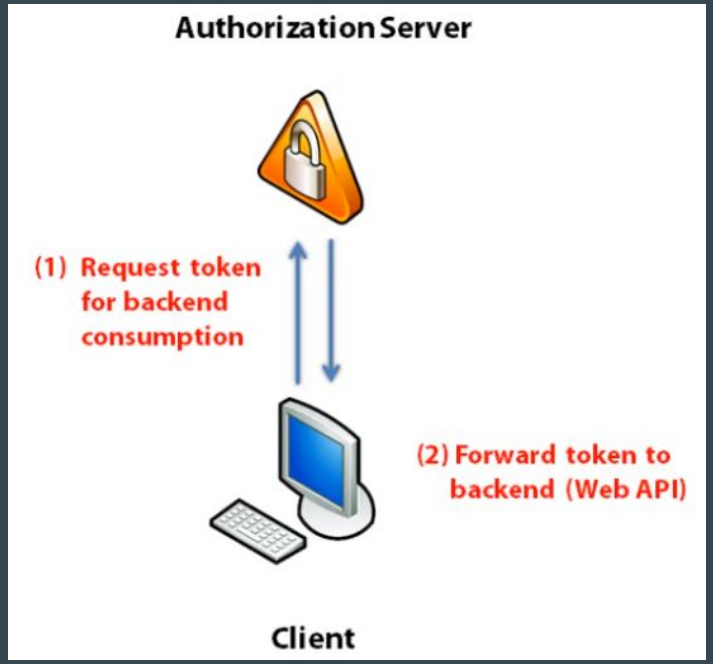


- No trust issues, everyone is known internally in intranet and very clear.
- User, security and client machine with LDAP to assign roles, Active Directory were all in
- It got more complex with business to business security with SOAP and SAML

Mobile Enterprise Apps

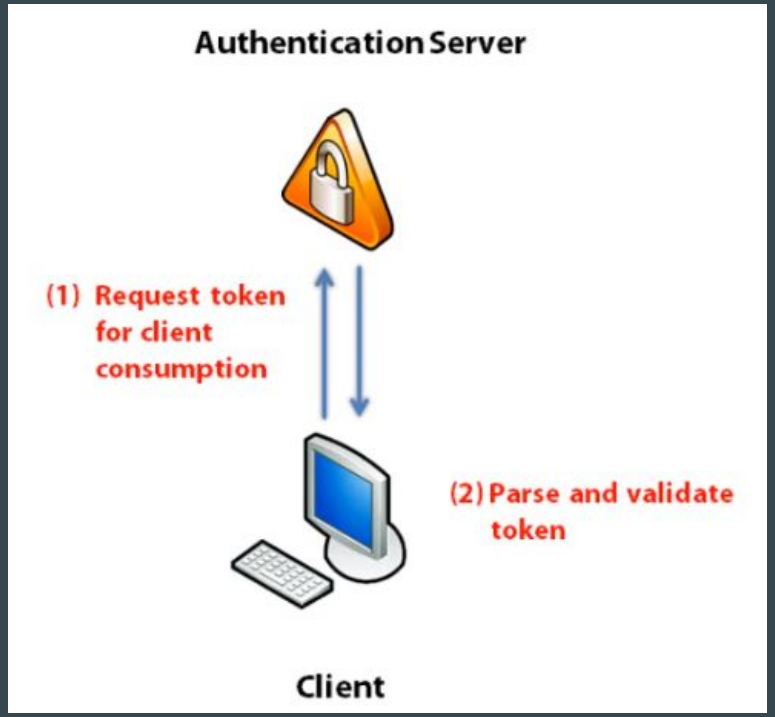
- Mobile Revolution was a game changer, these devices didn't care about enterprise security.
- There were consumer devices and didn't have SOAP, SAML, WS*, but they had HTTP and JSON
- Now we want to use these mobile devices to connect to Enterprise and access data

OAuth2



- We need to know who the client is and authorize them.
- The Client might be 3rd party external product that we don't want to give credentials.
- Handshaking must be done with OAuth2 first before accessing the main server.

OpenID Connect



- We might not need to communicate with backend service or server. In this case we just need to authenticate client and validate identity.
- There might use AuthOath later to get an authorization token to talk to backend server.

JSON Web Token

Purpose of Security Token

- Security tokens are (protected) data structures
  - contain information about issuer and subject (claims)
  - signed (tamper proof & authenticity)
  - typically contain an expiration time

- A client requests a token
- An issuer issues a token
- A resource consumes a token

History

- **SAML 1.1/2.0**
  - XML based
  - many encryption & signature options
  - very expressive
- **Simple Web Token (SWT)**
  - Form/URL encoded
  - symmetric signatures only
- **JSON Web Token (JWT)**
  - JSON encoded
  - symmetric and asymmetric signatures (HMACSHA256-384, ECDSA, RSA)
  - symmetric and asymmetric encryption (RSA, AES/CGM)
  - (the new standard)

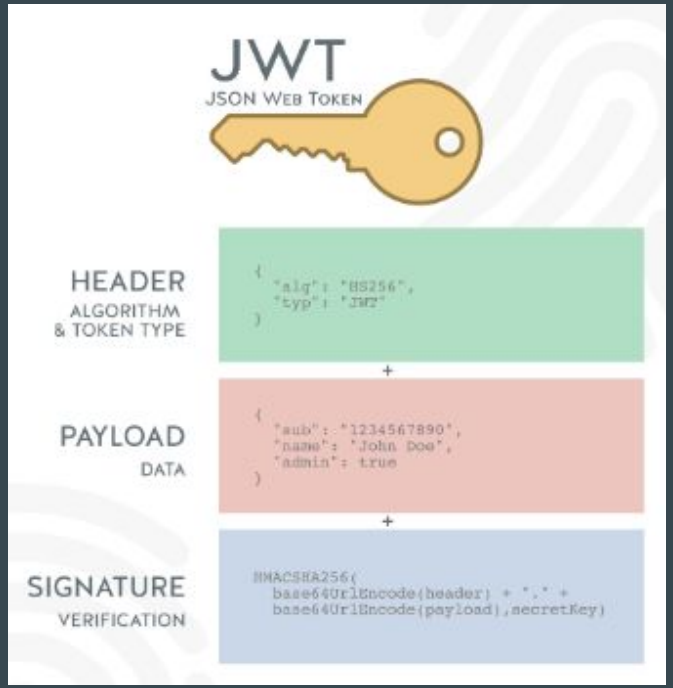
JSON Web Tokens

https://jwt.io/

- A JSON Web Token or JWT is an extremely powerful standard.
- It's a signed JSON object; a compact token format often exchanged in HTTP headers to encrypt web communications.
- It is often used as a standard for API security
- JWT should not be viewed as a complete solution, depending on it alone can be extremely dangerous!

What is JWT?

A JWT is a compact and self-contained way for securely transmitting information between parties as a JSON object



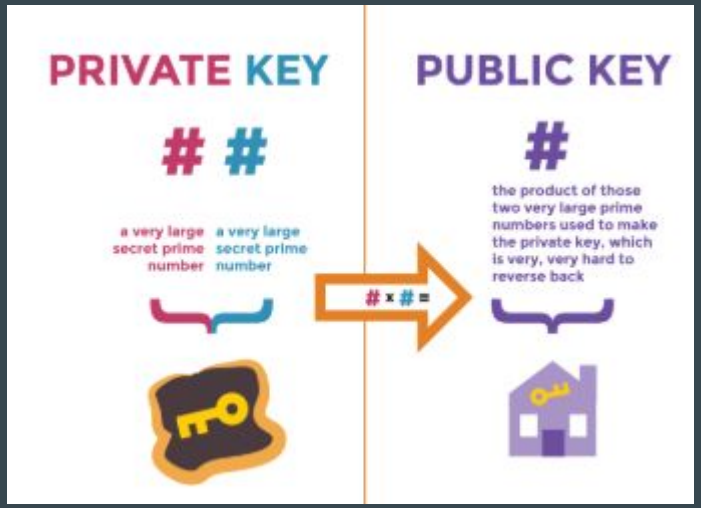
Header

- metadata
- algorithms & keys used

Claims

- Issuer (iss)
- Audience (aud)
- IssuedAt (iat)
- Expiration (exp) **EPOCH Time*
- Subject (sub)

Signed vs Encrypted Tokens

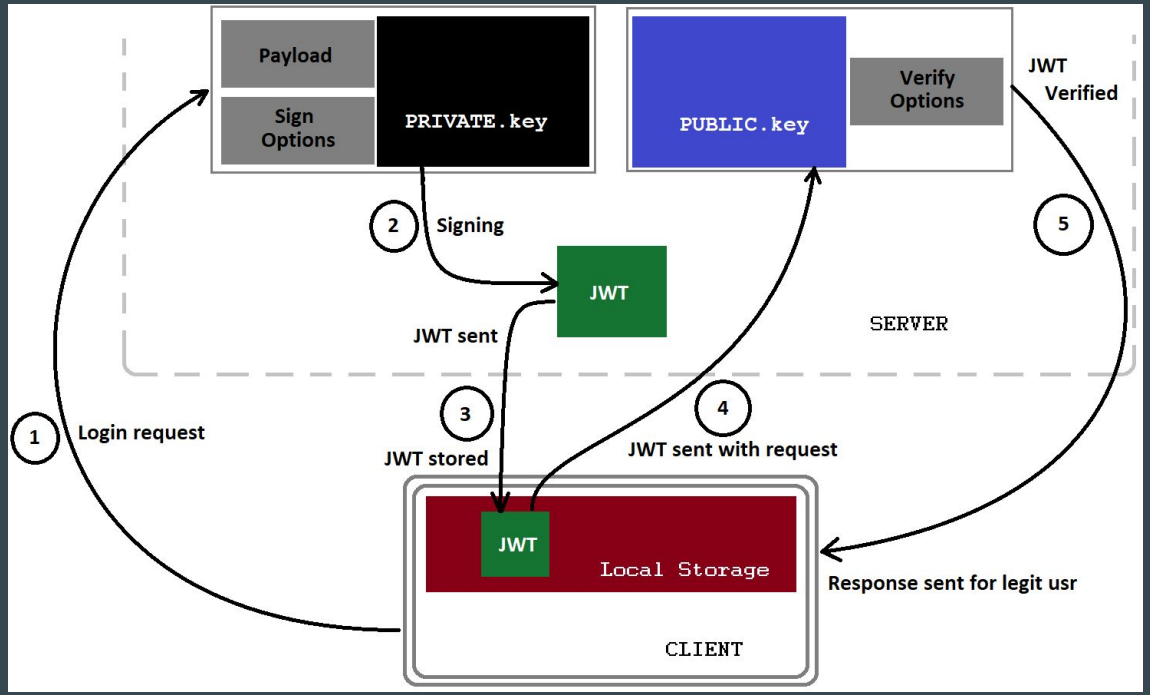


RSA Online Key Generator
https://travistidwell.com/jsencrypt/demo/

- The JWT information can be verified and trusted because it is digitally signed.
- JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.
- Two types of tokens:
  - Signed tokens can verify the integrity of the claims contained within it (public/private key pairs)
  - Encrypted tokens hide those claims from other parties
-

JWT Structure

Creating and Consume JWT Token



- NPM JSON Web Token
  https://www.npmjs.com/package/jsonwebtoken
- Express-JWT Middleware
  https://www.npmjs.com/package/express-jwt

Firebase