OpenID Connect Orlando Backend Meetup 2019

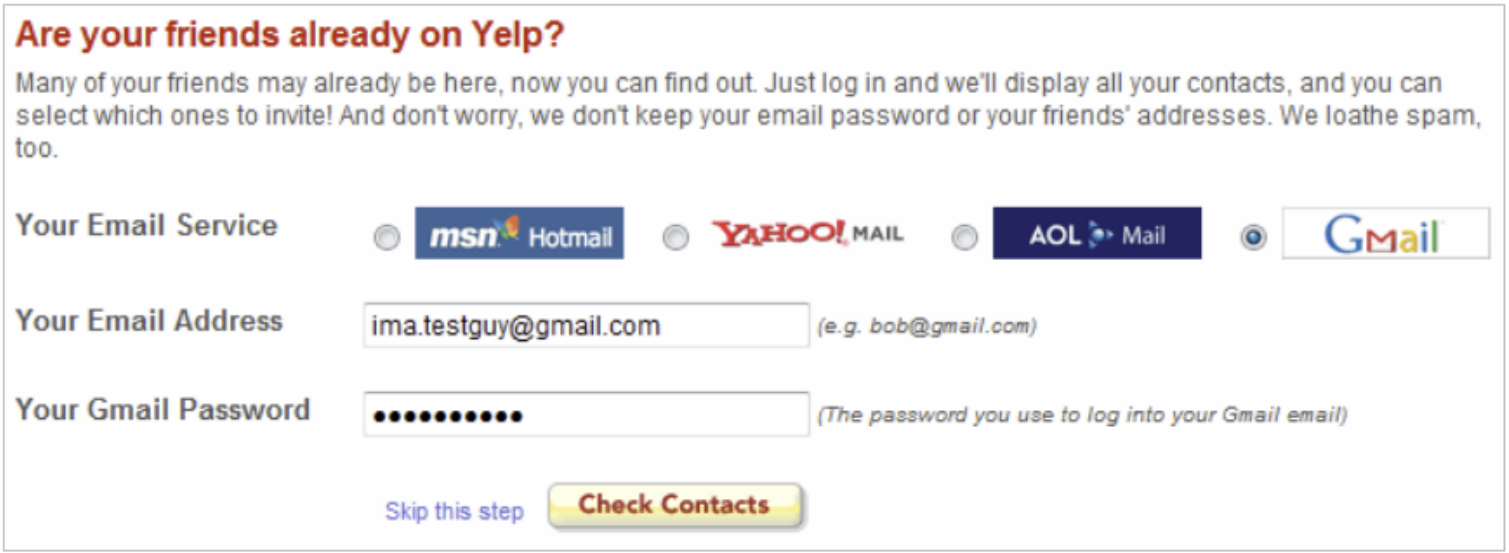
Hi Orlando!

Surya Suluh

https://github.com/ssuluh

https://speakerdeck.com/ssuluh

Yelp Circa 2008



Security Is Hard

- A Lot Of Protocols And Standards
  (Oauth, WS-*, Kerberos, SAML, etc.)
- Reading Specification Is Hard
- Conflicting Information In The Internet
- Not Easy To Implement Correctly
- Evolving Technique, New Exploit, Vulnerability

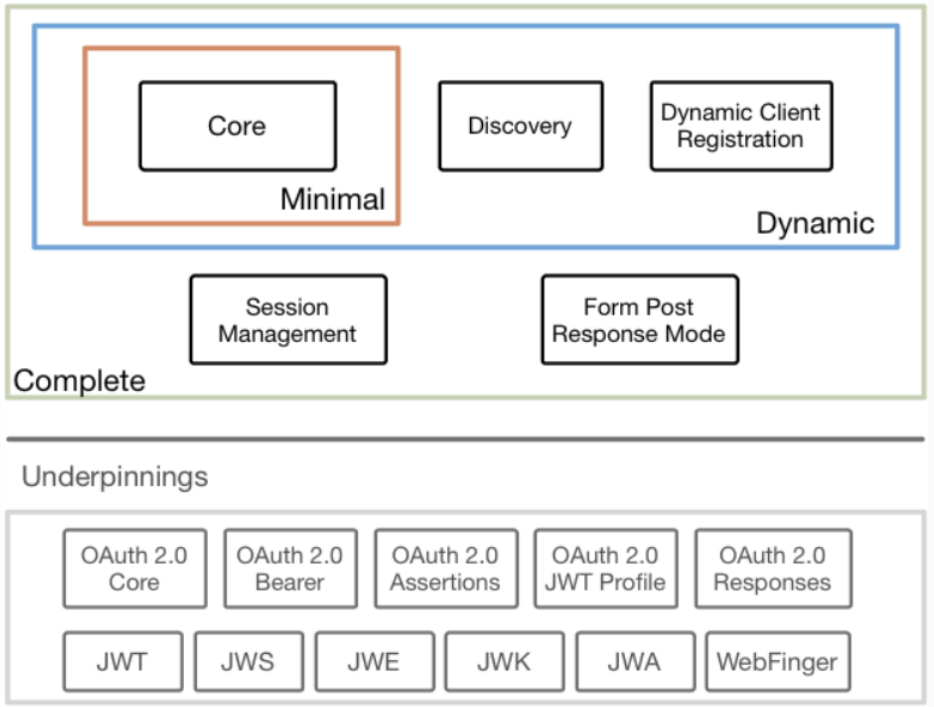
OpenID Connect

- OpenID Foundation
- February 2014
- Authentication
- Built on top of OAuth 2.0

OAuth 2.0

- IETF (RFC 6749)
- October 2012
- Authorization

OpenID Connect



Important Terms

- Client, Relying Party
- Resource Owner
- Authorization Server, STS, IAM
- Resource Server
- Authorization Grant
- Redirect URI
- Access Token

Public Client

- Can't safely store secret
- Exists in client machine
- Typically 1-many clientid-client
  - Dynamic registration to convert to unique clientid

Confidential Client

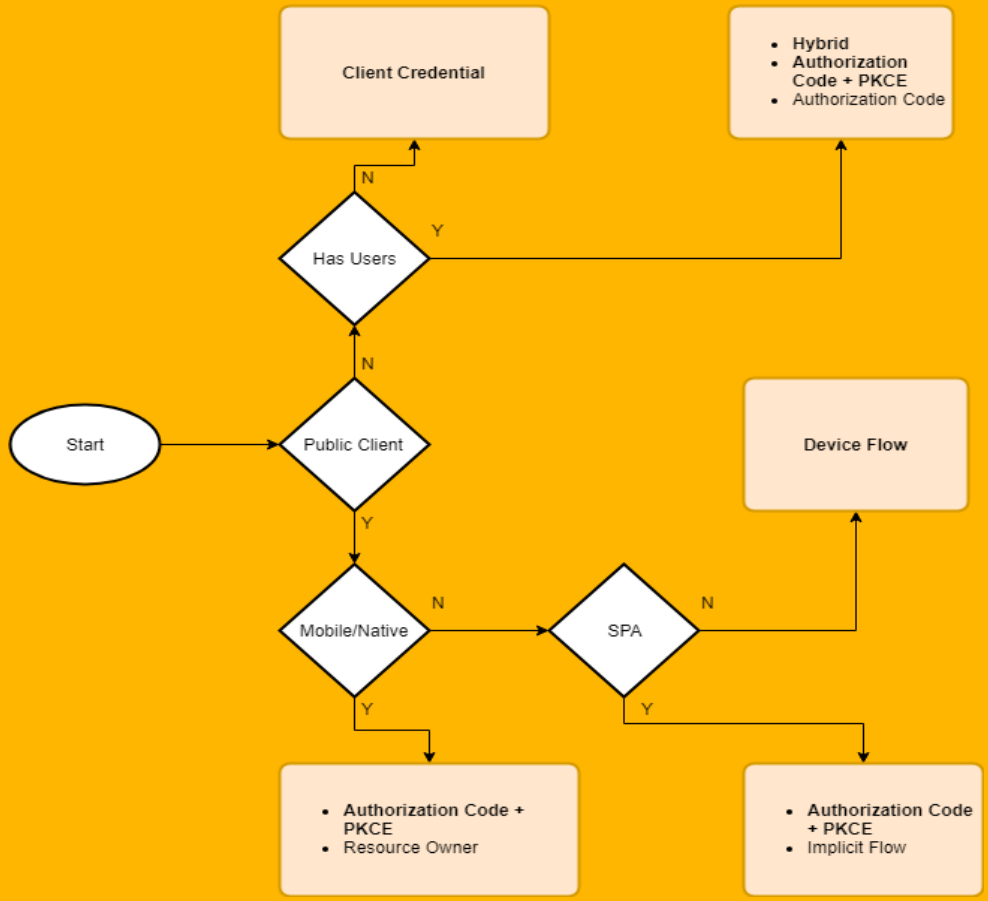
- Can safely store secret
- Exists in our server
- Typically 1-1 client-client id

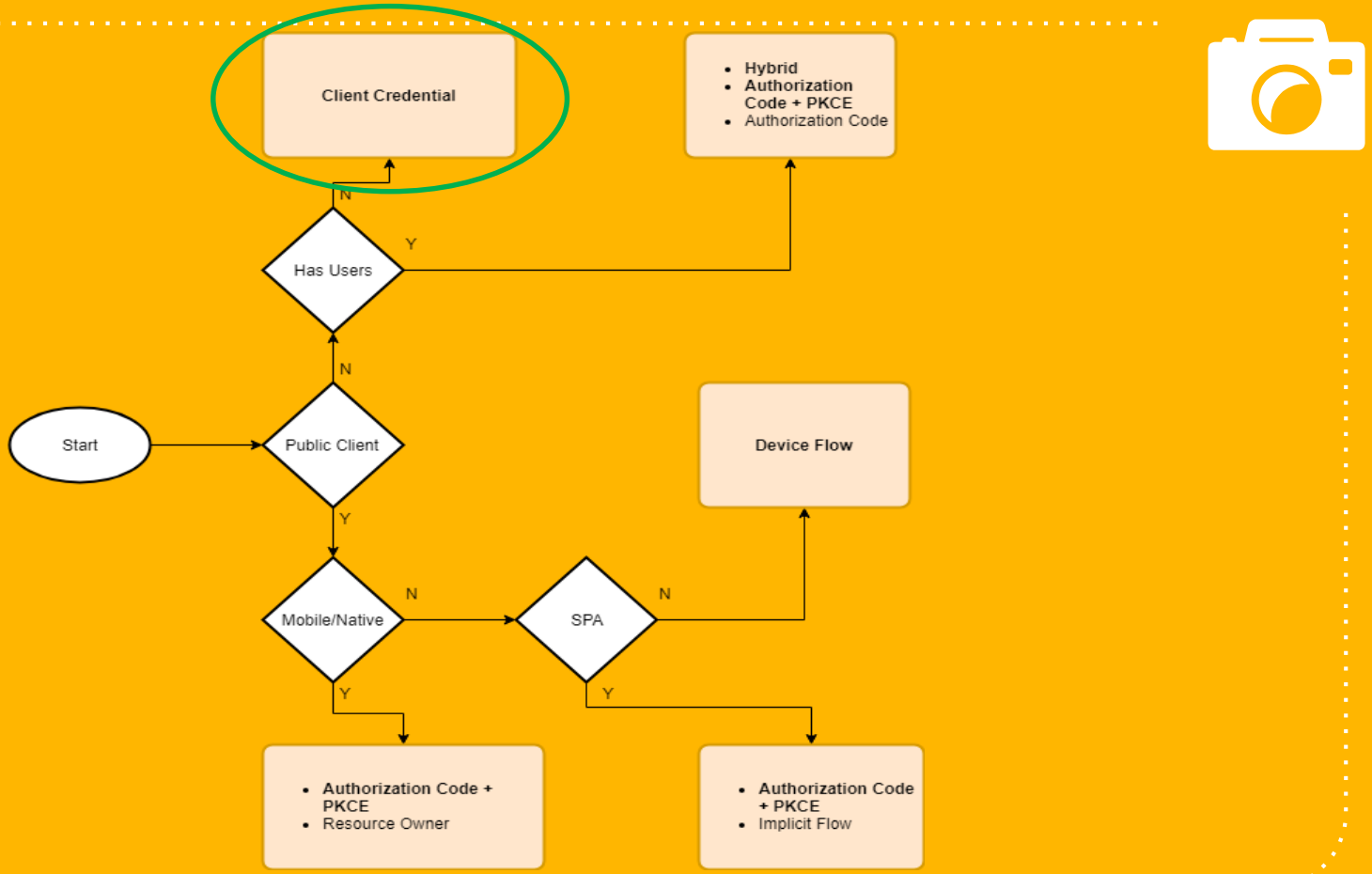
Public Client

- Front Channel
- SPA (Angular, React)
- Mobile App (Native, Cordova)
- PC App (Native, Electron)

Confidential Client

- Back channel and front channel
- Server to Server
- Server side web application (Asp.Net MVC)

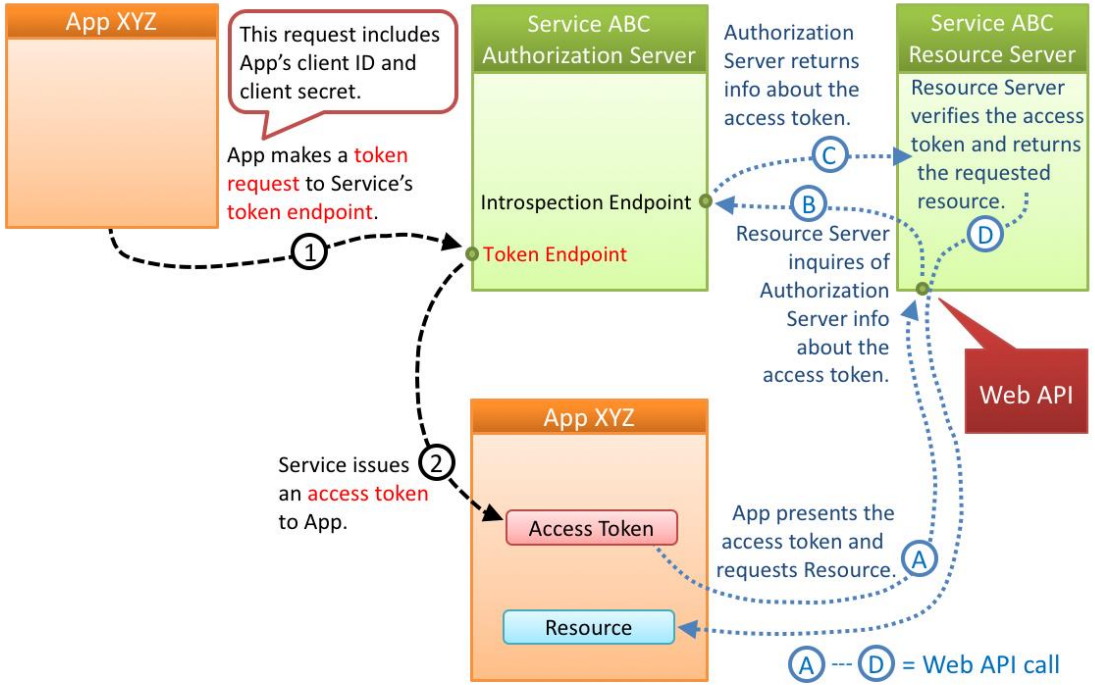

Client Credential
Hybrid
Authorization Code + PKCE
Authorization Code
N
Y
Has Users
N
Start
Public Client
Device Flow
Y
Mobile/Native
N
SPA
N
Y
Y
Authorization Code + PKCE
Resource Owner
Authorization Code + PKCE
Implicit Flow

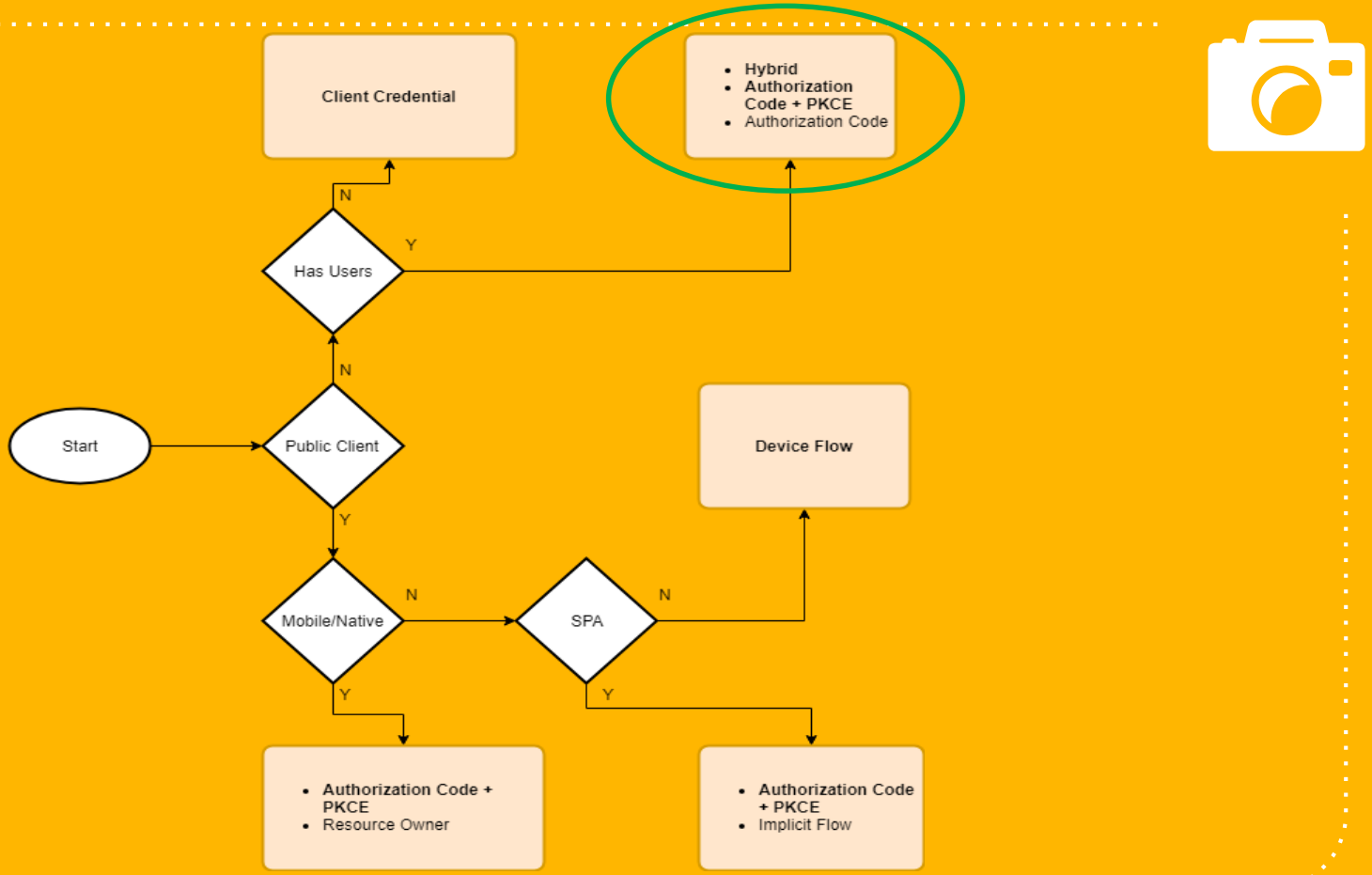

Client Credential
Hybrid
Authorization Code + PKCE
Authorization Code
N
Y
Has Users
N
Start
Public Client
Device Flow
Y
Mobile/Native
N
SPA
N
Y
Y
Authorization Code + PKCE
Resource Owner
Authorization Code + PKCE
Implicit Flow

Client Credential Flows

- Confidential Client
- Use Only Server To Server Communication Only
- Send ClientID, ClientSecret and Scope, get back access token

Client Credentials Flow (RFC 6749, 4.4)

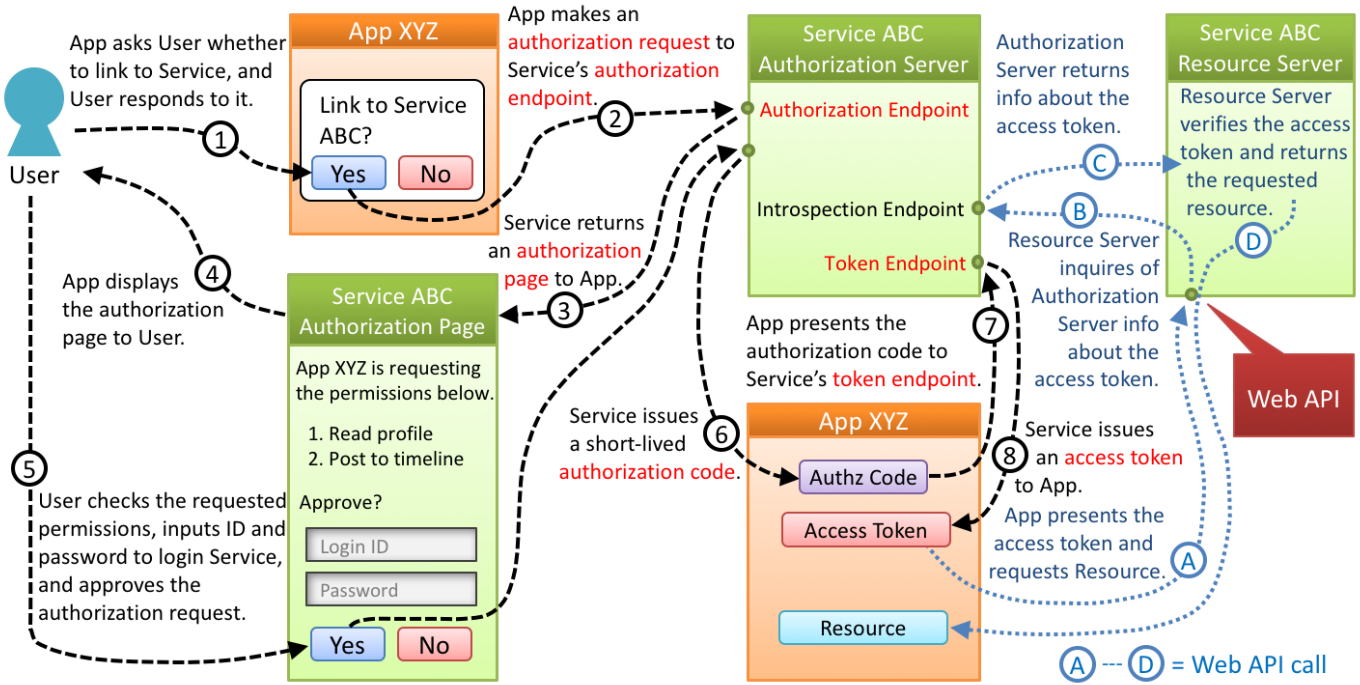



Client Credential
Hybrid
Authorization Code + PKCE
Authorization Code
N
Y
Has Users
N
Start
Public Client
Device Flow
Y
Mobile/Native
N
SPA
N
Y
Y
Authorization Code + PKCE
Resource Owner
Authorization Code + PKCE
Implicit Flow

Authorization Code Flow

- Confidential Client
- Front Channel: Authorization Code
- Back Channel: Identity Token, Access Token
- Refresh Token can be provided
- Other name: 3 Legged OAuth

Authorization Code Flow (RFC 6749, 4.1)



Authorization Code Flow

Problem:

- Code substitution attack

OpenID Connect mitigates this threat with Hybrid Flow

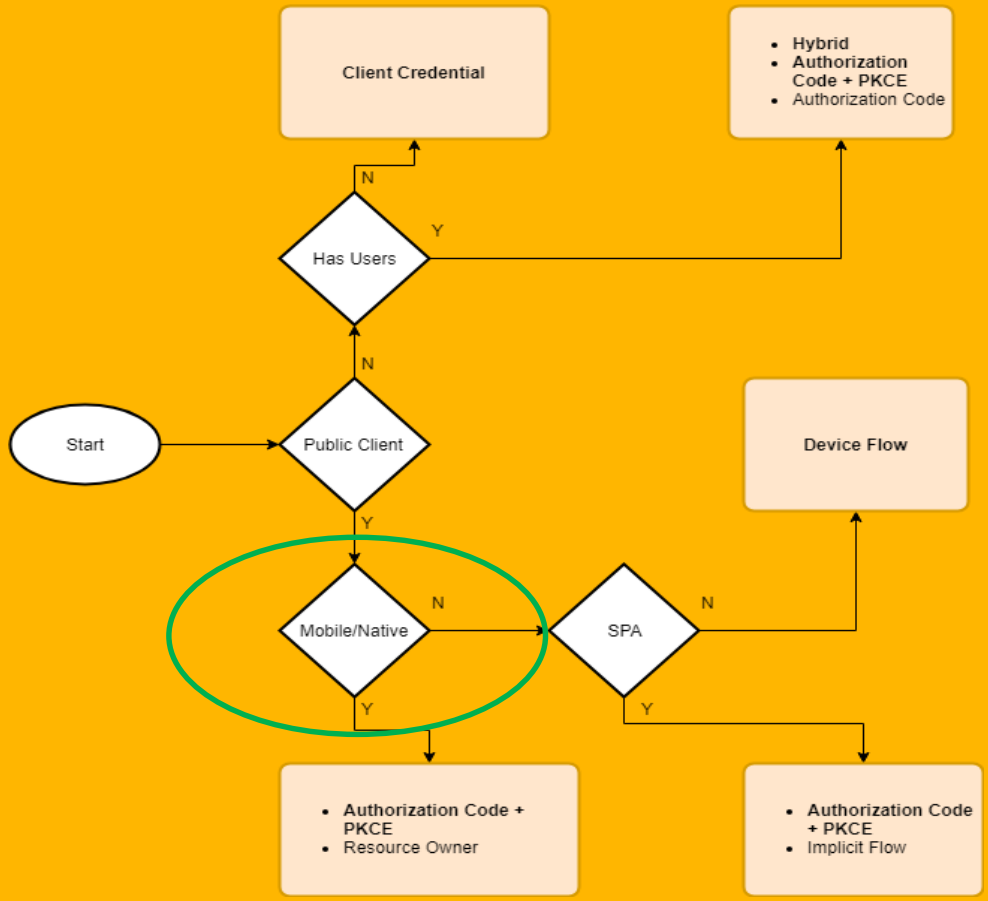
OAuth mitigates this threat with PKCE

Hybrid Flow

During step 6 in diagram, authorization code is accompanied by identity token and there's a hash (chash) that making sure the authorization code and identity token belong together

Problems:

- Heavy
- Client library more complex
- Check the provider for Hybrid Flow support
- Identity token may leak personal information

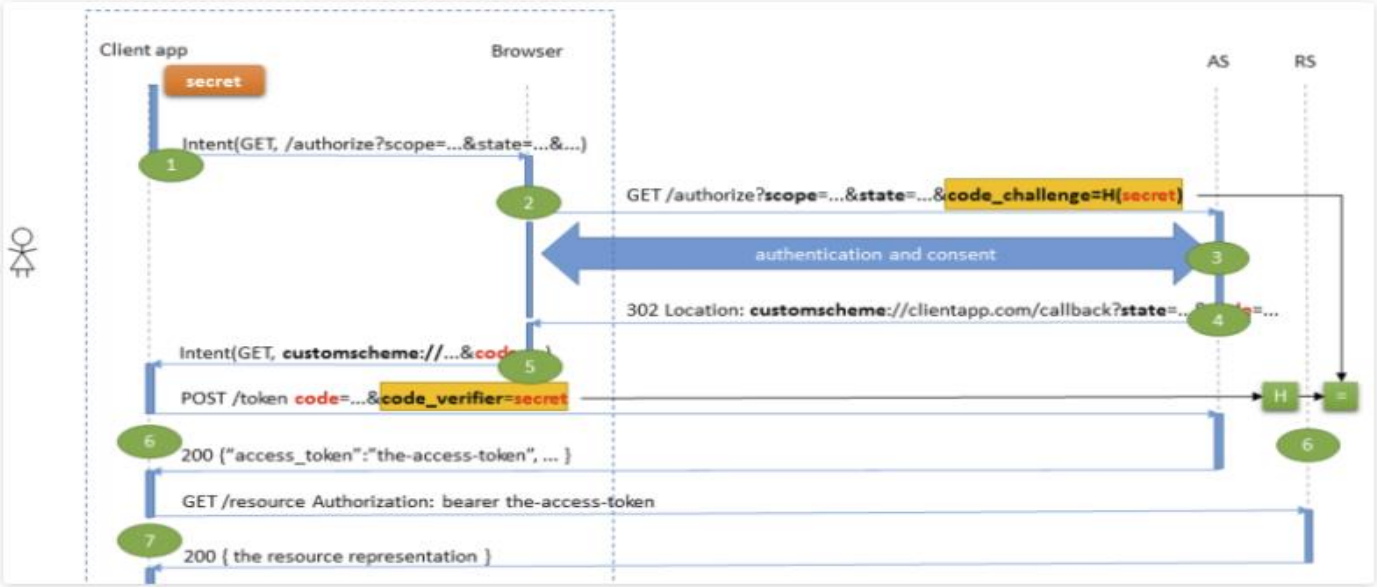

Client Credential
Hybrid
Authorization Code + PKCE
Authorization Code
N
Y
Has Users
N
Start
Public Client
Device Flow
Y
Mobile/Native
N
SPA
N
Y
Y
Authorization Code + PKCE
Resource Owner
Authorization Code + PKCE
Implicit Flow

Code Flow + PKCE

1. Client create code verifier, a strong cryptographic random string, save it on the device
2. Transform this code verifier to get code challenge
    - codechallenge = Base64(Sha256(codeverifier)))
3. Send code challenge and the transformation method during the code flow authorization request
4. STS saves this code challenge
5. When client request access token it sends clientid, code and codeverifier
6. STS transform codeverifier into codechallenge, and compared them to the original code challenge, if it's not the same, deny the request.

Detail: RFC7836 (https://tools.ietf.org/html/rfc7636)

Native/Mobile Client



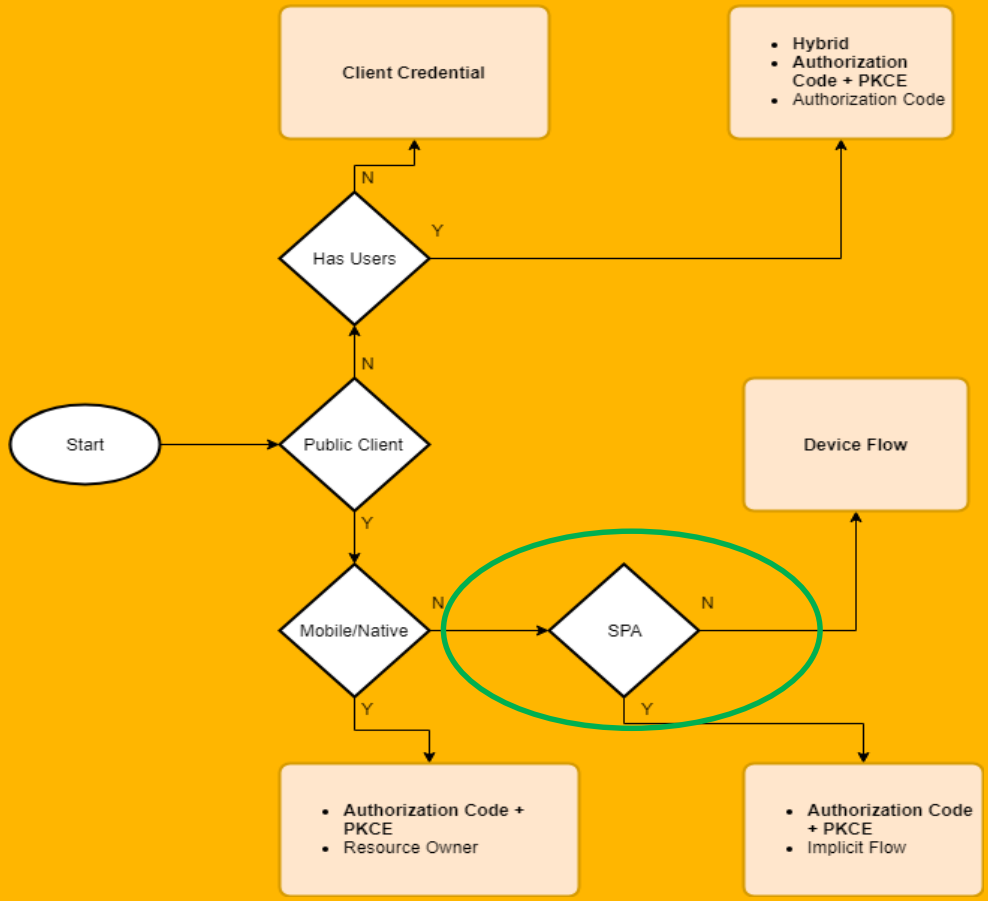
Native/Mobile Client

- Can't store client secret safely, don't use it
- ClientID is identical for the same app, use dynamic client registration
- OAuth 2.0 for Native Apps (https://tools.ietf.org/html/rfc8252)
  - Do not build your own login form
  - A lot of provider deprecate this flow
- Use system browser !
- Register custom URI handler for your app to received authorization redirect
- Store access and refresh token in secure storage

Native/Mobile Client

Client Library

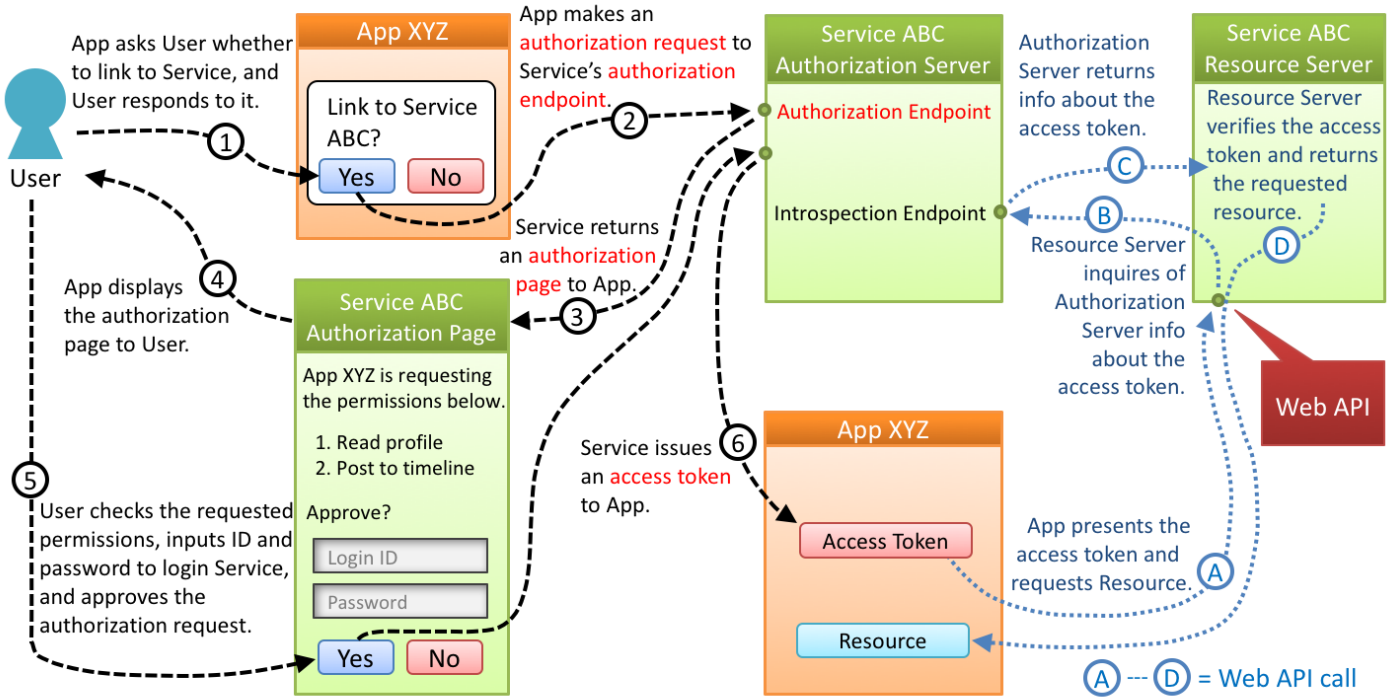
- AppAuth library ! (https://appauth.io/), available in ios, android and JS
- C# .Net IdentityModel.OidcClient2 (https://github.com/IdentityModel/IdentityModel.OidcClient2)


Client Credential
Hybrid
Authorization Code + PKCE
Authorization Code
N
Y
Has Users
N
Start
Public Client
Device Flow
Y
Mobile/Native
N
SPA
N
Y
Y
Authorization Code + PKCE
Resource Owner
Authorization Code + PKCE
Implicit Flow

SPA

- React, Angular, etc.
- Most common client but…
- The most difficult to secure
- Implicit Flow is widely adopted

Implicit Flow (RFC 6749, 4.2)



SPA

Problem with SPA and OAuth

- Access token is sent on the front channel
- Must use hash fragment in the URL
- XSS, XRF can cause access token exfiltration
- How to prevent token substitution attack ?
    - OpenID Implicit Flow added (at hash) claim

RFC 6819

- January 2019
- OAuth 2.0 Threat Model and Security Considerations (https://tools.ietf.org/html/rfc6819)
- Is Implicit Flow anti pattern ?
- What's the solution ?

Authorization Code + PKCE

- Unified Flow for 3 type of clients !
- Use library that support PKCE
- Most good client library is just a matter of configuration change

Securing SPA

- Use a certified OpenID library
- Don't use unsafe DOM manipulation:
  - Angular DomSanitizer.bypass*()
- Watch out on using non-reputable package from NPM !
- Utilize CSP !
- Use silent token renewal instead rather than refresh token

Alternate:

- Use same site cookie for same domain application
- BFF

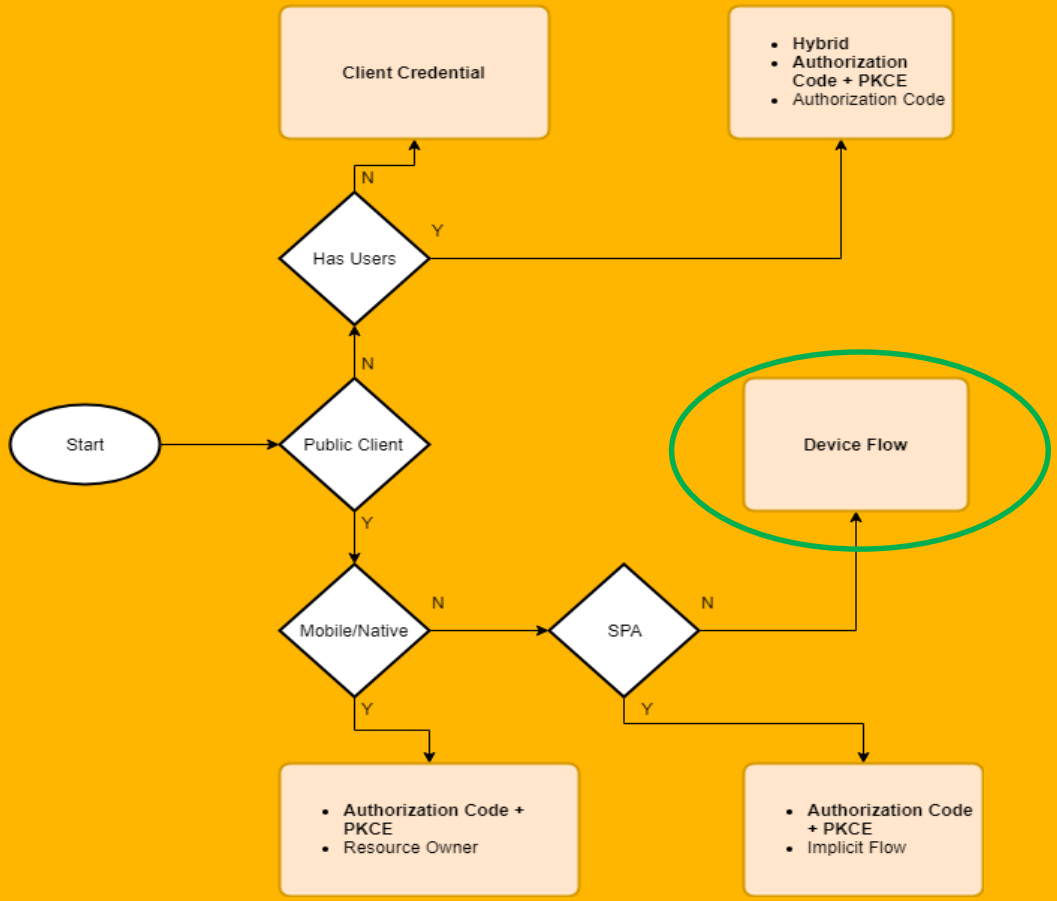
Securing SPA

Method	Access credentials can be securely stored	Access credentials secure during auth	Can be used across domains	Secure against CSRF	Speed
"Just use a damn cookie"	✗	✓	✓	✗	fast
OAuth Implicit Flow	✗	✗	✓	✓	fast
OAuth Auth Code + PKCE	✗	✓	✓	✓	Auth: average API: fast
Same-Domain Application	✓	✓	✗	✓	fast
OAuth + Backend for Front End	✓	✓	✓	✓	Auth: average API: slow

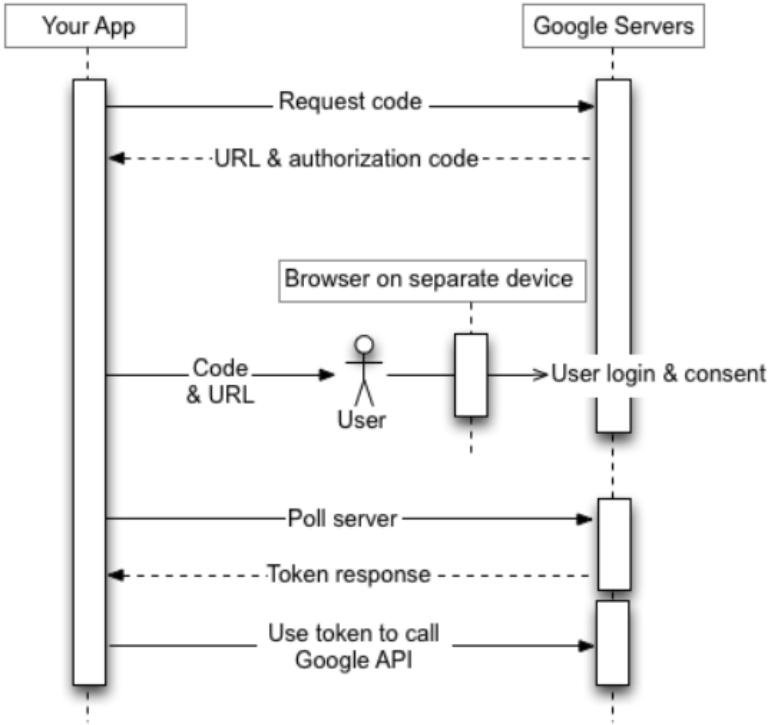
https://www.scottbrady91.com/OAuth/Cheat-Sheet-OAuth-for-Browser-Based-Applications

SPA OIDC Library

- oidc-client (https://www.npmjs.com/package/oidc-client) aka. oidc-client-js
- AuthJS (https://www.npmjs.com/package/@openid/appauth)
- angular-oauth2-oidc (https://github.com/manfredsteyer/angular-oauth2-oidc)
- Vendor specific (Octa, auth0, etc.)


Client Credential
Hybrid
Authorization Code + PKCE
Authorization Code
N
Y
Has Users
N
Start
Public Client
Device Flow
Y
Mobile/Native
N
SPA
N
Y
Y
Authorization Code + PKCE
Resource Owner
Authorization Code + PKCE
Implicit Flow

Device Flow


Your App
Google Servers
Request code
URL & authorization code
Browser on separate device
Code
& URL
User
User login & consent
Poll server
Token response
Use token to call
Google API

IDaaS

- Okta (https://www.okta.com/)
- Auth0 (https://auth0.com/)
- OneLogin (https://www.itcentralstation.com/products/onelogin-reviews)
- SailPoint(https://www.sailpoint.com/?elqct=Website&elqchannel=OrganicDirect)
- AWS (Cognito)
- Azure (Azure AD B2C)
- GCP (Identity Platform)_

OSS

- IdentityServer4 (.NET, http://docs.identityserver.io/en/latest/)
- Gluu (Java, https://www.gluu.org/)

Both are OpenID Foundation Certified

Libraries

- https://openid.net/developers/libraries/

IETF

- JWT (https://tools.ietf.org/html/rfc7519)
- OAuth 2.0 for Browser-Based Apps (https://tools.ietf.org/html/draft-parecki-oauth-browser-based-apps-02)
- OAuth 2.0 Security Best Current Practice (https://tools.ietf.org/html/draft-ietf-oauth-security-topics-12)
- OAuth 2.0 for Native Apps (https://tools.ietf.org/html/rfc8252)