Authentication & API Access for native/mobile Applications

Dominick Baier
dominick.baier@leastprivilege.com
http://leastprivilege.com
@leastprivilege

Dominick Baier

- **Independent Consultant**
  - Specializing on Identity & Access Control
  - Working with Software Development Teams (ISVs and in-house)

- **Creator and Maintainer of IdentityServer/IdentityModel OSS Projects**
  - OpenID Connect & OAuth 2.0 Implementation for ASP.NET / Core
  - .NET Foundation Advisory Board
  - http://identityserver.io

dominick.baier@leastprivilege.com
http://leastprivilege.com

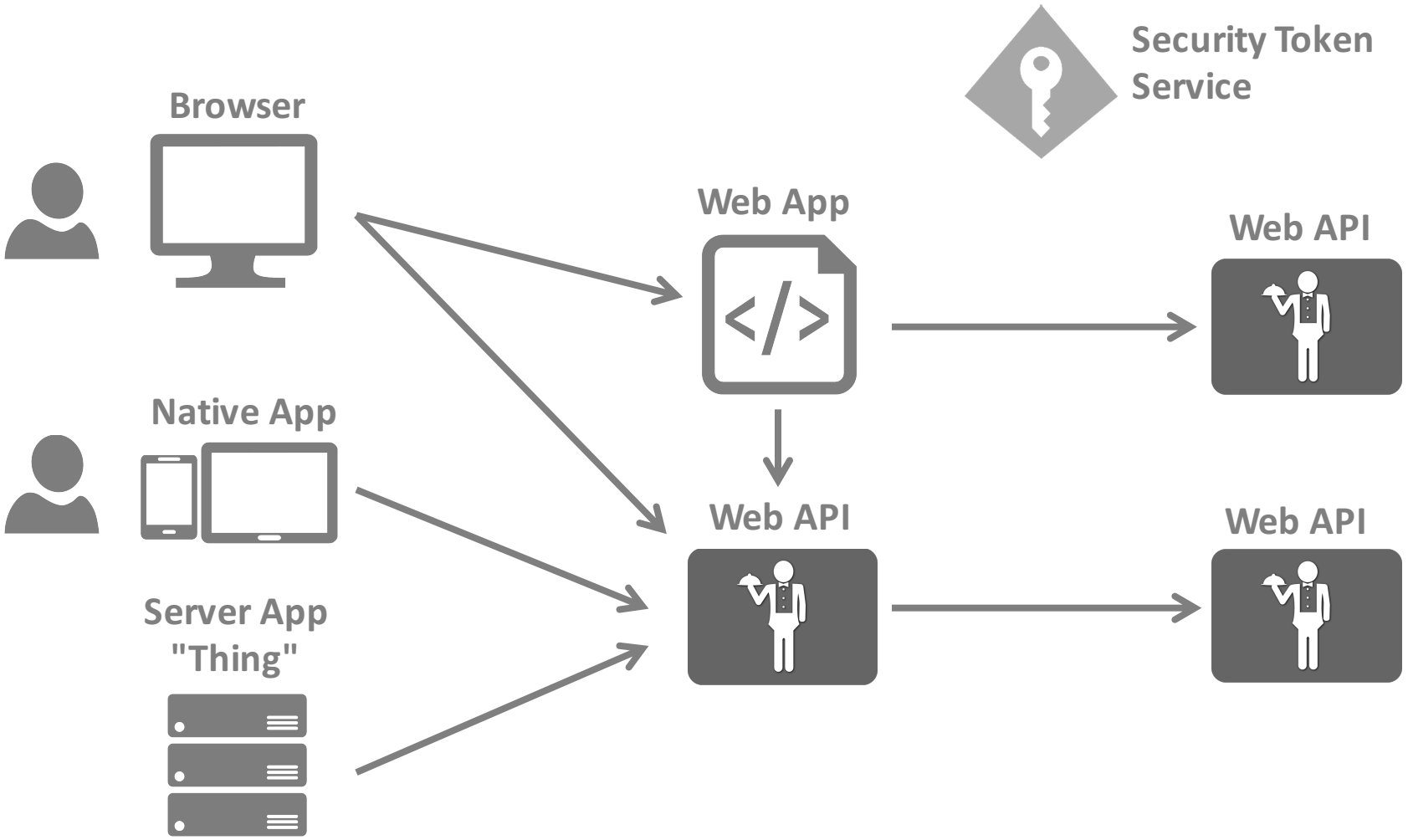
slides: https://speakerdeck.com/leastprivilege

tl;dr

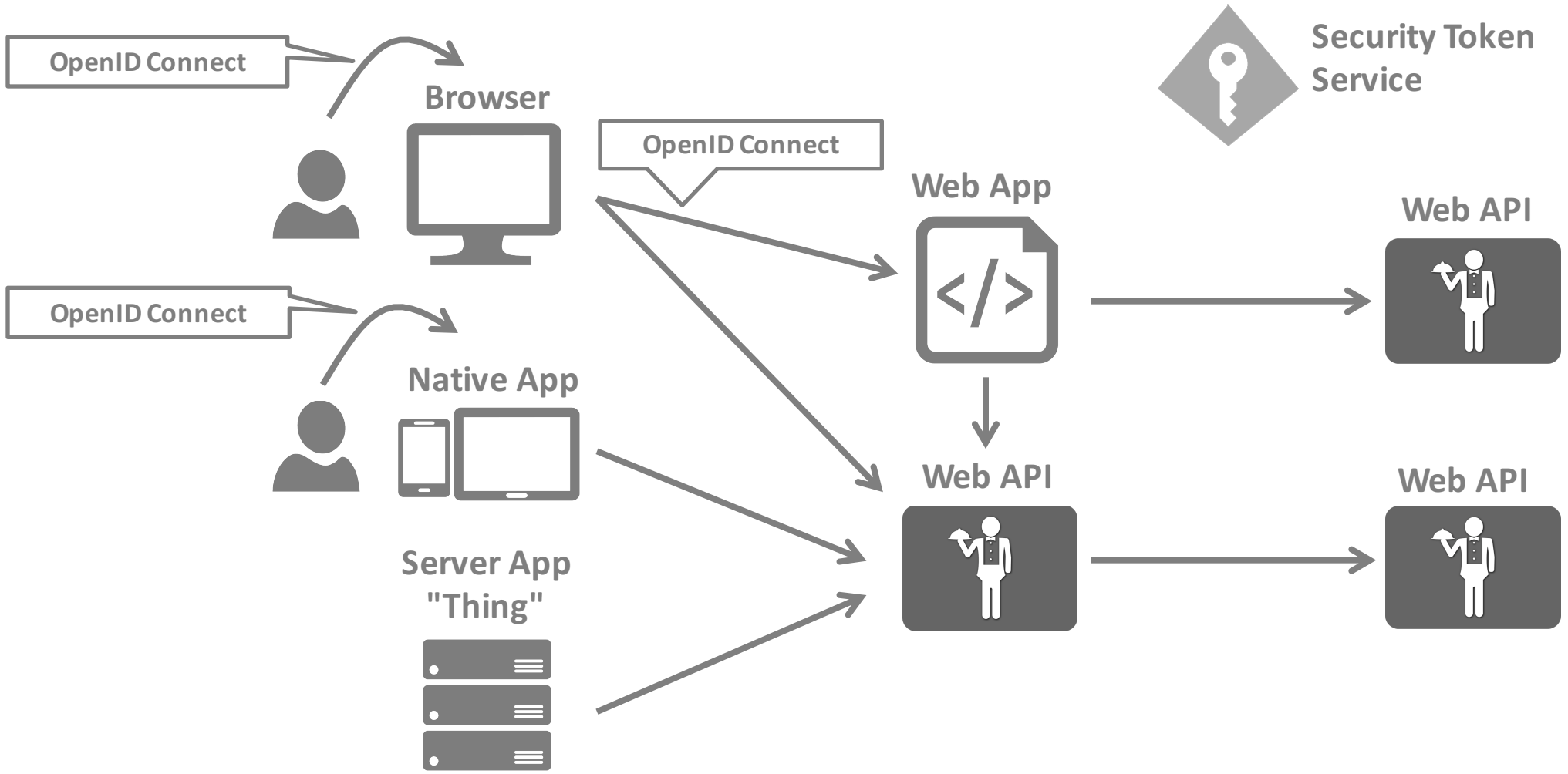
- **Implementing authentication and API access for native/mobile applications**
  - IOW applications that have access to native platform APIs
  - desktop or mobile

- **Following the guidance of "OAuth 2.0 for native Applications"**
  - https://tools.ietf.org/html/draft-wdenniss-oauth-native-apps-02

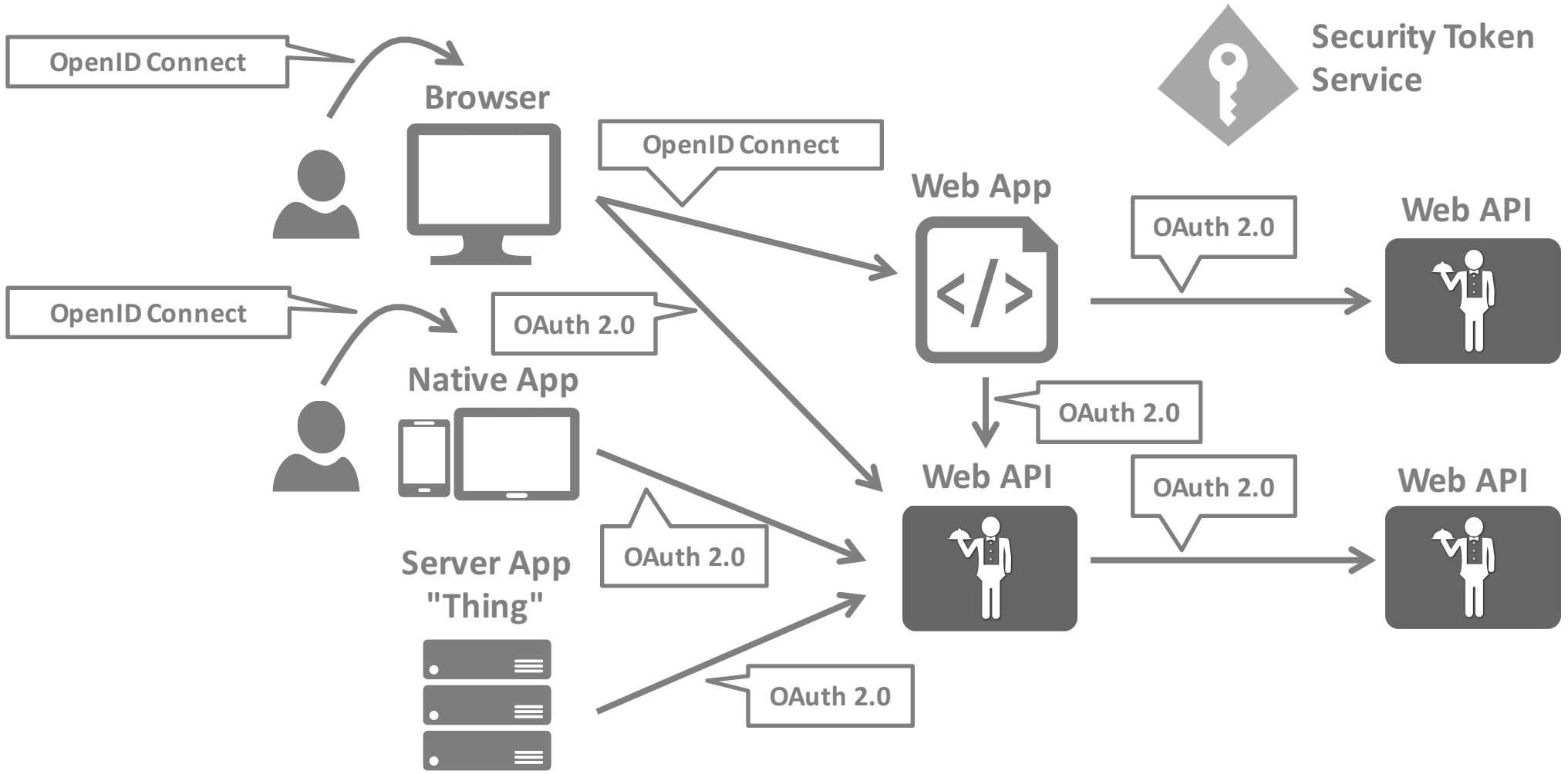
The big picture



Security protocols (I)



Security protocols (II)

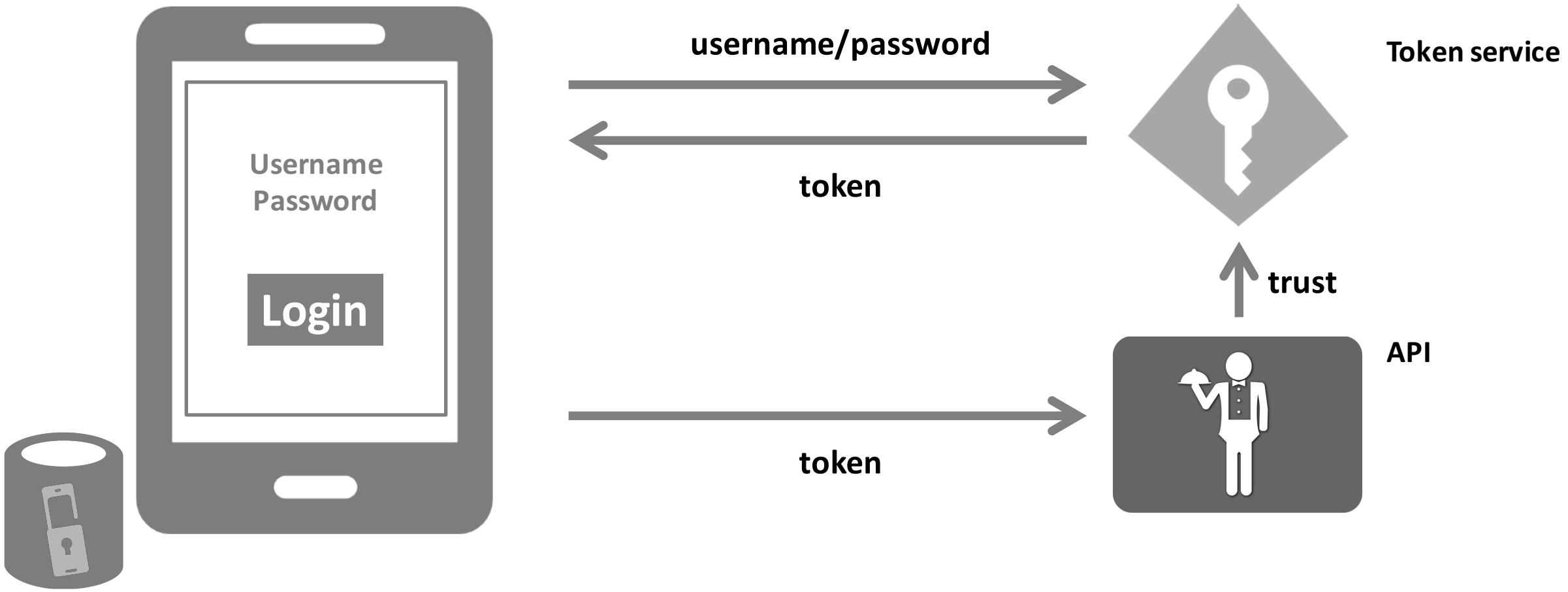


So many options…

- **Low hanging fruit**
  - OAuth 2.0 resource owner password credential flow
- **Better, but is missing out on some advanced features**
  - OAuth 2.0 implicit flow

- **Recommended**
  - OAuth 2.0 authorization code flow (with PKCE)
- **…and my favourite**
  - OpenID Connect Hybrid Flow (with PKCE)

Native login dialogs

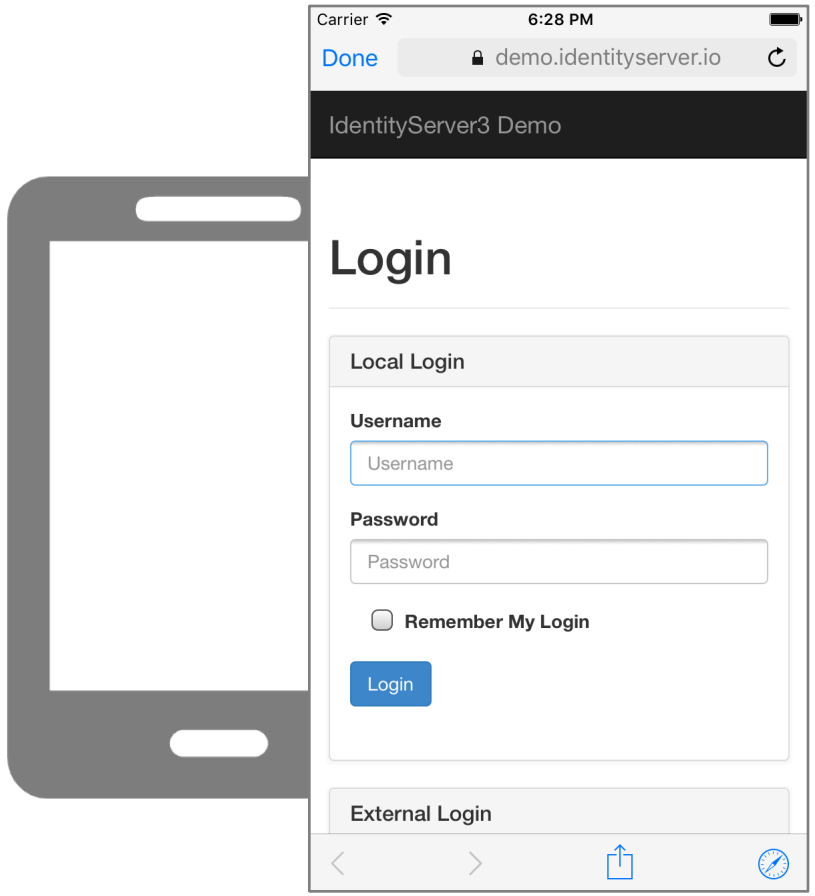


OAuth 2.0 Resource Owner Password Flow

- **Pros**
  - client app has full control over login UI
  - support for long lived API access without having to store a password

- **Cons**
  - user is encouraged to type in his master secret into "external" applications
    - especially problematic once applications also come from 3rd parties
  - no cross application single sign-on or shared logon sessions
  - no federation with external identity providers/business partners
  - every change in logon workflow requires versioning the application

Using a browser for driving the authentication workflow



authentication request

render UI & workflow

Using a browser for driving the authentication workflow

- **Centralize authentication logic**
  - consistent look and feel
  - implement once, all applications get it for free
  - allows changing the workflow without having to update the applications
    - e.g. consent, updated EULA, 2FA

- **Enable external identity providers and federation**
  - federation protocols are browser based only

- **Depending on browser, authentication sessions can be shared between apps and OS**

Browser types

- **Embedded web view**
  - private browser & private cookie container
  - e.g. WinForms or WPF browser control
- **Authentication broker**
  - "special" browsers (look private but share some cookies)
  - e.g. Win8 & UWP *WebAuthenticationBroker*
- **In-app browser tab**
  - full blown system browser (including address bar & add-ins)
  - shared cookie container
  - e.g. SafariViewController (iOS9) & Chrome Custom Tabs (Android 5)

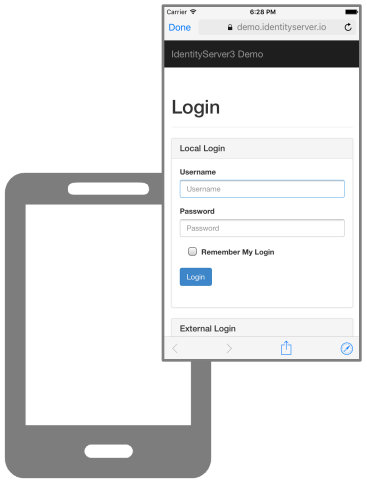
How to implement?

- **Implicit flow**
  - really designed for JS apps
  - access tokens transmitted over browser (and potentially cross process)
  - no refresh tokens

- **Authorization code-based flows**
  - access tokens only over back-channel communication
  - slightly more secure due to client secret
  - allows long lived API access via refresh tokens
  - authorization code itself needs to be protected though
    - cut'n paste attack
    - man in the middle

Starting the authentication request

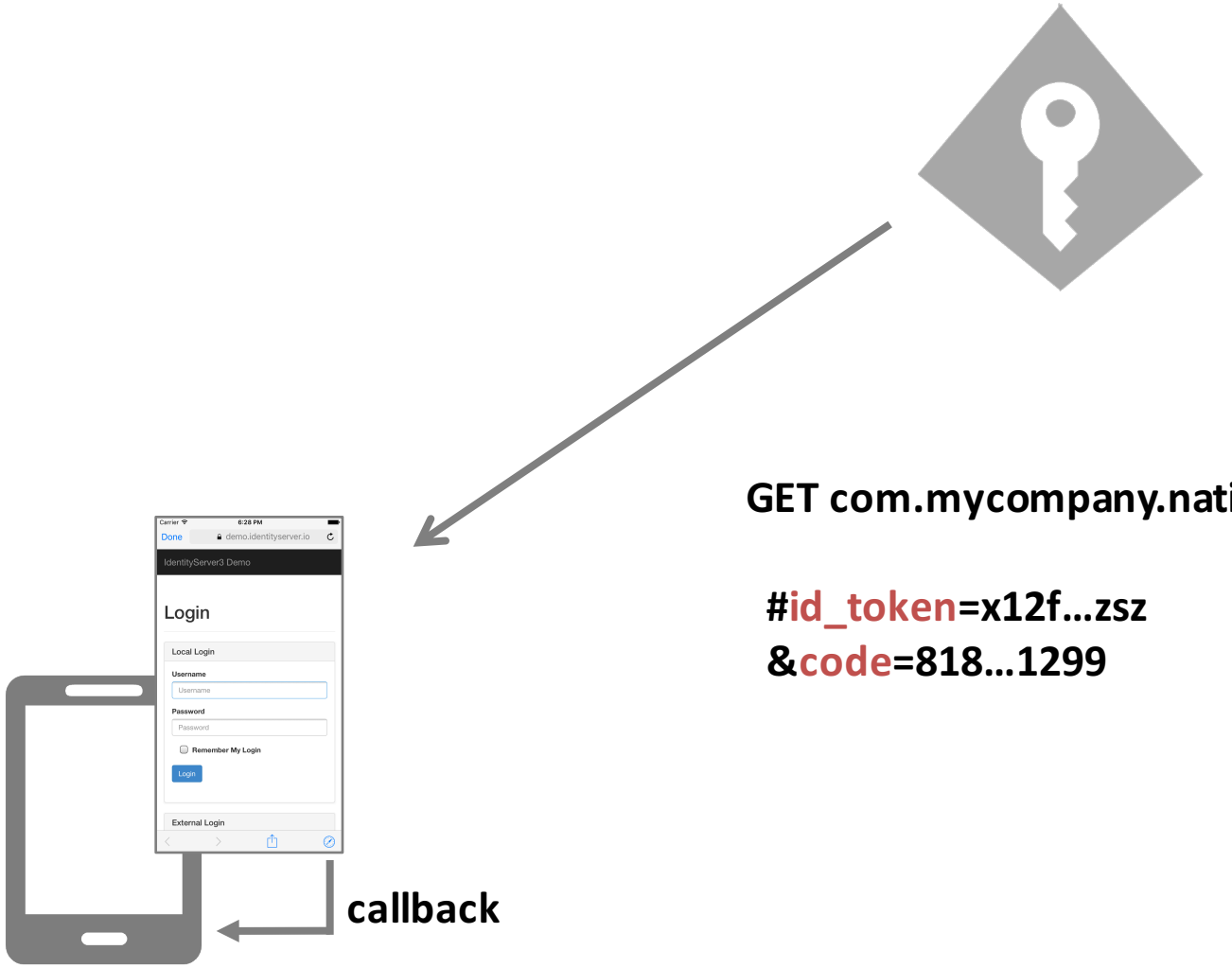
nonce = random_number
code_verifier = random_number
code_challenge = hash(code_verifier)



GET /authorize

?client_id=nativeapp
&scope=openid profile api1 api2 ***offline_access***
&redirect_uri=com.mycompany.nativeapp://cb
&response_type=*code id_token*
&nonce=j1y…a23
&code_challenge=x929..1921

Receiving the response



GET com.mycompany.nativeapp://cb

#id_token=x12f…zsz
&code=818…1299

Identity token

Header

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "mj399j…"
}
```

Payload

```
{
  "iss": "https://idsrv",
  "exp": 1340819380,
  "aud": "nativeapp",
  "nonce": "j1y…a23",
  "amr": [ "password", "sms" ],
  "auth_time": 12340819300

  "sub": "182jmm199"
}
```

base64url →

```
eyJhbGciOiJub25lIn0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt
```

Header **Payload** **Signature**

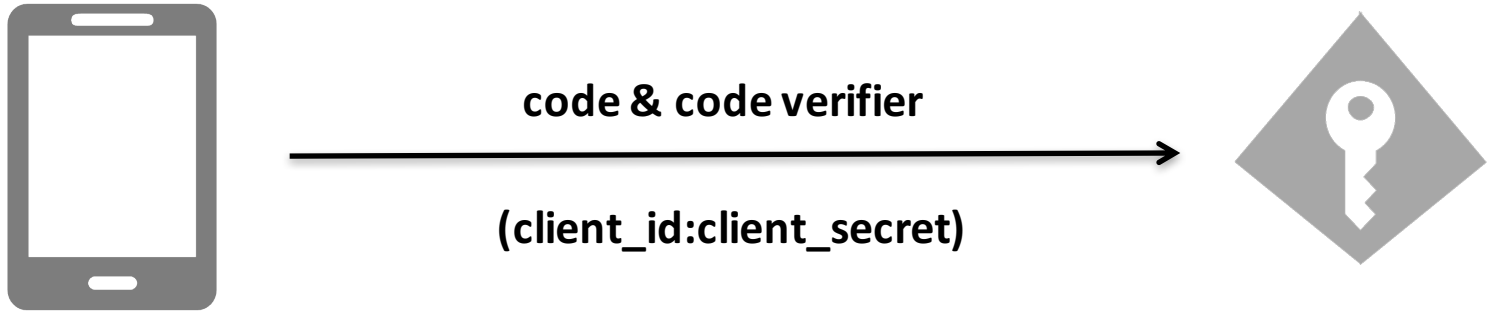
Validating the response

- **Identity token validation (section 3.1.3.7)**
  - validate signature
    - key material available via discovery endpoint
  - validate **`iss`** claim
  - validate **`exp`** (and **`nbf`**)
  - validate **`aud`** claim
- **Authorization code validation (section 3.3.2.10)**
  - hash authorization code and compare with **`c_hash`** claim

https://openid.net/specs/openid-connect-core-1_0.html

Requesting the access token

- **Exchange code for access token**
  - using client id and secret




```
{
  access_token: "xyz…123",
  refresh_token: "dxy…103"
  expires_in: 3600,
  token_type: "Bearer"
}
```

Optional: download more claims

- **OpenID Connect *UserInfo* endpoint provides claims as JSON object**


```
{
  "given_name": "Kendall",
  "preferred_username": "FluffyBunnySlippers"
  "profile_picture": "  "
}
```

Next steps

- **Persist the data in protected storage**
  - claims
  - access token
  - refresh token

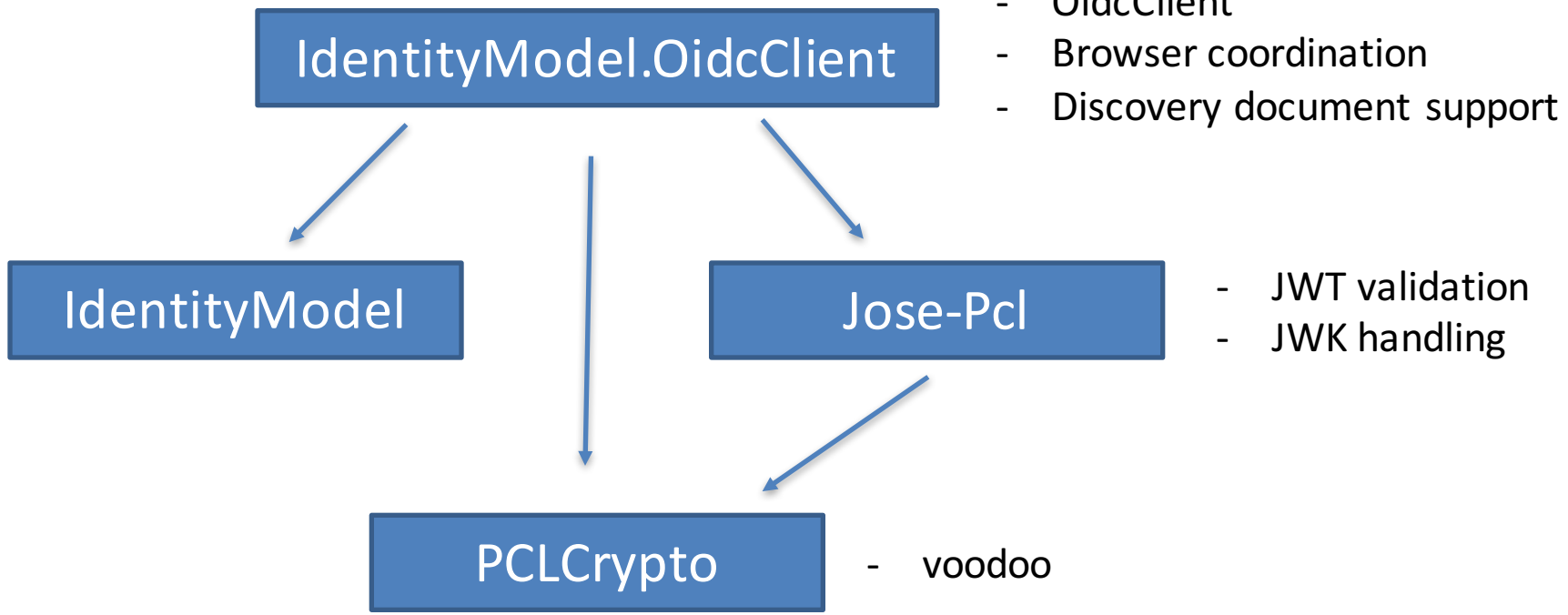
- **Use access token to communicate with APIs**
- **Use refresh token to get new access tokens when necessary**

That's a lot of work!

- **Native libraries**
  - https://github.com/openid/AppAuth-iOS
  - https://github.com/openid/AppAuth-Android

- **C# portable class library (desktop .NET, UWP, mobile, iOS, Android)**
  - https://github.com/IdentityModel/IdentityModel.OidcClient
  - https://github.com/IdentityModel/IdentityModel.OidcClient.Samples

OSS FTW!


IdentityModel.OidcClient
- OidcClient
- Browser coordination
- Discovery document support
- OpenID Connect & OAuth 2.0 protocol
- Refresh token handling
IdentityModel
Jose-Pcl
- JWT validation
- JWK handling
PCLCrypto
- voodoo

Setup

```
var options = new OidcClientOptions(
    authority:    authority,
    clientId:     "native",
    clientSecret: "secret",
    scope:        "openid profile api offline_access",
    redirectUri:  "com.mycompany.myapp://callback",
    webView:      webView);

var client = new OidcClient(options);
```

Authentication & requesting tokens

```
var result = await client.LoginAsync();

var claims = result.Claims;
var accessToken = result.AccessToken;
var refreshToken = result.RefreshToken;
```

Calling APIs and keeping tokens fresh

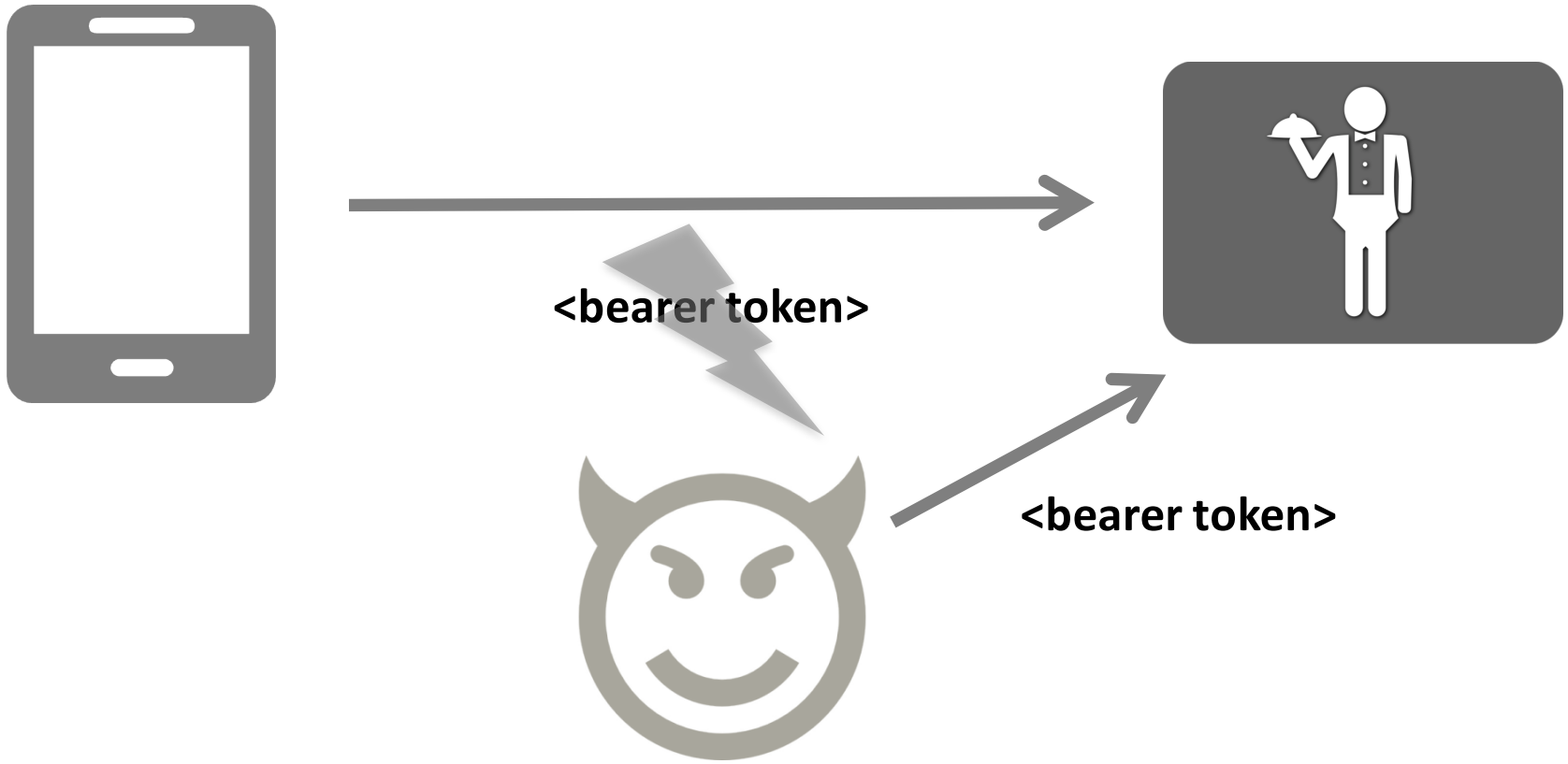
```
var apiClient = new HttpClient(result.Handler);
apiClient.BaseAddress = new Uri("https://www.mycompany.com/api/");
```

or…

```
var tokenClient = new TokenClient(
    address:      "https://demo.identityserver.io/connect/token",
    clientId:     "client",
    clientSecret: "secret");

var handler = new RefeshTokenHandler(tokenClient, refreshToken);
```

Stepping up security: bearer vs pop tokens



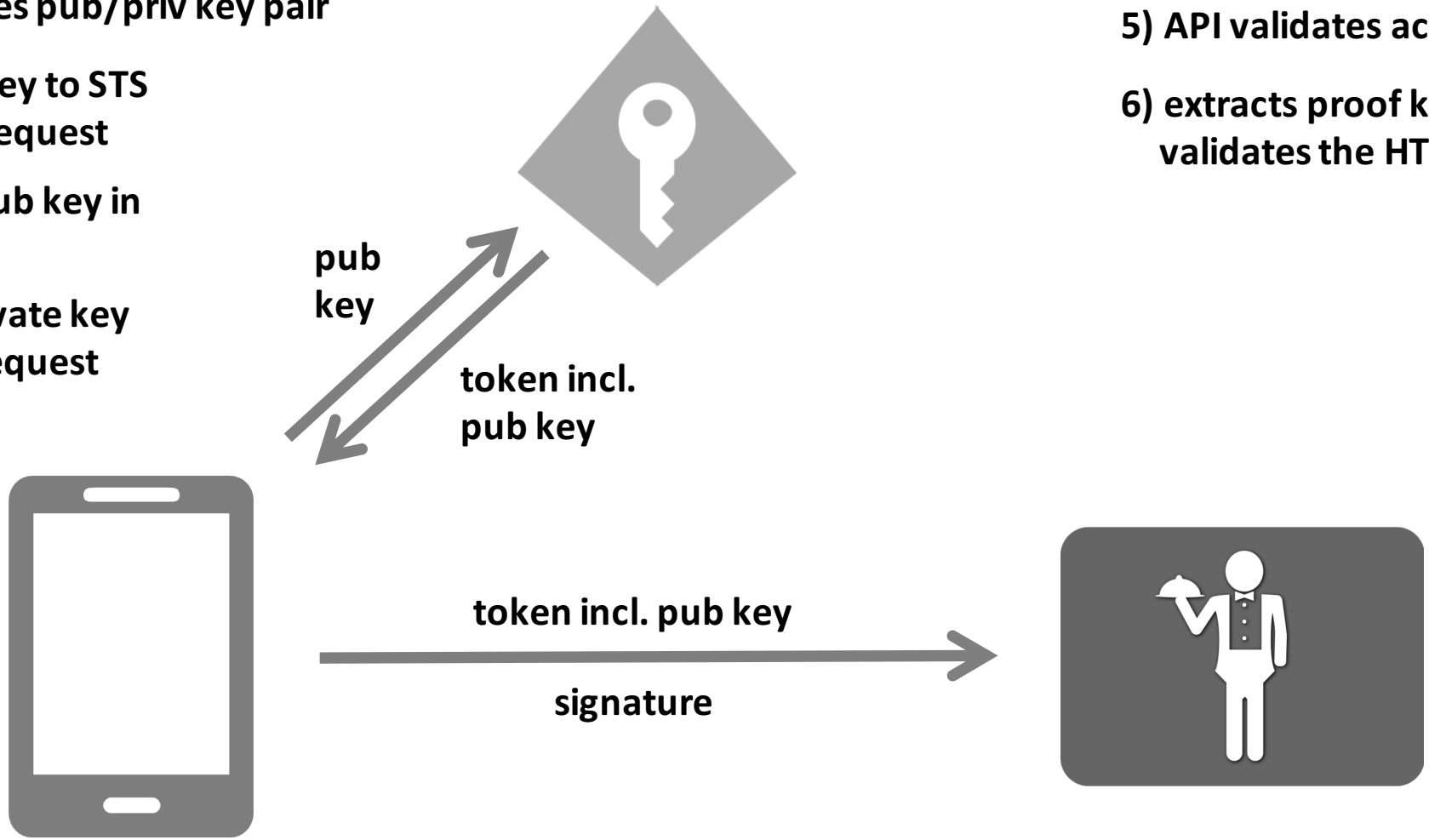
Adding a proof key & signature

1) client generates pub/priv key pair

2) sends public key to STS during token request

3) STS embeds pub key in access token

4) client uses private key to sign HTTP request



5) API validates access token

6) extracts proof key & validates the HTTP signature

That's even more work!!!

- **Helper libraries**
  - https://github.com/IdentityModel/IdentityModel.Owin.PopAuthentication
  - https://github.com/IdentityModel/IdentityModel.HttpSigning

- **The specs (not done yet)**
  - https://tools.ietf.org/wg/oauth/draft-ietf-oauth-pop-architecture/
  - https://tools.ietf.org/wg/oauth/draft-ietf-oauth-token-exchange/
  - https://tools.ietf.org/wg/oauth/draft-ietf-oauth-signed-http-request/

Summary

- **NAPPS spec recommends code based authentication flow**
  - with PKCE
  - id_token adds signed response and signature over authorization code
- **Use system level browser for best user experience**
  - shared logon sessions
  - password managers
- **Use refresh tokens for long lived API access**
  - store them securely
- **Look into PoP for further protection against man-in-the-middle**

thank you!

get slides from https://speakerdeck.com/leastprivilege