

# Introduction to OAuth2 in JavaScript

Basics

Authorizing Access To API

OpenID Connect

OpenID Connect On Clients

Impersonating The User

Credentials

# Basics

## Basics

Authorizing Access To API

OpenID Connect

OpenID Connect On Clients

Impersonating The User

Credentials

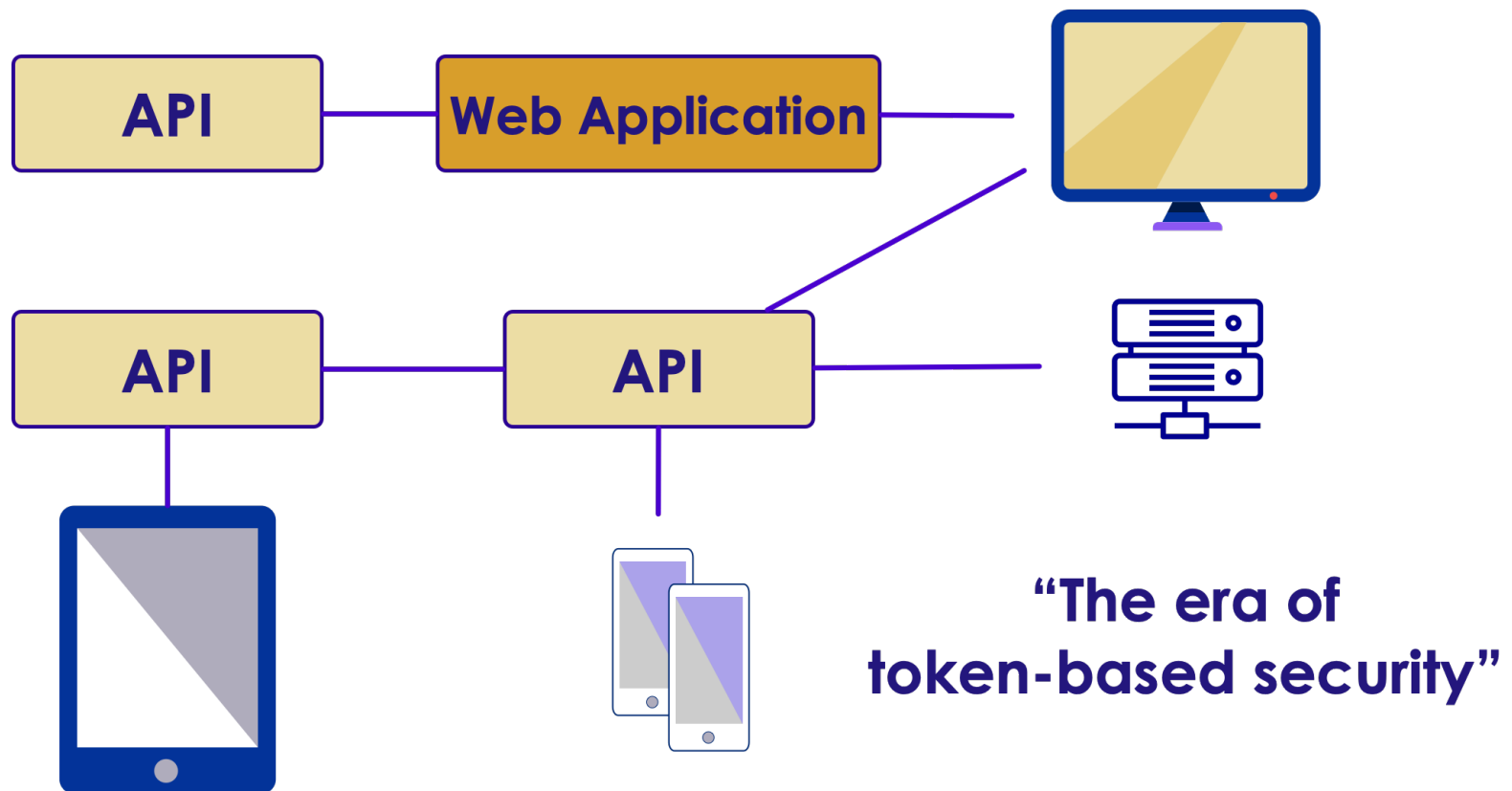
# What Is It



**OAuth 2.0 is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop application**

# Application

## A Modern Application



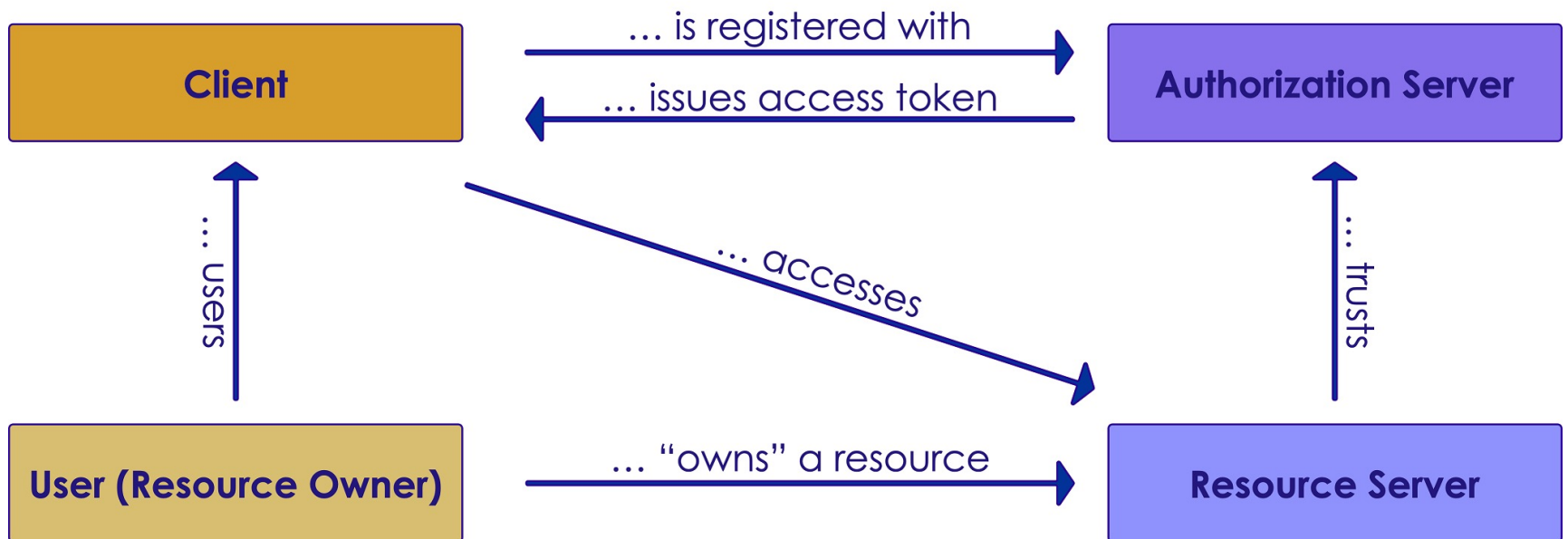
# Typical Payload of OAuth2 Access Token

```
1 {  
2   "client_id": "tripgalleryimplicit",  
3   "scope": "gallerymanagement",  
4   "sub": "b05d3546-6ca8-4d32-b95c-77e94d705ddf",  
5   "amr": "password",  
6   "auth_time": "1437400047",  
7   "idp": "idsrv",  
8   "iss": "https://elephantscale/identity",  
9   "aud": "https://elephantscale/identity/resources",  
10  "exp": 1437403647,  
11  "nbf": 1437403647  
12  
13 }
```

# Defining OAuth 2.0

- ◆ Is not an authentication method
- ◆ Is an authorization method
- ◆ Standard is silent about the user
- ◆ RFC6749

## The Main Actors



# Simple Questions

- ◆ Different types of applications require different means to achieve authorization
- ◆ Where can the token be delivered to?
- ◆ Can the client application safely store secrets?



# Making Decision

- ◆ How can you safely achieve authorization?
- ◆ By answering to this question choose your type of application you are building:
  - Client Credentials
  - Implicit
  - Authorization Code
  - Resource Owner Password Credentials
  - Device Code

# Clients

- ◆ Confidential clients:
  - Clients that can maintain the confidentiality of their credentials
  - Example: web applications
- ◆ Public clients:
  - Clients that cannot maintain the confidentiality of their credentials

# Client Examples

- ◆ JavaScript clients
  - Native apps
    - iOS
    - Android
    - Windows Phone apps built in a native or compile to native language
  - User-Agent based apps
  - JavaScript apps

# Endpoints: On Authorization Server

- ◆ Authorization endpoint
  - Used by the client to get authorization the owner of resource through user-agent redirection
- ◆ Token endpoint
  - Client uses this token to exchange an authorization grant for an access token usually with client authentication

# Endpoints: On Client

- ◆ Redirection endpoint
  - Authorization server uses it to return responses containing authorization credentials to the client through resource owner user=agent

# What About Authorization server?

- ◆ We don't have to implement Authorization server
- ◆ Identity server
  - For example: @leastprivilege and @brockallen Implement OAuth 2.0 and OpenID Connect

# Authorizing Access To API

Basics

**Authorizing Access To API**

OpenID Connect

OpenID Connect On Clients

Impersonating The User

Credentials

# Client Credentials Flow

- ◆ Machine to machine communication
  - No human or username and password involved
  - Can be used to get access token using client credentials
- ◆ Is used only by confidential clients
  - A public client doesn't safely store the client secret



## Client Credentials Flow



# How About Angular?

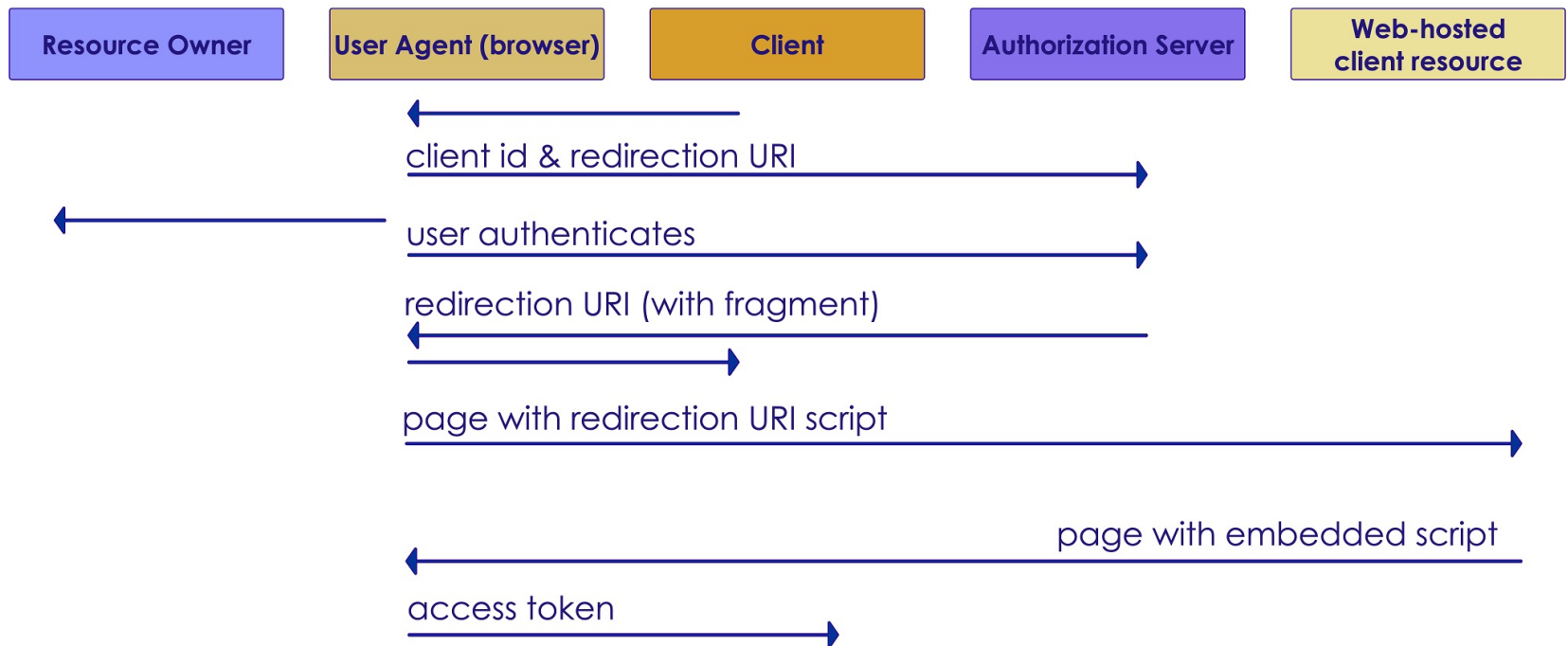
- ◆ Question: Can we use client credential for our Angular applications?
- ◆ Answer: Yes but \* It is not safe \*
- ◆ It's like to lock the door but leave the key on it

# Implicit Flow

- ◆ For public clients at predefined redirection URI and also might be used by confidential clients (JavaScript or Angular apps)
- ◆ To obtain access tokens not refresh tokens
- ◆ No client authentication because a public user cannot store the secret safely

# Implicit Flow In A Picture

## Implicit Flow

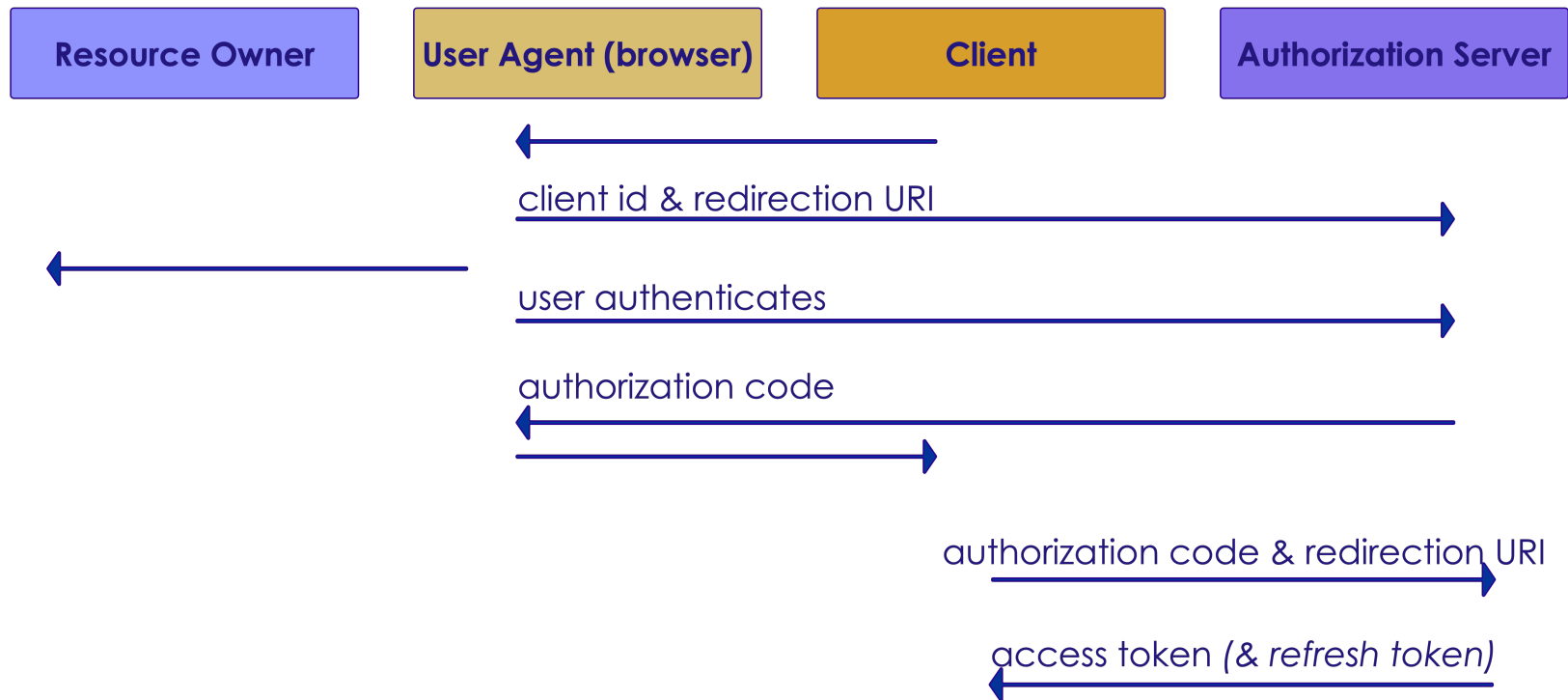


# Authorization Code Flow

- ◆ Optimized for confidential and public clients
- ◆ To get access and refresh tokens
- ◆ Includes a client authentication step

# Authorization Code Flow In A Picture

## Authorization Code Flow

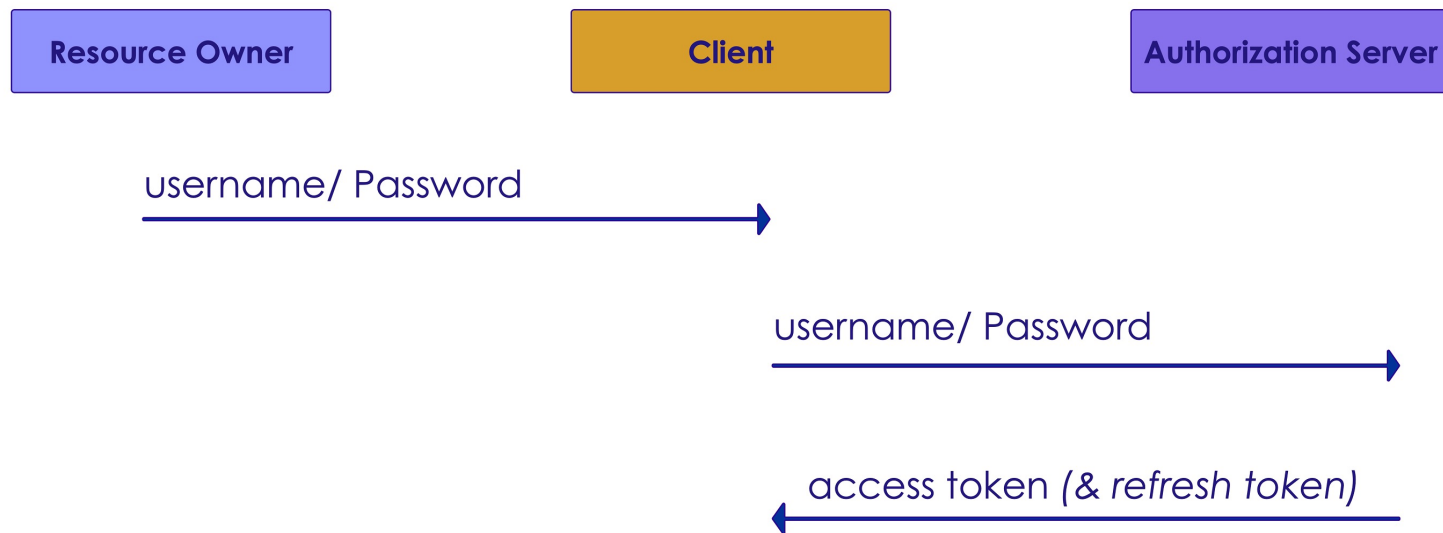


# Resource Owner Password Credentials Flow

- ◆ Client has to be able to obtain the resource owner's credentials (for example via in-app login screen)
- ◆ Just for trusted applications
- ◆ Is used to obtain access and refresh tokens
- ◆ Includes a client authentication step
- ◆ High risk in compared to other flows so it is the last choice

# Flow In A Picture

## Resource Owner Password Credentials Flow





# Device Code Flow

- ◆ Between devices that have Internet connection but not browser
- ◆ Flow between smart TVs, media consols, etc.

# Device Code Flow In A Picture

①



②



# Angular And Cross-Origin Resource Sharing

- ◆ Browsers prevent the web page from making AJAX requests to another domain
- ◆ Cross-Origin Resource Sharing (CORS) is W3C standard
- ◆ Allows server to relax the same origin policy

## Example:

- ◆ Origin: `http://elephantscale.com`
- ◆ Different domain: `http://elephantscale.org`
- ◆ Different port: `http://elephantscale.com:2546`
- ◆ Different scheme: `https://elephantscale.com`
- ◆ Different subdomain: `http://www.elephantscale.com`

# OpenID Connect

Basics

Authorizing Access To API

**OpenID Connect**

OpenID Connect On Clients

Impersonating The User

Credentials

# Problem?

- ◆ OAuth 2 has to do with authorization
- ◆ How can we handle authentication?
- ◆ How can we get information regarding identity
- ◆ Solution: OpenID Connect

# OpenID Connect

- ◆ A simple identity layer on the OAuth 2 protocol
- ◆ Core functionality:
  - Authentication
  - Claims about the user

# How It Works?

- ◆ New type of token : ID token
- ◆ Example: A typical ID token:

```
1 {  
2   "sub": "b05d3546-6ca8-4d32-b95c-77e94d705ddf",  
3   "amr": "password",  
4   "auth_time": "1437400047",  
5   "idp": "idsrv",  
6   "given_name": "Jack",  
7   "family_name": "Brown",  
8   "iss": "https://elephantscale/identity",  
9   "aud": "https://elephantscale",  
10  "exp": 1443101047,  
11  "nbf": 1443101047  
12  
13 }
```



# OpenID Connect token

- ◆ `id_token` contains claims about the authentication of an end user (and other requested claims)
- ◆ `id_token` can be used for signing in to an application
- ◆ Access tokens are for accessing resources
- ◆ `UserInfo` endpoint:
- ◆ Can be used by the client to get more user information of the authenticated user
- ◆ These claims are requested with the `access_token`

# OpenID Connect Flow

- ◆ Extends authorization code of OAuth 2 and implicit flow

## OpenID Connect Flows

OpenID Connect extends OAuth 2.0's Authorization Code and Implicit flows

**Authorization Code**

**Implicit**

... and introduces a new one

**Hybrid**

# Scopes And Claims

- ◆ OpenID Connect adds `identity` scopes to OAuth 2.0 resource scopes
- ◆ Example:
  - profile scope:
    - `name`
    - `family-name`
    - `middle-name`
    - `...`

# Scopes And Claims, cont'd

# OpenID Connect On Clients

Basics

Authorizing Access To API

OpenID Connect

**OpenID Connect On Clients**

Impersonating The User

Credentials

# Choosing The Right Flow

- ◆ Depends on the `response_type` is requested
- ◆ Authorization code: Confidential clients
- ◆ Implicit: Public clients
- ◆ Hybrid: Confidential or public clients if you send auth code to the server not user agent

# Helper Components

- ◆ **Helper components for JavaScript :-** **oidc client** - **oidc token manager** `bash`  
`https://github.com/IdentityModel/oidc-client`  
`https://github.com/IdentityModel/oidc-token-manager`

# Impersonating The User

Basics

Authorizing Access To API

OpenID Connect

OpenID Connect On Clients

**Impersonating The User**

Credentials



# A Bit API Level

## Impersonating the User



“ I can access the update method because I’m in the correct role”

“ I can get my private trips, because the API know who I am”

“ I can delete the pictures I added, because the API knows who I am”

**CLIENT**

**API**

# Role-Based Authentication

- ◆ We are not authorizing access to actions depending on the user (Because of functional requirements)
- ◆ We can add additional claims in the acces token and use them for authorization
- ◆ A role claim is an example, and allows role-based authorization

# Credentials

Basics

Authorizing Access To API

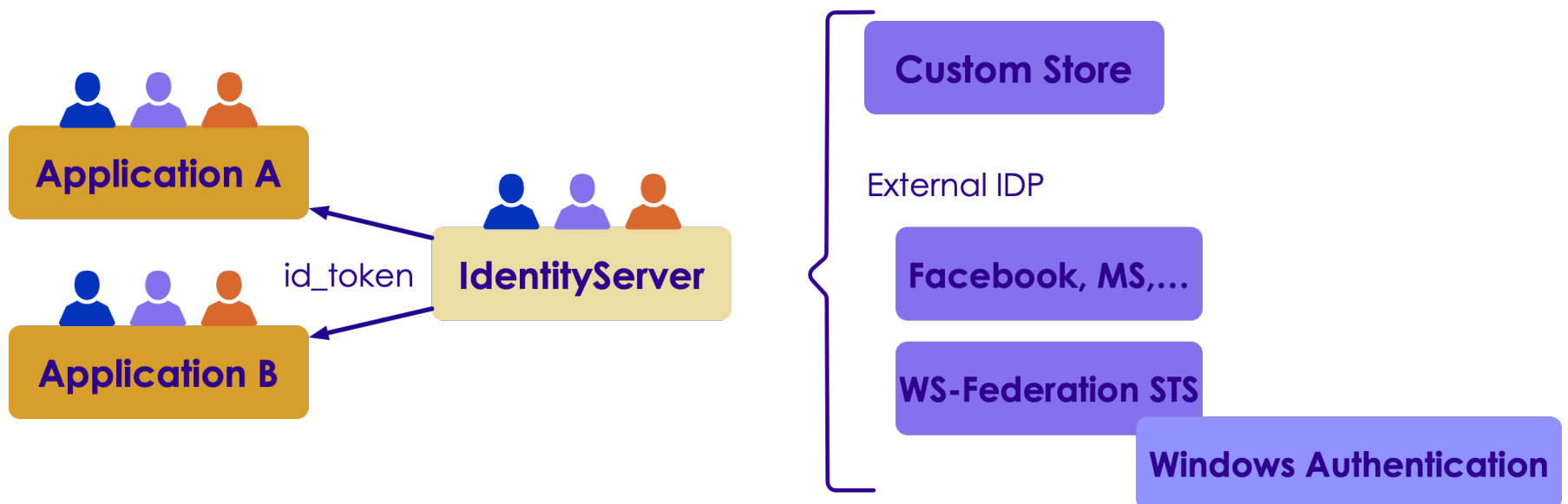
OpenID Connect

OpenID Connect On Clients

Impersonating The User

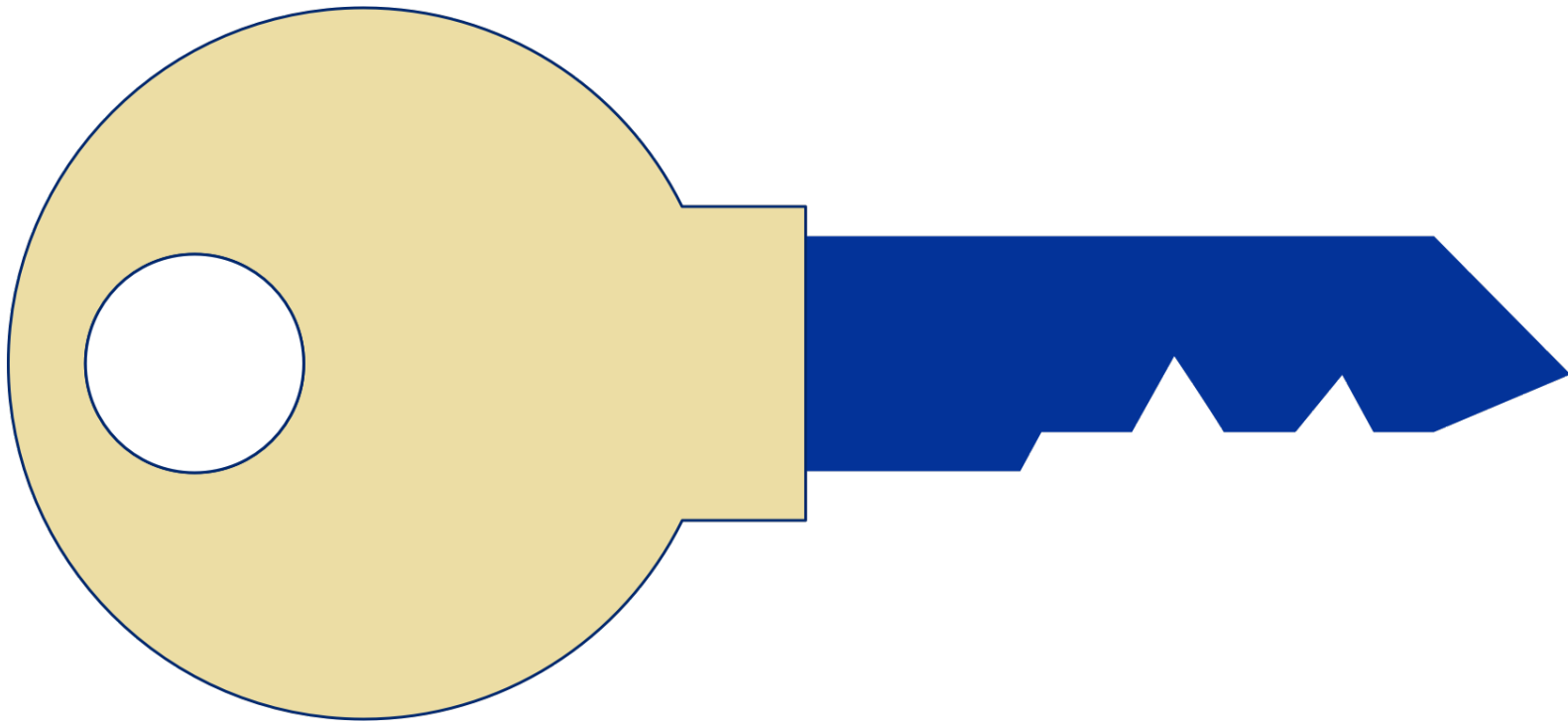
**Credentials**

## Dealing with Credentials



# Two-Factor Authentication

- ◆ 2FA provides identification of users by means of the combination of two different components
- ◆ Something you know, something you possess, something that's inseparable from you
- ◆ Not all applications might require 2FA
- ◆ Use `acr_values` parameter (to authorization endpoint)



# Additional Resources

- ◆ Entity Framework Persistence Layer

1 <https://github.com/IdentityServer/IdentityServer3.EntityFramework>

- ◆ Identity Manager

1 <https://github.com/IdentityManager/IdentityManager>