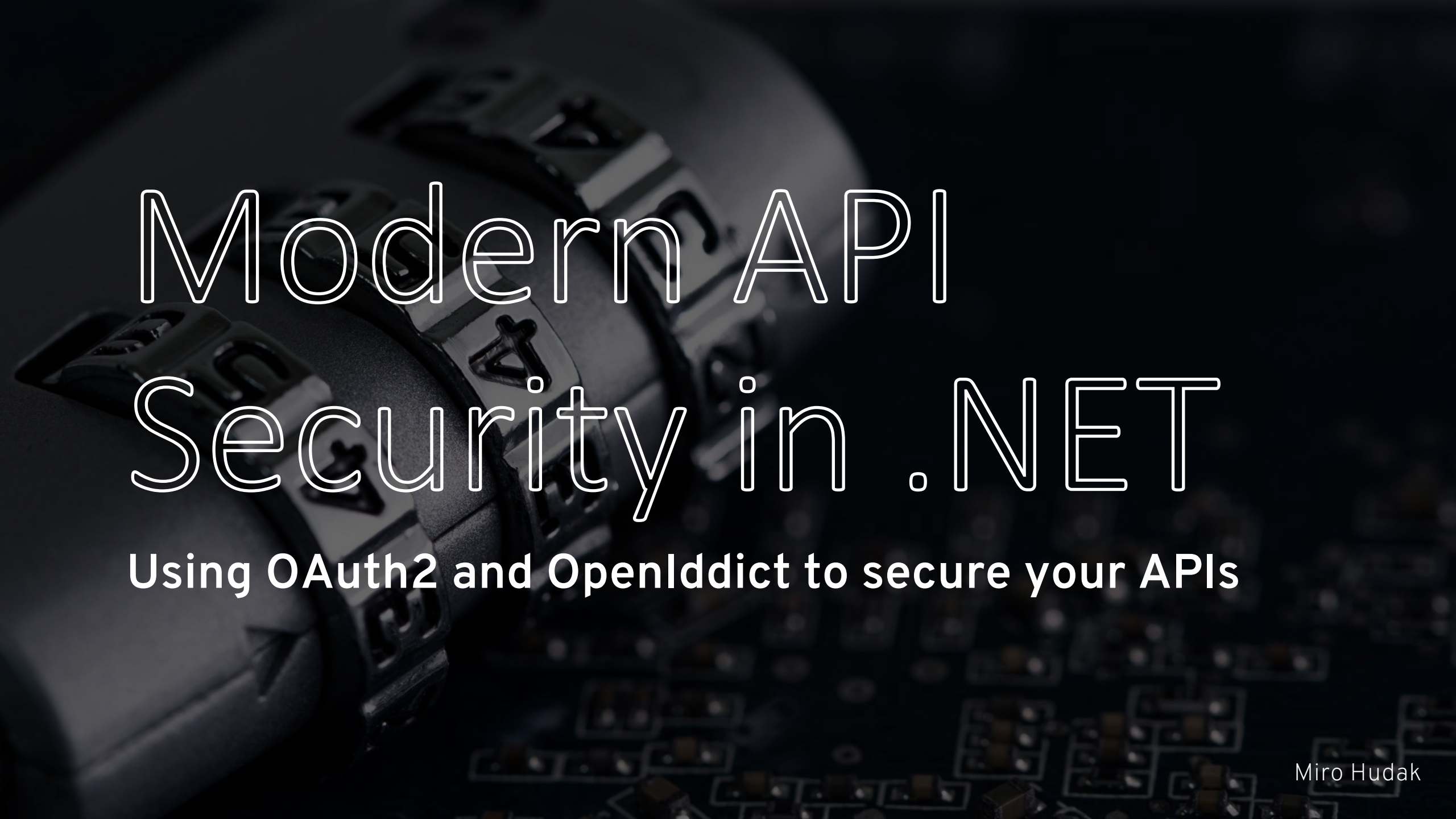


Modern API Security in .NET

Using OAuth2 and OpenIddict to secure your APIs

Miro Hudak

About Me

- **Miro Hudak**
- Senior Software Engineer in RABOT Charge GmbH

Authentication vs. Authorization

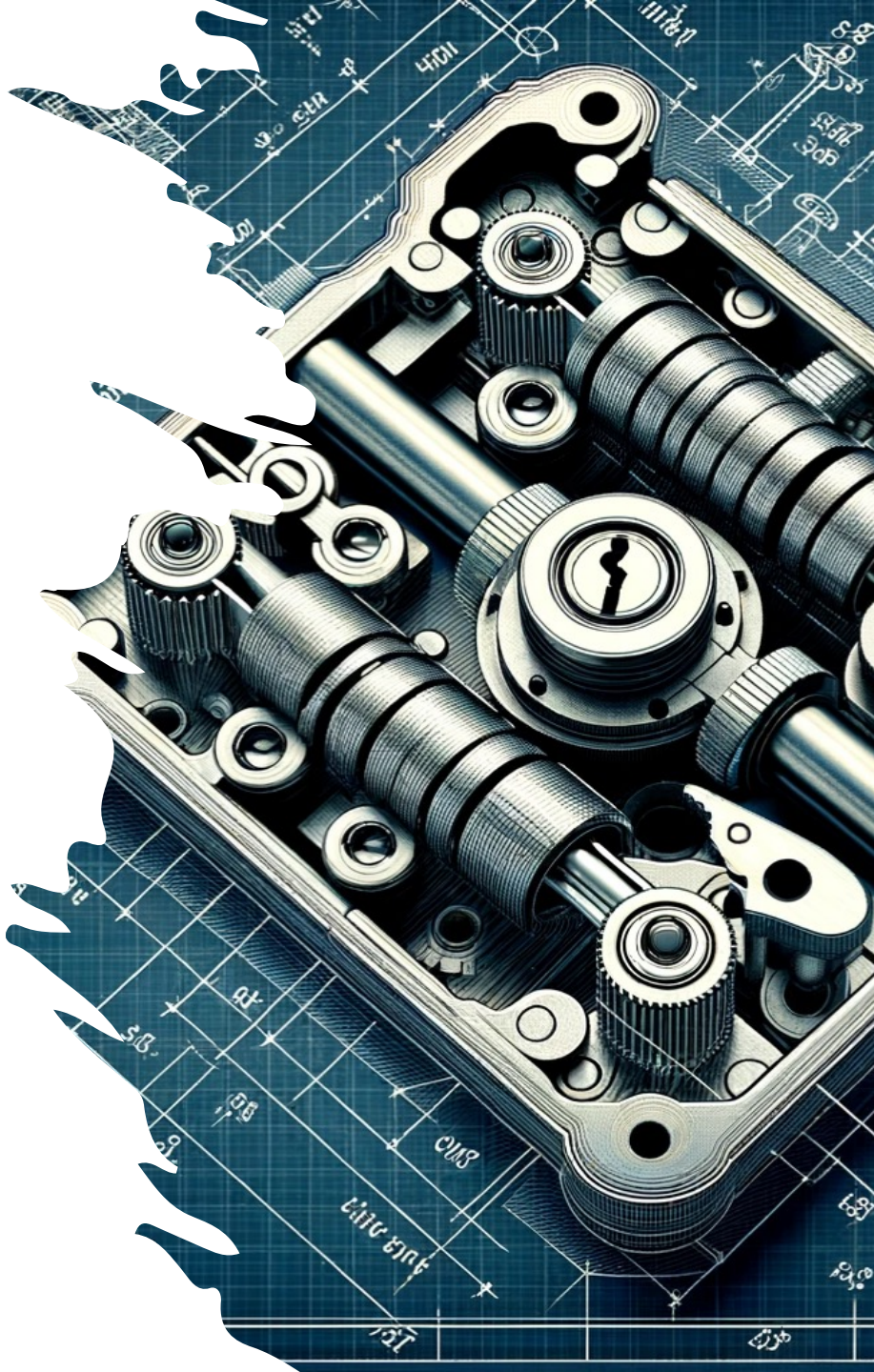
- **Authentication** is a process of verifying an identity:
  - Who are you?

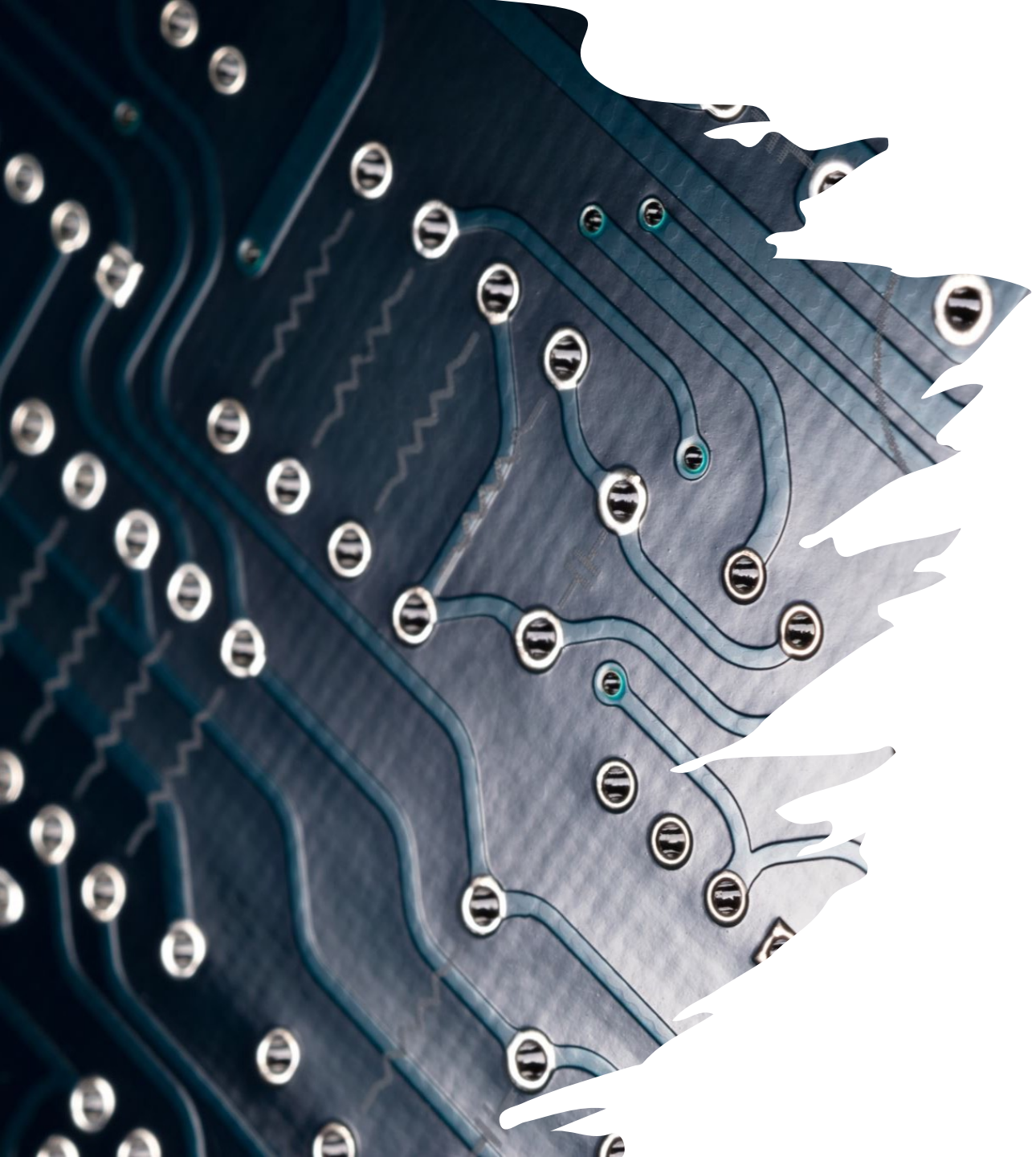
- **Authorization** is a process of establishing access to resources
  - What can you access?
  - What can you read / write?

API Tokens

- We need to consider, how we want to use tokens

- **Monolithic application**
  - Session token directly in database
- **Microservices**
  - Stateless Shared token
- **External Access**
  - No users' credentials disclosure





JSON Web Tokens

- De facto **Internet standard**
- Allows to **securely transmit claims** about identity and access
- Can be **signed** and **encrypted**
- Contains multitude of **standard fields**
- Allows specifying custom fields

Nomenclature

- **Claims** – facts, meta-data about the token
- **Scopes** – permission groups, that specify, which and how are resources accessed
- **Clients (Applications)** – systems, that can access our protected APIs
- **Authorizations** – information about tokens and consents

JWT Standard Claims

- **iss** – issuer – issuer of the token
- **sub** – subject – who has the access privileges
- **aud** – audience – who is the JWT intended for
- … and other.

- And also, you can define yours!

OAuth2

- **Open standard**
- Allows users to **share their data with third parties**, without disclosing their credentials
- Uses **Bearer** token (JWT)
- Provides **multiple grant types** for different use cases

OAuth2 Token Types

- **Access Token**
  - Used to access resources
  - Shorter expiration time
  - Can be cached / stored in local storage
- **Refresh Token**
  - Used to request a new access token
  - Long expiration
  - Should be stored in secure storage
  - In case, that it is not possible to store refresh token securely, PKCE should be used instead (if available)

OAuth2 Grant Types

- Authorization Code
- Resource Owner Password Credentials
- Client Credentials
- Refresh Token
- Device Authorization
- Proof Key for Code Exchange

Common OAuth2 Credentials

Client Credentials
Client ID and Client Secret

User Credentials
Username and Password

Authorization Code Grant

- Allows to provide access to third party without disclosing credentials

1. Third party **requests authorization code** via first party's system with specified scopes – groups of permissions to access resources
2. After logging in, user can **authorize access** to all or some scopes, requested by the third-party system
3. After authorizing, an **authorization code is returned**
4. This authorization code then can be used to **request access and refresh tokens**

Resource Owner Password Credentials Grant (ROPC)

- **Grant type removed** in OAuth 2.1, due to the credentials' disclosure
- Requires Client ID, Client Secret, Username and Password
- **Can be used as usual password entry**, when the calling system can have access to credentials
- **Not to be used for giving third parties access to APIs**

Client Credentials Grant

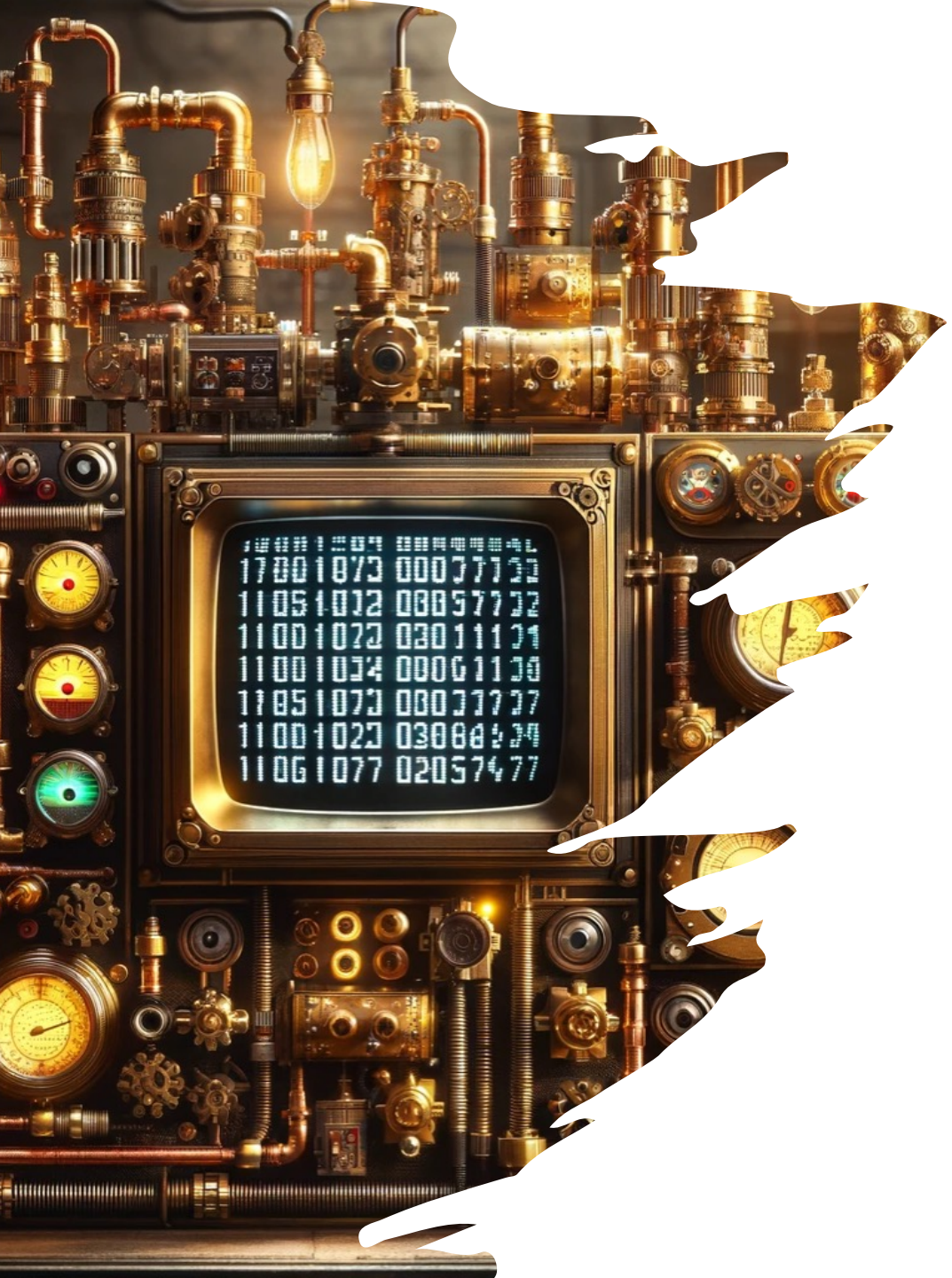
- Allows authentication of a client (system), **without specifying user**
- Useful for **API-level access**
- Only needs to specify Client ID and, optionally, Client Secret to obtain access token

Refresh Token Grant

- Grant used to **receive new Access Token** by providing previously obtained Refresh Token
- Usually, Refresh Token is then **rotated**
- Refresh Token **grant can request different scopes**, and as soon, as it's only a subset of the original scopes, it's provided without re-authorization; otherwise, re-authorization is necessary

PKCE - Proof Key for Code Exchange

- **Protection** against CSRF and injection attacks
- **Not an actual grant** - extension to existing Authorization Code grant
- Client first computes a ***Code Challenge*** derived from ***Code Verifier*** on Authorization Code request
- Then provides the original Code Verifier on Code Exchange to obtain Access Token



Device Authorization Grant

- Used to obtain tokens for devices, that have **limited input capabilities**
- For example, applications in set-top boxes, e.g., Netflix login on Apple TV
- Device identifies itself with **device code**
- It's provided with user code, **verification code** and **polling interval**
- It polls the token endpoint at internal and awaits Access Token, or Error

Identity Providers

- Identity & Access Management (IAM)
- Single Sign-On
- Identity Federation

- **IdentityServer** – .NET, commercialized
- **OpenIddict** - .NET open-source library
- **Keycloak** – Java, open-source
- **AWS Cognito**, **Azure AD B2C**
- **Auth0**, **Ory**,...

OpenIddict Library

- Open-source, **written in C#**
- Library only, **not a turn-key product**, needs custom implementation
- **Completely customizable**
- Integration with **Microsoft Identity Platform**, but does not require it
- Define custom grant types, also customization of claims, scopes,...
- Access **external user stores**

OpenIddict Challenges

- **Documentation is poor**
- There are **samples**, but those are also **limited**, yet readable
- If you want to avoid MS Identity Platform, there is more work to do
- Deeper **understanding of OAuth2 and/or OpenID is required**
- Custom implementation can lead to **security issues**

Why bother then?

- **Legacy userbase can be tricky to import** to external identity store and maintain in sync
  - **Passwords are usually hashed**, so you need to build some synchronization
  - You need to synchronize changes in userbase
- **Special requirements** for client handling
  - E.g. have customers bound to partners
- **Cost** is based on MAU or actual number of users
- **You can customize it** however you want

Live Demo!

Server Configuration

- OpenIddict **supports Entity Framework Core**
- Add **DbContext** and call some of the overloads of **UseOpenIddict()**
  - If you need to customize some of the built-in entities, use the overload with the most generic parameters and provide your own entities
- Call **services.AddOpenIddict()**, you need to call **AddCore()** and **AddServer()** in the fluent interface, where you configure behaviors and allow grants *(flows)*, provide endpoints, set issuer, certificates, etc.
  - There is a **nasty scope check**, that can cause problems in Authentication Code grant, that can be removed by removing event handler

Custom Entities

- When you need to add some fields to existing **OpenIddict** entities, they need to be implemented properly and then appropriate managers and stores need to be overridden as well to support those custom properties.
- Customized entities should be derived from **OpenIddict** entities and forced via [Table] attribute to be used instead.
- You need to explicitly replace them by calling **ReplaceDefaultEntities<..>** inside the **AddCore()** builder
- You need to implement all entity overrides even if you add just one field in one entity due to how the generics are structured

Client Credentials Grant Implementation

- **Client (called Application)** needs to be registered in **ApplicationManager**
- Library then automatically **verifies client credentials**, **grants** and allowed **scopes**
- Implementation of the flow is then only retrieving information from **OpenIddictServerRequest** and creating *claims identity* and using that to create *claims principal*
- Based on *claims principal*, library then generates *access token* response and, when **offline_access** scope is requested, also *refresh token*

ROPC Grant Implementation

- As in Client Credentials flow, client must be registered and ROPC grant must be allowed, otherwise the flow won't proceed
- **OpenIddictServerRequest** now contains also **Username** and **Password**, which **should be verified** against user store
- From there, the usual flow as previously is performed, creating *claims identity* and *claims principal*, then returning it as **SignIn()** call from controller

Authorization Code Grant Implementation

- Most complicated grant type with lots of moving parts
- Works as exchange of **authorization code**, retrieved via calling Authorization endpoint (e.g. **/oauth/authorize**) and then, after receiving **Authorization Code**, this code is exchanged for *Access Token* (and possibly *Refresh Token*) via calling **/oauth/token** with the code, returned from the Authorization endpoint.
- For the authorization to proceed, user must be logged in to the external *(identity providing)* system, that can store cookie in the context of the login website and after successful login, displays consent page.

Authorization Code Grant Implementation

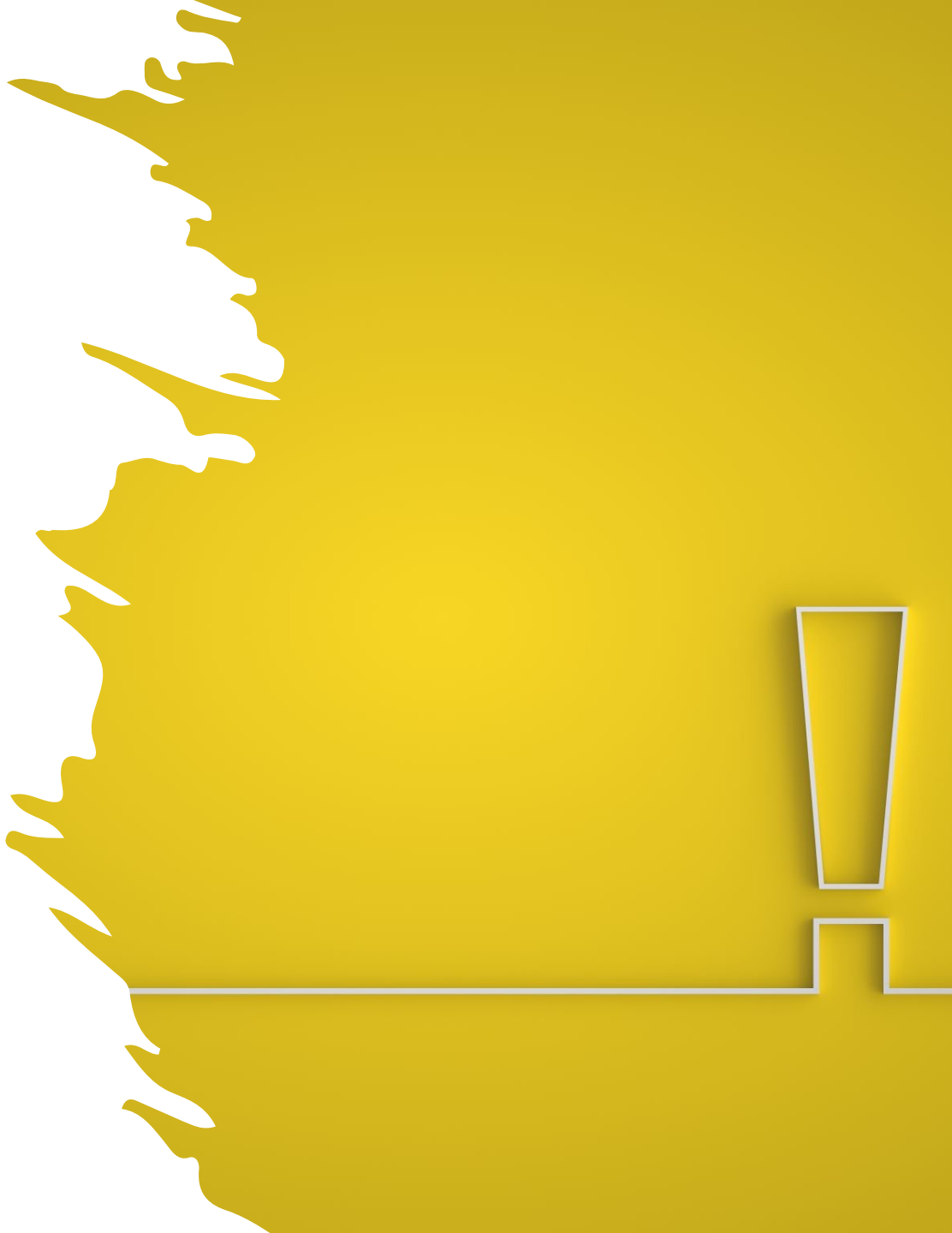
- On the *consent page*, user has option to **Allow scopes** (either all or they can select specific, depending on the implementation) or **Deny request**.
- When user **Allows** the access to requested scopes, a code is returned as a response to specified redirect URI, where client system receives it and can call **Token** endpoint to obtain tokens.
- As soon, as the tokens are received, the flow continues normally, access token can be used to access the API, refresh token (if requested) is used via Refresh Token flow to request new tokens.

Recommendations

- Use **Authorization Code Grant** to allow your third-party partners to call your API on behalf of customers
- Use **Client Credentials Grant** to allow your third-party partners to call your API directly
- Use **ROPC Grant** only when you are in full control over the client

Recommendations

- If you need to provide client credentials, **avoid storing them in the application**, if you don't have full control over it (SPA, mobile, desktop); rather **create additional API** that will invoke the flows without exposing the client credentials
- **Separate client credentials for user-related and API-only flows**, allow only required grants and scopes, separate audiences if possible
- **Use signing certificates and encryption**



Resources

- **OAuth2 Overview**
  - https://www.oauth.com
- **OpenIddict Library**
  - https://github.com/openiddict/openiddict-core
- **Example Implementation**
  - https://github.com/mx0r/openiddict-example
- **Keycloak**
  - https://www.keycloak.org


Questions?


Thank you!