

# Universita degli Studi di Parma



DIPARTIMENTO DI INGEGNERIA ED ARCHITETTURA

Corso di Laurea di ingegneria Informatica, Elettronica e delle  
Telecomunicazioni

---

## OAuth2

**Relatore:**

Prof. Ing. Poggi Agostino

**Tesi di laurea di:**

Corradi Alessandro

A.A. 2019/2020

# Contents

---

<b>Introduction</b>	<b>1</b>
The problem . . . . .	1
<b>OAuth2 Protocol</b>	<b>3</b>
. . . . .	3
Obtain an access token . . . . .	3
Revoke an access token . . . . .	3
<b>SSO with OpenID</b>	<b>4</b>
Second problem . . . . .	4
JWT . . . . .	4
JWS e JWE . . . . .	4
JWS Validation . . . . .	4

# Introduction

---

## The problem

Let's imagine that we have a server that manages a vast list of products. Those products could be searched, filtered and sorted by different fields, like the name, color, insertion date and so on. The search query is submitted, the server returns a web page with the products, represented in a tabular fashion.

The server becomes popular and is used by million of users, and you start to think that sharing those search results could be a cool feature.

We want to use third party API to integrate this feature.

## Possible solutions

Necessity of communication between client and server. Different ways to achieve the goal, here are a few of them:

1. Directly send the user id to the server

```
$ curl -v -X POST http://localhost:8080/db_auth '{"user-id": 1}'
```

2. Http Basic Auth: each API request is signed with a username and a password, encoded in base64

```
$ curl -u Aladdin:OpenSesamus -v http://localhost:8080/basic
```

```
> GET / HTTP/1.1
> Host: localhost:8080
> Authorization: Basic QWxhZGRpbjPcGVuU2VzYW11cw==
> User-Agent: curl/7.72.0
> Accept: */*
...
```

Since the credentials are neither hashed nor encrypted, your username and password are always visible by someone who could read the headers. Therefore this protocol requires an https channel in order to be somewhat secure.

3. Authorization via asymmetric cryptography. Two keys are generated: a public and a secret one. The public key is, as the name suggests, public and everyone could read it. The client encodes his messages with the private key; the receivers could validate the authenticity of those messages using the public key.

This is one of the safest methods, but it's not viable as we need to distribute in some way the private key to trusted users.

4. Assign an authorization token at each client where it could be used for API requests

```
$ curl -v --header 'Authorization: Bearer 1234' http://localhost:8080/api
```

```
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.72.0
> Accept: */*
> Authorization: Bearer 1234
...
```

The token could be in different states: valid, expired, revoked, invalid.  
OAuth2 provides a secure way to obtain and exchange these tokens.

# OAuth2 Protocol

## Obtain an access token

1. Redirect to an authorization provider.

`http://google.apis.com?response_type=code&client_id...`

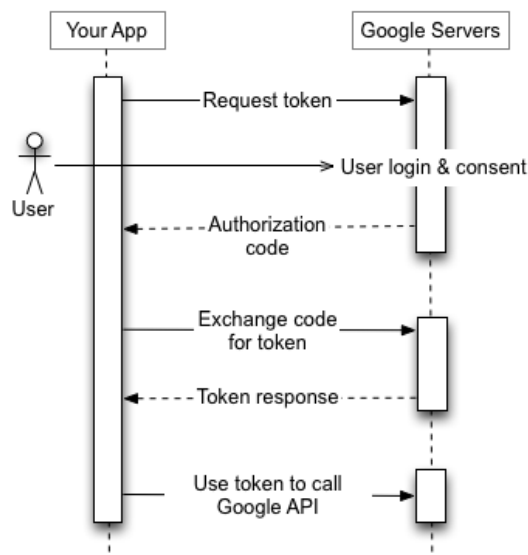
2. If the authorization is successful, the oauth server will redirect to `redirect_uri` passing a `code` as GET parameter. Otherwise `error` contains the reason why the authorization was not successful.

```
POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=authorization_code
&code=xxxxxxxxxxx
&redirect_uri=https://example-app.com/redirect
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx
```

*If an authorization code is used more than once, the authorization server must deny the subsequent request.*

3. The client exchanges the `code`, called also "grant token", with the server to obtain an `access_token`.
4. The server returns the access token with additional informations, such as expire date and JWT.
5. The client uses the access token for APIs requests.



## Revoke an access token

Something something

# SSO with OpenID

---

## Second problem

We want to delegate the process of authentication of username and password to an external service, then if the user is registered in our platform we allow him in.

## JWT

Shorthand JSON Web Token, encoded token that contains the necessary informations to identify the user.

## JWS e JWE

Token could be signed, encrypted or both.

## JWS Validation

Verify the SHA256 signature

# References

---

- [1] Using OAuth 2.0 to Access Google APIs  
<https://developers.google.com/identity/protocols/oauth2>