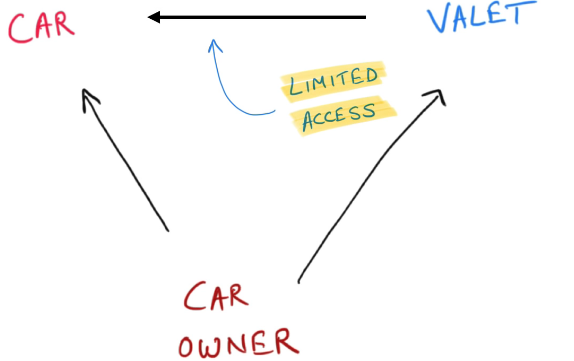
# OAuth Security

**OAuth** is mainly for **authorization** protocol. It’s a standard for **access delegation.** OAuth is first created for a service to authorize service.

The current most widely used version of OAuth is OAuth2.0

A real world non-technical example where a car owner handover valet key to valet boy, which has very limited functionality.



Using OAuth services can access each other on our behalf and this is commonly called **Access Delegation.**

There is a certain flow needs to happen for all these to work called OAuth flow.

## OAuth flow

Considering an example say we are accessing photo printing application which will print the photo from our Google drive.

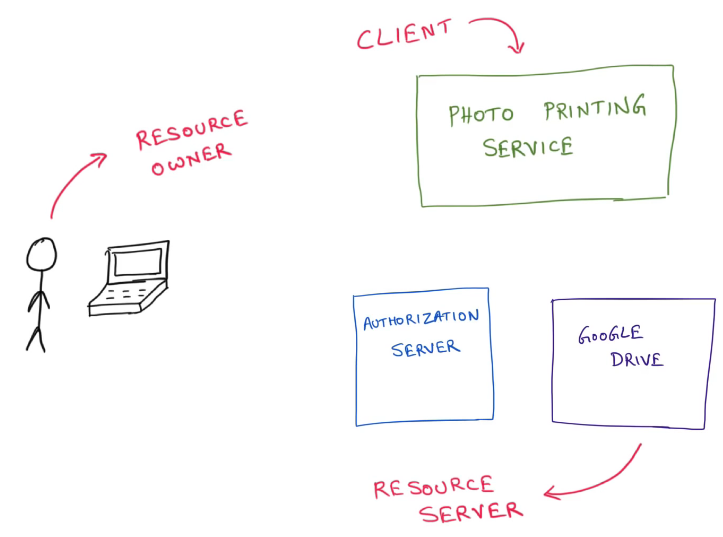
There is no way we can share our Google credential to the photo printing application for drive access. But if both of the service are in agreement and both implemented OAuth then delegating communication can happen.

Here when the photo printing application tries to access Google. Then Google can go to the user with an authentication screen for user if the user wants to give the permission. If user authenticated then Google will generate a **Token (**with limited access) and handover to photo printing application and every time the app will come to Google it will come with the token to prove it’s authorized to have limited access.

Here the **access token** is **JWT.** It’s a secured and signed token.

### Terminologies used in OAuth

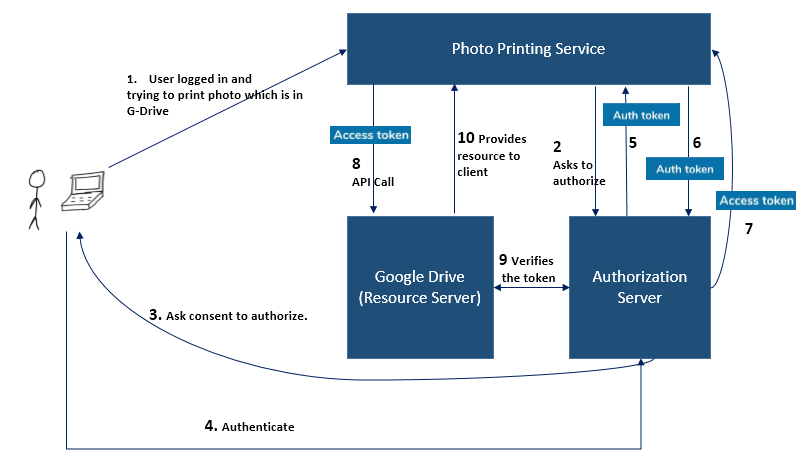
* **Resource/Protected resource** – Photos in Google drive
* **Resource owner** – Person who has the access to resource. This is also the person who can gives access to the photo printing application.
* **Resource Server**- Server holding protected resource. Here the Google
* **Client-** The application that making request to resource server for the protected resource in behalf of client. Here the photo printing application.
* **Authorization Server:** Here the resource server and authorization server are coupled (Google). Resource server implemented OAuth. Authorization server has the burden of security. It could be a separate server or could be the same as resource server. Authorization server issues **Access Token** to client.



## OAuth Flow

OAuth has multiple flows to handle this communication.

### Authorization Code Flow



**Auth Token** are the short lived token used to exchange for **Access Token.**

The Resource server and Authorization server can be same (in most of cases). Then it will validate the flow itself.

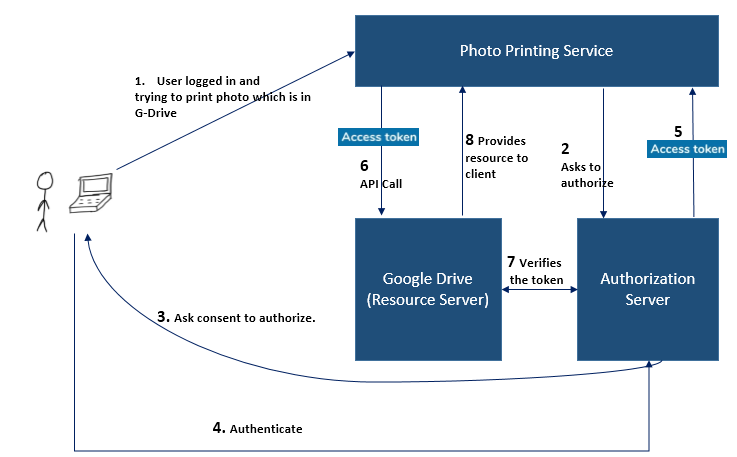
This is the most secured and trusted flow.

### Implicit Flow

It’s a simplified flow where Authorization server sends the Access Token directly without Auth Token/Key.

**Drawback:** Here because of this there is no secure exchange of **Access Token** and which makes it bit unsecure, if someone got holds to Access Token, they could impersonate the user.

This flow is useful for javascript application here we know access token will be exchanged over network and will sit on browser which is not very secure to begin with. So here there couldn’t have secure exchange of **Auth Token.** So we can use implicit flow anyway which is bit less secure.



The **Access Token** is **short lived** token.

### Client Credential Flow

This flow is heavily used for communication between **microservices.** Here the clients is well trusted (confidential client).

When all the clients we are writing by our self. Which is the case in microservices.

