BITS Pilani
Pilani Campus

Full Stack Application Development- SE ZG503

Akshaya Ganesan

CSIS, WILP

BITS Pilani
Pilani Campus

Lecture No: 15 Security

Agenda

- Basic Authentication
- API Keys
- JWT
- OAuth

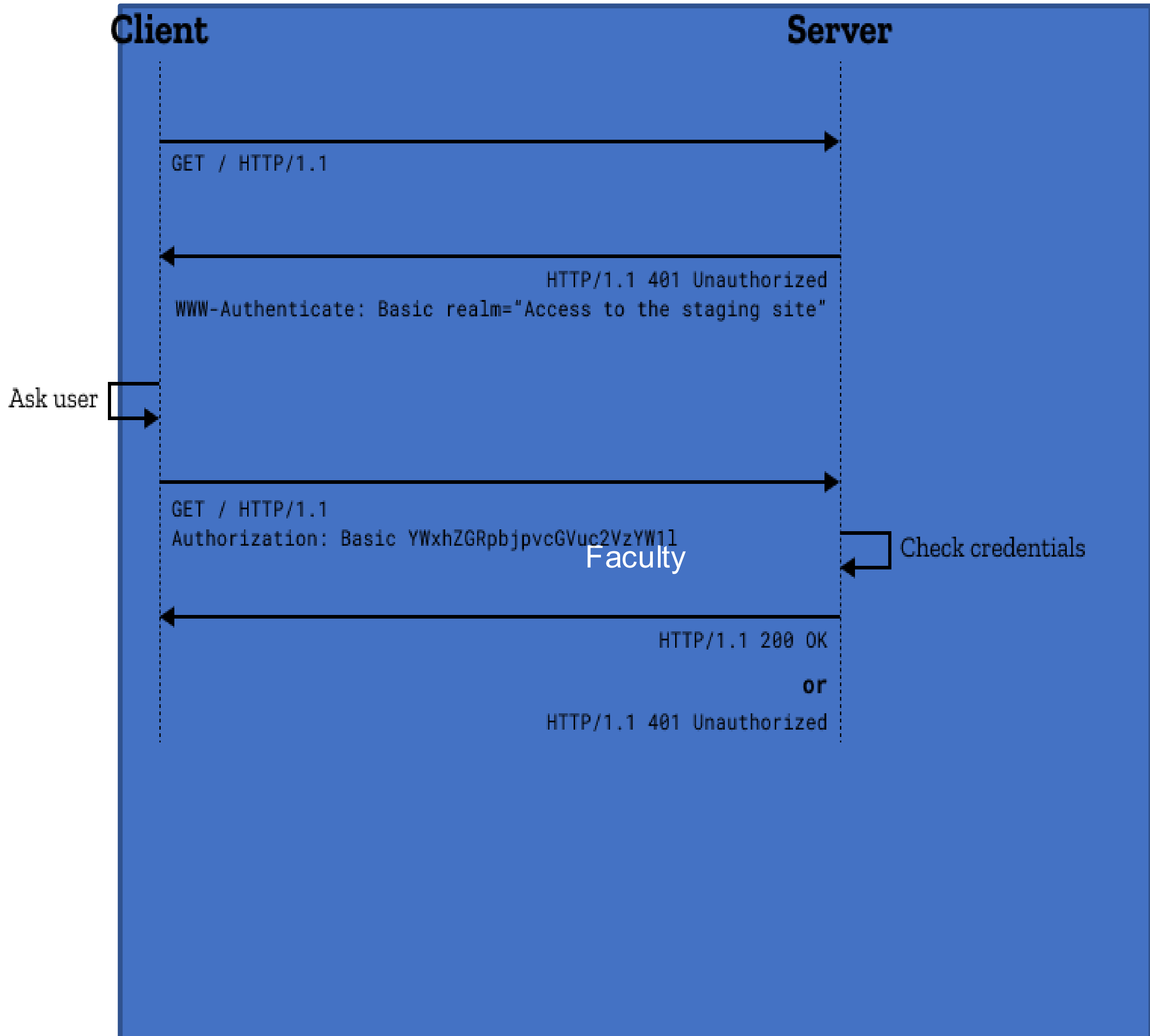
Security

- Authentication and authorization are two foundation elements of security:
- Authentication is the process of verifying who a user is.
- Authorization is the process of verifying what they have access to.

Basic Authentication

- HTTP provides a general framework for access control and authentication.
- In basic HTTP authentication, a request contains a header field in the form of

  Authorization: Basic <credentials>
- The credentials is the Base64 encoding of username and password joined by a single colon
- Basic authentication is typically used in conjunction with HTTPS to provide confidentiality.



```
GET / HTTP/1.1

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to the staging site"

GET / HTTP/1.1
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l

HTTP/1.1 200 OK
or
HTTP/1.1 401 Unauthorized
```

Image Ref:https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

Basic Authentication

- Here are some reasons why it is still used:
  - Simplicity
  - Compatibility
  - Statelessness
- Limitations
  - Security
  - Single Factor
  - Risk of Credential Exposure

API Keys

- An **API key** is a unique identifier used to authenticate and authorize the access to an **API**
- Instead of a username and password, the client application is issued a unique API key, typically a long alphanumeric string.
- The API key is sent in the HTTP request as a parameter in the query string or the request headers
- (e.g., api_key=your_api_key or Authorization: API-Key your_api_key).
- API keys are commonly used for machine-to-machine communication or applications interacting with the API.
- Some APIs use API keys to enforce rate limits.

API Keys

- **Security**: Treat API keys as sensitive information. Avoid hardcoding them in client-side code or exposing them publicly.
- **Rotation**: Regularly rotate API keys to enhance security. You can invalidate a key and issue a new one if a key is compromised.
- **Scopes**: Consider using different keys for different purposes (e.g., read-only vs. administrative access).
- **HTTPS**: Always use HTTPS to transmit API keys securely.
- Examples: **Google Maps API, Cloud Services**:

Token-based authentication system

- Token-based authentication allows users to verify their identity, and in return receive a unique access token.
- Token-based authentication is different from traditional password-based technique.
- In stateless communication, each request that the user makes to the server contains all the necessary information for authentication, typically in the form of token.
- The server validates the token and responds accordingly for each request.

Types Of Tokens

- **Opaque tokens**
- The opaque token is a random, unique string of characters the authorization server issues.
- The opaque token does not pass any identifiable information
- To validate the token and retrieve the information on the token and the user, the resource server calls the authorization server and requests the token introspection.
- **Structured token:**
- Its format is well-defined so the resource server can decode and verify the token without calling the authorization server.
- JWT is a structured token

JSON Web Tokens (JWT)

- JSON Web Tokens (JWTs) are a format of tokens used in web development and security.
- JWT is a standard way to securely represent claims, such as user identity and roles, between two parties.
- A JWT has a payload, which is a JSON object that contains information about the user, such as their identity and roles, and other metadata, such as an expiration date.
- It’s signed with a secret that’s only known to the creator of the JWT.
- The secret ensures a malicious third party can’t forge or tamper with a JWT.

JSON Web Tokens (JWT)

- JSON Web Tokens consist of three parts separated by dots (.), which are:
  - Header
  - Payload
  - Signature
- Therefore, a JWT typically looks like the following: xxxxx.yyyyy.zzzzz

JSON Web Tokens (JWT)

- **Header**
- The header typically consists of two parts: the type of the token, which is JWT, and the hashing algorithm such as

  HMAC SHA256 or RSA.
- Then, this JSON is Base64Url encoded to form the first part of the JWT.
- **Payload**
- The second part of the token is the payload, which contains the claims.
- Claims are statements about an entity (typically, the user) and additional metadata.
- The payload is then **Base64Url** encoded to form the second part of the JWT.

```
Header:
{
"alg": "HS256",
  "typ": "JWT"
}
```

Faculty

```
Payload:
{
    "sub": "1234567890",
    "name": "John Doe",
     "admin": true
}
```

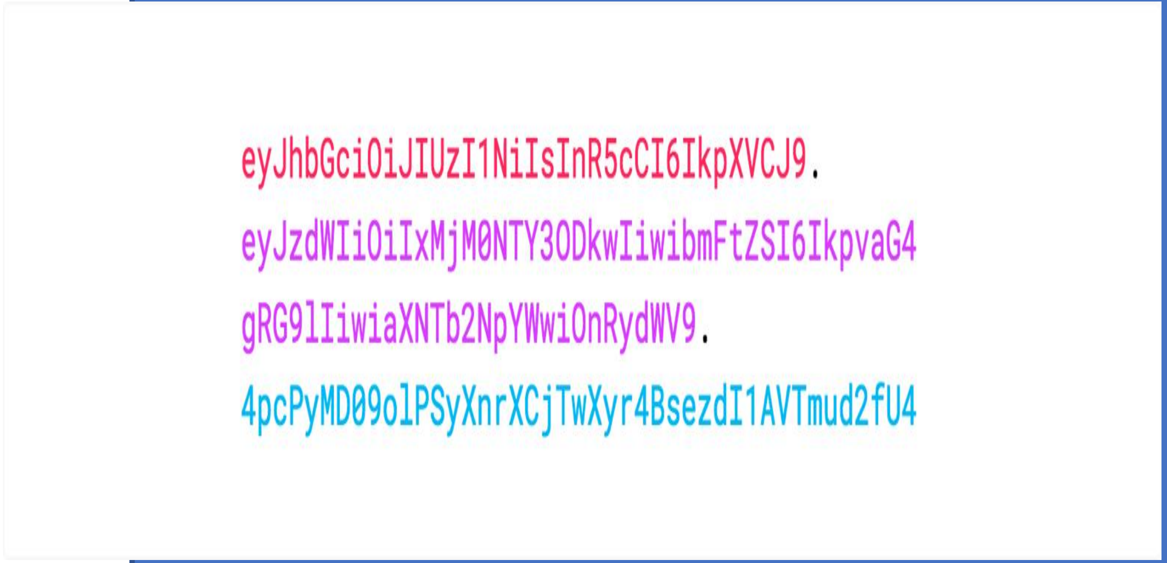
JSON Web Tokens (JWT)

- **Signature**

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

```
HMACSHA256(
  base64UrlEncode(header) + '.' +
  base64UrlEncode(payload),
  secret)
```

- The signature is used to verify that the sender of the JWT
- The output is three Base64 strings separated by dots



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Uses of JWT

- Information Exchange
- Single Sign on
- Authentication and Authorization

BITS Pilani
Pilani Campus

OAuth

OAuth

- OAuth (Open Authorization) is an open standard for access delegation.
- It is commonly used as a way for internet users to grant websites or applications access to their information on other websites.
- It specifies a process for resource owners to authorize third-party access to their server resources without providing credentials.
- OAuth works over HTTPS

Without OAuth

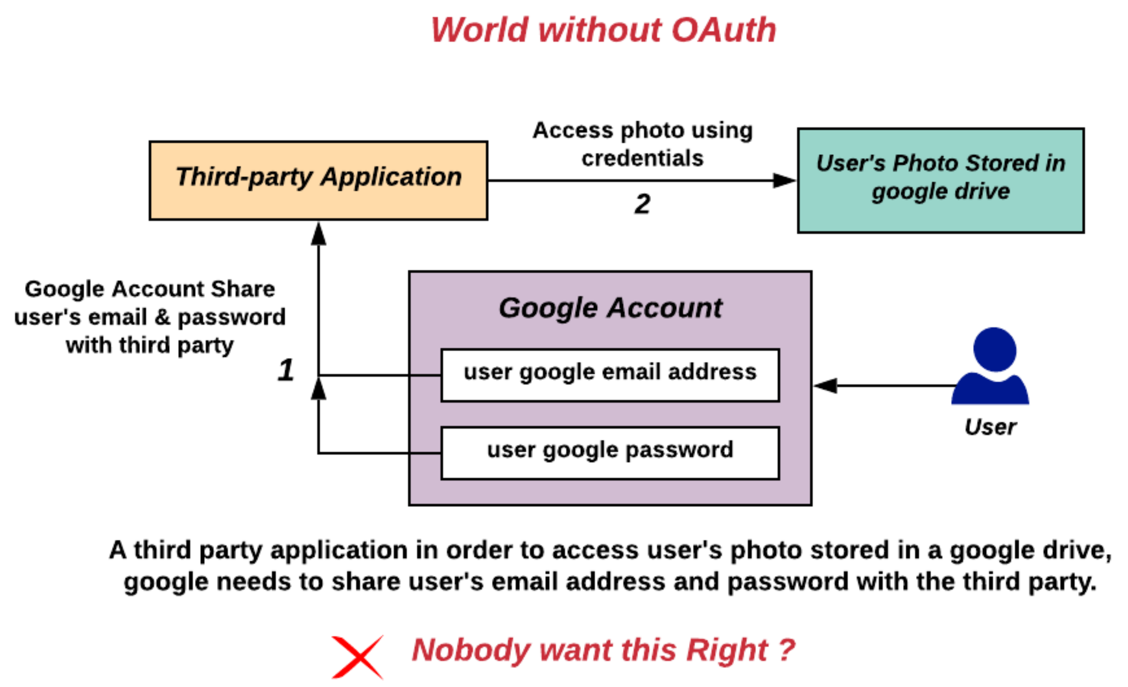

World without OAuth
Third-party Application
Access photo using credentials
2
User's Photo Stored in google drive
Google Account Share user's email & password with third party
1
Google Account
user google email address
user google password
User
A third party application in order to access user's photo stored in a google drive, google needs to share user's email address and password with the third party.
Nobody want this Right ?

Image Ref: By Devansvd - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=109591037

OAuth

- OAuth is a delegated authorization framework for REST/APIs.
- It enables apps to obtain limited access (scopes) to a user’s data without giving away a user’s password.
- It decouples authentication from authorization and supports multiple use cases addressing different device capabilities.
- It supports server-to-server apps, browser-based apps, mobile/native apps, and consoles/TVs.

OAuth

- Analogy: If you have a hotel key card, you can access your room.
- How do you get a hotel key card?
- You have to do an authentication process at the front desk to get it.
- After authenticating and obtaining the key card, you can access the room and resources permitted across the hotel.
- Similarly ,App requests authorization from User

OAuth Flow

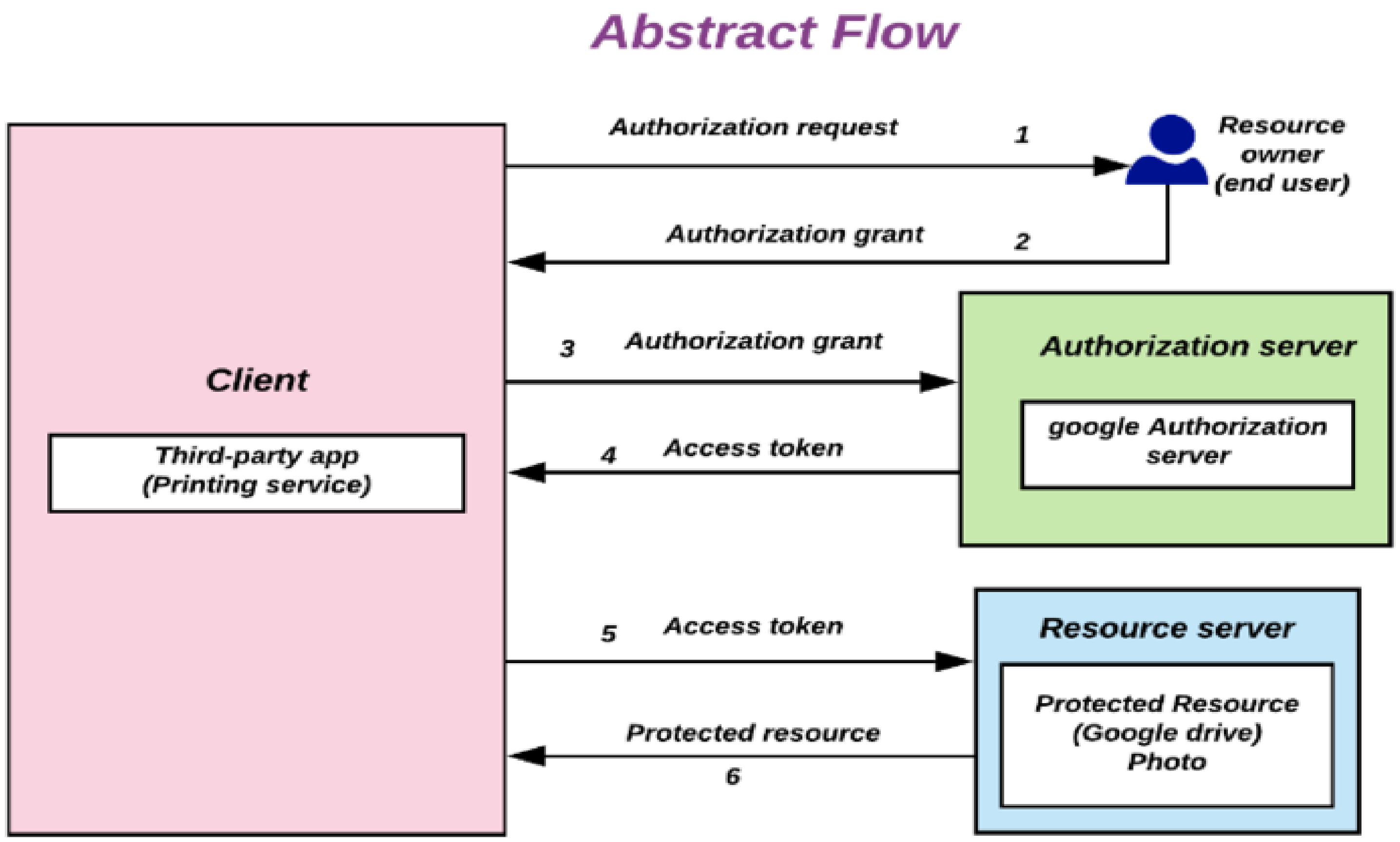


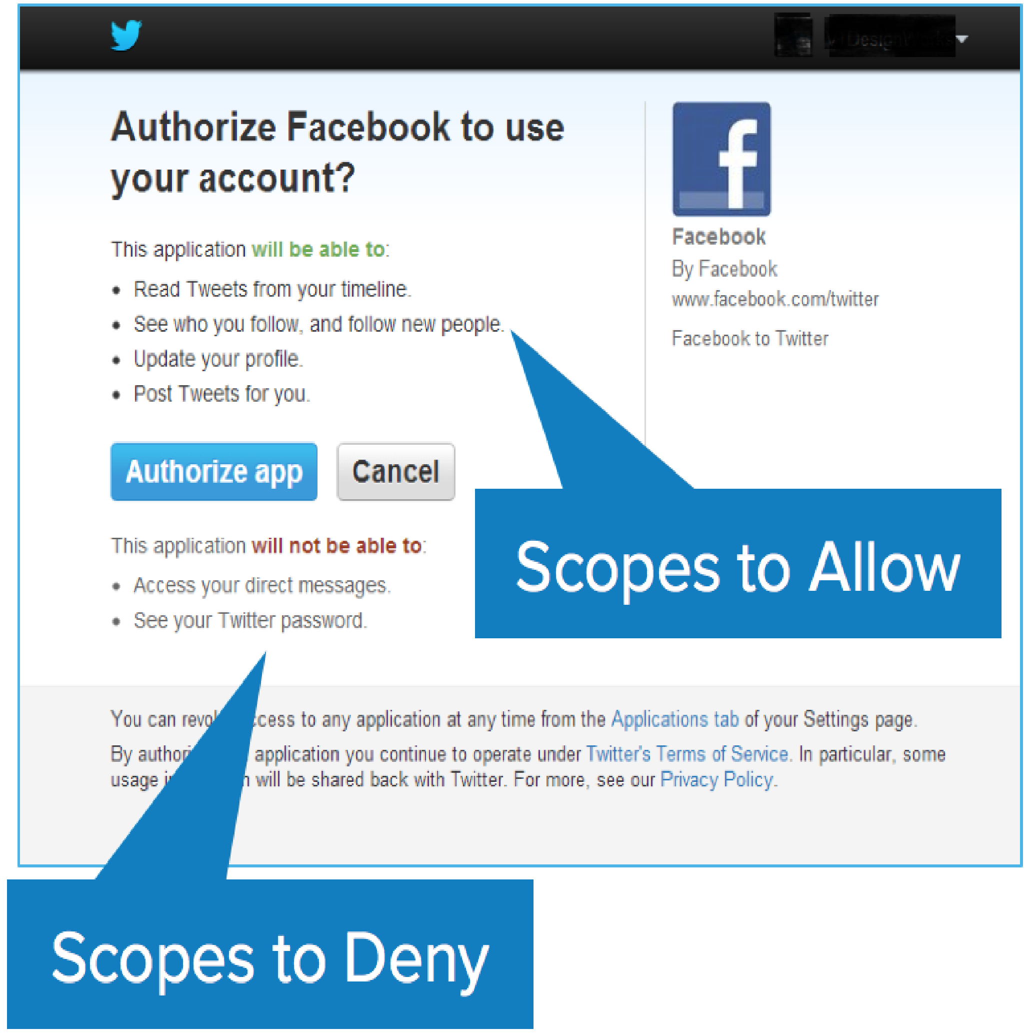
Image Reference: By Devansvd - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=109591026

OAuth Central Components

- OAuth is built on the following central components:
  - Scopes and Consent
  - Actors
  - Tokens
  - Flows

Scopes

- Scopes are what you see on the authorization screens when an app requests permissions.
- They're bundles of permissions asked for by the client when requesting a token.
- These are coded by the application developer when writing the application.


[redacted]

Authorize Facebook to use your account?

This application **will be able to**:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Authorize app   Cancel

This application **will not be able to**:

- Access your direct messages.
- See your Twitter password.

Facebook
By Facebook
www.facebook.com/twitter
Facebook to Twitter

You can revo cess to any application at any time from the Applications tab of your Settings page.
By author application you continue to operate under Twitter's Terms of Service. In particular, some usage i n will be shared back with Twitter. For more, see our Privacy Policy.

Actors

- The actors in OAuth flows are as follows:
- **Resource Owner**: owns the data in the resource server.
- **Resource Server**: The API which stores data the application wants to access
- **Client**: the application that wants to access your data
- **Authorization Server**: The main engine of OAuth

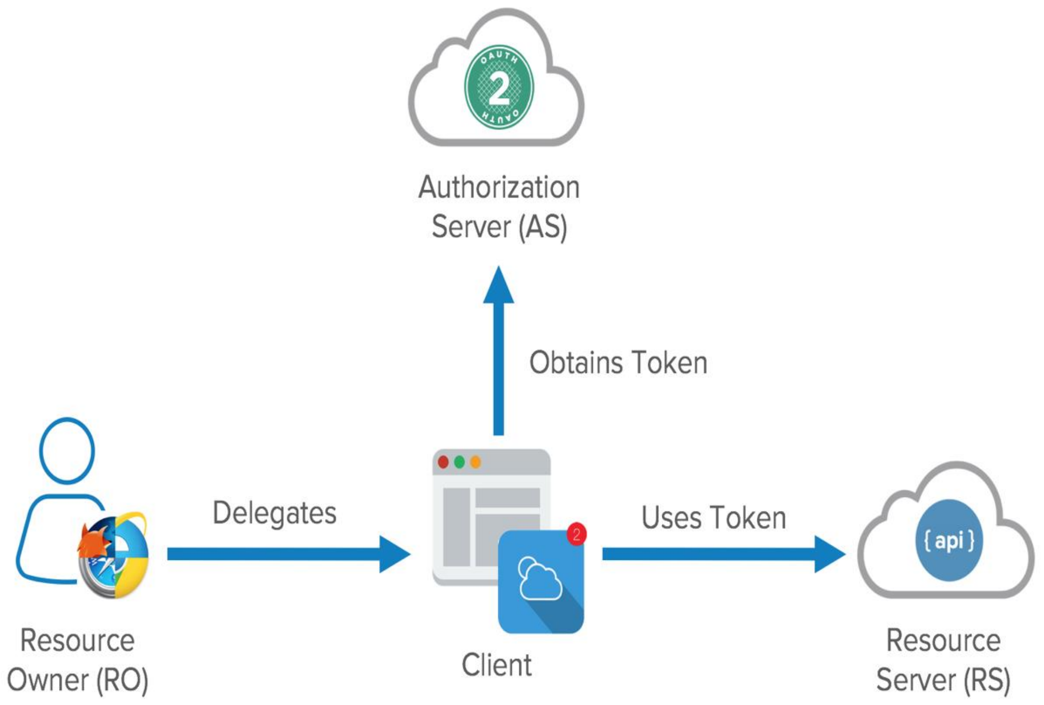


Image Reference:https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth

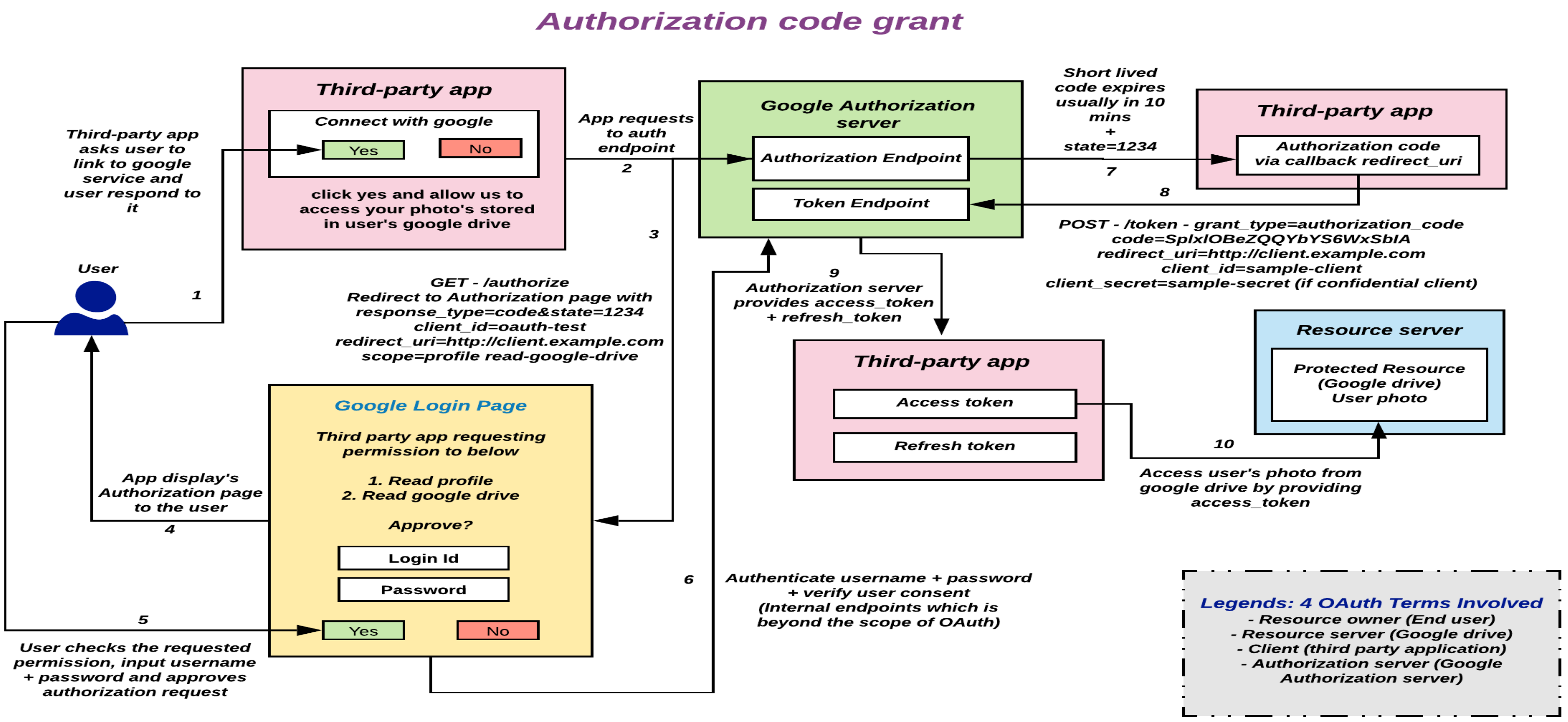
Tokens

- Access tokens are the token the client uses to access the Resource Server (API).
- They're meant to be short-lived.
- Refresh Tokens can be used to get new tokens
- The OAuth spec doesn't define what a token is
- Usually JWT is used
- Tokens are retrieved from endpoints on the authorization server.
- The two main endpoints are the authorize endpoint and the token endpoint.

Flows

- OAuth framework specifies several grant types for different use cases.
- OAuth grant types
  - Authorization Code
  - Client Credentials
  - Implicit Flow
  - Resource Owner Password Flow

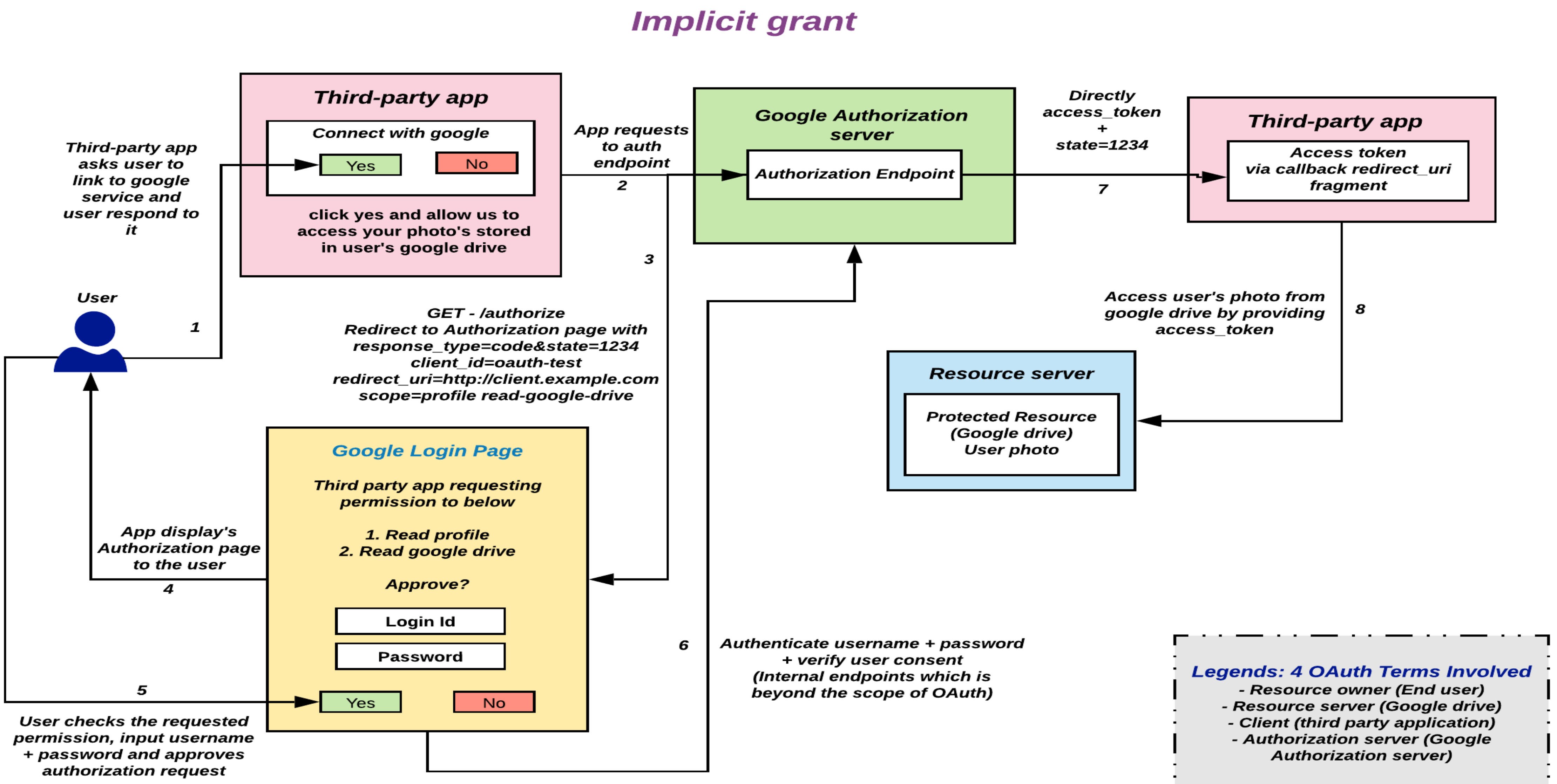
Authorization code


Authorization code grant
Third-party app asks user to link to google service and user respond to it
Third-party app
Connect with google
Yes
No
click yes and allow us to access your photo's stored in user's google drive
App requests to auth endpoint
2
Google Authorization server
Authorization Endpoint
Token Endpoint
Short lived code expires usually in 10 mins + state=1234
7
Third-party app
Authorization code via callback redirect_uri
8
POST - /token - grant_type=authorization_code
code=SplxlOBeZQQYbYS6WxSbIA
redirect_uri=http://client.example.com
client_id=sample-client
client_secret=sample-secret (if confidential client)
User
1
3
GET - /authorize
Redirect to Authorization page with
response_type=code&state=1234
client_id=oauth-test
redirect_uri=http://client.example.com
scope=profile read-google-drive
9
Authorization server provides access_token + refresh_token
Third-party app
Access token
Refresh token
Resource server
Protected Resource (Google drive) User photo
10
Access user's photo from google drive by providing access_token
Google Login Page
Third party app requesting permission to below
1. Read profile
2. Read google drive
Approve?
Login Id
Password
Yes
No
App display's Authorization page to the user
4
5
User checks the requested permission, input username + password and approves authorization request
6
Authenticate username + password + verify user consent (Internal endpoints which is beyond the scope of OAuth)
Legends: 4 OAuth Terms Involved
- Resource owner (End user)
- Resource server (Google drive)
- Client (third party application)
- Authorization server (Google Authorization server)

Authorization code flow

- Use Case: Regular web apps executing on a server.
- Example: A web application that needs to securely retrieve an access token.
- The client (web app) exchanges an authorization code for an access token. It's considered safe because the token is passed directly to the server without going through the user's browser.

Implicit Flow

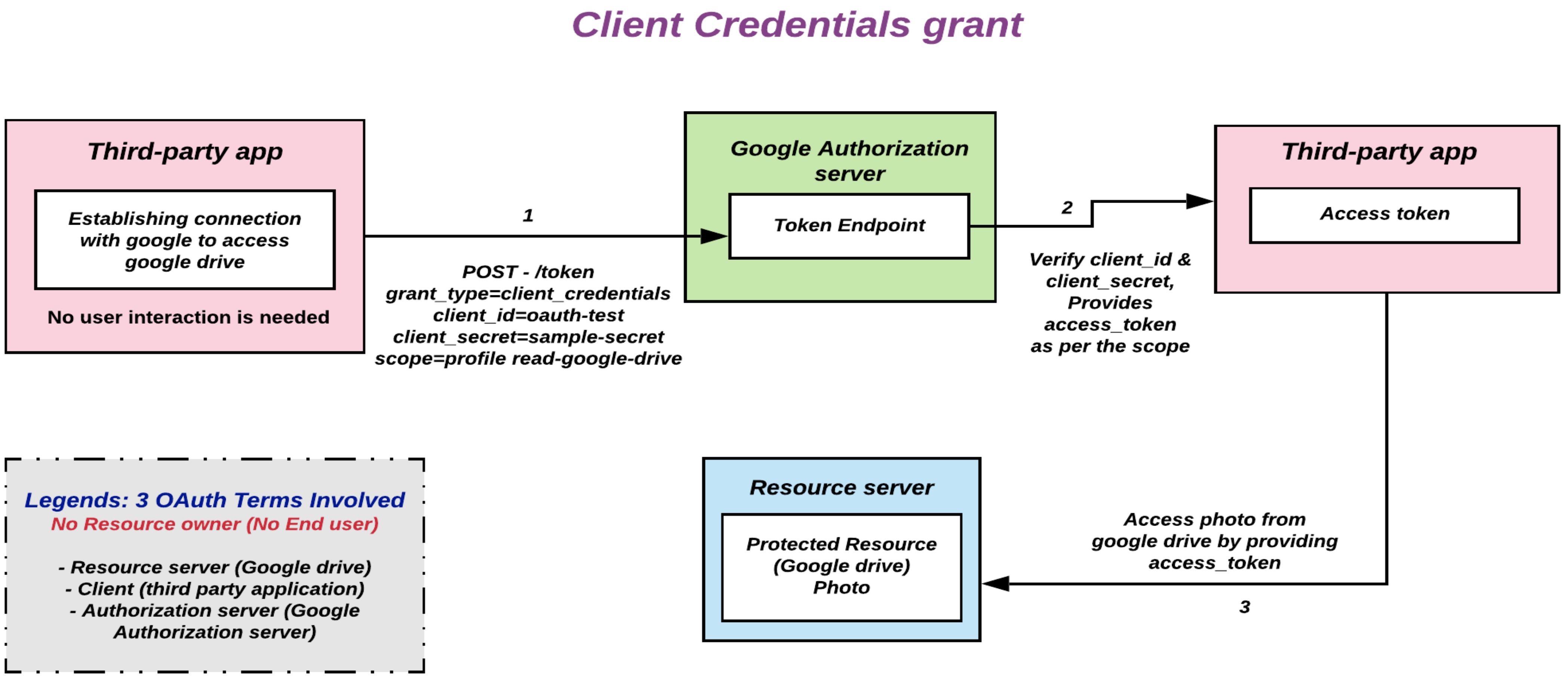


Implicit grant
Third-party app
Connect with google
Yes
No
click yes and allow us to access your photo's stored in user's google drive
Third-party app asks user to link to google service and user respond to it
User
1
App requests to auth endpoint
2
Google Authorization server
Authorization Endpoint
Directly access_token + state=1234
7
Third-party app
Access token via callback redirect_uri fragment
3
GET - /authorize
Redirect to Authorization page with
response_type=code&state=1234
client_id=oauth-test
redirect_uri=http://client.example.com
scope=profile read-google-drive
Access user's photo from google drive by providing access_token
8
Resource server
Protected Resource (Google drive) User photo
Google Login Page
Third party app requesting permission to below
1. Read profile
2. Read google drive
Approve?
Login Id
Password
Yes
No
App display's Authorization page to the user
4
5
User checks the requested permission, input username + password and approves authorization request
6
Authenticate username + password + verify user consent (Internal endpoints which is beyond the scope of OAuth)
Legends: 4 OAuth Terms Involved
- Resource owner (End user)
- Resource server (Google drive)
- Client (third party application)
- Authorization server (Google Authorization server)

Implicit Flow

- An access token is returned directly from the authorization request.
- It typically does not support refresh tokens.
- Since everything happens on the browser, it’s the most vulnerable to security threats.
- An SPA is a good example of this flow’s use case.

Client Credentials flow


Client Credentials grant
Third-party app
Establishing connection with google to access google drive
No user interaction is needed
1
POST - /token
grant_type=client_credentials
client_id=oauth-test
client_secret=sample-secret
scope=profile read-google-drive
Google Authorization server
Token Endpoint
2
Verify client_id & client_secret, Provides access_token as per the scope
Third-party app
Access token
Resource server
Protected Resource (Google drive) Photo
Access photo from google drive by providing access_token
3
Legends: 3 OAuth Terms Involved
No Resource owner (No End user)
- Resource server (Google drive)
- Client (third party application)
- Authorization server (Google Authorization server)

Client Credentials Flow

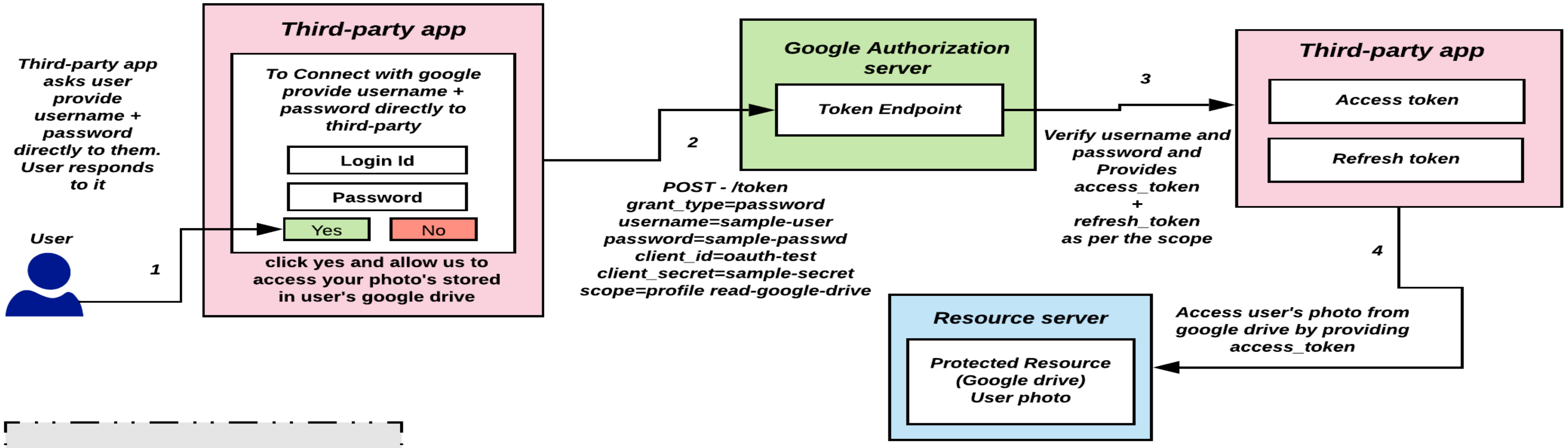
- For server-to-server scenarios, a Client Credential Flow is used
- In this scenario, the client application is a confidential client that's acting on its own.
- It's a back channel only flow to obtain an access token using the client's credentials.
- It supports shared secrets or assertions as client credentials
- Use Case: Machine-to-machine authorization where no end-user interaction is needed.
- Example: A cron job that imports data to a database using an API.
- How It Works: The client (e.g., the cron job) directly obtains an access token from the authorization server using its client ID and client secret.

Resource Owner Password Flow

- It’s a legacy grant type for native username/password apps like desktop applications.
- In this flow, you send the client application a username and password and it returns an access token from the Authorization Server.

Resource Owner Password Flow

Resource owner password credentials grant

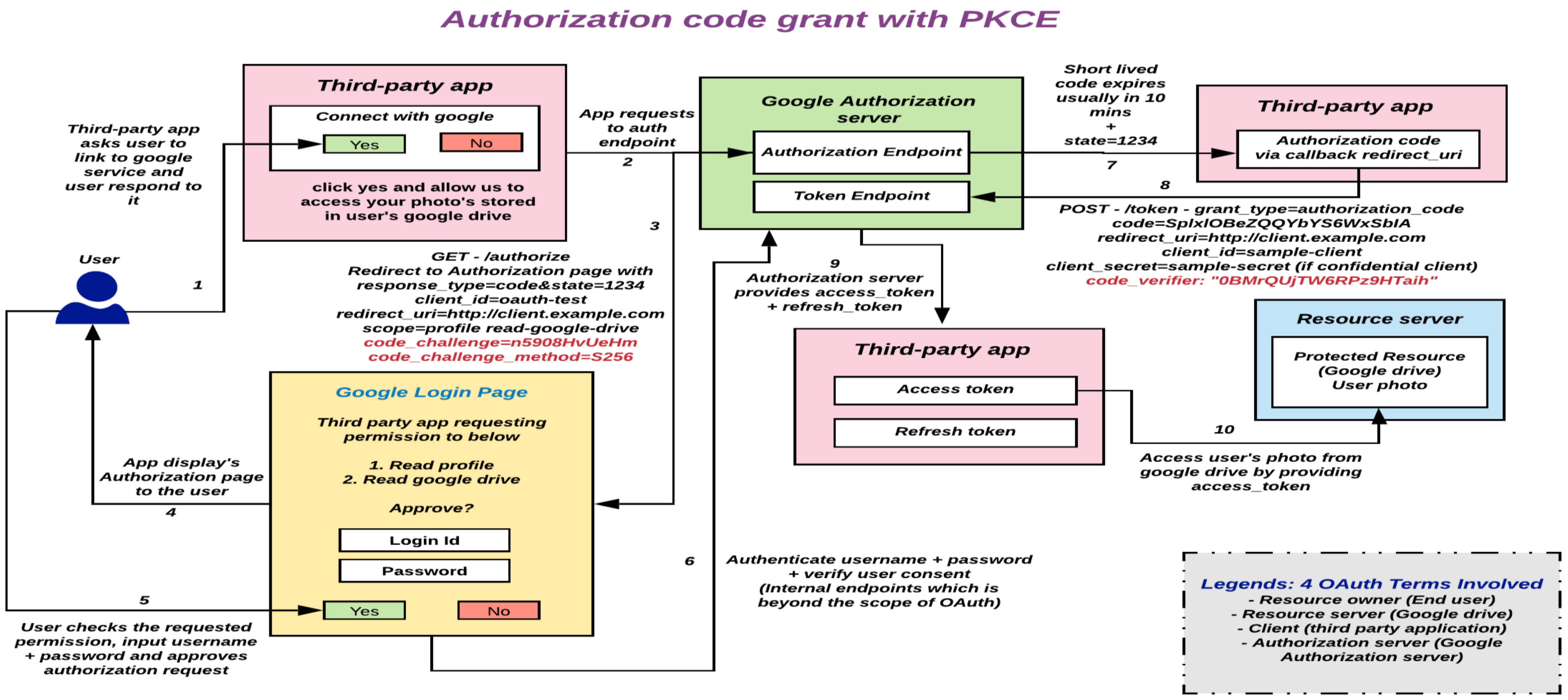

Third-party app asks user provide username + password directly to them. User responds to it
User
1
Third-party app
To Connect with google provide username + password directly to third-party
Login Id
Password
Yes
No
click yes and allow us to access your photo's stored in user's google drive
2
POST - /token
grant_type=password
username=sample-user
password=sample-passwd
client_id=oauth-test
client_secret=sample-secret
scope=profile read-google-drive
Google Authorization server
Token Endpoint
3
Verify username and password and Provides access_token + refresh_token as per the scope
Third-party app
Access token
Refresh token
4
Access user's photo from google drive by providing access_token
Resource server
Protected Resource (Google drive) User photo

Legends: 4 OAuth Terms Involved
- *Resource owner (End user)*
- *Resource server (Google drive)*
- *Client (third party application)*
- *Authorization server (Google Authorization server)*

Authorization code with PKCE

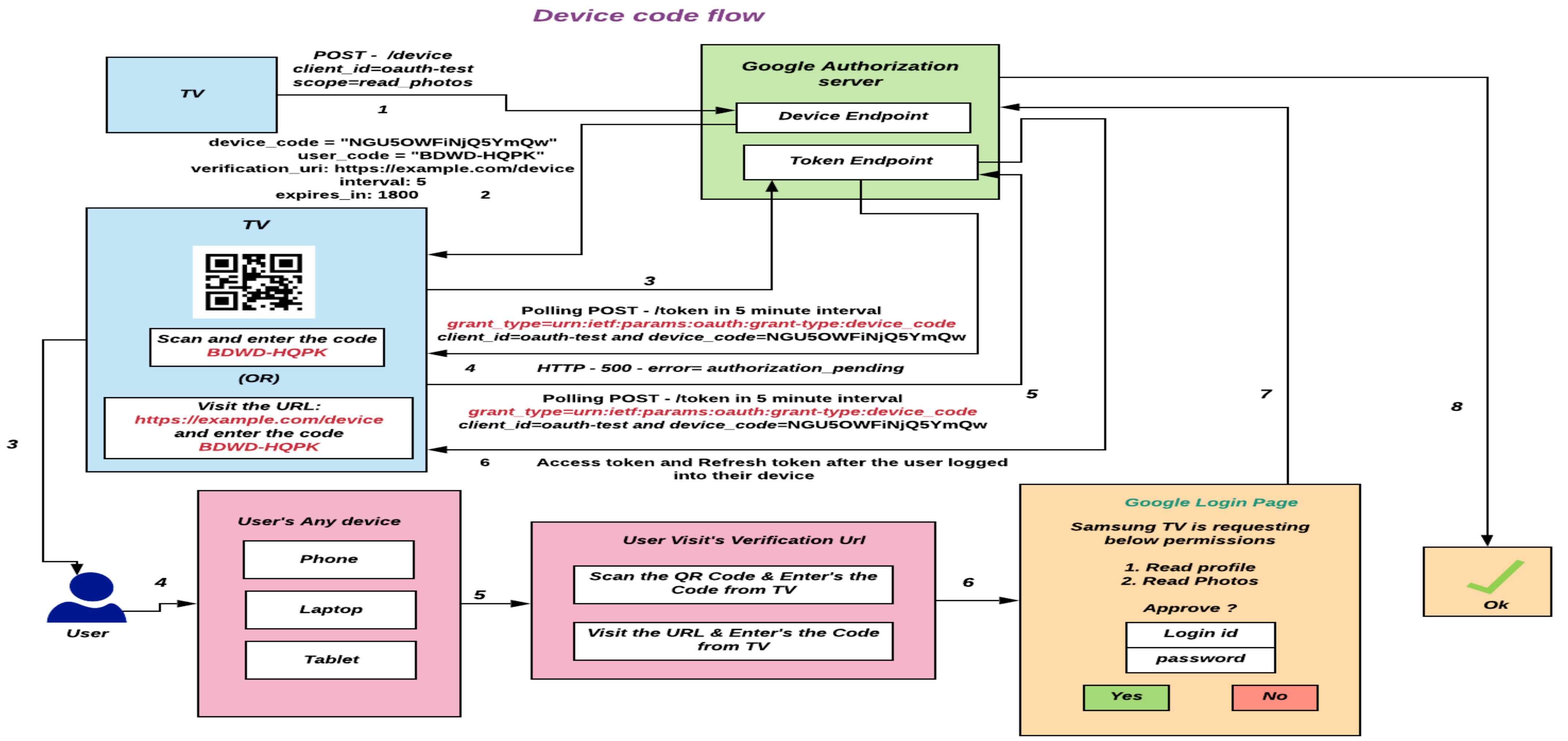
- This flow is an extension to Authorization grant flow.
- Authorization code grant is vulnerable to authorization code interception attacks when used with public clients
- Proof Key for Code Exchange(PKCE)

Authorization code with PKCE



Authorization code grant with PKCE
Third-party app
Connect with google
Yes
No
click yes and allow us to access your photo's stored in user's google drive
Third-party app asks user to link to google service and user respond to it
User
1
App requests to auth endpoint
2
Google Authorization server
Authorization Endpoint
Token Endpoint
Short lived code expires usually in 10 mins + state=1234
7
Third-party app
Authorization code via callback redirect_uri
8
POST - /token - grant_type=authorization_code
code=SplxlOBeZQQYbYS6WxSbIA
redirect_uri=http://client.example.com
client_id=sample-client
client_secret=sample-secret (if confidential client)
code_verifier: "0BMrQUjTW6RPz9HTaih"
3
GET - /authorize
Redirect to Authorization page with
response_type=code&state=1234
client_id=oauth-test
redirect_uri=http://client.example.com
scope=profile read-google-drive
code_challenge=n5908HvUeHm
code_challenge_method=S256
9
Authorization server provides access_token + refresh_token
Third-party app
Access token
Refresh token
Resource server
Protected Resource (Google drive) User photo
10
Access user's photo from google drive by providing access_token
Google Login Page
Third party app requesting permission to below
1. Read profile
2. Read google drive
Approve?
Login Id
Password
Yes
No
App display's Authorization page to the user
4
5
User checks the requested permission, input username + password and approves authorization request
6
Authenticate username + password + verify user consent (Internal endpoints which is beyond the scope of OAuth)
Legends: 4 OAuth Terms Involved
- Resource owner (End user)
- Resource server (Google drive)
- Client (third party application)
- Authorization server (Google Authorization server)

Device Code Flow


Device code flow
TV
POST - /device
client_id=oauth-test
scope=read_photos
1
Google Authorization server
Device Endpoint
Token Endpoint
device_code = "NGU5OWFiNjQ5YmQw"
user_code = "BDWD-HQPK"
verification_uri: https://example.com/device
interval: 5
expires_in: 1800
2
TV
Scan and enter the code
BDWD-HQPK
(OR)
Visit the URL:
https://example.com/device
and enter the code
BDWD-HQPK
3
Polling POST - /token in 5 minute interval
grant_type=urn:ietf:params:oauth:grant-type:device_code
client_id=oauth-test and device_code=NGU5OWFiNjQ5YmQw
4
HTTP - 500 - error= authorization_pending
5
Polling POST - /token in 5 minute interval
grant_type=urn:ietf:params:oauth:grant-type:device_code
client_id=oauth-test and device_code=NGU5OWFiNjQ5YmQw
7
8
6
Access token and Refresh token after the user logged into their device
3
User
4
User's Any device
Phone
Laptop
Tablet
5
User Visit's Verification Url
Scan the QR Code & Enter's the Code from TV
Visit the URL & Enter's the Code from TV
6
Google Login Page
Samsung TV is requesting below permissions
1. Read profile
2. Read Photos
Approve ?
Login id
password
Yes
No
Ok

pseudo-authentication using OAuth

- OAuth is an authorization protocol, rather than an authentication protocol.
- OAuth does not provide user’s information via an access token
- Access tokens are meant to be opaque.
- They’re meant for the API, they’re not designed to contain user information.
- Custom Hacks were used to fill this gap
- Using OAuth on its own as an authentication method may be referred to as pseudo-authentication

OpenID vs OAuth

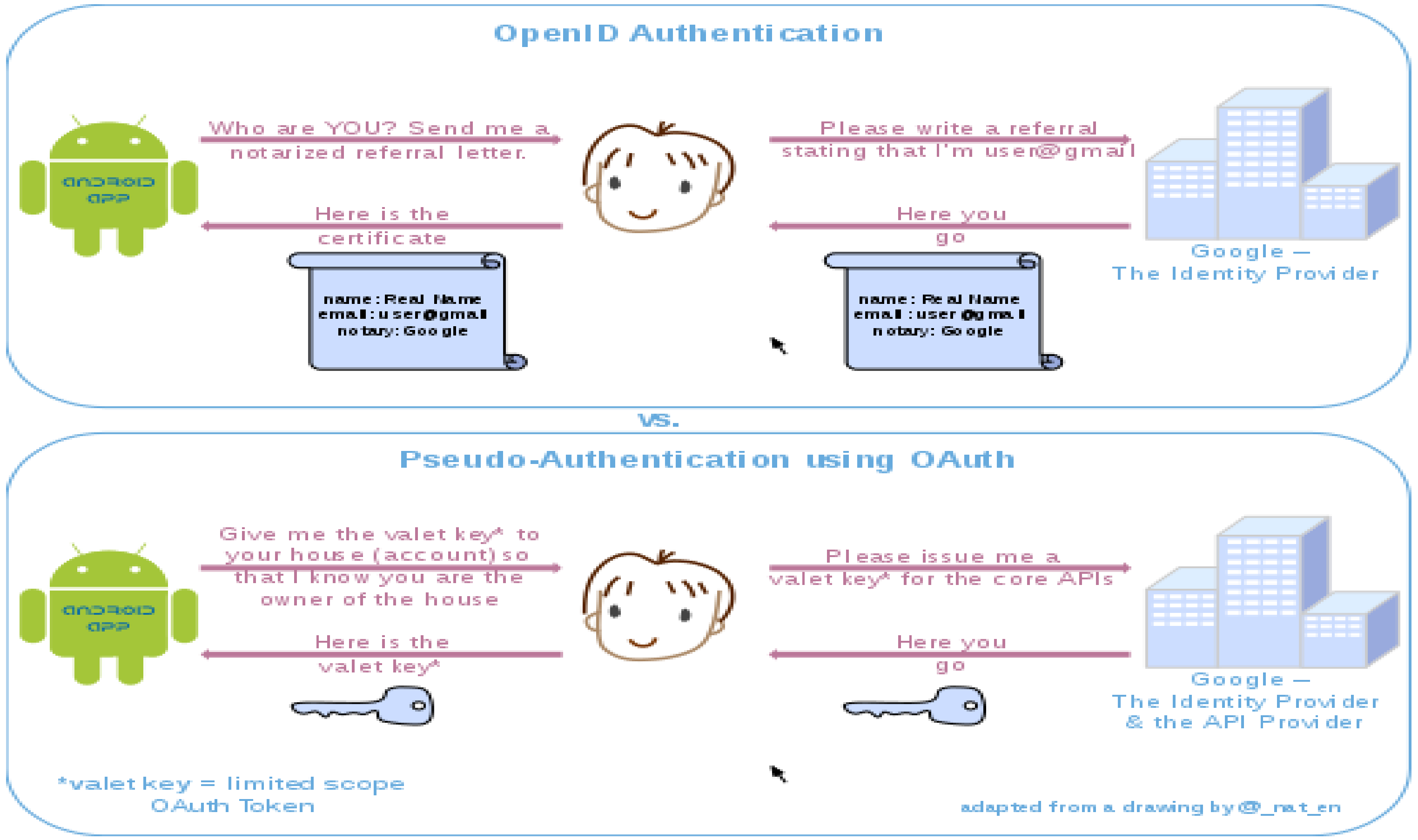

OpenID Authentication
ANDROID APP
Who are YOU? Send me a notarized referral letter.
Here is the certificate
name: Real Name
email: user@gmail
notary: Google
Please write a referral stating that I'm user@gmail
Here you go
name: Real Name
email: user@gmail
notary: Google
Google – The Identity Provider
vs.
Pseudo-Authentication using OAuth
ANDROID APP
Give me the valet key* to your house (account) so that I know you are the owner of the house
Here is the valet key*
Please issue me a valet key* for the core APIs
Here you go
Google – The Identity Provider & the API Provider
*valet key = limited scope OAuth Token
adapted from a drawing by @_nat_en

Image Reference: https://commons.wikimedia.org/wiki/File:OpenIDvs.Pseudo-AuthenticationusingOAuth.svg

OpenID Connect

- OAuth is directly related to OpenID Connect (OIDC).
- OIDC is an authentication layer built on top of OAuth 2.0.
- OpenID Connect (OIDC) extends OAuth 2.0 with a new signed id_token for the client and a UserInfo endpoint to fetch user attributes
- OpenID Connect is the standard for identity provision on the Internet.

OpenID Connect

- What it adds:
  - ID token
  - User endpoint to get more userinfo
  - Standardized
- Its formula for success: simple JSON-based identity tokens (JWT), delivered via OAuth 2.0 flows that fit web, browser-based and native / mobile applications.

References

- Demystifying OAuth 2.0 - A Tutorial & Primer :: Devansvd — Personal website
- https://blog.postman.com/pkce-oauth-how-to/
- https://auth0.com/docs/get-started/authentication-and-authorization-flow/which-oauth-2-0-flow-should-i-use

BITS Pilani
Pilani Campus

Thank you