PKCE-enhanced Authorization Code

PKCE - **P**roof **K**ey for **C**ode **E**xchange

User

Mobile App

Connect to API Service

Facebook, Google, Your API

API Service

Authorization Server

Authorization request

?**response_type**=code&**scope**={Scopes}&**state**={CSRF}&...

GET /authorize

?response_type=code // Required. Must be “code”

&state=hv8hf0h2i7d // Recommended

&redirect_uri= {Redirect uri} // Optional

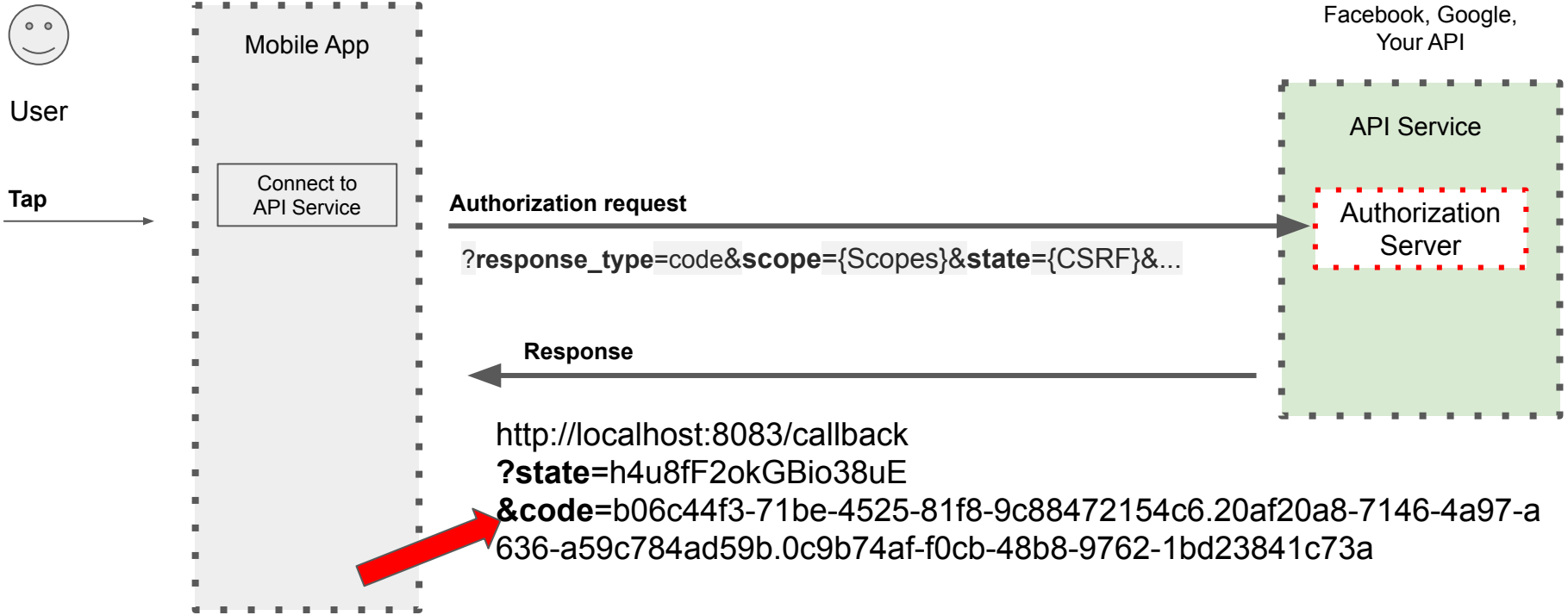
&scope={Scopes} // Optional

&client_id={Client Id} // Required

&code_challenge=-sUEoAV-txYvhniiuJ4-gwNCtsiD2XiIPvLQYm-sUsE

&code_challenge_method=S256 or “plain”

code_verifier = jWJS7olsI78LF-hcNHO1QBMqVX06iN5Z837vD6UXO3g



curl --location --request POST 'http://localhost:8080/auth/realms/appsdeveloperblog/protocol/openid-connect/token' **'code**=b06c44f3-71be-4525-81f8-9c88472154c6.20af20a8-7146-4a97-a636-a59c784ad59b.0c9b74af-f0cb-48b8-9762-1bd23841c73a' **'code_verifier**=c3cxd2UzNHJmZGUzNHJneWh1NzhpazFxd2U0cmZkZXI1NnI1N3lnZnJ0NmpraW85NHJkc3dlcg'

Request for an OAuth Code

```
curl --location --request GET
'http://localhost:8080/auth/realms/appsdeveloperblog/protocol/openid-connect/auth
?client_id=photo-app-pkce
&response_type=code
&scope=openid
&redirect_uri=http://localhost:8083/callback
&state=h4u8fF2okGBio38uE
&code_challenge=NDEyYjM0YzhkZTZhNWVlMzE3YWVjYmJkZWJiYTg4ZDFhMTIxNjQyMGQwZTU0NjE1NjlmZjMzNTg0NzkwODVlYQ
&code_challenge_method=S256'
```

Exchange OAuth Code for an Access Token

```
curl --location --request POST
'http://localhost:8080/auth/realms/appsdeveloperblog/protocol/
openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=authorization_code' \
--data-urlencode 'client_id=photo-app-pkce' \
--data-urlencode
'code=b06c44f3-71be-4525-81f8-9c88472154c6.20af20a8-7146-4a97-
a636-a59c784ad59b.0c9b74af-f0cb-48b8-9762-1bd23841c73a' \
--data-urlencode 'redirect_uri=http://localhost:8083/callback'
\
--data-urlencod
'code_verifier=c3cxd2UzNHJmZGUzNHJneWh1NzhpazFxd2U0cmZkZXI1Nnl
1N3lnZnJ0NmpraW85NHJkc3dlcg'
```

Generating
Code Verifier

4. Protocol

4.1. Client Creates a Code Verifier

The client first creates a code verifier, "code_verifier", for each OAuth 2.0 [RFC6749] Authorization Request, in the following manner:

code_verifier = high-entropy cryptographic random STRING using the unreserved characters [A-Z] / [a-z] / [0-9] / "-" / "." / "_" / "~" from Section 2.3 of [RFC3986], with a minimum length of 43 characters and a maximum length of 128 characters.

ABNF for "code_verifier" is as follows.

```
code-verifier = 43*128unreserved
unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"
ALPHA = %x41-5A / %x61-7A
DIGIT = %x30-39
```

NOTE: The code verifier SHOULD have enough entropy to make it impractical to guess the value. It is RECOMMENDED that the output of a suitable random number generator be used to create a 32-octet sequence. The octet sequence is then base64url-encoded to produce a 43-octet URL safe string to use as the code verifier.

```
package com.appsdeveloperblog.pkce;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.Base64;

public class PkceUtil {

    String generateCodeVerifier() throws UnsupportedEncodingException {
        SecureRandom secureRandom = new SecureRandom();
        byte[] codeVerifier = new byte[32];
        secureRandom.nextBytes(codeVerifier);
        return
Base64.getUrlEncoder().withoutPadding().encodeToString(codeVerifier);
    }

}
```

5GluDRih4mQPRoG4C4WylsHp0l--aBbOcwGO1MPEfLA

Generating
Code Challenge

4.2. Client Creates the Code Challenge

The client then creates a code challenge derived from the code verifier by using one of the following transformations on the code verifier:

plain
   code_challenge = code_verifier

S256 4 3 2 1
   code_challenge = BASE64URL-ENCODE(SHA256(ASCII(code_verifier)))

If the client is capable of using "S256", it MUST use "S256", as "S256" is Mandatory To Implement (MTI) on the server. Clients are permitted to use "plain" only if they cannot support "S256" for some technical reason and know via out-of-band configuration that the server supports "plain".

The plain transformation is for compatibility with existing deployments and for constrained environments that can't use the S256 transformation.

ABNF for "code_challenge" is as follows.

```
code-challenge = 43*128unreserved
unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"
ALPHA = %x41-5A / %x61-7A
DIGIT = %x30-39
```

```
    String generateCodeChallange(String codeVerifier) throws
UnsupportedEncodingException, NoSuchAlgorithmException {

        byte[] bytes = codeVerifier.getBytes("US-ASCII");
        MessageDigest messageDigest =
MessageDigest.getInstance("SHA-256");
        messageDigest.update(bytes, 0, bytes.length);
        byte[] digest = messageDigest.digest();

        return
Base64.getUrlEncoder().withoutPadding().encodeToString(digest);

    }
```

ZQNn8kfypf2X6j0HFdoApXgndA1LJr2-jm6kKJgsNDQ