The `code_challenge` is derived from the `code_verifier` using one of the two possible transformations: plain and S256. Plain can *only* be used when S256 is not possible. For the majority of use cases, the `code_challenge` will be a base 64 encoding of an SHA256 hash made with the `client_verifier`. This string gets decrypted server-side and is used to verify that the requests are coming from the same client.

The `code_challenge_method` tells the server which function was used to transform the `code_verifier` (plain or S256). It will default to plain if left empty.

These new parameters are used to supplement the authorization code flow to create a powerful system of checks that allow the server to verify that the authorization request and token request both come from the same client.

When a user kicks off a PKCE authorization flow in your app, here's what takes place:

1. Client (your app) creates the `code_verifier`. (RFC 7636, Section 4.1)
2. Client creates the `code_challenge` by transforming the `code_verifier` using S256 encryption. (RFC 7636, Section 4.2)
3. Client sends the `code_challenge` and `code_challenge_method` with the initial authorization request. (RFC 7636, Section 4.3)
4. Server responds with an `authorization_code`. (RFC 7636, Section 4.4)
5. Client sends `authorization_code` and `code_verifier` to the token endpoint. (RFC 7636, Section 4.5)
6. Server transforms the `code_verifier` using the `code_challenge_method` from the initial authorization request and checks the result against the `code_challenge`. If the value of both strings match, then the server has verified that the requests came from the same client and will issue an `access_token`. (RFC 7636, Section 4.6)

Now that we understand the flow, let's see what it looks like in practice.

3 Testing the PKCE Flow

In these samples, we're using Node.js to generate the dynamic strings and curl to send our requests to the Dropbox API. In production, the string generation and API requests would happen in the same app. You'll need a Dropbox app to follow along.

Dropbox.Tech

const crypto = require("crypto")

Step 1: Client creates `code_verifier` and subsequent `code_challenge`

Add the following snippet to your JavaScript file

Copy

```
const base64Encode = (str) => {
    return str.toString('base64')
        .replace(/\+/g, '-')
        .replace(/\//g, '_')
        .replace(/=/g, '');
}
const codeVerifier = base64Encode(crypto.randomBytes(32));
console.log(`Client generated code_verifier: ${codeVerifier}`)

const sha256 = (buffer) => {
    return crypto.createHash('sha256').update(buffer).digest();
}
const codeChallenge = base64Encode(sha256(codeVerifier));
console.log(`Client generated code_challenge: ${codeChallenge}`)
```

Run the script with `node <your_filename>`

The console will output the generated strings

```
Client generated code_verifier: kiNgBo0-r4GdQld6ShdPoxGq9SheI2m5moxtX-tFce4
Client generated code_challenge: lSEB3zK2TM-X38Baht80CvC4E_a5DnpCG52y5a7dQyk
```

Step 2: Client sends `code_challenge` and `code_challenge_method` to /oauth2/authorize

Manually assemble the authorization URL and replace the variables with your own information

https://www.dropbox.com/oauth2/authorize?client_id=<APP_KEY>&response_type=code&code_challenge=<CHALLENGE>&code_challenge_method=<METHOD>

Dropbox.Tech