How to do an Ersatz client exchange, aka fork a flow.

In OA4MP, you can designate an ersatz (or replacement) client which can take over an existing flow.

P = id of provisioner (starts the flopw)
E = id of ersatz client
AT-P, RT-P, IDT-P = access, refresh, ID token for provisioner
AT-E, RT-E, IDT-E = access, refresh, ID token for ersatz client

To start

Begin a standard the flow with P however you do it, getting the tokens AT-P, RT-P, IDT-P.

To fork, you need to use the token endpoint and the token exchange grant (details below).

Normally when you use the token exchange grant you can request a single token type.
This still works but requires 3 exchanges to get all 3 tokens for the fork.
This is a little clunky so OA4MP will return all 3 in the initial fork.

To do the fork

1. You send E's credentials and AT-P
2. You get back AT-E, RT-E, IDT-E
3. Do standard operations with E from this point forward

More details

This next section we give the outline of what goes over the wire. Since
tokens can be immense, it is very easy to get confused. These are all
in the body of the POST to the server.

It is not helped by the RFC 8693 specification! In that, the token is *always* denoted
as "access_token" and you need to look at the returned token type in the response.

Typical token exchange for P to get another IDT-P

REQUEST

(Header contains the credentials for P)

```
subject_token_type=urn:ietf:params:oauth:token-type:access_token&  | The type of token you send
subject_token=AT-P&                                                | The token itself
grant_type=urn:ietf:params:oauth:grant-type:token-exchange&        | grant type so the server knows
                                                                     how to handle request
```

```
requested_token_type=urn:ietf:params:oauth:token-type:id_token     | The type of token you want back
```

RESPONSE (JSON always!)

```
{
 "issued_token_type":"urn:ietf:params:oauth:token-type:id_token",    | The type of access_token
                                                                      |   element returned
        "token_type":"N_A",                                           | the type of an ID token
      "access_token":"IDT-P",                                         | An ID token. As per spec this
                                                                      |   must be done this way
        "expires_in":1800                                             | When (in seconds) the token
                                                                        expires.
 }
```

Typical exchange to fork

REQUEST

(Header contains the credentials for E!)

```
subject_token_type=urn:ietf:params:oauth:token-type:access_token&    | Token type you send
subject_token=AT-P&                                                   | The token itself
grant_type=urn:ietf:params:oauth:grant-type:token-exchange&           | grant type
requested_token_type=urn:ietf:params:oauth:token-type:access_token   | the type of token you want
back
```

RESPONSE

```
{
"issued_token_type":"urn:ietf:params:oauth:token-type:access_token", | Type of access_token element
                                                                      |   returned
       "token_type":"Bearer",                                         | Required type if issued type
                                                                      |   is access_token
    "refresh_token": "RT-E",                                          | The refresh token for E
     "access_token": "AT-E",                                          | The access token for E
         "id_token": "IDT-E"                                           | The ID token for E
}
```

Final notes

- There is that there is no limit on how often you can fork a flow, so every time you present E's credentials and AT-P youwill create a fork.
- You may do the fork sending any of AT-P, RT-P or IDT-P as the subject token. Best practices are to use the AT-P.
- You may send along scopes, resources, etc. -- anything that goes in a normal exchange and the resulting tokens will conform to these. This lets P provision for an complex lifecyle and delelgate to E only what it is allowed.
- In particular, E may only downscope (i.e., have more restrictive scopes) from P.

Being able to fork repeatedly is very useful for, e.g., having P start a flow with many permissions and forking flows to processes with much more limited scopes etc.