RFC 7523 and OA4MP

This is a quick note about how to get a client using RFC 7523 under OA4MP. The aim is to get a developer who needs this to work now up to speed.

RFC:https://www.rfc-editor.org/rfc/rfc7523

Executive Summary

Register your client with a public key and you can just send requests (as a signed JWT) directly for access tokens – no user involved. This is § 2.1 of the spec. This is far better than that client credential flow – none of the vulnerabilities – and OA4MP supports it right now.

You can alternately just use your public key in place of a client password . This is §2.2 of the spec.

OA4MP requires you get authorized as per §2.2 to make the request as per §2.1, so we need both of these. We may relax that in the future. May.

I'll stick to an annotated example of how to make the request to the token endpoint. The initial kickoff for the flow is RFC 7523, but the flow after that is identical to the device or auth code flow.

Registration with OA4MP

If you use the registration endpoint (oauth2/register) then there is a box for the public key. You put your public keys in JWK (JSON webkey) format. This can include multiple keys with identifiers (**`kid`**). You keep the private keys and sign your request JWTs with them. Note that you can use a single key, in which case no kid is required in the request, but if you have multiple, you must include it.

Authorization (§2.2)

A typical request to the token endpoint POSTs the following two things

```
 client_assertion_type=
   urn:ietf:params:oauth:client-assertion-type:jwt-bearer
client_assertion=signed auth JWT
```

The **`client_assertion_type`** is fixed and required. The "signed auth JWT" is a JWT that is signed with your private key. OA4MP uses the stored public key to verify.
Typical example is (line breaks added, signature truncated)

```
eyJraWQiOiJCMzNGODZBMzI3QTIzMkU5IiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ
.eyJhdWQiOiJodHRwczovL2xvY2FsaG9zdDo5NDQzL29hdXRoMi90b2tlbiIsImV4cCI6MTcxNzYyMjY1MC
wiaWF0IjoxNzE3NjIxNzUwLCJpc3MiOiJhdXRvLXRlc3Q6L29hdXRoL3JmYzkwNjgvcWRsIiwianRpIjoiY
XV0by10ZXN0Oi9vYXV0aC9yZmM5MDY4L3FkbC9yZmM3NTIzL0ZIZlRab042WHdwSjk5Y1ZaMGFETWJpTXg1
Z2xHTG1EY1hHSWtyTUM2UmMiInN1YiI6ImF1dG8tdGVzdDovb2F1dGgvcmZjOTA2OC9xZGwiLH0
.ukG2PD8_InSZ8cdFn1FHs_tSv_cpFUEIGMfv-0y2xXDNkk73NM1sIuBfgSLS_0v...
```

which has header

```
{
  "kid":"B33F86A327A232E9",
  "typ":"JWT",
  "alg":"RS256"
}
```

Key	Description
kid	The key identifier in the JWK used at registration.
typ	Type of payload. This is fixed at **JWT**
alg	The algorithm used. Required

Again, if you uploaded a single key, you don't need a kid. The payload decodes as

```
{
  "aud": "https://localhost:9443/oauth2/token",
  "exp": 1717622650,
  "iat": 1717621750,
  "iss": "auto-test:/oauth/rfc9068/qdl",
  "jti":
"auto-test:/oauth/rfc9068/qdl/rfc7523/FHfTZoN6XwpJ99cVZ0aDMbiMx5glGLmDcXGIkrMC6Rc"
  "sub": "auto-test:/oauth/rfc9068/qdl"
}
```

Notes

key	Req?	Description
aud	Y	The server address
exp	Y	Timestamp in seconds when this request expires
iat	N	Timestamp in seconds when this request was created
iss	Y	The client identifier
jti	N	An identifier created by the client. This is ignored by the server but is passed back at times.
sub	Y	The client identifier. Fixed!

Authorization Grant (§2.1)

Quick review: A standard authorization code flow sends a request, the user needs to authenticate, then the client gets a response which includes the authorization grant. This response is used at the token endpoint to get an access token.

RFC 7523 lets you simply write your own authorization grant and send it to the token endpoint. The trick, of course, is that your client has a trust relationship with the server (the public key) so we know it can only have come from a specific client and cannot be forged.

The request POSTs the following

```
grant_type=
```

```
   urn:ietf:params:oauth:grant-type:jwt-bearer
assertion=auth grant JWT
```

The **`grant_type`** is fixed and must be as given. The auth grant JWT is a JWT that is signed with your private key. OA4MP uses the stored public key to verify.
Typical example is (line breaks added, signature truncated)

```
eyJraWQiOiJCMzNGODZBMzI3QTIzMkU5IiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ
.eyJhdWQiOiJodHRwczovL2xvY2FsaG9zdDo5NDQzL29hdXRoMi90b2tlbiIsImV4cCI6MTcxNzYyMjY1MC
wiaWF0IjoxNzE3NjIxNzUwLCJpc3MiOiJhdXRvLXRlc3Q6L29hdXRoL3JmYzkwNjgvcWRsIiwianRpIjoiY
XV0by10ZXN0Oi9vYXV0aC9yZmM5MDY4L3FkbC9yZmM3NTIzL044Sk81VW9wN18tb2NSR1lfWHRocVRYenct
aVNWN0hRYmVEUGUza0FQbXMiIm5vbmNlIjoidXRyVmpoZ2ZGZ0V4NHlLYnRQNDNHU3B2YkZtbXBqWkR0eGN
Vbk9pMkFOayIsInJlc291cmNlIjoiQU5ZIiwic2NvcGUiOlsib3BlbmlkIiwib3JnLmNpbG9nb24udXNlcm
luZm8iXSwic3ViIjoiZGF2ZW5wb3J0In0
.orJxMrEbQa-Q7Mue...
```

The header decodes as

```
{
  "kid":"B33F86A327A232E9",
  "typ":"JWT",
  "alg":"RS256"
}
```

See above, since this is identical. The payload decodes as

```
{
  "aud": "https://localhost:9443/oauth2/token",
  "exp": 1717622650,
  "iat": 1717621750,
  "iss": "auto-test:/oauth/rfc9068/qdl",
  "jti": "auto-test:/oauth/rfc9068/qdl/rfc7523/N8JO5Uop7_-ocRGY_XthqTXzw-
iSV7HQbeDPe3kAPms"
  "nonce": "utrVjhgfFgEx4yKbtP43GSpvbFmmpjZDtxcUnOi2ANk",
  "resource": "ANY",
  "scope": ["openid","org.cilogon.userinfo"],
  "sub": "davenport"
}
```

Note that any parameters you can normally send in a request can be encoded here. Hence the scope and resource.

Key	Req?	Description
aud	Y	The address of the server
exp	Y	The expiration timestamp, in seconds
iat	N	The issued at timestamp, in seconds
iss	Y	The client identifier
jti	N	A client generated identifier
nonce	N	A nonce (one-time random string) a client may use to track requests
resource	N	OA4MP request parameter
scope	N	OA4MP request parameter

sub	Y	Usually the user name. This will be used as the subject of the identity token.

RFC7523 and the device flow (RFC 8628)

You may register your client’s key(s) and simply use them for authorization in the device code flow.

```
{
  "alg": "RS256",
  "kid": "EC9FCFCB3716AC4C2279DF42EC98CABF",
  "typ": "JWT"
}
```

The only slight change is to note that, as per the spec., the aud claim is the endpoint, so it must be the device code flow endpoint:

```
{
  "aud": "https://localhost:9443/oauth2/device_authorization",
  "exp": 1717709668,
  "iat": 1717708768,
  "iss": "ashigaru:command.line2",
  "jti":
"ashigaru:command.line2/rfc7523/SpPddDaTx6SFJ7Cd9x4DYbwhQ4RtaM6FWSLUU3KJJr4"
  "sub": "ashigaru:command.line2",
}
```