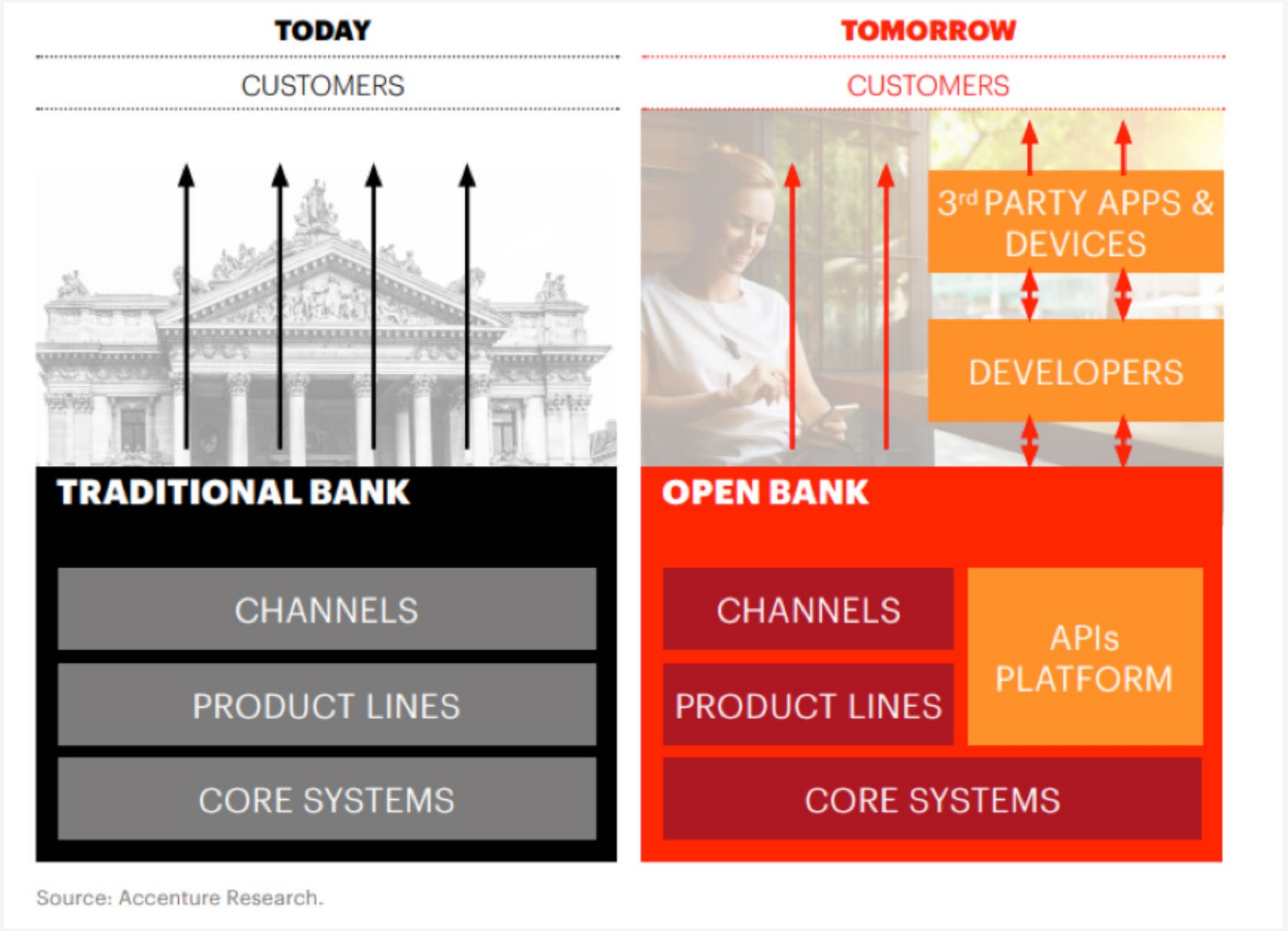
Financial Grade API

Vinod Kannan
Principal Consultant – Synopsys Inc
https://www.linkedin.com/in/vinod-kannan/

What is open banking?

- In the 2000s, screen scraping became a regular method for payment initiation services and third-party providers to process payments.
- In 2007, first payment Iban Services Directive (PSD) to open EU based banks to Fintech companies to open up banking sector.
- In 2018, second directive was passed which opened up the sector more formally and the banks were mandated to provide APIs and interfaces for third party.
- Financial Grade API(FAPI) will be an enabler critical for these API security requirements.
- In the United States and other markets too, financial organizations are looking for FAPI and Open banking model to process consumer data.

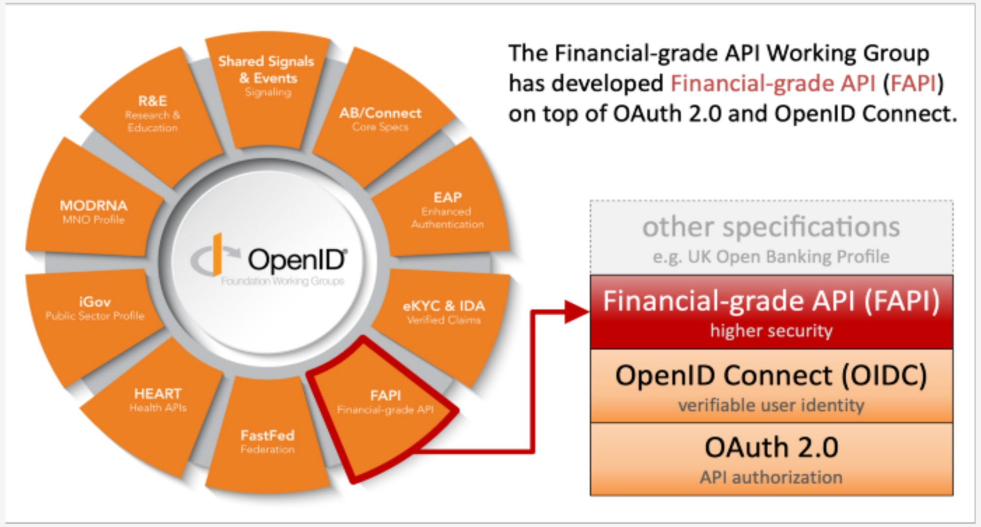
Source : https://fapi.openid.net/



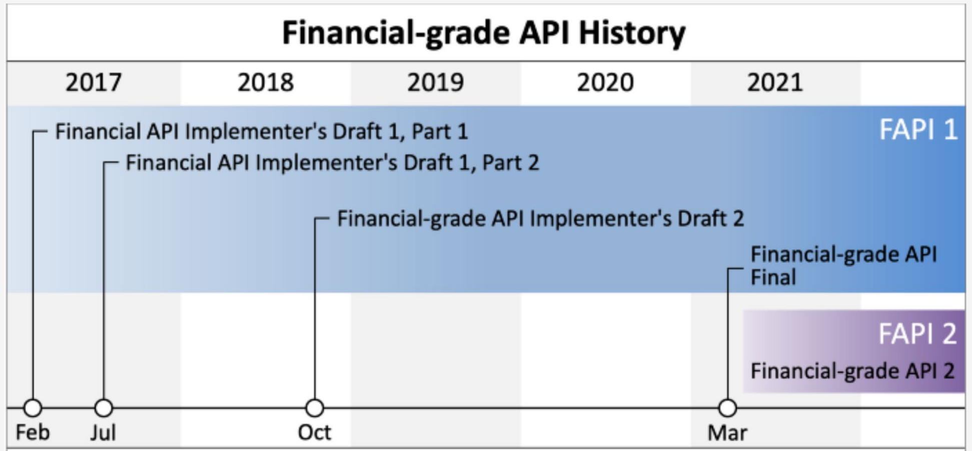
What is Financial – Grade API (FAPI)?

- FAPI is a security and interoperability profile for OAuth mainly intended to be used in the Open Banking scenario.
- FAPI is a technical specification the FAPI Working group of the Openid foundation has created.
- FAPI adds additional technical requirements to improve the security posture of OAuth/OIDC.
- This presentation speaks about FAPI 1.0 which has been operationalized , the 2.0 version is in the works.

Source : https://fapi.openid.net/



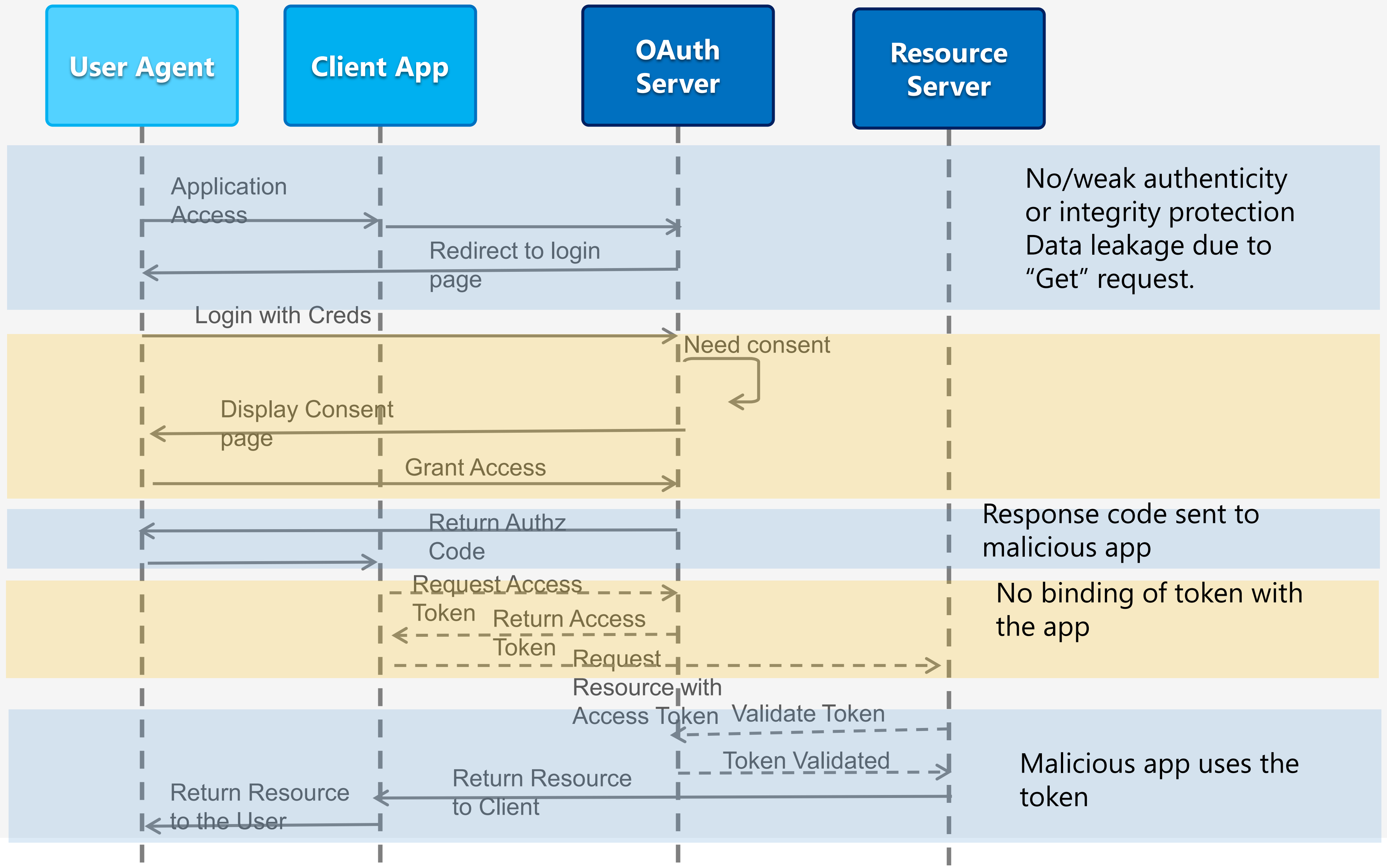
The Financial-grade API Working Group has developed Financial-grade API (FAPI) on top of OAuth 2.0 and OpenID Connect.



- "Financial API" was renamed to "Financial-grade API" by Implementer's Draft 2.
- "Part 1: Read-Only Security Profile" was renamed to "Part 1: Baseline Security Profile" by FInal.
- "Part 2: Read and Write API Security Profile" was renamed to "Part 2: Advanced Security Profile" by Final.

History of Financial-grade API

OAuth + OIDC flow


User Agent
Client App
OAuth Server
Resource Server
Application Access
Redirect to login page
Login with Creds
Need consent
Display Consent page
Grant Access
Return Authz Code
Request Access Token
Return Access Token
Request Resource with Access Token
Validate Token
Token Validated
Return Resource to Client
Return Resource to the User
No/weak authenticity or integrity protection Data leakage due to "Get" request.
Response code sent to malicious app
No binding of token with the app
Malicious app uses the token

FAPI standards

Part 1 : Baseline Profile Key requirements:

(Previously Read Only API Security Profile*)*

a) Client Authentication:
   Mutual TLS (tls_client_auth, self_signed_tls_client_auth)
   JWT (client_secret_jwt or private_key_jwt)

b) ID token as detached signature/JWT-based Response Mode

Part 2: Advanced Security Profile

(Previously Read & Write API Security Profile*)*

a) Client Authentication
   Mutual TLS (tls_client_auth, self_signed_tls_client_auth)
   JWT (private_key_jwt)

b) Use of signed JWT to bundle the request parameters

c) Mandating client applications to be the holder of the Key (Proof of possession)

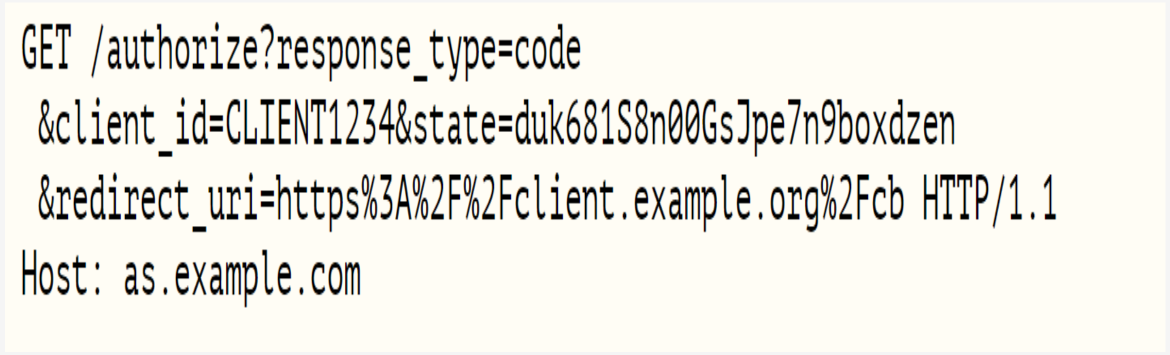
d) ID token as detached signature

3) JWT Secured Authorization Response Mode for OAuth 2.0 (JARM)

4) Client-Initiated Backchannel Authentication (CIBA) Profile

Pushed Authorization request

1. Issues with the OAuth Auth Request:
   a) Conventional OAuth request has no Cryptographic integrity and authenticity protection. An attacker could modify the scope or context of the request.
   b) There is no confidentiality of the request, though HTTPS is a requirement in basic OAuth, query string with sensitive data may be leaked to the logs of the UserAgent and other places.
   c) The browser-based URLs can be quite large, which may lead to request processing issues.



```
GET /authorize?response_type=code
 &client_id=CLIENT1234&state=duk681S8n00GsJpe7n9boxdzen
 &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1
Host: as.example.com
```

Ref: [RFC 9126 - OAuth 2.0 Pushed Authorization Requests (ietf.org)]

Pushed Authorization request

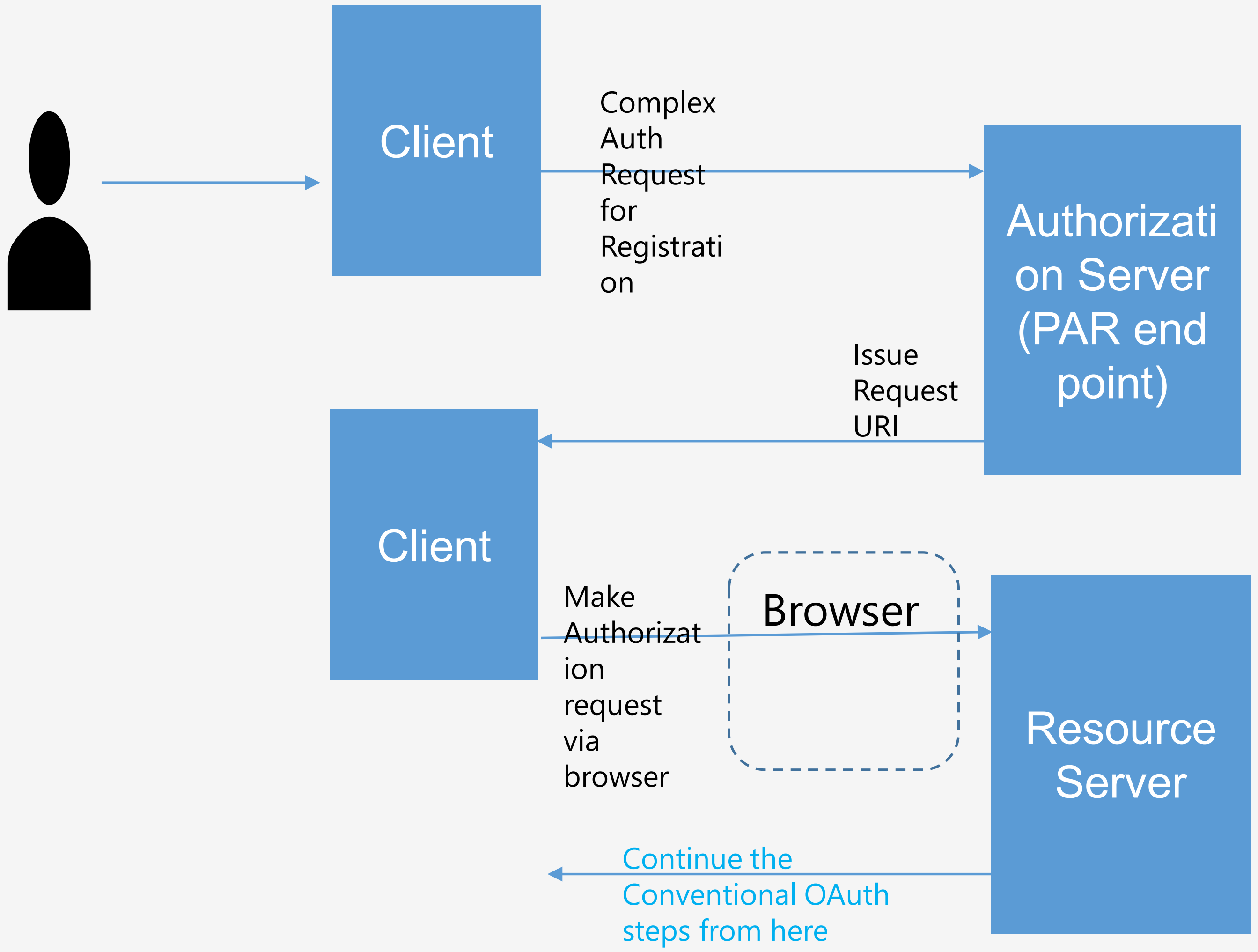
a) Initial Authorization request is pushed to the Authz Server's PAR end point.

b) Pushed authorization request endpoint is an HTTP API at the Authorization server that accepts POST with application/x- www-form-urlencoded" format

c) Request format: Client sends the request with the parameter set which may include- Client_id, response_type, redirect_uri, redirect_type, scope, state, etc. Initial Authorization request is pushed to the Authz Server's PAR end point.

d) The steps above comprise the registration process.

Ref: [RFC 9126 - OAuth 2.0 Pushed Authorization Requests (ietf.org)]


Client
Complex Auth Request for Registration
Authorization Server (PAR end point)
Issue Request URI
Client
Make Authorization request via browser
Browser
Resource Server
Continue the Conventional OAuth steps from here

Pushed Authorization request

e) The initial request can include the client authentication, shown here is the JWT based assertion. (This is from the conventional OAuth spec).

f) In response to the Authorization server issues the request_uri.

g) Client in turn specifies the request_uri in the authorization request through the browser.

```
 POST /as/par HTTP/1.1
 Host: as.example.com
 Content-Type: application/x-www-form-urlencoded

 &response_type=code
 &client_id=CLIENT1234&state=duk681S8n00GsJpe7n9boxdzen
 &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
 &client_assertion_type=
  urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
 &client_assertion=eyJraWQiOiI0MiIsImFsZyI6IkVTMjU2In0.eyJpc3MiOiJDTE
  lFTlQxMjM0Iiwic3ViIjoiQ0xJRU5UMTIzNCIsImF1ZCI6Imh0dHBzOi8vc2VydmVyL
  mV4YW1wbGUuY29tIiwiZXhwIjoxNjI1ODY4ODc4fQ.Igw8QrpAWRNPDGoWGRmJumLBM
  wbLjeIYwqWUu-ywgvvufl_0sQJftNs3bzjIrP0BV9rRG-3eI1Ksh0kQ1CwvzA

The authorization server responds with a request URI:

 HTTP/1.1 201 Created
 Cache-Control: no-cache, no-store
 Content-Type: application/json

 {
   "request_uri": "urn:example:bwc4JK-ESC0w8acc191e-Y1LTC2",
   "expires_in": 90
 }
```

Subsequent Client request

```
GET /authorize?client_id=CLIENT1234
 &request_uri=urn%3Aexample%3Abwc4JK-ESC0w8acc191e-Y1LTC2 HTTP/1.1
Host: as.example.com
```

Ref: RFC 9126 - OAuth 2.0 Pushed Authorization Requests (ietf.org)

JWT Authorization Request (JAR)

1. Request can be wrapped up as an object - JAR.
2. Request object either sent as a value in the request or as reference to the location of the object. If sent as a reference, the AuthZ accesses the location to pick the object.
3. Adds ability to send request parameters in Json Web Token with JWS (signing) or Encryption (JWE)
   a) Support Source authentication, integrity and confidentiality of the request.
   b) This also provides non-repudiation confirmation.

Request object construct

```
 {
  "iss": "s6BhdRkqt3",
  "aud": "https://server.example.com",
  "response_type": "code id_token",
  "client_id": "s6BhdRkqt3",
  "redirect_uri": "https://client.example.org/cb",
  "scope": "openid",
  "state": "af0ifjsldkj",
  "nonce": "n-0S6_WzA2Mj",
  "max_age": 86400
 }
```

Request Using the "request_uri" Request Parameter

```
https://server.example.com/authorize?
   client_id=s6BhdRkqt3
   &request_uri=https%3A%2F%2Ftfp.example.org%2Frequest.jwt
   %2FGkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
```

Authorization Server Fetches Request Object

```
GET /request.jwt/GkurKxf5T0Y-
mnPFCHqWOMiZi4VS138cQO_V7PZHAdM HTTP/1.1
Host: tfp.example.org
```

Ref: RFC 9101: The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR) (rfc-editor.org)

JWT Authorization Request (JAR)

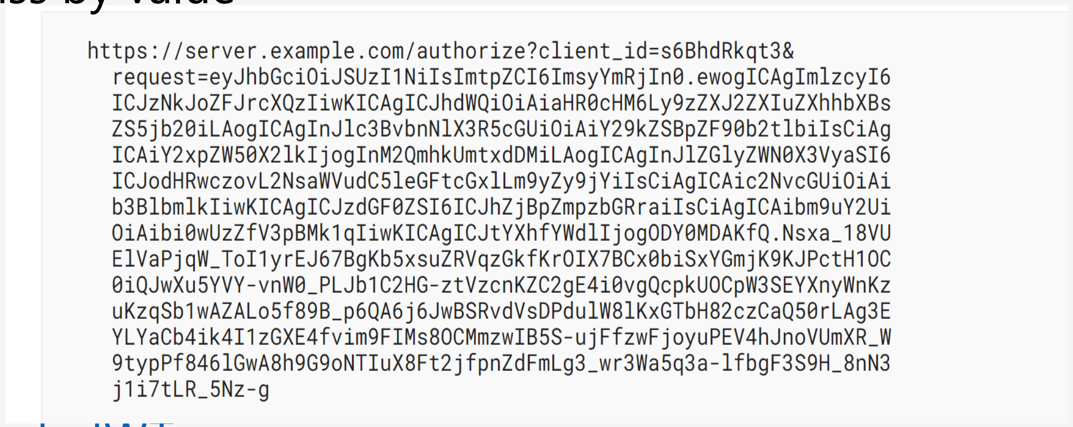
4. If the object is sent as a parameter (pass by value), it will be sent in the query string as shown.

AuthZ response

```
HTTP/1.1 200 OK
Date: Thu, 20 Aug 2020 23:52:39 GMT
Server: Apache/2.4.43 (tfp.example.org)
Content-type: application/OAuth-authz-req+jwt
Content-Length: 797
Last-Modified: Wed, 19 Aug 2020 23:52:32 GMT

eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ewogICAgImlzcyI6ICJzNkJoZF
JrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLAog
ICAgInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsCiAgICAiY2xpZW50X2
lkIjogInM2QmhkUmtxdDMiLAogICAgInJlZGlyZWN0X3VyaSI6ICJodHRwczovL2Ns
aWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAib3BlbmlkIiwKICAgIC
JzdGF0ZSI6ICJhZjBpZmpzbGRraiIsCiAgICAibm9uY2UiOiAibi0wUzZfV3pBMk1q
IiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VUElVaPjqW_ToI1yrEJ67BgK
b5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC0iQJwXu5YVY-vnW0_PLJb1C2
HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKzuKzqSb1wAZALo5f89B_p6QA6j6
JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3EYLYaCb4ik4I1zGXE4fvim9FIMs8O
CMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W9typPf846lGwA8h9G9oNTIuX8Ft2jf
pnZdFmLg3_wr3Wa5q3a-lfbgF3S9H_8nN3j1i7tLR_5Nz-g
```

Pass by value



```
https://server.example.com/authorize?client_id=s6BhdRkqt3&
  request=eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ewogICAgImlzcyI6
  ICJzNkJoZFJrcXQzIiwKICAgICJhdWQiOiAiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBs
  ZS5jb20iLAogICAgInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsCiAg
  ICAiY2xpZW50X2lkIjogInM2QmhkUmtxdDMiLAogICAgInJlZGlyZWN0X3VyaSI6
  ICJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsCiAgICAic2NvcGUiOiAi
  b3BlbmlkIiwKICAgICJzdGF0ZSI6ICJhZjBpZmpzbGRraiIsCiAgICAibm9uY2Ui
  OiAibi0wUzZfV3pBMk1qIiwKICAgICJtYXhfYWdlIjogODY0MDAKfQ.Nsxa_18VU
  ElVaPjqW_ToI1yrEJ67BgKb5xsuZRVqzGkfKrOIX7BCx0biSxYGmjK9KJPctH1OC
  0iQJwXu5YVY-vnW0_PLJb1C2HG-ztVzcnKZC2gE4i0vgQcpkUOCpW3SEYXnyWnKz
  uKzqSb1wAZALo5f89B_p6QA6j6JwBSRvdVsDPdulW8lKxGTbH82czCaQ50rLAg3E
  YLYaCb4ik4I1zGXE4fvim9FIMs8OCMmzwIB5S-ujFfzwFjoyuPEV4hJnoVUmXR_W
  9typPf846lGwA8h9G9oNTIuX8Ft2jfpnZdFmLg3_wr3Wa5q3a-lfbgF3S9H_8nN3
  j1i7tLR_5Nz-g
```

Ref: [RFC 9101: The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR) (rfc-editor.org)]

JWT Authorization Request (JAR) Request Object with PAR

Pushing a signed request object (JAR) to the PAR Endpoint. This ties up the request with the payload (JAR)

a) Authz server would decrypt the object.
b) Verify the signature
c) If the Authz has knowledge of the client_id, it would reject request if the client_id in the JWT does not match the actual id.

```
POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

client_assertion_type=
 urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
&client_assertion=eyJraWQiOiJrMmJkYyIsImFsZyI6IlJTMjU2In0.eyJpc3Mi
 OiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0dHBzOi8vc
 2VydmVyLmV4YW1wbGUuY29tIiwiZXhwIjoxNjI1ODY5Njc3fQ.te4IdnP_DK4hWrh
 TWA6fyhy3fxlAQZAhfA4lmzRdpoP5uZb-E90R5YxzN1YDA8mnVdpgj_Bx1lG5r6se
 f5TlckApA3hahhC804dcqlE4naEmLISmN1pds2WxTMOUzZY8aKKSDzNTDqhyTgE-K
 dTb3RafRj7tdZb09zWs7c_moOvfVcQIoy5zz1BvLQKW1Y8JsYvdpu2AvpxRPbcP8W
 yeW9B6PL6_fy3pXYKG3e-qUcvPa9kan-mo9EoSgt-YTDQjK1nZMdXIqTluK9caVJE
 RWW0fD1Y11_tlOcJn-ya7v7d8YmFyJpkhZfm8x1FoeH0djEicXTixEkdRuzsgUCm6
 GQ
&request=eyJraWQiOiJrMmJkYyIsImFsZyI6IlJTMjU2In0.eyJpc3MiOiJzNkJoZ
 FJrcXQzIiwiYXVkIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJleHAiOj
 E2MjU4Njk2NzcsInJlc3BvbnNlX3R5cGUiOiJjb2RlIiwiY2xpZW50X2lkIjoiczZ
 CaGRSa3F0MyIsInJlZGlyZWN0X3VyaSI6Imh0dHBzOi8vY2xpZW50LmV4YW1wbGUu
 b3JnL2NiIiwic2NvcGUiOiJhY2NvdW50LWluZm9ybWF0aW9uIiwic3RhdGUiOiJhZ
 jBpZmpzbGRraiIsImNvZGVfY2hhbGxlbmdlIjoiSzItbHRjODNhY2M0aDBjOXc2RV
 NDX3JFTVRKM2J3dy11Q0hhb2VLMXQ4VSIsImNvZGVfY2hhbGxlbmdlX21ldGhvZCI
 6IlMyNTYifQ.l9R3RC9bFBHry_8acObQjEf4fX5yfJkWUPfak3J3iiBm0aaQznPw5
 BZ0B3VQZ9_KYdPt5bTkaflS5fSDklM3_7my9MyOSKFYmf46INk6ju_qUuC2crkOQX
 ZWYJB-0bnYEbdHpUjazFSUvN49cEGstNQeE-dKDWHNgEojgcuNA_pjKfL9VYp1dEA
 6-WjXZ_OlJ7R_mBWpjFAzc0UkQwqX5hfOJoGTqB2tE4a4aB2z8iYlUJp0DeeYp_hP
 N6svtmdvte73p5bLGDFpRIlmrBQIAQuxiS0skORpXlS0cBcgHimXVnXOJG7E-A_lS
 _5y54dVLQPA1jKYx-fxbYSG7dp2fw
&client_id=s6BhdRkqt3
```

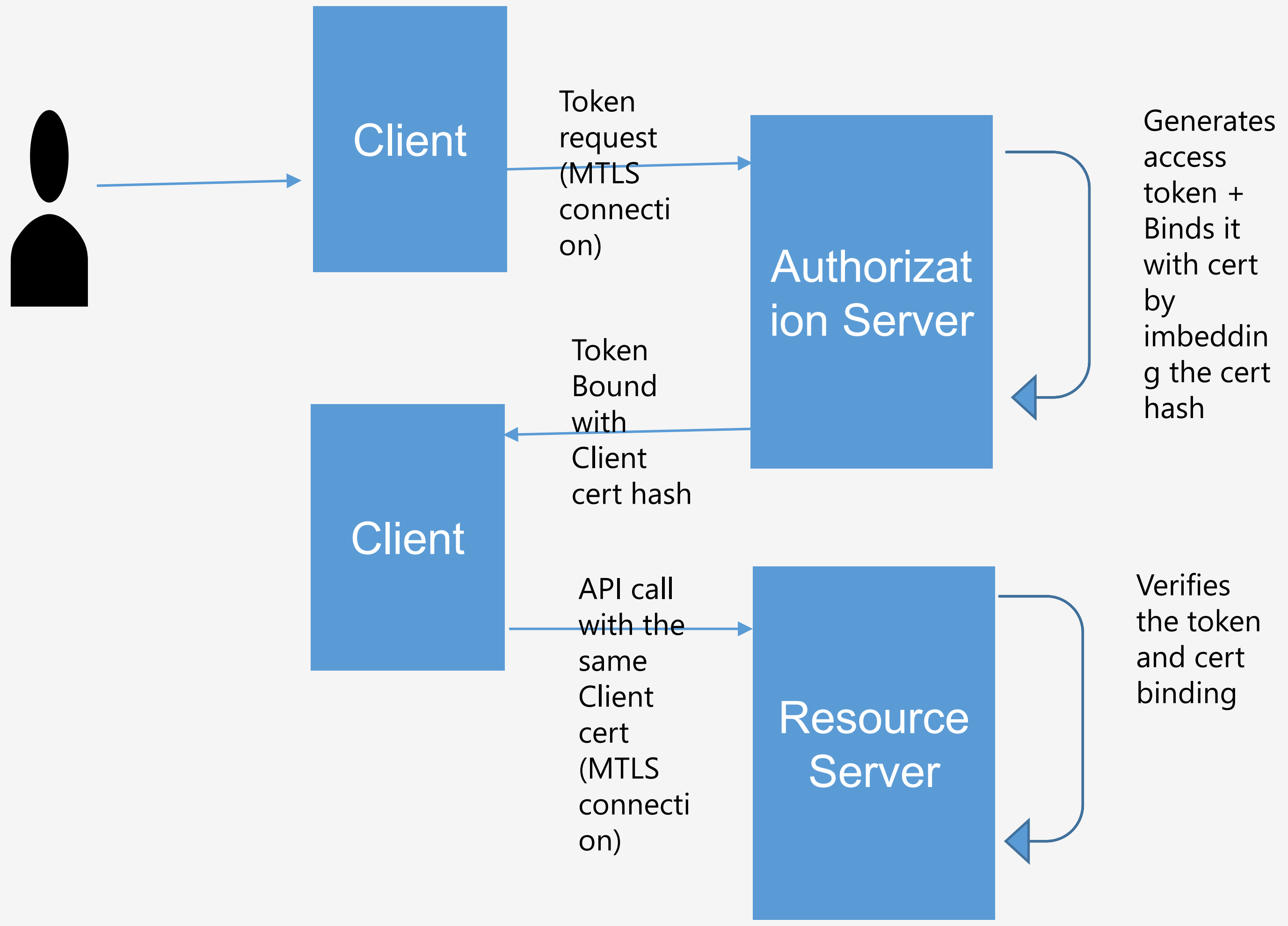
[RFC 9126 - OAuth 2.0 Pushed Authorization Requests (ietf.org)]

Client Authentication with Certificate bound Tokens

1.OIDC core client defines various Authentication methods continues to be used with FAPI:

a) client_secret_basic
b) client_secret_post
c) client_secret_jwt
d) private_key_jwt
e) tls_client_auth
f) self_signed_tls_client_auth

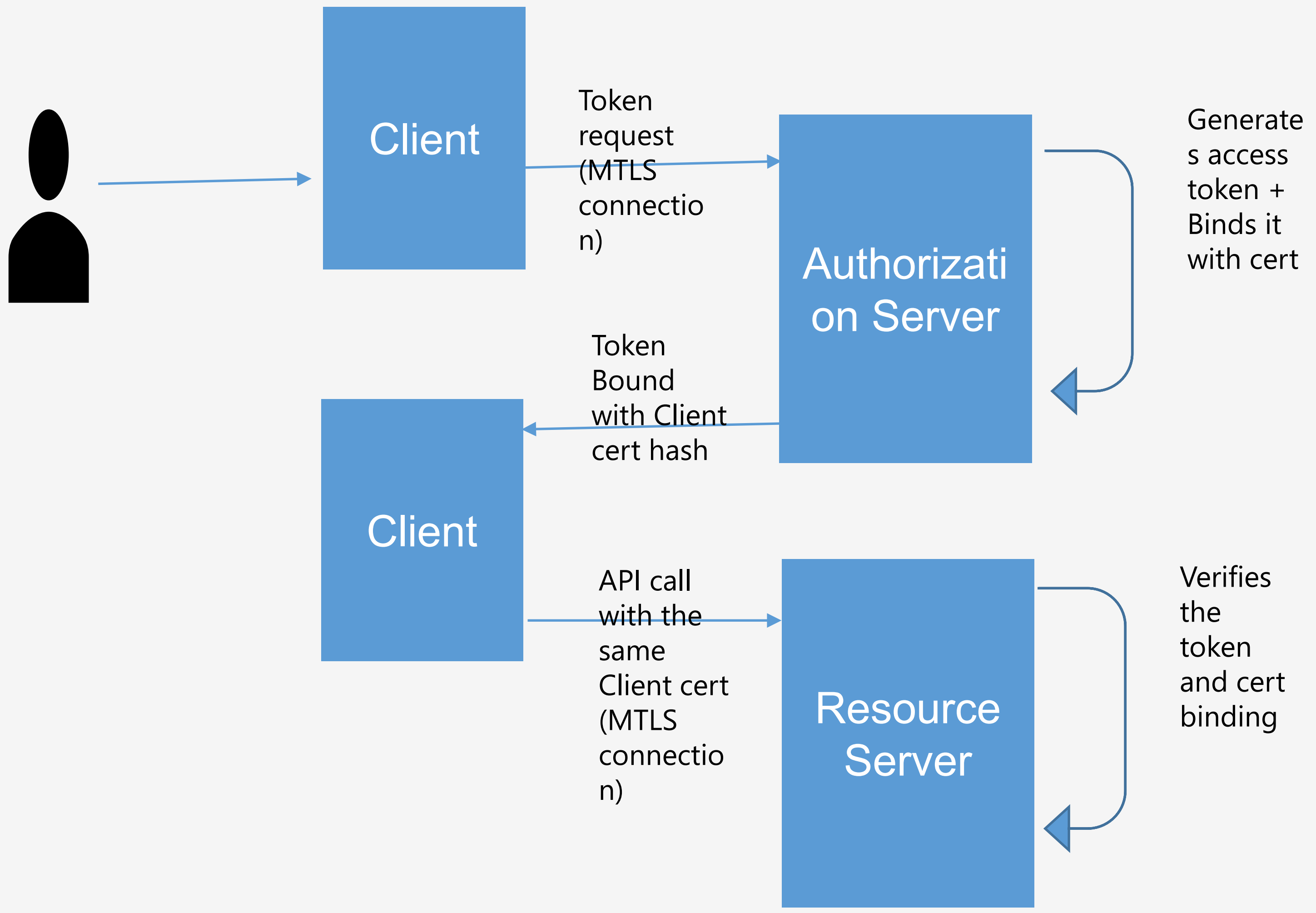
2.There is nothing specified to attach ownership of a specific token to the client.
3.Once the access token gets leaked, an attacker in possession of the token can access the resource.


Client
Token request (MTLS connection)
Authorization Server
Generates access token + Binds it with cert by imbedding the cert hash
Token Bound with Client cert hash
Client
API call with the same Client cert (MTLS connection)
Resource Server
Verifies the token and cert binding

Client Authentication with Authz endpoint for token

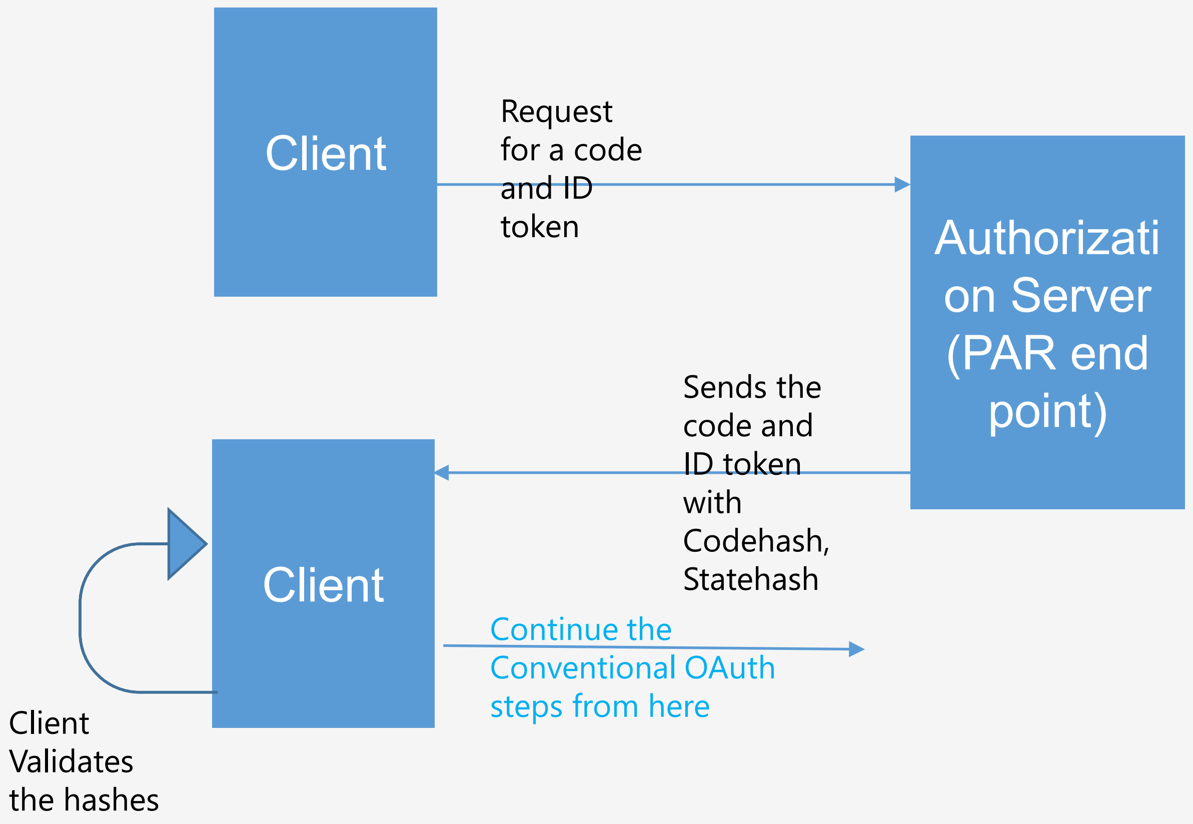
4. FAPI addresses this issue by adding something unique to the client to the token and brings in the concept of "Proof of possession (PoP)". Hash of the client cert is imbedded into the token, which the resource server could verify later on.

4. Current version of FAPI only identifies Mutual TLS (MTLS) as the only mechanism for PoP.


Client
Token request (MTLS connection)
Authorization Server
Generates access token + Binds it with cert
Token Bound with Client cert hash
Client
API call with the same Client cert (MTLS connection)
Resource Server
Verifies the token and cert binding

ID token as Detached Signature

1. Aimed at securing the response from Authz server back to the client
2. When the client requests a code and ID token; Authz server uses the ID token as a detached signature for the code :
   a) Client sends the request for Code alongside State other request parameters.
   b) The AuthZ server generates the Code and the ID token, it also creates the hash of the code and the "State" value originally sent by the client and adds to the id token.
   c) On receiving the response from AuthZ server, the client can validate the hash and confirm the authenticity of the server if the state value matches.


Client
Request for a code and ID token
Authorization Server (PAR end point)
Sends the code and ID token with Codehash, Statehash
Client
Continue the Conventional OAuth steps from here
Client Validates the hashes

State hash value. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the state value, where the hash algorithm used is the hash algorithm used in the alg Header Parameter of the ID Token's JOSE Header. For instance, if the alg is HS512, hash the code value with SHA-512, then take the left-most 256 bits and base64url encode them. The s_hash value is a case sensitive string.

JWT-based Response Mode

1. Aimed at securing the response from Authz server back to client when the response does not include an ID token :
   a) A new JWT definition has been put in place for this purpose.
   b) In JARM, the Authz server packs the parameters like iss,aud,code and state into a signed JWT token and sends to the client.

2. Response Types:
   a) Response Type "code" – JWT will contain
      - Code – the auth code
      - State – if a state value was sent by the client

   b) Response Type "token" – JWT will contain
      - access_token – the access token
      - Token_type – the type of the token
      - Expires_in- access token expiry
      - Scope – scope granted to the access token
      - State – if the client sent a state value in the request.

Conventional OAuth Response for Code:
http://example.com?code=xxxxx

"Code" response with JARM:

```
{
  "iss":"https://accounts.example.com",
  "aud":"s6BhdRkqt3",
  "exp":1311281970,
  "code":"PyyFaux2o7Q0YfXBU32jhw.5FXSQpvr8akv9CeRDSd0QA",
  "state":"S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw"
}
```

Conventional OAuth Token response:

```
{
     "access_token":"2YotnFZFEjr1zCsicMWpAA",
     "token_type":"access",
     "expires_in":3600,
}
```

"Token" response with JARM:

```
{
 "iss":"https://accounts.example.com",
   "aud":"s6BhdRkqt3",
   "exp":1311281970,
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "state":"S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw",
   "token_type":"bearer",
   "expires_in":3600,
   "scope":"example"
}
```

openid / fapi / Financial_API_JWT_Secured_Authorization_Response_Mode.md — Bitbucket

JWT-based Response Mode

3. The JWT can be signed with the Server’s private key the client can verify with its public key (for JWS). If its JWE, the JWT is signed and encrypted too

4. Response Encoding:
   a) query.jwt – the response JWT will be sent as part of the query string
   b) fragment.jwt – the response JWT will be sent as part of the fragment part of the URL
   c) form_post.jwt – the response will contain an auto-submitted HTML form response JWT will be in a hidden field
   d) Jwt – this will be used as the default response mode for the response type (query.jwt for “code” response type and fragment.jwt for “token” response type)

[openid / fapi / Financial_API_JWT_Secured_Authorization_Response_Mode.md — Bitbucket]()

“query.jwt” mode :

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?
response=eyJraWQiOiJsYWViIiwiYWxnIjoiRVMyNTYifQ.eyAgImlzcyI6ICJodHRwcz
ovL2FjY291
bnRzLmV4YW1wbGUuY29tIiwgICJhdWQiOiAiczZCaGRSa3F0MyIsICAiZXhwIjogM
TMxMTI4MTk3MCwg
ICJjb2RlIjogIlB5eUZhdXgybzdRMFlmWEJVMzJqaHcuNUZYU1FwdnI4YWt2OUNlU
kRTZDBRQSIsICAi
c3RhdGUiOiAiUzhOSjd1cWs1Zlk0RWpOdlBfR19GdHlKdTZwVXN2SDlqc1luaTlkTU
FKdyJ9.4VdtknV
Z9zFYDVLagJpVBD436bjPMcSgOaPDPFgTEkNyCs2uIHYJ2XML6d2w1AUsm5GBG
77DBisZNhLWfug6dA
```

“fragment.jwt” response type:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#
response=eyJraWQiOiJsYWViIiwiYWxnIjoiRVMyNTYifQ.eyAgImlzcyI6ICJodHRwc
zovL2FjY291
bnRzLmV4YW1wbGUuY29tIiwgICJhdWQiOiAiczZCaGRSa3F0MyIsICAiZXhwIjog
MTMxMTI4MTk3MCwg
ICJhY2Nlc3NfdG9rZW4iOiAiMllvdG5GWkZFanIxekNzaWNNV3BBQSIsICAic3Rhd
GUiOiAiUzhOSjd1
cWs1Zlk0RWpOdlBfR19GdHlKdTZwVXN2SDlqc1luaTlkTUFKdyIsICAidG9rZW5fd
HlwZSI6ICJiZWFy
ZXIiLCAgImV4cGlyZXNfaW4iOiAzNjAwLCAgInNjb3BlIjogImV4YW1wbGUifQ.g_9
6IM2t_6Dazm1Jp
b2gbO2EXe5IKTw2bYS7I9Y1RI8HbNPYN5EdNjxcWeL5LTQaUAZ2PtJoHbTdjMvN
a3xbVg
```

JWT-based Response Mode

Form_post.jwt

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
Cache-Control: no-cache, no-store
Pragma: no-cache

<html>
 <head><title>Submit This Form</title></head>
 <body onload="javascript:document.forms[0].submit()">
  <form method="post" action="https://client.example.com/cb">
    <input type="hidden" name="response"
     value="eyJraWQiOiJsYWViIiwiYWxnIjoiRVMyNTYifQ.eyAgImlzcyI6ICJodHRw
     czovL2FjY291bnRzLmV4YW1wbGUuY29tIiwgICJhdWQiOiAiczZCaGRSa3F0MyIsIC
     AiZXhwIjogMTMxMTI4MTk3MCwgICJhY2Nlc3NfdG9rZW4iOiAiMllvdG5GWkZFanIx
     ekNzaWNNV3BBQSIsICAic3RhdGUiOiAiUzhOSjd1cWs1Zlk0RWpOdlBfR19GdHlKdT
     ZwVXN2SDlqc1luaTlkTUFKdyIsICAidG9rZW5fdHlwZSI6ICJiZWFyZXIiLCAgImV4
     cGlyZXNfaW4iOiAzNjAwLCAgInNjb3BlIjogImV4YW1wbGUifQ.g_96IM2t_6Dazm1
     Jpb2gbO2EXe5IKTw2bYS7l9Y1RI8HbNPYN5EdNjxcWeL5LTQaUAZ2PtJoHbTdjMvNa
     3xbVg"/>
    </form>
   </body>
  </html>
```

Generates a request as below:

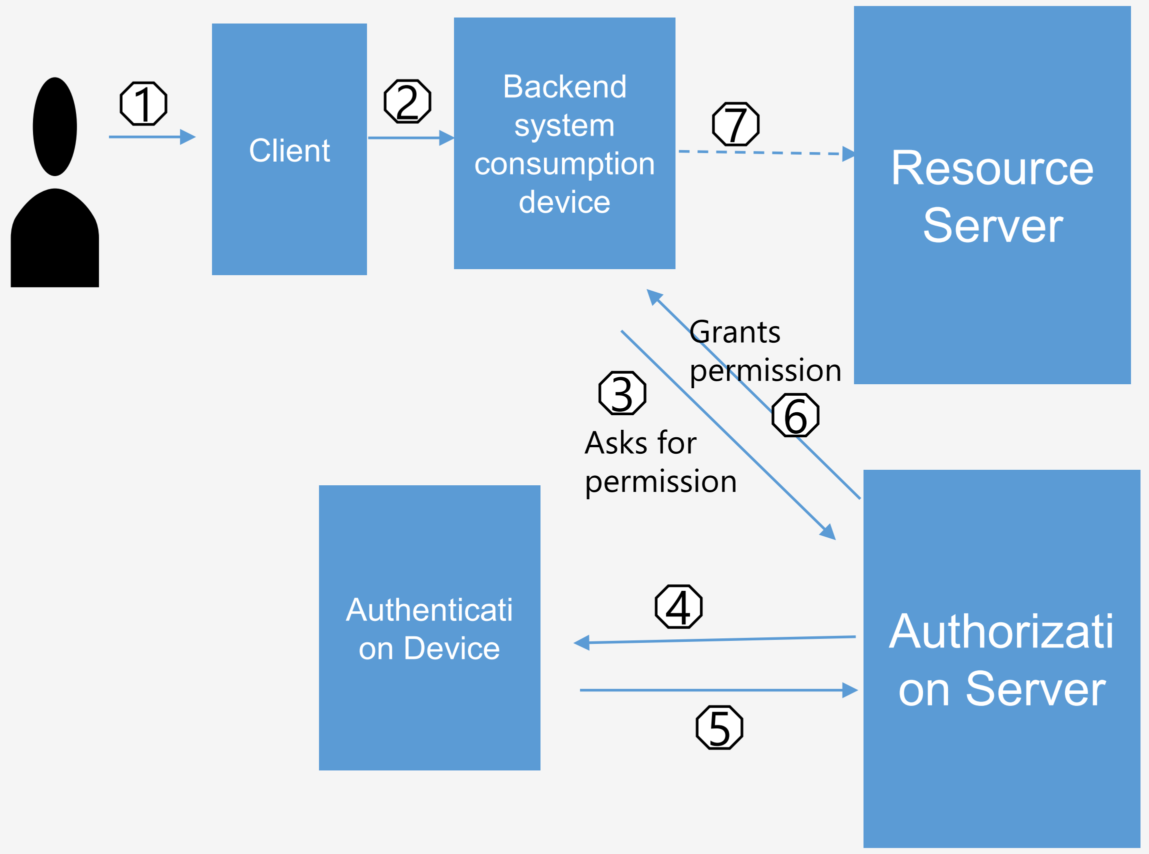
```
POST /cb HTTP/1.1
  Host: client.example.org
  Content-Type: application/x-www-form-urlencoded

   response=eyJraWQiOiJsYWViIiwiYWxnIjoiRVMyNTYifQ.eyAgImlzcyI6ICJodHRw
   czovL2FjY291bnRzLmV4YW1wbGUuY29tIiwgICJhdWQiOiAiczZCaGRSa3F0MyIsICAi
   ZXhwIjogMTMxMTI4MTk3MCwgICJhY2Nlc3NfdG9rZW4iOiAiMllvdG5GWkZFanIxekNz
   aWNNV3BBQSIsICAic3RhdGUiOiAiUzhOSjd1cWs1Zlk0RWpOdlBfR19GdHlKdTZwVXN2
   SDlqc1luaTlkTUFKdyIsICAidG9rZW5fdHlwZSI6ICJiZWFyZXIiLCAgImV4cGlyZXNf
   aW4iOiAzNjAwLCAgInNjb3BlIjogImV4YW1wbGUifQ.g_96IM2t_6Dazm1Jpb2gbO2EX
   e5IKTw2bYS7l9Y1RI8HbNPYN5EdNjxcWeL5LTQaUAZ2PtJoHbTdjMvNa3xbVg
```

Ref: openid / fapi / Financial_API_JWT_Secured_Authorization_Response_Mode.md — Bitbucket

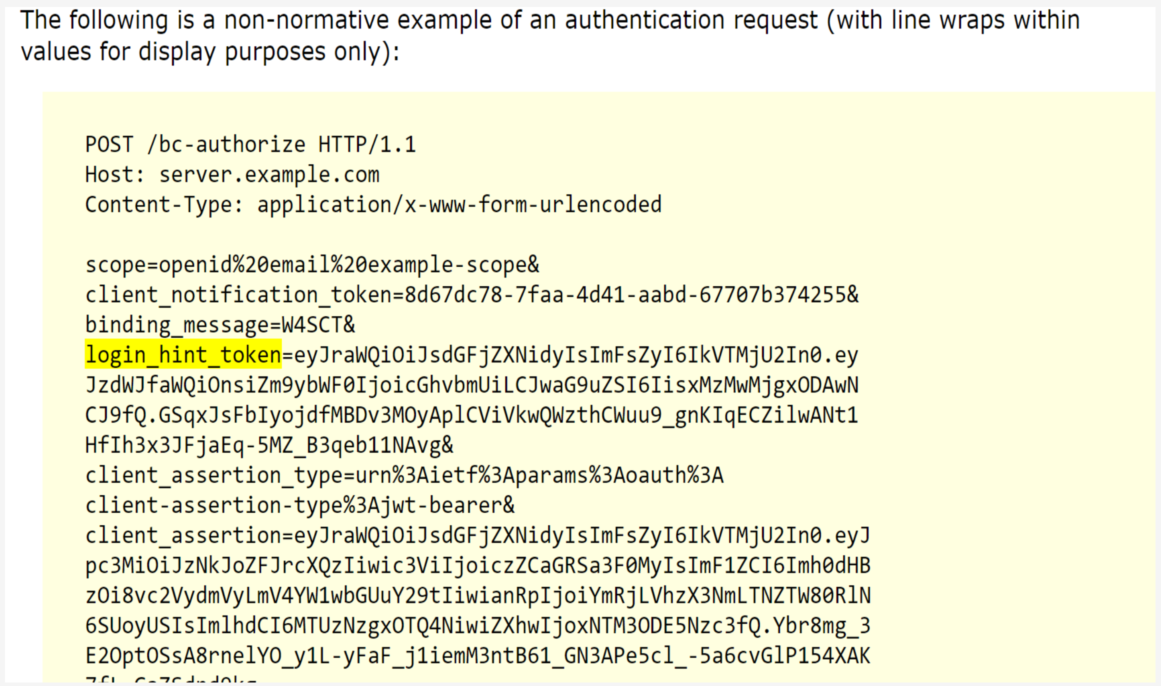
Client Initiated Backchannel Authentication

1. The process decouples the flow between device from which the authorization is requested and the actual device where the auth creds are input.
2. Use cases:
    1. Helps multiple human beings to access and authenticate from different devices
    2. Helps an end user to access the protected resource from a device while authenticating from a different device.



Client Initiated Backchannel Authentication

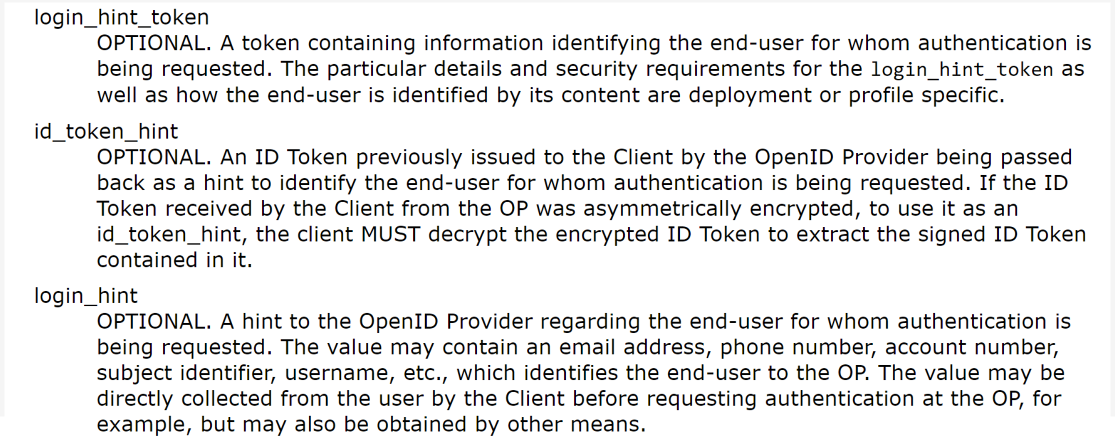
3. Process flow:
   a) The client app makes an Authorization request with a hint about the user that needs to be authenticated.
   b) The hint provided known to both parties, like email address, shared identifier or an ID token.
   c) If the authentication request is successful, authorization server initiates a backend request for user authorization.



The following is a non-normative example of an authentication request (with line wraps within values for display purposes only):

```
POST /bc-authorize HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

scope=openid%20email%20example-scope&
client_notification_token=8d67dc78-7faa-4d41-aabd-67707b374255&
binding_message=W4SCT&
login_hint_token=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.ey
JzdWJfaWQiOnsiZm9ybWF0IjoicGhvbmUiLCJwaG9uZSI6IisxMzMwMjgxODAwN
CJ9fQ.GSqxJsFbIyojdfMBDv3MOyAplCViVkwQWzthCWuu9_gnKIqECZilwANt1
HfIh3x3JFjaEq-5MZ_B3qeb11NAvg&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJraWQiOiJsdGFjZXNidyIsImFsZyI6IkVTMjU2In0.eyJ
pc3MiOiJzNkJoZFJrcXQzIiwic3ViIjoiczZCaGRSa3F0MyIsImF1ZCI6Imh0dHB
zOi8vc2VydmVyLmV4YW1wbGUuY29tIiwianRpIjoiYmRjLVhzX3NmLTNZTW80RlN
6SUoyUSIsImlhdCI6MTUzNzgxOTQ4NiwiZXhwIjoxNTM3ODE5Nzc3fQ.Ybr8mg_3
E2OptOSsA8rnelYO_y1L-yFaF_j1iemM3ntB61_GN3APe5cl_-5a6cvGlP154XAK
```



login_hint_token
: OPTIONAL. A token containing information identifying the end-user for whom authentication is being requested. The particular details and security requirements for the `login_hint_token` as well as how the end-user is identified by its content are deployment or profile specific.

id_token_hint
: OPTIONAL. An ID Token previously issued to the Client by the OpenID Provider being passed back as a hint to identify the end-user for whom authentication is being requested. If the ID Token received by the Client from the OP was asymmetrically encrypted, to use it as an id_token_hint, the client MUST decrypt the encrypted ID Token to extract the signed ID Token contained in it.

login_hint
: OPTIONAL. A hint to the OpenID Provider regarding the end-user for whom authentication is being requested. The value may contain an email address, phone number, account number, subject identifier, username, etc., which identifies the end-user to the OP. The value may be directly collected from the user by the Client before requesting authentication at the OP, for example, but may also be obtained by other means.

[Ref : OpenID Connect Client-Initiated Backchannel Authentication Flow - Core 1.0]

What’s Next

FAPI 2.0 still under drafting:
Introduces concepts such Rich Authorization Request (RAR) & Demonstrating Proof-of-Possession (DPoP)

a) Rich Authorization Request (RAR):
   - Conventional OAuth token allows course grained definition of contents through it scope parameter.
   - RAR allows specifying more fine-grained information such account number, payment amount, creditor name etc to be mentioned.

b) Demonstrating Proof-of-Possession at the Application Layer :
   - Instead of cert hashing, the proof of possession is demonstrated by adding a header at the application layer that is tied to client’s keys.
   - The AuthZ server attaches the public key to the token while generating instead of the cert hash.
   - Does not replace MTLS but a simpler mechanism for apps like SPA.

RAR object

```
[
   {
      "type": "payment_initiation",
      "actions": [
         "initiate",
         "status",
         "cancel"
      ],
      "locations": [
         "https://example.com/payments"
      ],
      "instructedAmount": {
         "currency": "EUR",
         "amount": "123.50"
      },
      "creditorName": "Merchant A",
      "creditorAccount": {
         "iban": "DE02100100109307118603"
      },
      "remittanceInformationUnstructured": "Ref Number Merchant"
   }
]
```

THANK YOU