Financial-grade API (FAPI) and CIBA

DeveloperWeek NYC 2019 @ Brooklyn Expo Center on June 20, 2019

Co-founder, Authlete, Inc.
Takahiko Kawasaki <taka@authlete.com>
Company Profile

<table>
<thead>
<tr>
<th>Name</th>
<th>Authlete, Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establishment</td>
<td>September 18, 2015</td>
</tr>
<tr>
<td>Capital</td>
<td>444,710,000 JPY (including the capital reserve)</td>
</tr>
<tr>
<td>Website</td>
<td><a href="https://www.authlete.com/">https://www.authlete.com/</a></td>
</tr>
</tbody>
</table>

Offices

<table>
<thead>
<tr>
<th>City</th>
<th>Office Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokyo</td>
<td>FINOLAB, Otemachi Bldg 4F, Otemachi 1-6-1, Chiyoda-ku, Tokyo, 100-0004, Japan</td>
</tr>
<tr>
<td>London</td>
<td>Level39, One Canada Square, Canary Wharf, London E14 5AB, UK</td>
</tr>
</tbody>
</table>

Product

Authlete, SaaS providing Web APIs whereby developers implement servers that support OAuth 2.0 and OpenID Connect.

Team

Takahiko Kawasaki – Co-founder, software engineer
Ali Adnan – Co-founder, multilingual serial entrepreneur
Joseph Heenan – Lead of the official OpenID test suite
Justin Richer – Author of "OAuth 2 in Action" and other wonderful members

History

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan. 2014</td>
<td>Starts to implement Authlete</td>
</tr>
<tr>
<td>Sep. 2015</td>
<td>Establishes Authlete, Inc.</td>
</tr>
<tr>
<td>Sep. 2016</td>
<td>Establishes Authlete UK, Ltd.</td>
</tr>
<tr>
<td>Nov. 2016</td>
<td>Joins FINOLAB</td>
</tr>
<tr>
<td>Feb. 2017</td>
<td>Joins OpenID Foundation</td>
</tr>
<tr>
<td>Mar. 2017</td>
<td>Wins FIBC 2017 Grand Prize</td>
</tr>
<tr>
<td>May 2017</td>
<td>Joins Level39</td>
</tr>
<tr>
<td>May 2017</td>
<td>Fund Raising (seed round)</td>
</tr>
<tr>
<td>Jul. 2017</td>
<td>Gets OpenID Certification</td>
</tr>
<tr>
<td>Aug. 2017</td>
<td>Cyber39 Founding Member</td>
</tr>
<tr>
<td>Sep. 2017</td>
<td>Tech in Asia Tokyo 2017 Finalist</td>
</tr>
<tr>
<td>Feb. 2018</td>
<td>Fund Raising (pre-series A)</td>
</tr>
<tr>
<td>Apr. 2018</td>
<td>Wins IBM Prize at Draper Nexus B2B Summit 2018</td>
</tr>
<tr>
<td>Jul. 2018</td>
<td>Joins Fintech Association of Japan</td>
</tr>
<tr>
<td>Jul. 2018</td>
<td>Organizes Japan/UK Open Banking and APIs Summit 2018</td>
</tr>
<tr>
<td>Jul. 2018</td>
<td>Supports Financial-grade API (Authlete 2.0)</td>
</tr>
<tr>
<td>Aug. 2018</td>
<td>Passes Open Banking Security Profile Test</td>
</tr>
<tr>
<td>Jan. 2019</td>
<td>Supervises &quot;OAuth徹底入門&quot; (book)</td>
</tr>
<tr>
<td>Feb. 2019</td>
<td>Supports CIBA</td>
</tr>
<tr>
<td>Apr. 2019</td>
<td>Gets Certified Financial-grade API (FAPI) OpenID Provider</td>
</tr>
</tbody>
</table>
Chapter 1: Financial-grade API
The Financial-grade API (FAPI) Working Group has developed **Financial-grade API (FAPI)** on top of OAuth 2.0 and OpenID Connect.
The specification was renamed from Financial API to Financial-grade API because the specification can apply to not only the financial industry but also other industries that need high security.

<table>
<thead>
<tr>
<th>Year</th>
<th>Issue</th>
<th>Description</th>
<th>Draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>2</td>
<td>Part 1 of Financial API Implementer's Draft 1</td>
<td></td>
</tr>
<tr>
<td>2017</td>
<td>7</td>
<td>Part 2 of Financial API Implementer's Draft 1</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>10</td>
<td>Financial-grade API Implementer's Draft 2</td>
<td></td>
</tr>
</tbody>
</table>
FAPI Certification

OpenID Foundation started **FAPI Certification Program** on April 1, 2019.

---

### Certified Financial-grade API (FAPI) OpenID Providers

These deployments have been granted certifications for these Financial-grade API (FAPI) conformance profiles:

<table>
<thead>
<tr>
<th>Organization</th>
<th>Implementation</th>
<th>FAPI R/W ID2 OP w/ MTLS</th>
<th>FAPI R/W ID2 OP w/ Private Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authlete</td>
<td>Authlete 2.1</td>
<td>1-Apr-2019</td>
<td>1-Apr-2019</td>
</tr>
<tr>
<td>ForgeRock</td>
<td>ForgeRock Financial 3.1.0-credence</td>
<td>1-Apr-2019</td>
<td></td>
</tr>
<tr>
<td>Ozone</td>
<td>Ozone Sandbox v3.1</td>
<td>6-Jun-2019</td>
<td>6-Jun-2019</td>
</tr>
</tbody>
</table>

(Certified FAPI OPs, as of June 12, 2019)
FAPI Parts

From the foreword of FAPI specification:

Financial-grade API consists of the following parts:

• **Part 1**: Read-Only API Security Profile
• **Part 2**: Read and Write API Security Profile
• **Part 3**: Client Initiated Backchannel Authentication Profile

**CIBA specification adds new authorization flows.**

NEW

2019 2 CIBA Core 1.0
Enhanced Security

- Entropy Requirement for Client Secret
- JWT-Based Client Authentication
- Certificate-Based Client Authentication
- Key Size Requirement for Client Authentication
- Proof Key for Code Exchange
- Redirect URI Pre-registration
- Redirect URI Mandatory Request Parameter
- Redirect URI Exact Match
- Level of Assurance for End-User Authentication
- Explicit Consent for Requested Scopes
- Prohibition of Authorization Code Reuse
- Scope Mandatory Response Parameter
- Entropy Requirement for Access Token
- Access Token Revocation

- Claimed HTTP Scheme URI Redirection
- Prohibition of Access Token in Query Part
- Detached Signature
- State Hash
- Certificate-Bound Access Token
- Token Binding
- Request Object Mandatory Request Parameter
- Request Object including All Request Parameters
- Request Object EXP Claim
- Request Object Mandatory Signing
- Essential ACR Claim
- JWT Secured Authorization Response Mode
- TLS Cipher Suite Restriction
- JWS Signature Algorithm Restriction
Client Authentication
Client Authentication is required when a confidential client accesses the token endpoint.
The traditional ways described in RFC 6749 use **Client ID** and **Client Secret** for client authentication.

### 1. Basic Authentication (**client_secret_basic**)

```
"{Client ID}:{Client Secret}"
```

Encode by BASE64

- **POST** `{Token Endpoint}` HTTP/1.1
- **Host:** `{Authorization Server}`
- **Authorization:** `Basic {BASE64-encoded Credentials}`
- **Content-Type:** `application/x-www-form-urlencoded`

(abbrev)
2. Form Parameters (client_secret_post)

POST {Token Endpoint} HTTP/1.1
Host: {Authorization Server}
Content-Type: application/x-www-form-urlencoded

client_id={Client ID}&
client_secret={Client Secret}&
(abbrev)

These traditional ways (client_secret_basic and client_secret_post) are not allowed in FAPI.
<table>
<thead>
<tr>
<th>Client Authentication Method</th>
<th>Part 1</th>
<th>Part 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_secret_basic</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>client_secret_jwt</td>
<td>○</td>
<td>✗</td>
</tr>
<tr>
<td>private_key_jwt</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>tls_client_auth</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>self_signed_tls_client_auth</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>client_secret_post</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>client_secret_advanced</td>
<td></td>
<td></td>
</tr>
<tr>
<td>client_secret_jwt</td>
<td>○</td>
<td>✗</td>
</tr>
<tr>
<td>private_key_jwt</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>tls_client_auth</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>self_signed_tls_client_auth</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

- **Part 1**: Blue color indicates availability.
- **Part 2**: Red color indicates non-availability.
JWT-based Client Authentication (RFC 7523)

- Generate **JWT** and pass it to the token endpoint instead of passing a pair of **client ID** & **client secret** directly.

- The JWT is passed as the value of `client_assertion`.

- The JWT is signed using either
  - (a) the client's **client secret** (`client_secret_jwt`), or
  - (b) the client's **private key** (`private_key_jwt`).
POST {Token Endpoint} HTTP/1.1
Host: {Authorization Server}
Content-Type: application/x-www-form-urlencoded

client_assertion_type=
   urn:ietf:params:oauth:client-assertion-type:jwt-bearer&
client_assertion={JWT}&
(abbrev)

```json
{
   "iss": "{Client ID}",
   "sub": "{Client ID}",
   "aud": "{Token Endpoint}",
   "jti": "{JWT ID}",
   "exp": {Expiration Time},
   "iat": {Issue Time}
}
```

The **iss** claim and the **sub** claim in the JWT hold the **client ID**.
Certificate-based Client Authentication

✓ Establish **mutual TLS** connection to the token endpoint.

✓ The **client certificate** presented in the connection is used for client authentication.

✓ The client certificate is either
  (a) **PKI certificate** (**tls_client_auth**), or
  (b) **self-signed certificate** (**self_signed_tls_client_auth**).
A client certificate is sent through the TLS connection.

Authorization server uses the client certificate for client authentication.
Certificate-Bound Access Token
Client Application

access token

Resource Server

access token

API call

extract

verify

API call

Malicious Client

access token

stolen
Certificate-Bound Access Token

1. Client Application
   - token request (Mutual TLS)

2. Authorization Server
   - generate an access token and bind the certificate to it

3. issue an access token

4. Resource Server
   - API call (Mutual TLS)

5. check the binding

The same client certificate as used in the token request
JWT Secured Authorization Response Mode (JARM)
JARM is a specification to pack response parameters from the authorization endpoint into a JWT.

In normal cases

HTTP/1.1 302 Found
Location: https://client.example.com/callback?

In JARM

HTTP/1.1 302 Found
Location: https://client.example.com/callback?
  [response]=\{JWT\}
Example of an authorization response in JARM

HTTP/1.1 302 Found
Location: https://client.example.com/cb?response=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2FjY291bnRzLmV4YW1wbGUuY29tIiwic2l6ZSI6MCwiZG9tYWluIjoiYXNzZXRzLXN0b3JlIiwiaWRva2ciOjIiLCJzdF90d29yZCI6IjJfNjQiLCJzdF9zY3JpcHRhdGUiOjEiLCJzdF9waWNzIjoiYWRtaW4ifSwicm9vdCI6Mn0.

Decoded payload

```json
{
  "iss": "https://accounts.example.com",
  "aud": "s6BhdRkqt3",
  "exp": 1311281970,
  "code": "PyyFaux2o7Q0YfXBU32jhw.5FXSQpvr8akv9CeRDSd0QA",
  "state": "S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw"
}
```
To use JARM, include the `response_mode` parameter with `*.jwt`.

```
response_mode=query.jwt
|fragment.jwt
|form_post.jwt
|jwt
```

```
GET {Authorization Endpoint}
?response_type={Response Type}
&client_id={Client ID}
&response_mode=jwt
HTTP/1.1
Host: {Authorization Server}
```
Chapter 2: **Client Initiated Backchannel Authentication**
CIBA (Client Initiated Backchannel Authentication) defines new authorization flows.

<table>
<thead>
<tr>
<th></th>
<th>CIBA POLL Mode</th>
<th>CIBA PING Mode</th>
<th>CIBA PUSH Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The flows enable to separate the **authentication device** on which a user is authenticated and API authorization is granted from the **consumption device** on which a client application that calls APIs runs.
The system is asking for the permission. Approve?

Purchase ABC.

smart speaker

consumption device

backend system

authentication device

authorization server that supports CIBA

1. smart speaker

2. backend system

3. asks for the permission

4. backchannel authentication request

5. grants the permission

6. calls APIs

7. resource server that provides APIs
Every CIBA flow starts from a backchannel authentication request.

Client sends a backchannel authentication request to the backchannel authentication endpoint of the authorization server.
Backchannel Authentication Endpoint returns a response immediately.

Authorization Server delegates the tasks of end-user authentication and consent confirmation to the Authentication Device.

Authentication Device passes the result to the authorization server.
CIBA POLL mode

1. backchannel authentication request
2. backchannel authentication response
3. communicate
4. token request
5. token response

(4)-(5) is repeated until (3) finishes.

Authorized Server

Backchannel Authentication Endpoint

Token Endpoint

Authentication Device

End-User
CIBA PING mode

Client Notification Endpoint

Authorization Server

Backchannel Authentication Endpoint

Token Endpoint

Authentication Device

End-User

1. Backchannel authentication request
2. Backchannel authentication response
3. Communicate
4. Notification
5. Token request
6. Token response
CIBA PUSH mode

This notification includes an access token & an ID token.
Thank You

Contact

https://www.authlete.com/contact/

<table>
<thead>
<tr>
<th>General</th>
<th><a href="mailto:info@authlete.com">info@authlete.com</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td><a href="mailto:sales@authlete.com">sales@authlete.com</a></td>
</tr>
<tr>
<td>PR</td>
<td><a href="mailto:pr@authlete.com">pr@authlete.com</a></td>
</tr>
<tr>
<td>Technical</td>
<td><a href="mailto:support@authlete.com">support@authlete.com</a></td>
</tr>
</tbody>
</table>

@authlete
References
Specifications

- **Financial-grade API, Part 1: Read-Only Security Profile**
  https://openid.net/specs/openid-financial-api-part-1-ID2.html

- **Financial-grade API, Part 2: Read and Write API Security Profile**

- **Financial-grade API: JWT Secured Authorization Response Mode for OAuth 2.0 (JARM)**

- **OpenID Connect Client Initiated Backchannel Authentication Flow – Core 1.0**
  https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html

- **OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens**
  https://datatracker.ietf.org/doc/draft-ietf-oauth-mtls/

- **RFC 7523 – JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants**

Articles

- **Financial-grade API (API), explained by an implementer**

- **"CIBA", a new authentication/authorization technology in 2019, explained by an implementer**
  https://medium.com/@darutk/ciba-a-new-authentication-authorization-technology-in-2019-explained-by-an-implementer-d1e0ac1311b4

Others

- **Financial-grade API (FAPI) Working Group**
  https://openid.net/wg/fapi/

- **Official Conformance Suite**
  https://gitlab.com/openid/conformance-suite