

いまどきのOAuth/OIDC一挙おさらい

2020年2月

工藤達雄

Authlete, Inc.



Welcome!

「ポストPKCE/JWT」を中心に
Authlete社の独断と偏見により
2020年に注目すべきOAuth 2.0 /
OIDC関連仕様とプラクティスを
ご紹介します



About Me

- サン・マイクロシステムズ、野村総合研究所、NRI セキュアを経て、2018年からAuthlete 社にて VP of Solution Strategy を担当
- 専門はデジタル・アイデンティティを中心とするプリセールス、コンサルティング、事業開発、エバンジェリズム

- LinkedIn <https://www.linkedin.com/in/tatsuokudo>
- Twitter @tkudos



OAuth 2.0 and OpenID Connect in the early 2010s





OAuth 2.0とは

- トークンを用いたAPIアクセス権移譲のフレームワーク
 - RFC 6749: トークン付与プロセスと基本的な付与フロー
(認可コード、クライアントクレデンシャルなど)
- 広範なユースケースに適用可能
 - トークン付与方式やトークン形式の拡張
 - ユーザー、サーバー、クライアント、デバイス、...

OAuth 2.0 認可サーバーが行うやりとり

(「認可リクエスト」を受け取り
「認可grant」を提供)

「認可grant」を受け取り
「トークン」を提供

(「トークン」を受け取り
「関連情報」を提供)

1.2. Protocol Flow

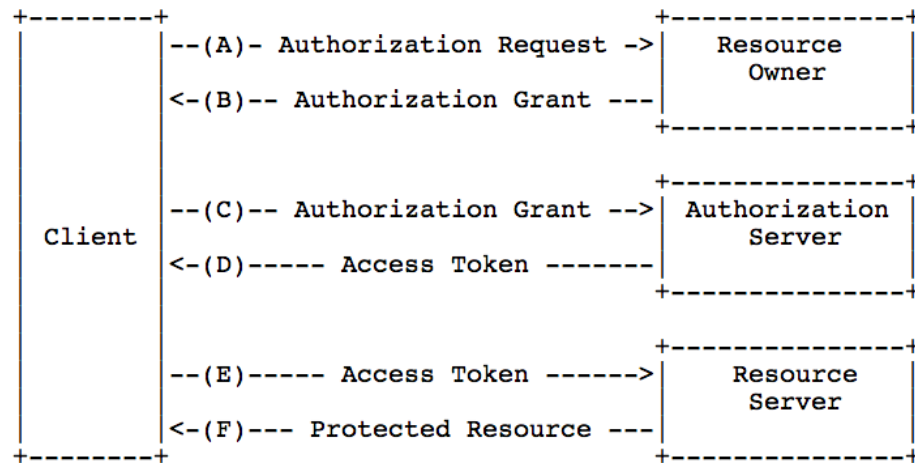
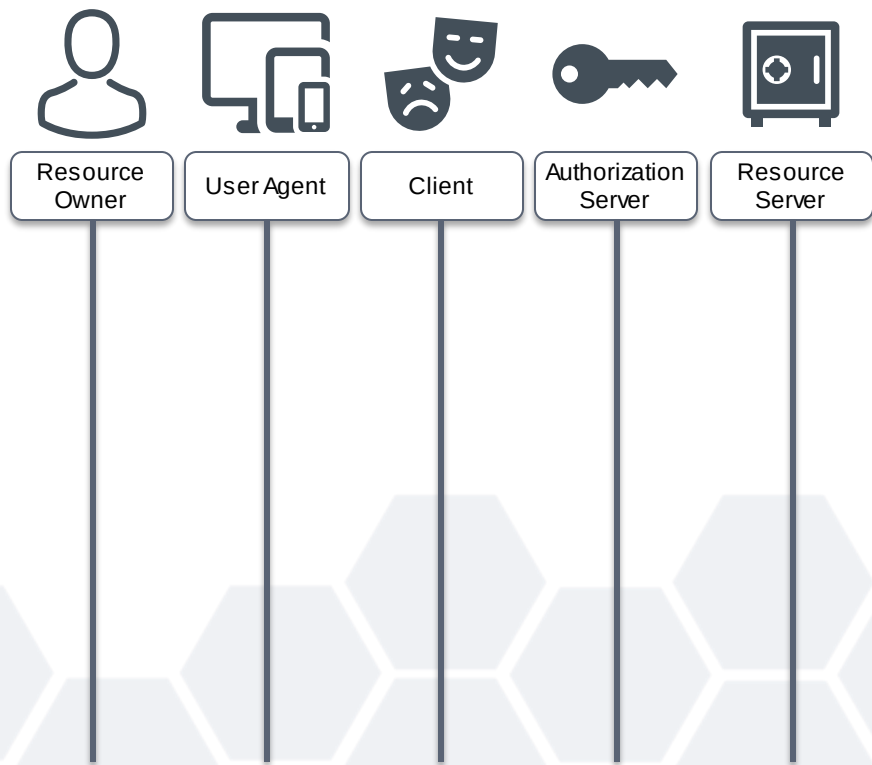


Figure 1: Abstract Protocol Flow

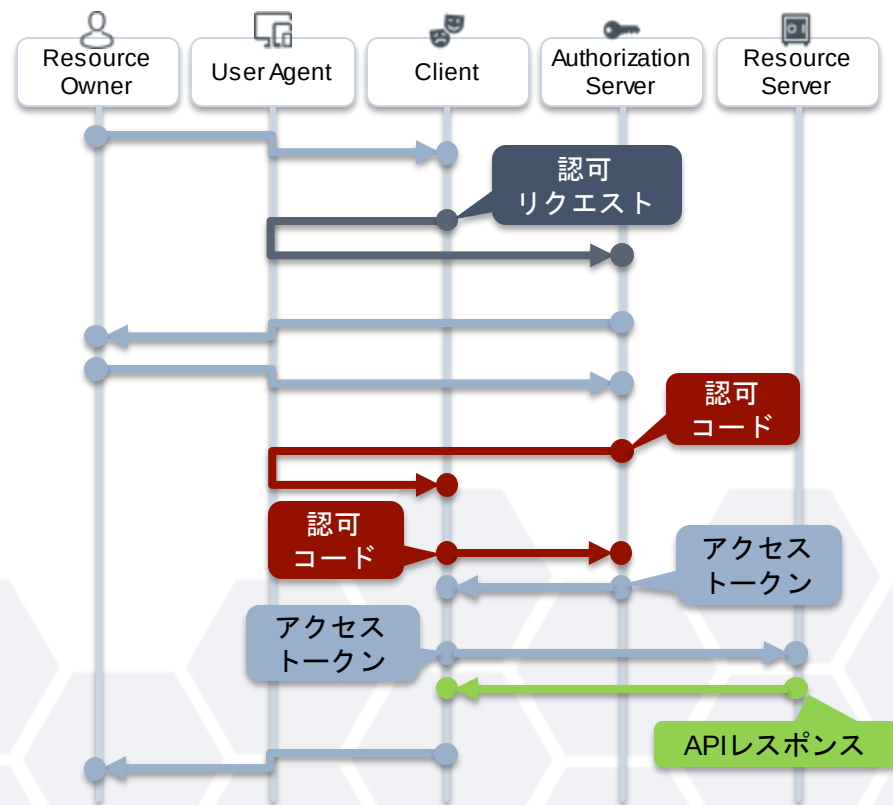
Source: RFC 6749 1.2. Protocol Flow <https://tools.ietf.org/html/rfc6749>

“OAuth Dance”



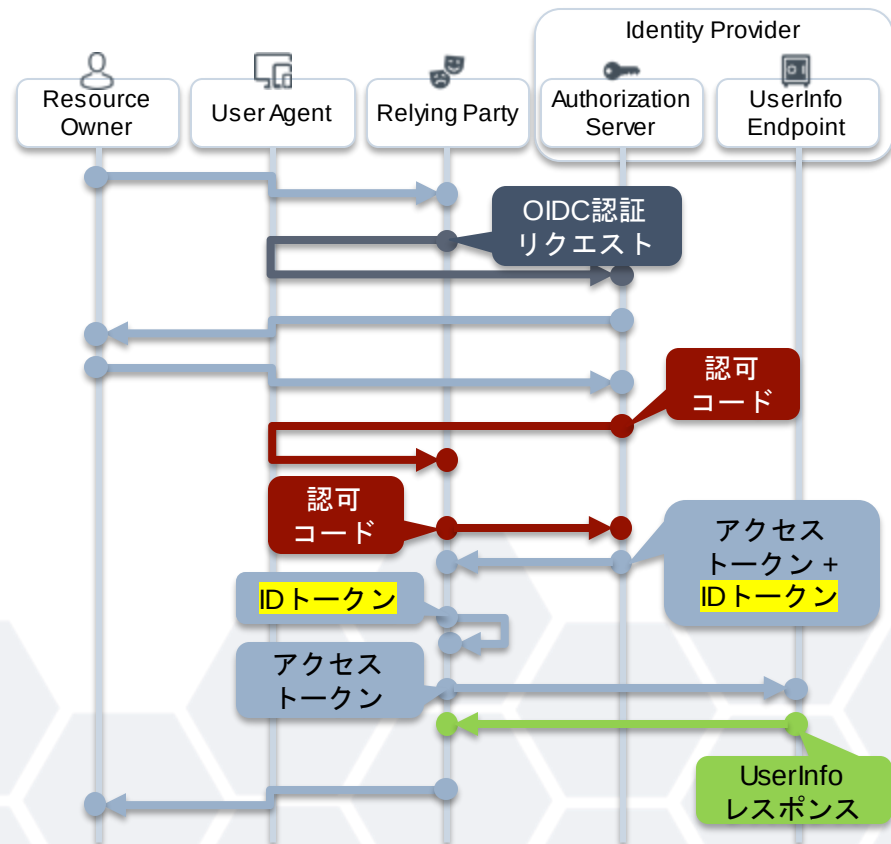
- **Resource Owner**
e.g. end user
- **User Agent**
e.g. Web browser
- **Client**
e.g. Web application using APIs
- **Authorization Server**
e.g. user authentication server
- **Resource Server**
e.g. API server

OAuth 2.0 認可コードグラントフロー



- 認可リクエストを受け付けたら
認可コードを返す
- 認可コードをもらったら
アクセストークンを返す
- (アクセストークン付きの
APIリクエストを受け付けて
レスポンスを返す)

OpenID Connect 認可コードフロー



- **OIDC認証リクエスト**を受け付けたら
認可コードを返す
- **認可コード**をもらったら
アクセストークンと
IDトークンを返す
 - (RPは) **IDトークン**を用いて
ユーザーを識別する
- **アクセストークン**つきの
UserInfoリクエストを受け付けて
レスポンスを返す

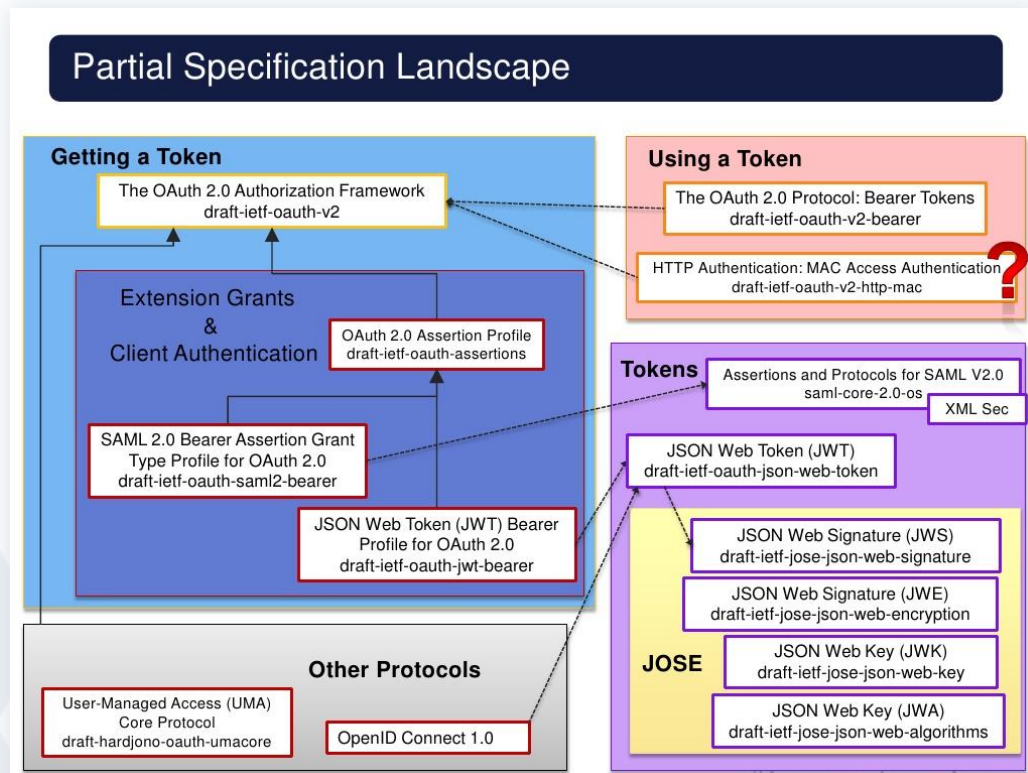
- コンパクトかつURL
セーフな「クレーム」
の表現方式
- 署名・暗号化に対応
- OAuth 2.0 における
トークン形式として用
いられることもある
- OpenID Connectでは
JWTを用いてIDトーク
ンを定義している



OAuth 2.0 関連仕様

- 2012年ごろには
全体の構成が確立
- 2015年、主要な
仕様がすべて標準化

...2020年もそのまま!?



Source: OAuth 101 & Secure APIs 2012 Cloud Identity Summit

<https://www.slideshare.net/briandavidcampbell/oauth-101-secure-apis-2012-cloud-identity-summit-2012/66>

OAuth/OIDC in 2020



PKCE / Resource Indicators / MTLS / private_key_jwt

OAUTH 2.0 SECURITY BCP

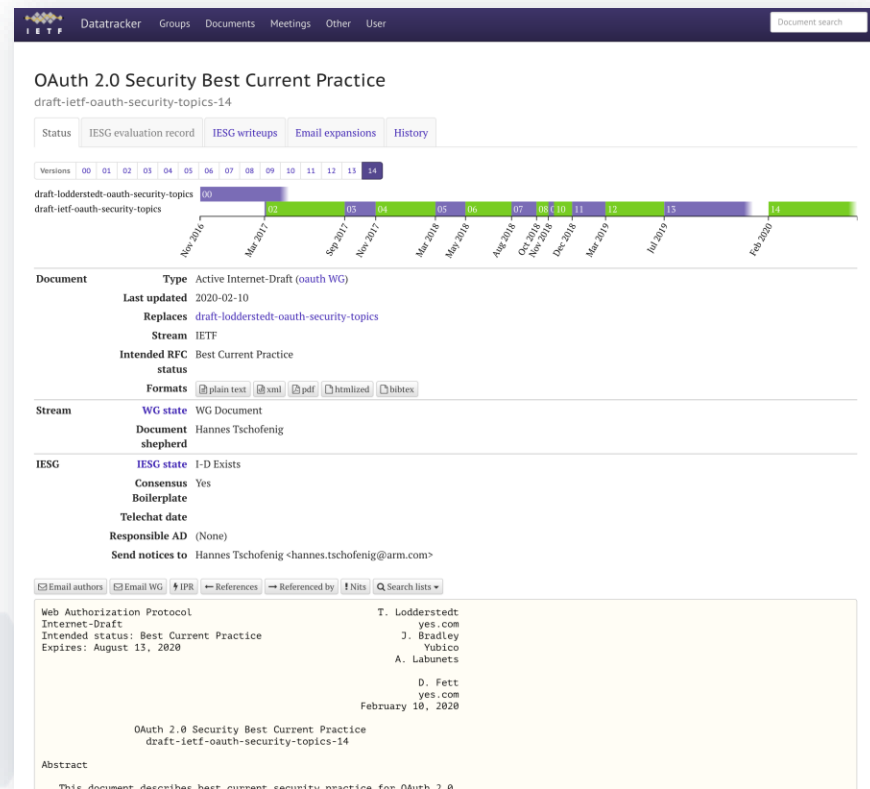
OAuth 2.0 Security Best Current Practice (BCP)

<https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics/>

- 現在のOAuth 2.0の利用状況は当初 (2012年) の想定から乖離
 - 実装時の問題: アンチパターンの認識不足 (仕様に書いてあるのに)、OAuth周辺の技術の変化
 - High-stakesな分野への適用: 金融・医療
 - 多数との動的な連携: クライアントが複数のAS/RSと連携。かつ動的な連携も

→ 過去の同種のドキュメントを更新

- OAuth 2.0 Threat Model and Security Considerations (RFC 6819)
- OAuth 2.0 Security Considerations (RFC 6749 & 6750)



Source: draft-ietf-oauth-security-topics-14 - OAuth 2.0 Security Best Current Practice
<https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics/>

OAuth 2.0 Security BCPの構成

Table of Contents

1. Introduction	3
1.1. Structure	4
1.2. Conventions and Terminology	4
2. Recommendations	5
2.1. Protecting Redirect-Based Flows	5
2.1.1. Authorization Code Grant	6
2.1.2. Implicit Grant	6
2.2. Token Replay Prevention	7
2.3. Access Token Privilege Restriction	7
2.4. Resource Owner Password Credentials Grant	8
2.5. Client Authentication	8
2.6. Other Recommendations	8
3. The Updated OAuth 2.0 Attacker Model	8
4. Attacks and Mitigations	10
4.1. Insufficient Redirect URI Validation	11
4.1.1. Redirect URI Validation Attacks on Authorization Code Grant	11
4.1.2. Redirect URI Validation Attacks on Implicit Grant	13
4.1.3. Countermeasures	14
4.2. Credential Leakage via Referer Headers	15
4.2.1. Leakage from the OAuth Client	15
4.2.2. Leakage from the Authorization Server	15
4.2.3. Consequences	16
4.2.4. Countermeasures	16
4.3. Credential Leakage via Browser History	17
4.3.1. Authorization Code in Browser History	17
4.3.2. Access Token in Browser History	17
4.4. Mix-Up Attacks	18
4.4.1. Attack Description	18
4.4.2. Countermeasures	20

4.5. Authorization Code Injection	21
4.5.1. Attack Description	21
4.5.2. Discussion	22
4.5.3. Countermeasures	23
4.5.4. Limitations	24
4.6. Access Token Injection	24
4.6.1. Countermeasures	25
4.7. Cross Site Request Forgery	25
4.7.1. Countermeasures	25
4.8. Access Token Leakage at the Resource Server	25
4.8.1. Access Token Phishing by Counterfeit Resource Server	26
4.8.2. Compromised Resource Server	31
4.9. Open Redirection	31
4.9.1. Client as Open Redirector	32
4.9.2. Authorization Server as Open Redirector	32
4.10. 307 Redirect	32
4.11. TLS Terminating Reverse Proxies	33
4.12. Refresh Token Protection	34
4.12.1. Discussion	34
4.12.2. Recommendations	35
4.13. Client Impersonating Resource Owner	36
4.13.1. Countermeasures	36
4.14. Clickjacking	36
5. Acknowledgements	37
6. IANA Considerations	38
7. Security Considerations	38
8. References	38
8.1. Normative References	38
8.2. Informative References	39
Appendix A. Document History	42
Authors' Addresses	46

2.1. リダイレクトベースのフローの保護

- 認可サーバーによる不適切なリダイレクトの防止
- クライアントが「オープンリダイレクター」になることの防止
- CSRF対策（**PKCE** or nonce, もしくは state)
- “Mix-up Attack” 対策
- 認可サーバーにおけるユーザークレデンシャルの想定外の漏洩防止

Native Appsにおける戻し先の乗っ取り

カスタムURLスキームの上書きにより正当なAppではなく攻撃者Appに認可コードが渡る

カスタムURLスキームを上書き

被害者のスマートフォン

被害者

OS / 標準
ブラウザ

正当なApp

攻撃者の
App

Authorization
Server

Resource
Server

連携開始

認可リクエスト

```
GET /authorize
?response_type=code
&client_id=s6BhdRkqt3
&scope=message.readonly
&redirect_uri=<AppのカスタムURLスキーム>
HTTP/1.1
Host: server.example.com
```

認可レスポンス

```
HTTP/1.1 302 Found
Location: <AppのカスタムURLスキーム>
?code=<被害者にひもづく認可コード>
&state=xyz
```

処理開始を指示

認可リクエスト

ユーザー認証・
アクセス承認

認可レスポンス

認可コード

攻撃者の
Appに処理を
引き継ぎ

被害者の
認可コードを
入手

RFC 7636: Proof Key for Code Exchange by OAuth Public Clients (PKCE)

<https://datatracker.ietf.org/doc/rfc7636/>

カスタムURLスキームを上書き

被害者のスマートフォン

被害者

OS / 標準
ブラウザ

正当なApp

攻撃者の
App

Authorization
Server

Resource
Server

連携開始

認可リクエスト

```
GET /authorize
?response_type=code
&client_id=s6BhdRkqt3
&scope=message.readonly
&redirect_uri=<AppのカスタムURLスキーム>
&code_challenge=<code verifierから
code_challenge methodに従って生成した値>
&code_challenge_method=S256
HTTP/1.1
Host: server.example.com
```

認可レスポンス

```
HTTP/1.1 302 Found
Location: <AppのカスタムURLスキーム>
?code=<被害者にひもづく認可コード>
&state=xyz
```

処理開始を指示

認可リクエスト
code_challenge

code_challenge
受け取り

ユーザー認証・
アクセス承認

認可レスポンス
認可コード

トークン
リクエスト
認可コード

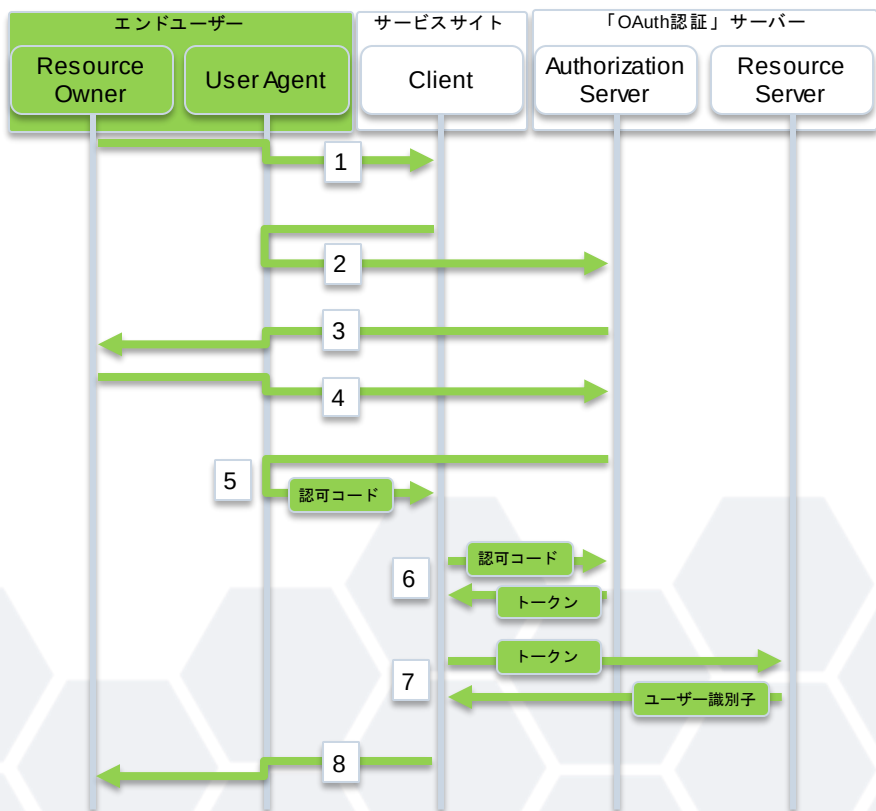
code_verifier
が無い
ため
トークン
を取
得
でき
ない

攻撃者の
Appに処理を
引き継ぎ

被害者の
認可コードを
入手

CSRF + 「OAuth認証」

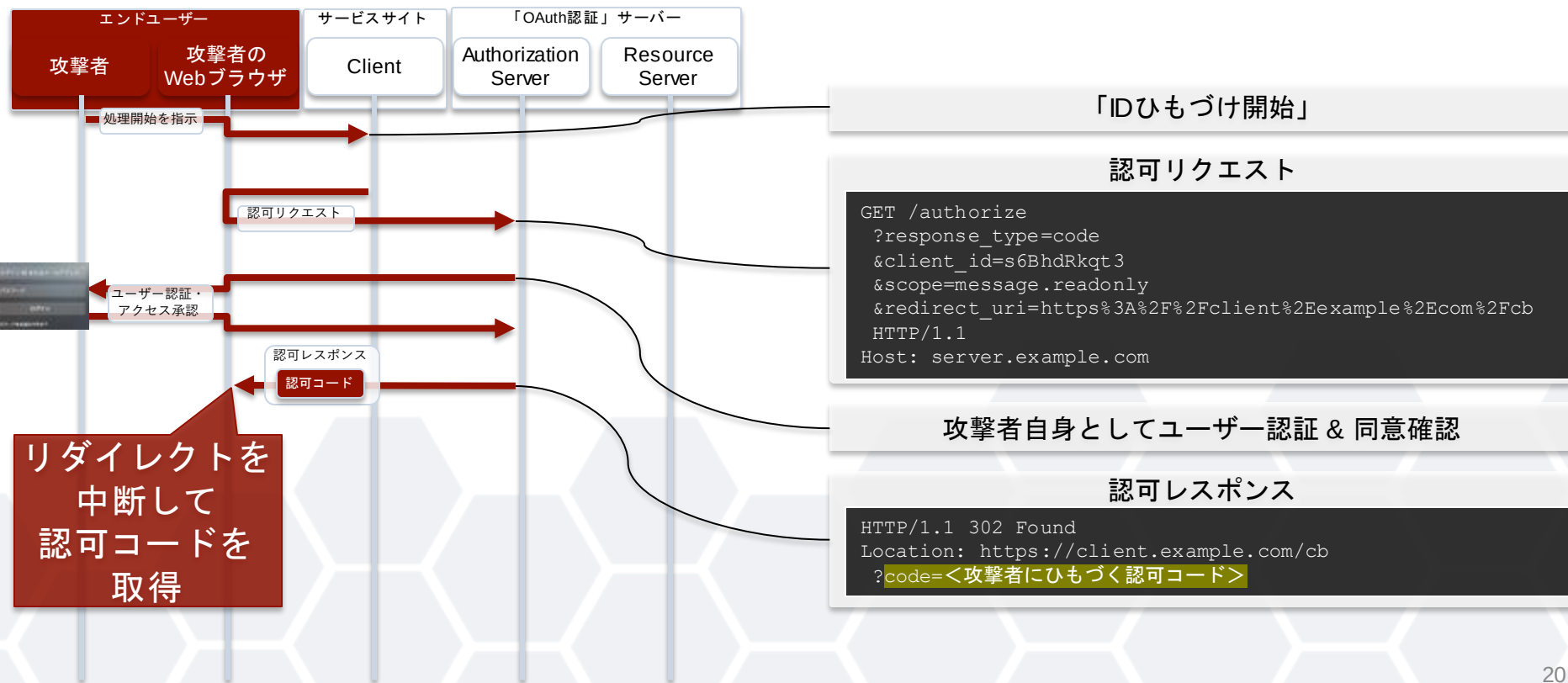
アクセストークンを用いてユーザー識別子を取得



1. 「OAuth認証でログイン」をクリック
2. 「OAuth認証サーバー」にリダイレクト
3. 「ログインしてください」
4. ID/パスワードを送信
5. サービスサイトにリダイレクト
6. 認可コードを送信しアクセストークンを取得
7. アクセストークンを用いてユーザー情報APIにアクセスし、ユーザー識別子を取得
8. 取得したユーザー識別子をもとにユーザーを特定してログイン完了

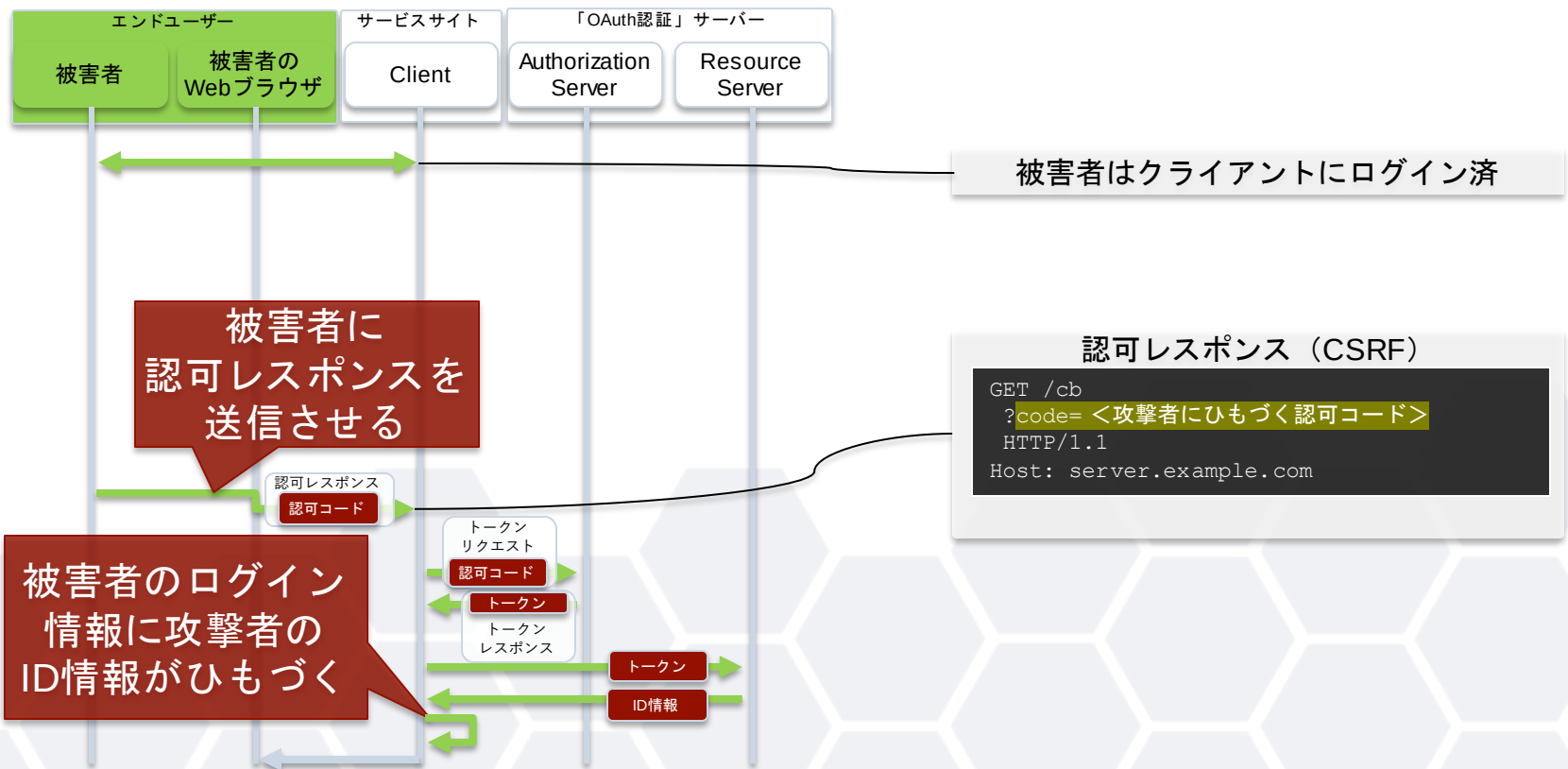
CSRF + 「OAuth認証」

攻撃者にひもづく認可コードを生成



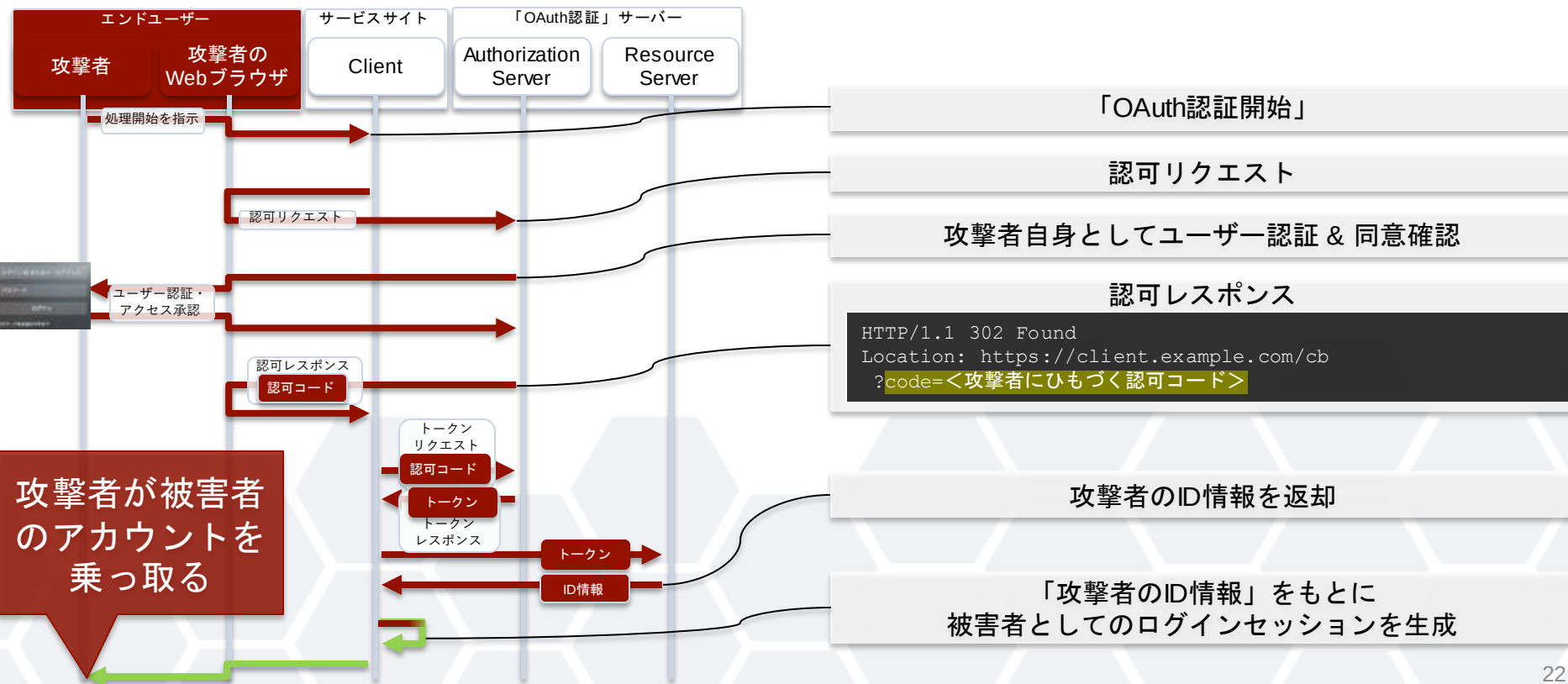
CSRF + 「OAuth認証」

被害者が「攻撃者にひもづく認可コード」を送信



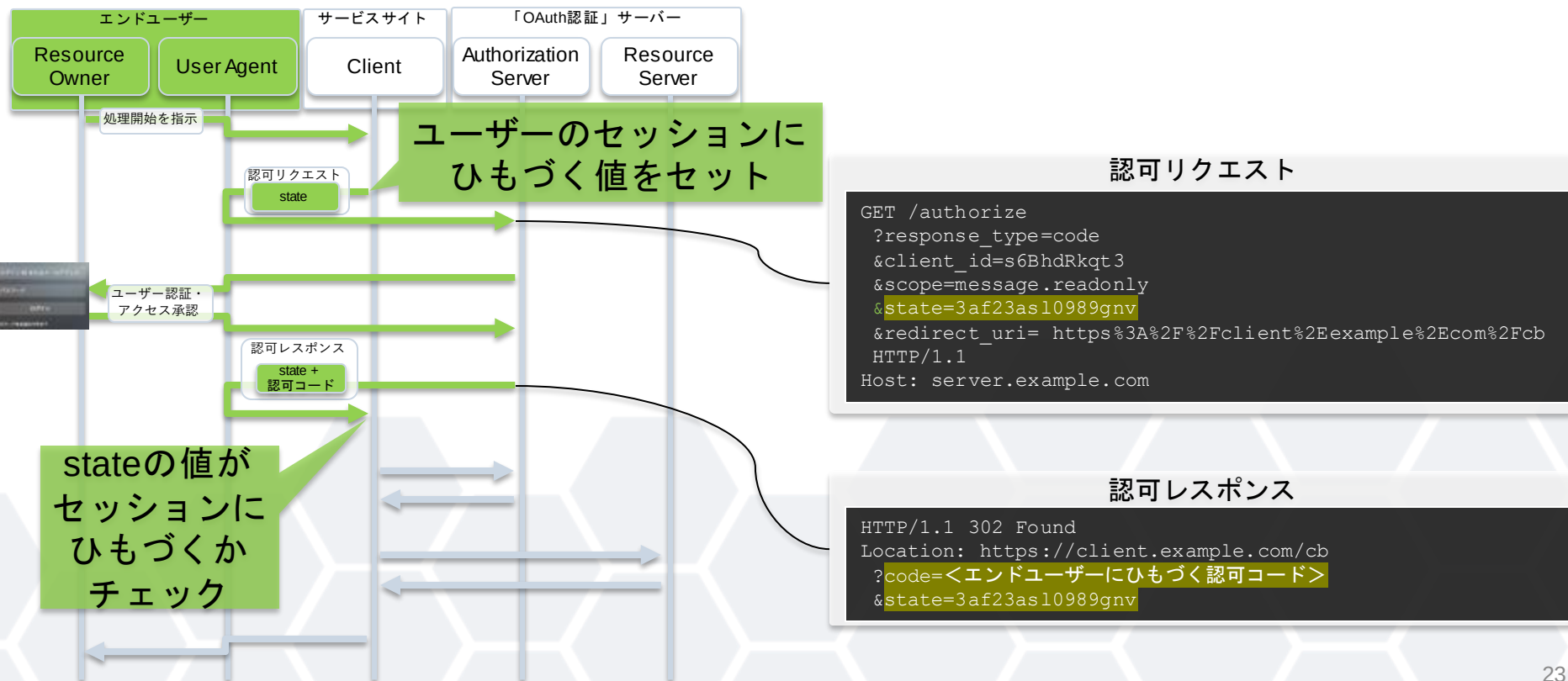
CSRF + 「OAuth認証」

攻撃者が「被害者としてログイン」できる



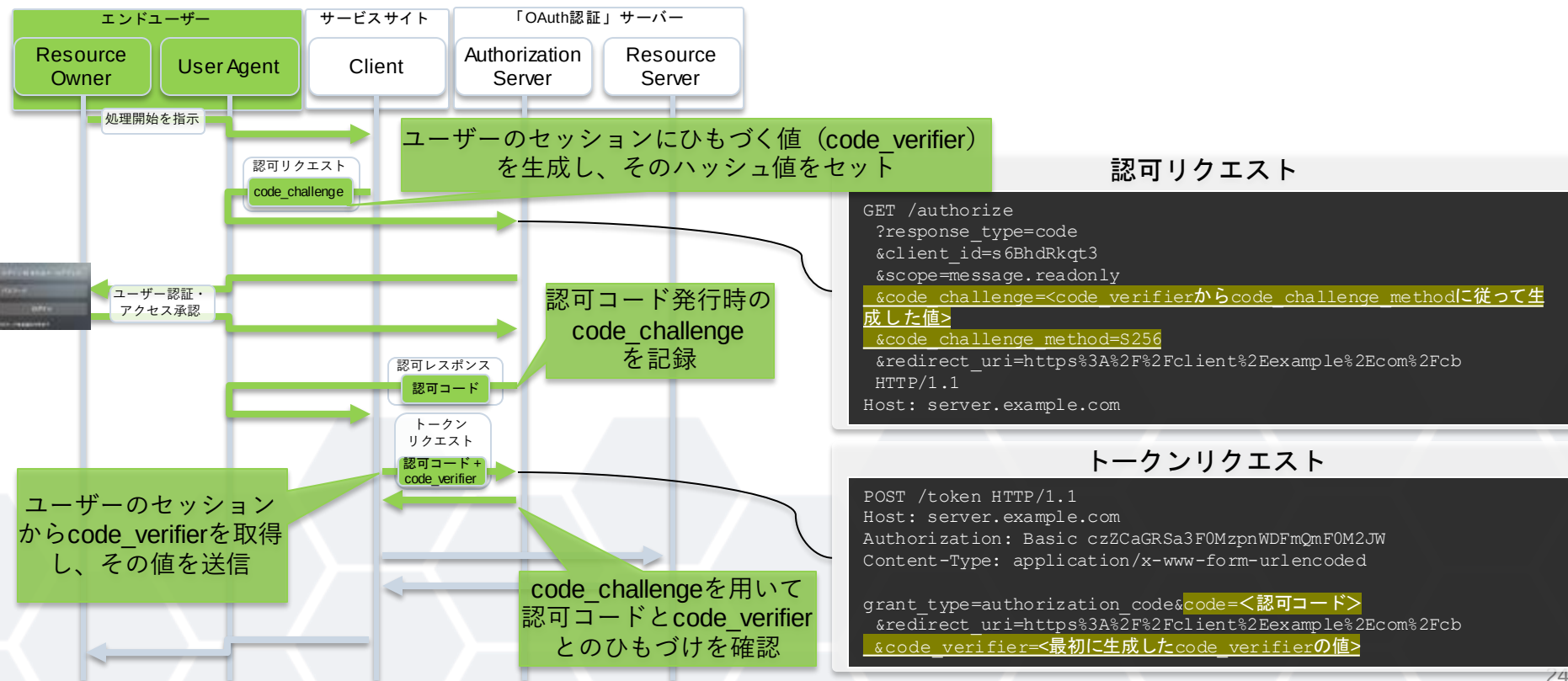
stateパラメーターによるCSRF対策

「認可リクエストの送信元」と「認可コードの送信元」が同一であることをクライアントが確認



PKCEによるCSRF対策

「認可リクエストの送信元」と「認可コードの送信元」が同一であることを認可サーバーが確認

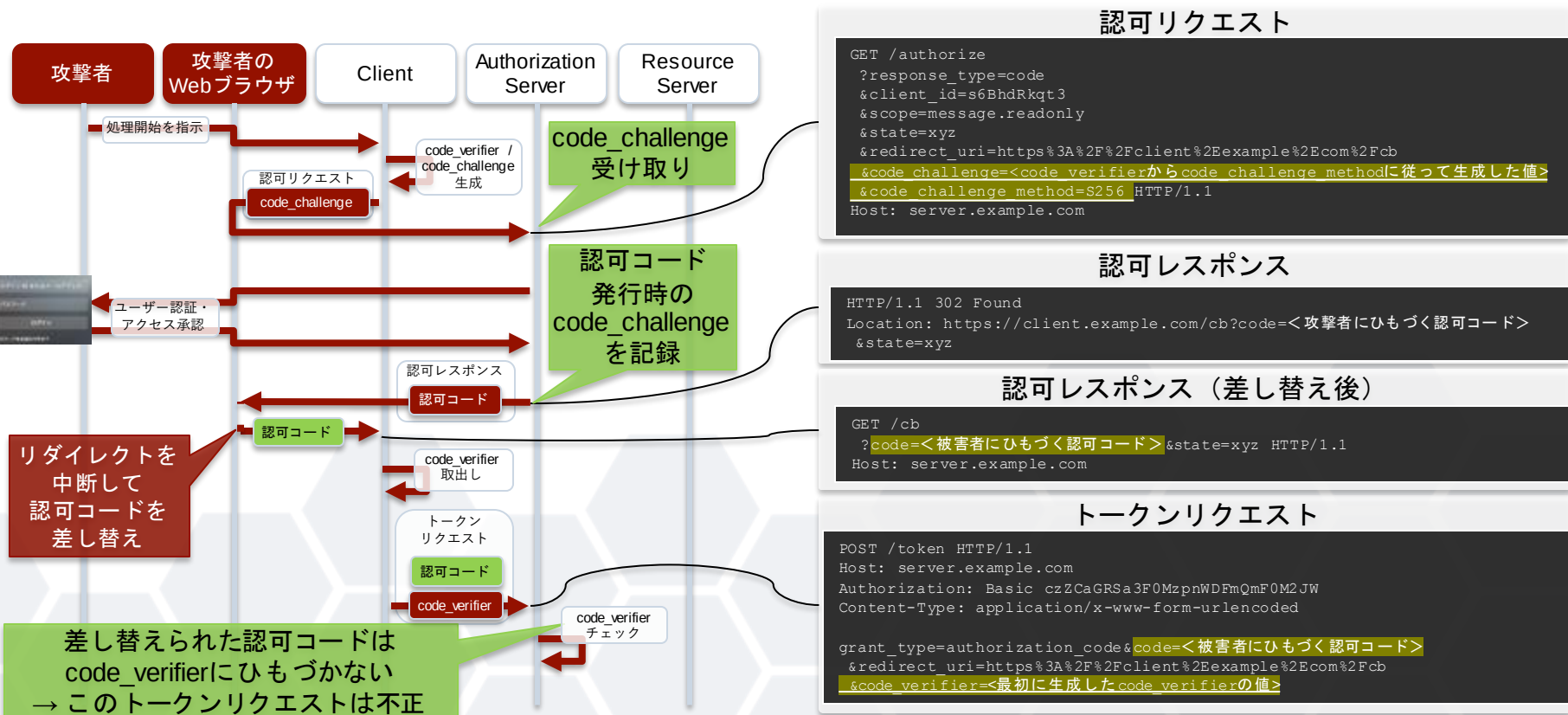


2.1.1. 認可コードグラント

- クライアントにおける認可コード注入（リプレイ）攻撃
対策として**PKCE**を推奨（対応していればnonceも可）
- 認可サーバーはPKCE対応が必須

認可コード差し替え攻撃対策としてのPKCE

認可コードの不正利用によるアクセストークン窃取を防止



2.1.2. Implicit Grant

- 認可レスポンスからのアクセストークン返却は非推奨

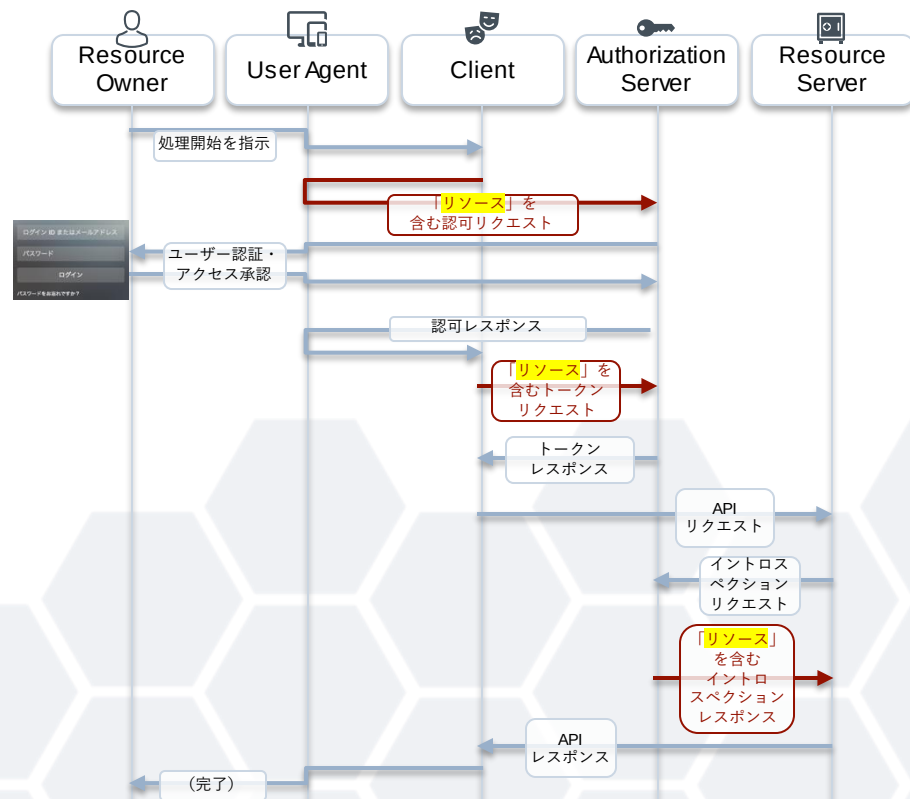
2.3. アクセストークンの権限の制限

- アクセストークンの受け手（オーディエンス）となるリソースサーバーの限定と、リソースサーバーにおけるアクセストークンのオーディエンスのチェック
 - クライアントおよび認可サーバーにおける、“scope” パラメーターや “resource” [I-D.ietf-oauth-resource-indicators] パラメーターの指定
- 加えてアクセストークンを「特定のリソース」や「特定のアクション」に限定
 - クライアントおよび認可サーバーにおける、“scope” パラメーターや “authorization_details” [I-D.ietf-oauth-rar] パラメーターの指定

Resource Indicators for OAuth 2.0

<https://datatracker.ietf.org/doc/draft-ietf-oauth-resource-indicators/>

- 認可リクエストにてresourceパラメーターを用いて対象リソース（絶対URI）を指定
 - `[...]&resource=https%3A%2F%2Fcal.example.com%2F&[...]`
- さらにトークンリクエストでもresourceを指定し、効力を限定したトークンを取得可能



draft-ietf-oauth-security-topics-14/ 2. Recommendations

2.4. リソースオーナーパスワードクレデンシャルズグラント

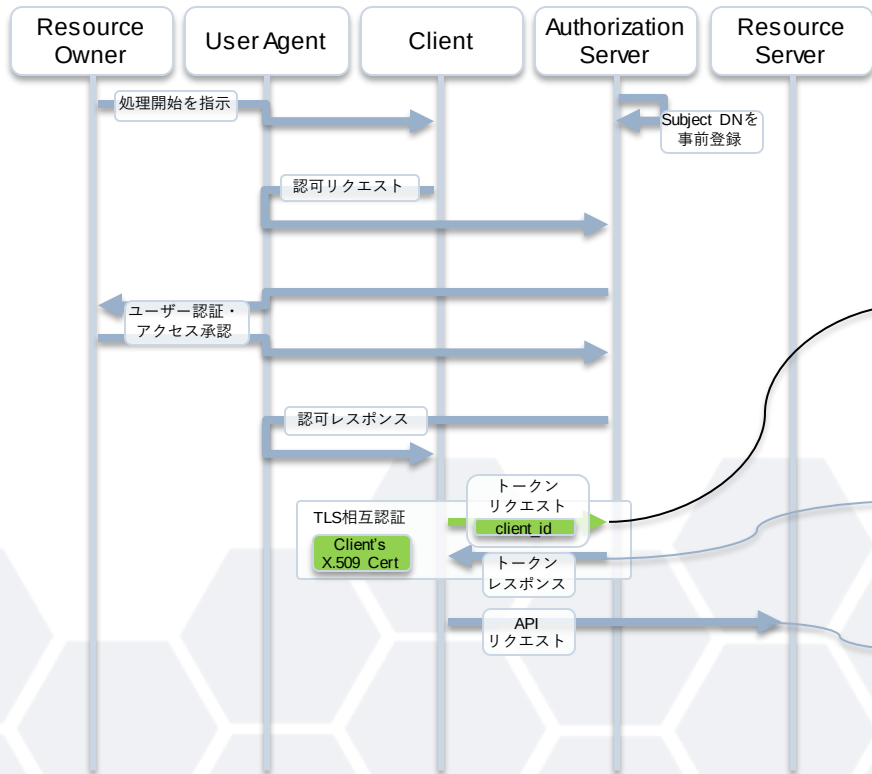
- リソースオーナーパスワードクレデンシャルズグラント
は利用禁止

2.5. クライアント認証

- 認可サーバーにおけるクライアント認証の実施を推奨
- mTLS [I-D.draft-ietf-oauth-mtls] もしくは private_key_jwt [OIDC] のような非対称（公開鍵ベース）の方式を推奨

OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens (MTLS) <https://datatracker.ietf.org/doc/draft-ietf-oauth-mtls/>

- クライアントから提示されたX.509証明書の「サブジェクト識別名（DN）」を登録値と照合



トークンリクエスト

- TLS相互認証を行ってクライアントのX.509証明書を取得し、その証明書のサブジェクトDNをもとにクライアントを認証
 - リクエスト内のclient_idの値からクライアントを識別
 - そのクライアント情報として事前登録されている「証明書のサブジェクトDN」の値と照合し認証

トークンレスポンス

- 証明書に結びつけた（バインドした）アクセストークンを返却

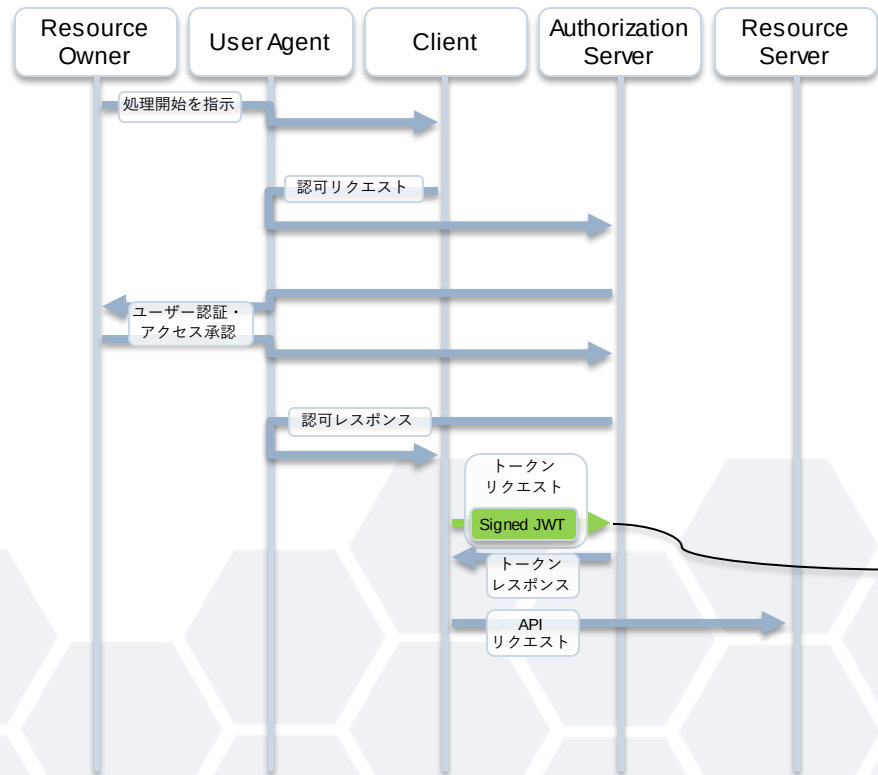
APIリクエスト

- TLS相互認証を行いクライアントのX.509証明書を取得
- 証明書とアクセストークンの結びつき（バインド）を確認

OIDC Core 1.0 / 9. Client Authentication / private_key_jwt

https://openid.net/specs/openid-connect-core-1_0.html#ClientAuthentication

- 公開鍵暗号により署名された情報（クレームセット）をもとに認証



クレームセット

```
{
  "iss": "55f9f559-2496-49d4-b6c3-351a586b7484",
  "sub": "55f9f559-2496-49d4-b6c3-351a586b7484",
  "aud": "https://idp-p.example.com/token",
  "iat": 1418698788,
  "exp": 1418698848,
  "jti": "1418698788/107c4da5194df463e52b56865c5af34e5595"
}
```

Signed JWT生成

トークンリクエスト

```
POST /token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: idp-p.example.com

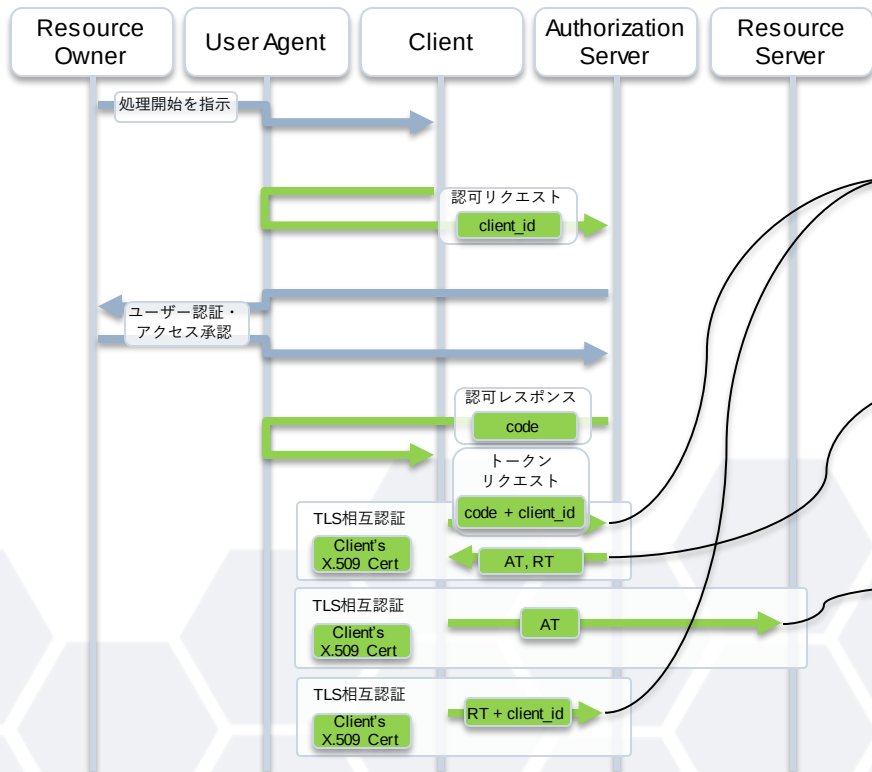
grant_type=authorization_code
&code=sedaFh
&scope=openid+email
&client_id=55f9f559-2496-49d4-b6c3-351a586b7484
&client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer
client_assertion=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.ew0KIICAg[...omitted for brevity...].t-gX8JQ[...omitted for brevity...]
```

2.2. トークンリプレイの防止

- 認可サーバーおよびリソースサーバーにおいて「送信者限定アクセストークン」を用いること
 - そのしくみとして **Mutual TLS for OAuth 2.0 [RFC8705]** を推奨
- リフレッシュトークンは送信者限定にすること、もしくはローテーションすることを必須
- エンドツーエンドのTLSを推奨

OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens (MTLS) <https://datatracker.ietf.org/doc/draft-ietf-oauth-mtls/>

- トークンリクエスト時のクライアント証明書に、発行するアクセストークンをバインド



トークンリクエスト

- TLS相互認証を行ってクライアントのX.509証明書を取得し、その証明書のサブジェクトDNをもとにクライアントを認証
 - リクエスト内のclient_idの値からクライアントを識別
 - そのクライアント情報として事前登録されている「証明書のサブジェクトDN」の値と照合し認証

トークンレスポンス

- 証明書に結びつけた（バインドした）アクセストークンを返却

APIリクエスト

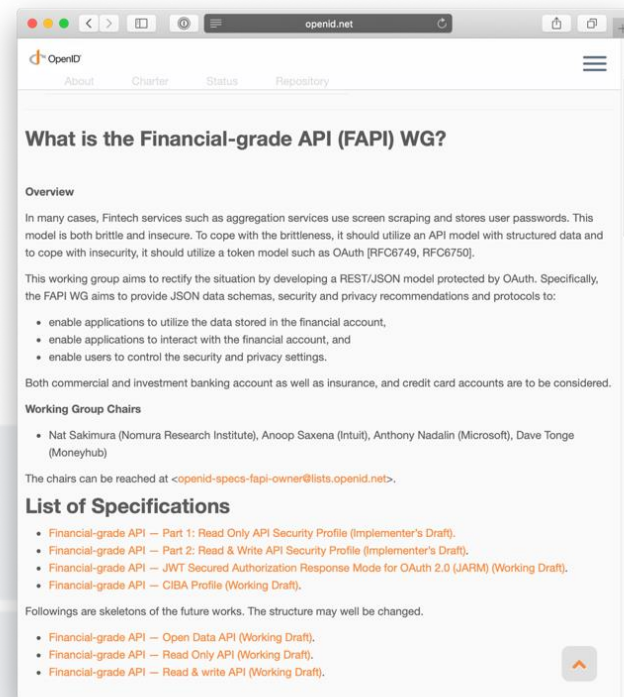
- TLS相互認証を行いクライアントのX.509証明書を取得
- 証明書とアクセストークンの結びつき（バインド）を確認
 - トークンに含まれる or イントロスペクションの結果として得られる、証明書の「サンプリント」を利用

Request Object / Hybrid Flow / JARM

FINANCIAL-GRADE API (FAPI)

Financial-grade API (FAPI)

- 金融 API 向けの OAuth 2.0 適用
プラクティス
- 2016年にOpenID Foundation傘下に
設置されたFAPI WGにて策定中
- 英国Open Bankingや豪州CDRなど
が既に採用



Source: <https://openid.net/wg/fapi/>

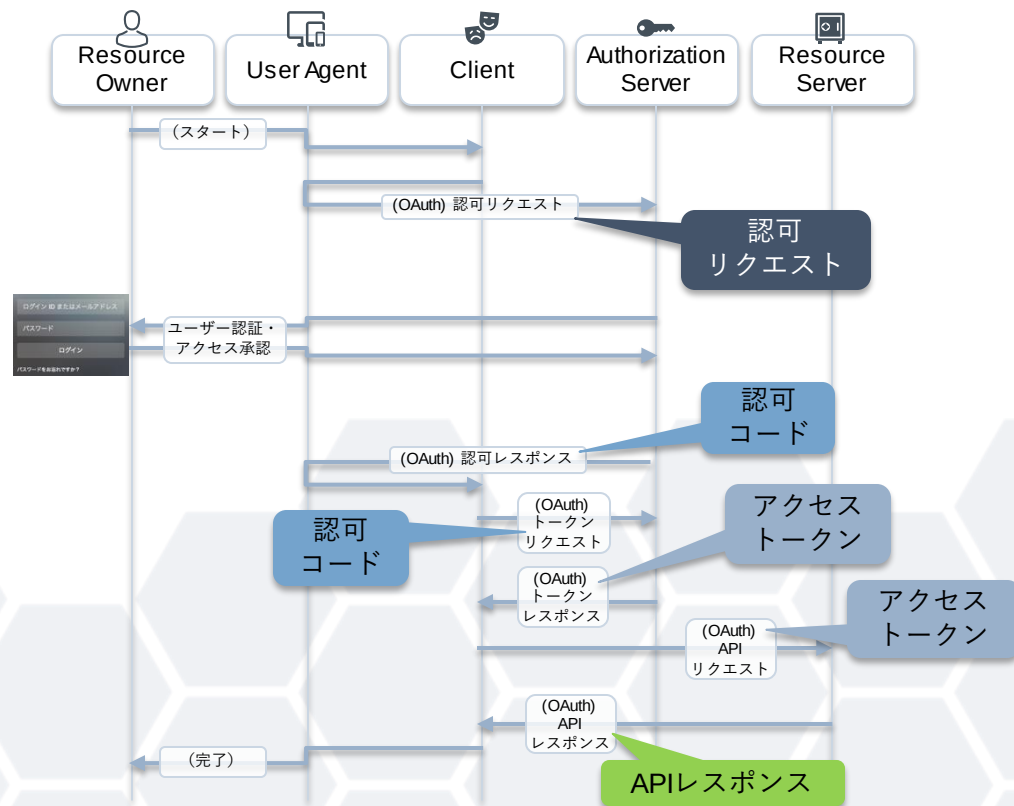
FAPIセキュリティ・プロファイル

- Part 1 (Read Only)
 - OAuth 2.0 適用のプラクティス
 - OAuth 2.0 拡張仕様
 - OIDC によるユーザー識別子の授受
- Part 2 (Read & Write): OIDC の積極的な活用 + 新たな拡張仕様
 - Request Object, Hybrid Flow, MTLS, OAUTB
 - JARM



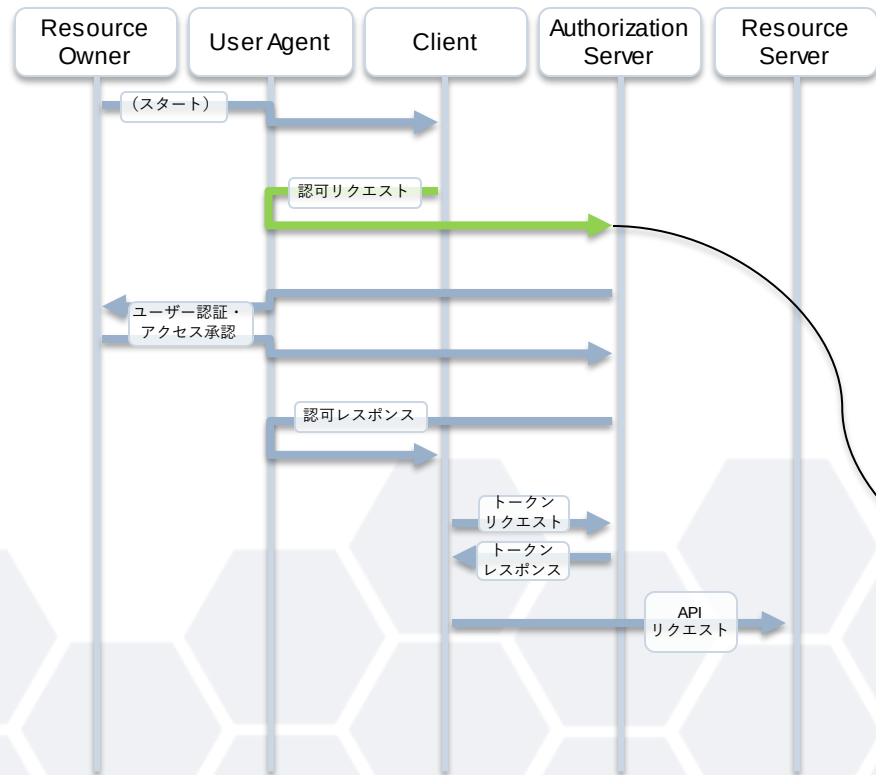
FAPIにおける“OAuth Dance”の改善・改良

- 認可リクエスト / レスポンスの送信者
詐称・改ざん防止
 - Request Objectの利用
 - Hybrid FlowもしくはJARMの利用
- 認可コードの漏洩・盗用防止
 - redirect_uriの厳密な管理
- クライアントのなりすまし防止
 - Mutual TLSもしくはJWTによるクライアント認証
- トークンの漏洩・盗用防止
 - OAUTB（トークン・バインディング）もしくはMTLS（双方向TLS接続へのバインド）の利用



リクエストオブジェクト

OIDC認証リクエストへの署名や暗号化



リクエストオブジェクトのクレーム

```

{
  "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,
  "claims": {
    {
      "userinfo": { ... "email": {"essential": true},...},
      "id_token": { "gender": null, ...
        "acr": {"values": ["urn:mace:incommon:iap:silver"]}
      }
    }
  }
}

```

リクエストオブジェクト生成

OIDC認証リクエスト

- 値渡し

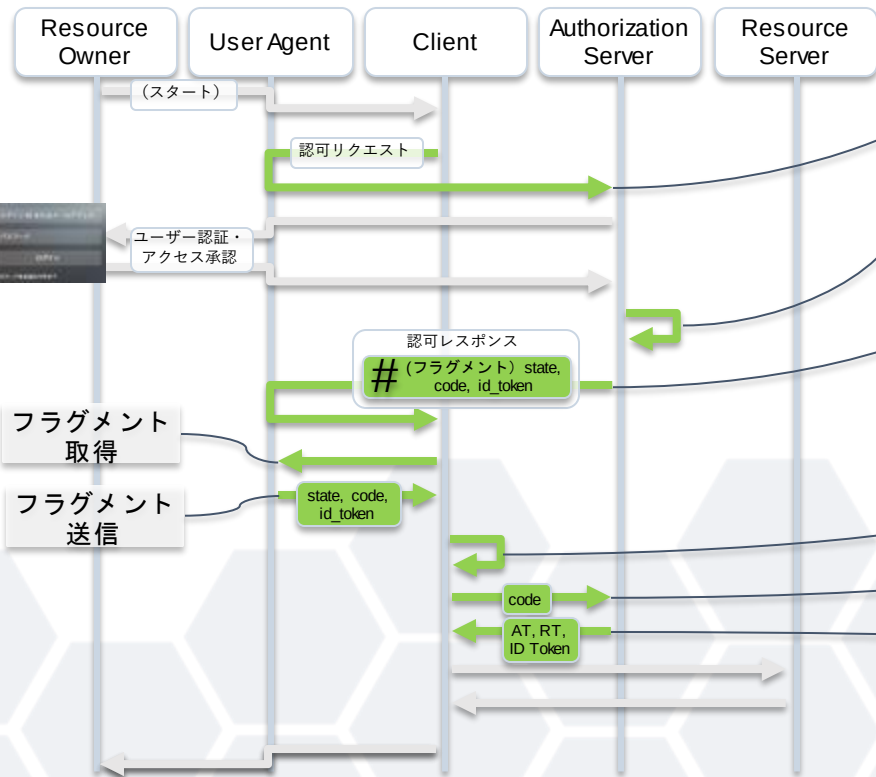
```
GET /authorize?request=eyJhbGciOiJSUzI1NiIsImtpZCI6ImN5YmRjIn0.ewOKICJpc3MiOiAicZCaGRS...
```

- 参照渡し

```
GET /authorize?request_uri=%3A%2F%2Fclient.example.org%2Frequest.jwt...
```


Hybrid Flow

認可コードとIDトークンを認可EPから返却



認可リクエスト

```
GET /authorize
?response_type=code%20id_token
[...]
&state=af0ifjsldkj
```

Detached Signature生成

認可レスポンス

```
HTTP/1.1 302 Found
Location: https://api.mytp.com/cb#
code=Splxl0BeZQQYbYS6WxSbIA
&id_token=eyJ0...NiJ9.eyJ1c...
I6IjIifX0.DeWt4Qu...ZXSo
&state=af0ifjsldkj
```

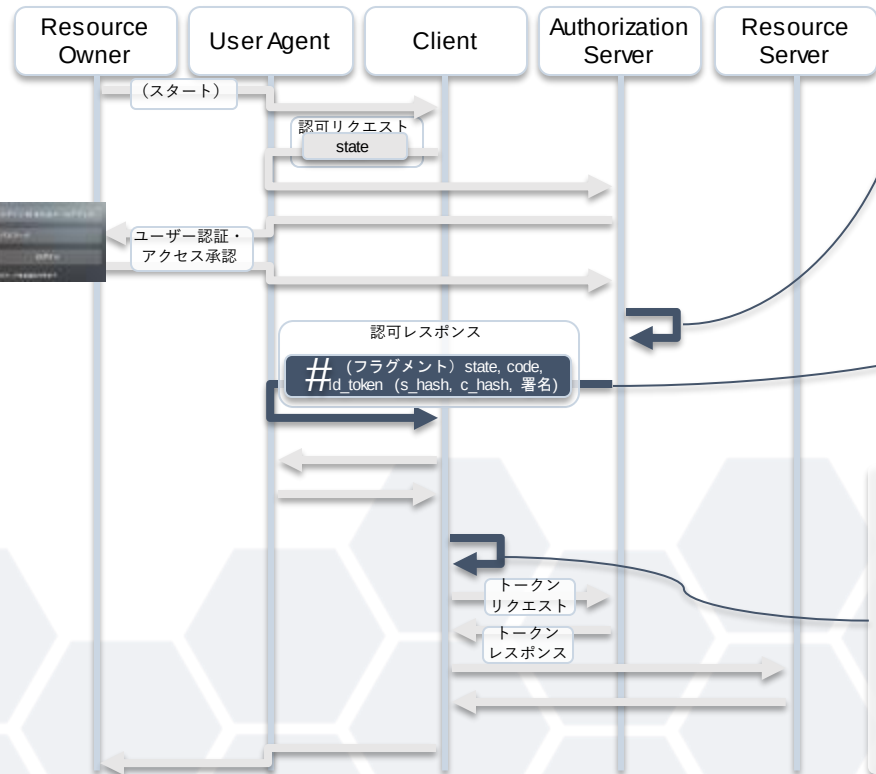
Detached Signatureによる検証

トークンリクエスト

トークンレスポンス

Detached Signature

検証用の値に署名して本文と別に返却



state, code の値が確定

s_hash, c_hash生成

クレームセット

```
{ ...  
  "s_hash": "76sa5dd",  
  "c_hash": "asd097d" ...  
}
```

IDトークン (Signed JWT) 生成

認可レスポンス

```
HTTP/1.1 302 Found  
Location: https://api.mytpp.com/cb#  
code=Sp1x10BeZQQYbYS6WxSbIA  
&id_token=eyJ0...NiJ9.eyJ1c...  
I6IjIifX0.DeWt4Qu...ZXs0  
&state=af0ifjsldkj
```

IDトークン (Signed JWT) を検証した上で、c_hash, s_hashを抽出し、それぞれをもとにcode, stateを検証

```
{ ...  
  "s_hash": "76sa5dd",  
  "c_hash": "asd097d" ...  
}
```

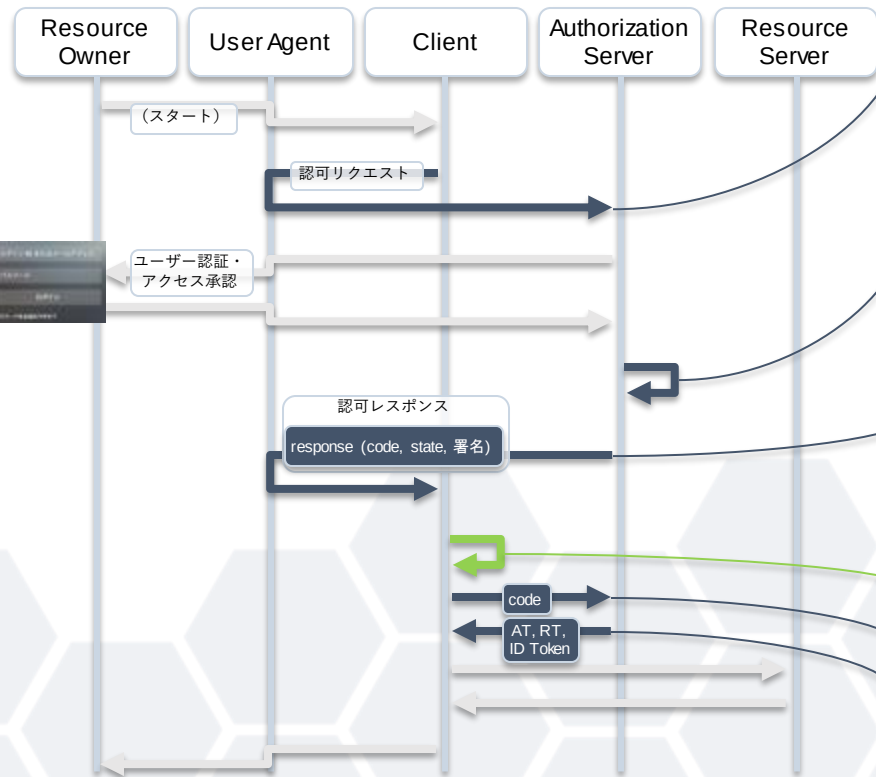
IDトークン (Signed JWT) として
検証できた (改ざんされていない) 値

```
code=Sp1x10BeZQQYbYS6WxSbIA  
state=af0ifjsldkj
```

クエリパラメーターとして
受け取った値

JARM

JWT Secured Authorization Response Mode for OAuth 2.0



認可リクエスト

```
GET /authorize?  
response_type=code&response_mode=jwt&[...]
```

クレームセット

```
{  
  "iss": "https://accounts.example.com",  
  "aud": "s6BhdRkqt3",  
  "exp": 1311281970,  
  "code": "PyyFaux2o7Q0YfXBU32jhw.5FXSQpvr8akv9CeRDSd0QA",  
  "state": "S8NJ7uqk5fY4EjNvp_G_FtyJu6pUsvH9jSYni9dMAJw"  
}
```

response (Signed JWT) 生成

認可レスポンス

```
HTTP/1.1 302 Found  
Location: https://client.example.com/cb?  
response=eyJhbGciOiJIJSUzIiNiIsInR5cCI6IkpXVCJ9.  
eyJpc3MiOiJodHRwczovL2FjY291bnRzLmV4YW1wbGUuY29tIiwiaXVkaW  
joicZCaGRSa3F0MyIsImV4cCI6MTY4MTMxMTI4MTk3MCwiY29kZSI6IjB5eU  
ZhdXgybzdRMFlnWEJVMzJqaHcuNUZyU1FwdnI4YWt2OUNlUkRTZDBRQSI  
sInN0YXRlIjoiaUZhOSjdlcWslZlZkO0RwOdlBfRl9GdHlKdTZwVXN2S1lq  
c1luaTlktUEFKdyJ9.  
HkdJ_TYgwBBj10C-aWuNUIA062Amq2b0_oyuc5P0aMTQpAqC2o9WbGsk  
pfuHVBowlb-zJ15tBvXDIABL_t83q6ajvjtg_pqsByiRK2dLVdUwKhW3P  
9wvjvI0K2ogdoTNbNlP9Z4lmhart4BqraIoI8e-L_EfAHfhCG_DDDv7Yg
```

Signed JWT検証

トークンリクエスト

トークンレスポンス

FAPIセキュリティプロファイルの現状

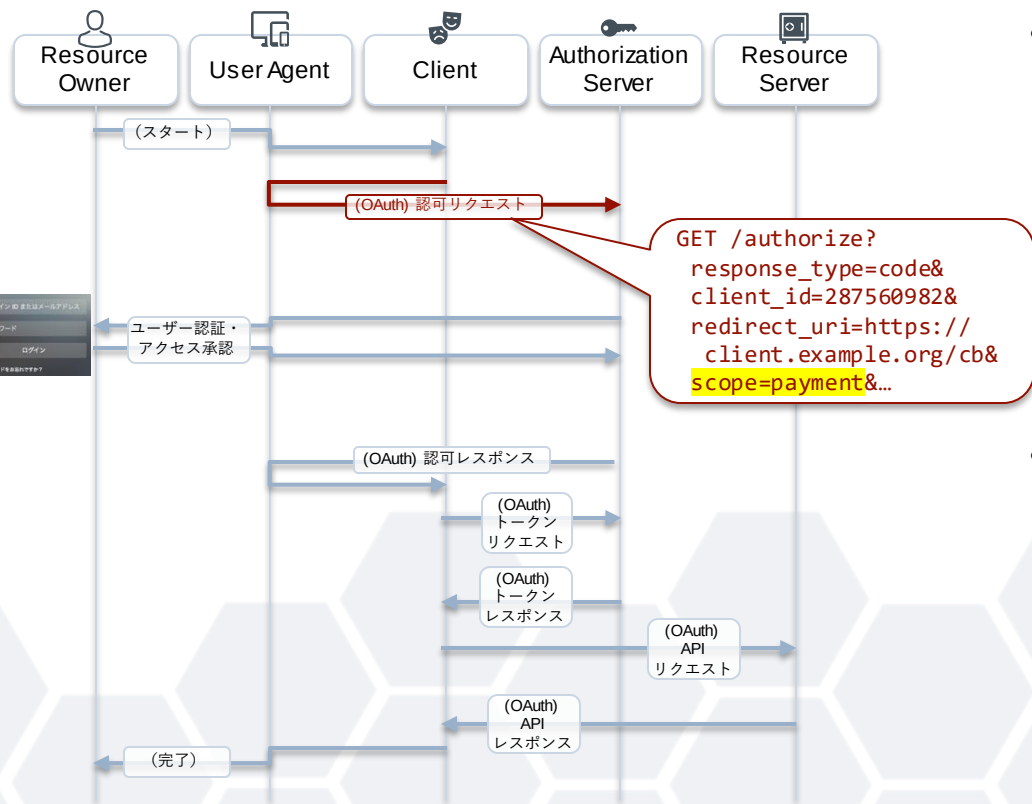
(実装者向けドラフト第2版以降)

- Removed
 - OAuth 2.0 Token Binding
 - code id_token and code id_token token
 - LoA3
 - Public client support
- Separated
 - Pushed Request Object https://bitbucket.org/openid/fapi/src/master/Financial_API_Pushed_Request_Object.md
- Promoted
 - JARM as an alternative to Hybrid Flow
- Added
 - 60-minute lifetime for exp in Request Object

PAR / RAR

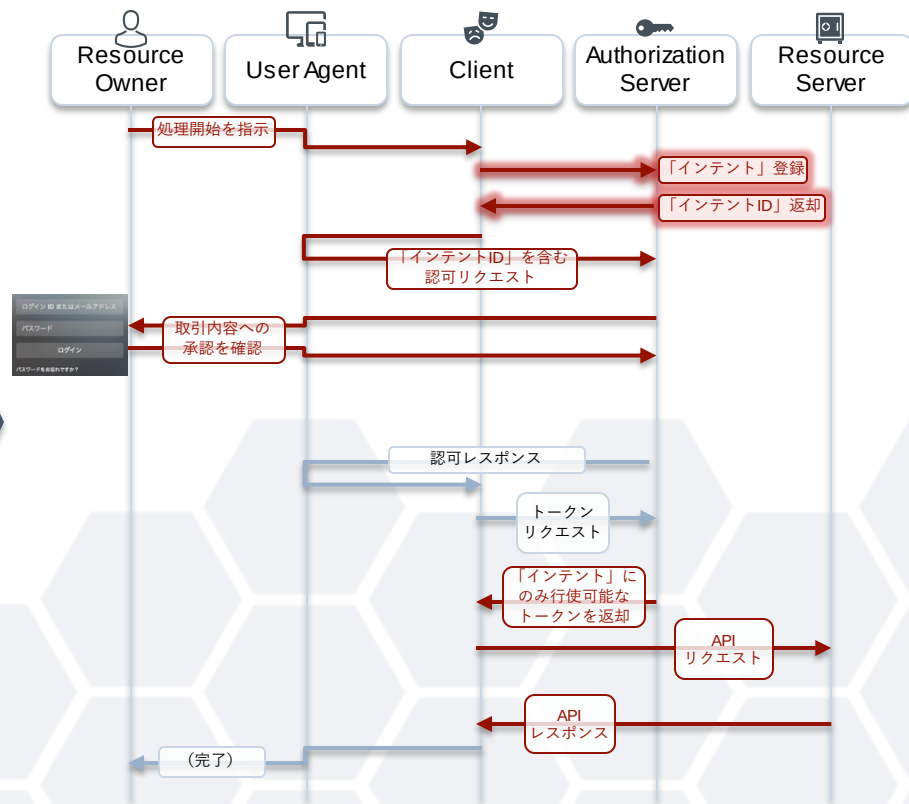
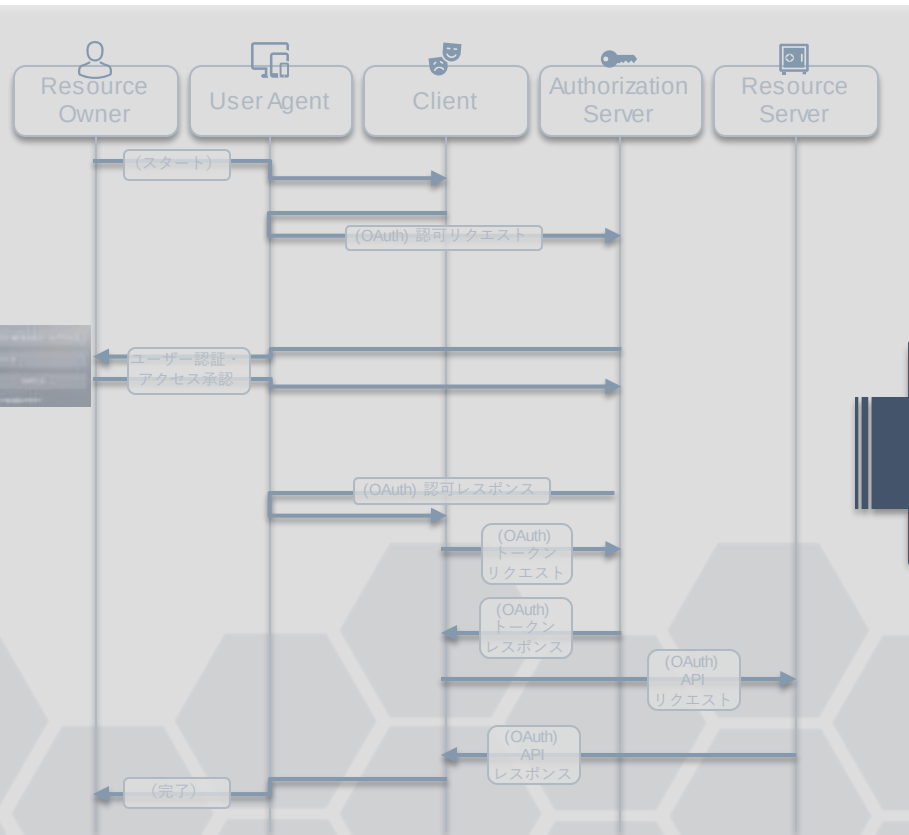
“LODGING INTENT PATTERN”

OAuth における「スコープ」 (scope)



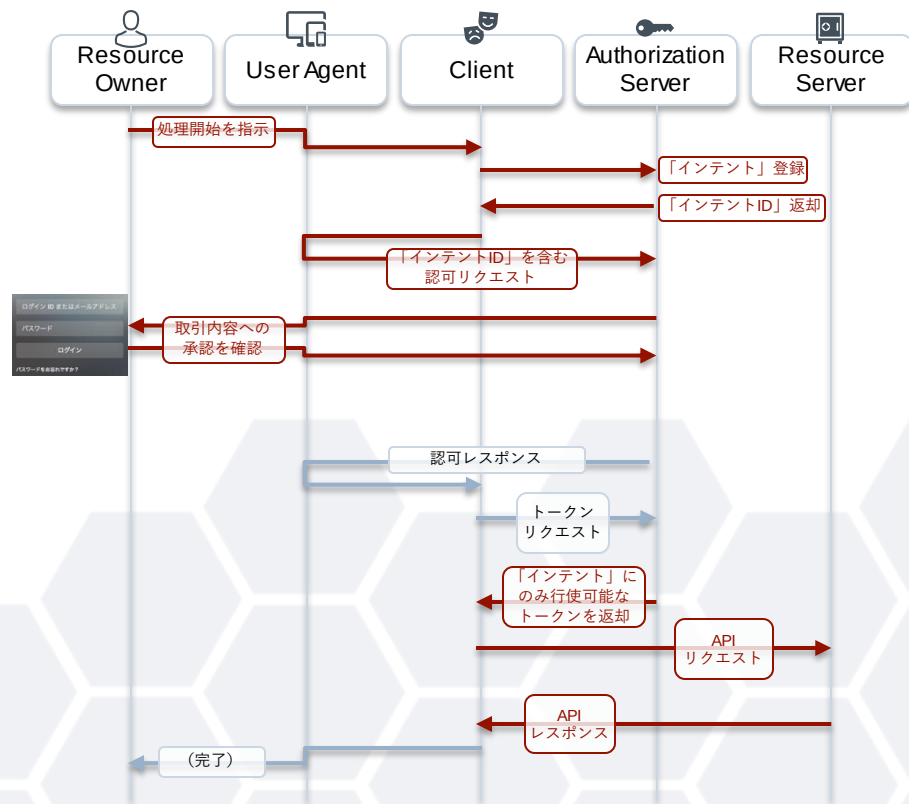
- リソースオーナー（ユーザー）がクライアントに移譲する権限の粒度
 - APIによってアクセス可能なデータの種類・範囲
(e.g. 口座情報、取引履歴)
 - APIによって実行可能な機能の範囲 (e.g. 参照、決済、送金)
- 「ある特定の取引」「同意した範囲でのデータ取得」のような粒度を表現するにはどうするか？

「インテント」の導入



インテントによる「取引単位」のアクセス認可

- サードパーティが銀行のAPIに、口座情報取得や決済指図伝達などの「インテント」を事前登録
- 登録された内容を利用者が銀行のWebサイトやアプリにて確認・承認
- 承認後、サードパーティが「インテント」の内容の実行を銀行のAPIにリクエスト



「インテント」の標準化

- FAPI WG が “Lodging Intent Pattern” に注目
<https://bitbucket.org/openid/fapi/issues/142/standardising-lodging-intent>
 - 事前にクライアントが認可サーバーに “intent” を登録し、返却されたその intent の「ハンドル」を認可リクエストに付加
 - UK Open Banking では essential claim、Berlin Group NextGenPSD2 では dynamic scope にて表現
- OAuth 2.0の認可リクエストを拡張する仕様として現在IETF OAuth WGにて議論が進行中
 - OAuth 2.0 Rich Authorization Requests
 - OAuth 2.0 Pushed Authorization Requests



Source: <https://www.slideshare.net/tkudo/osw2019/7>

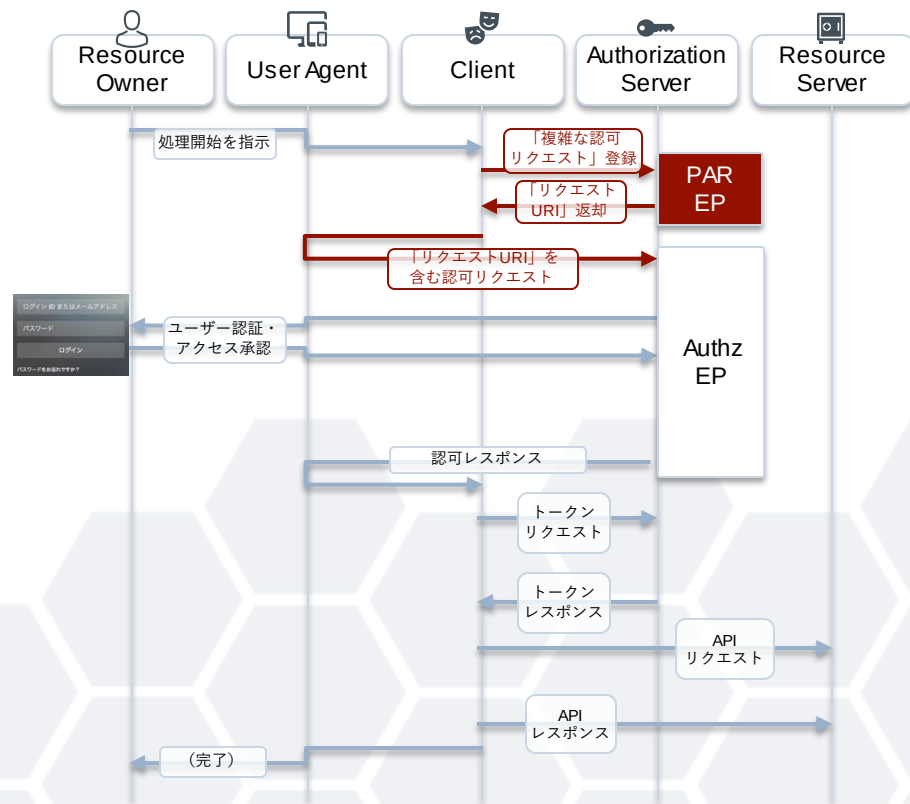


Source: <https://medium.com/oauth-2/rich-oauth-2-0-authorization-requests-87870e263ecb>

OAuth 2.0 Pushed Authorization Requests (PAR)

<https://datatracker.ietf.org/doc/draft-ietf-oauth-par/>

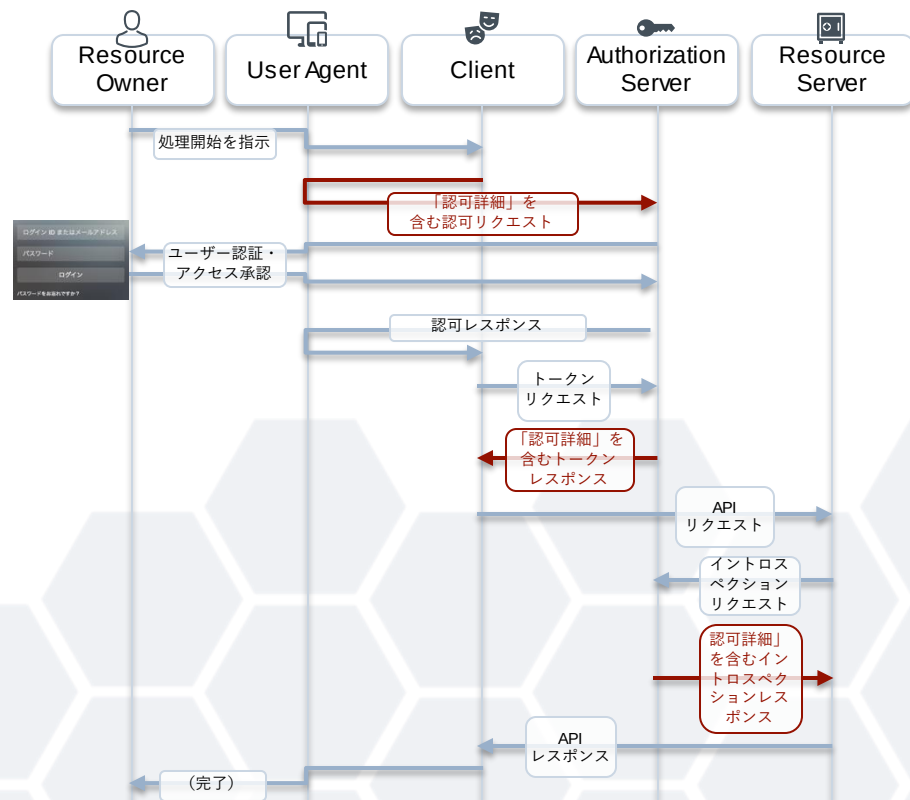
- PARエンドポイント（新規）
 - クライアントが認可サーバーに「複雑な認可リクエスト」を登録し、認可サーバーが「リクエストURI」を返却
 - 認可サーバーがクライアントを認証可能
- 認可エンドポイント（拡張）
 - クライアントが認可サーバーに、リクエストURIを含む「単純な認可リクエスト」を送信
 - `https://as.example.com/authorize?request_uri=<リクエストURI>&...`
 - ユーザーエージェントによって中継される情報を最小化



OAuth 2.0 Rich Authorization Requests (RAR)

<https://datatracker.ietf.org/doc/draft-ietf-oauth-rar/>

- “authorization_details” パラメーター (新規)
 - クライアントが認可サーバーに「詳細な認可リクエスト」を送信
 - `https://as.example.com/authorize?response_type=code&client_id=6493509078&scope=email&authorization_details=[{"type":"type0", "locations":["loc0","loc1"],"hello":"world"}]`
 - 認可サーバーはこの内容を、認可決定に用いると同時に、トークンレスポンスやイントロスペクションレスポンスとして、それぞれクライアントとリソースサーバーに提供



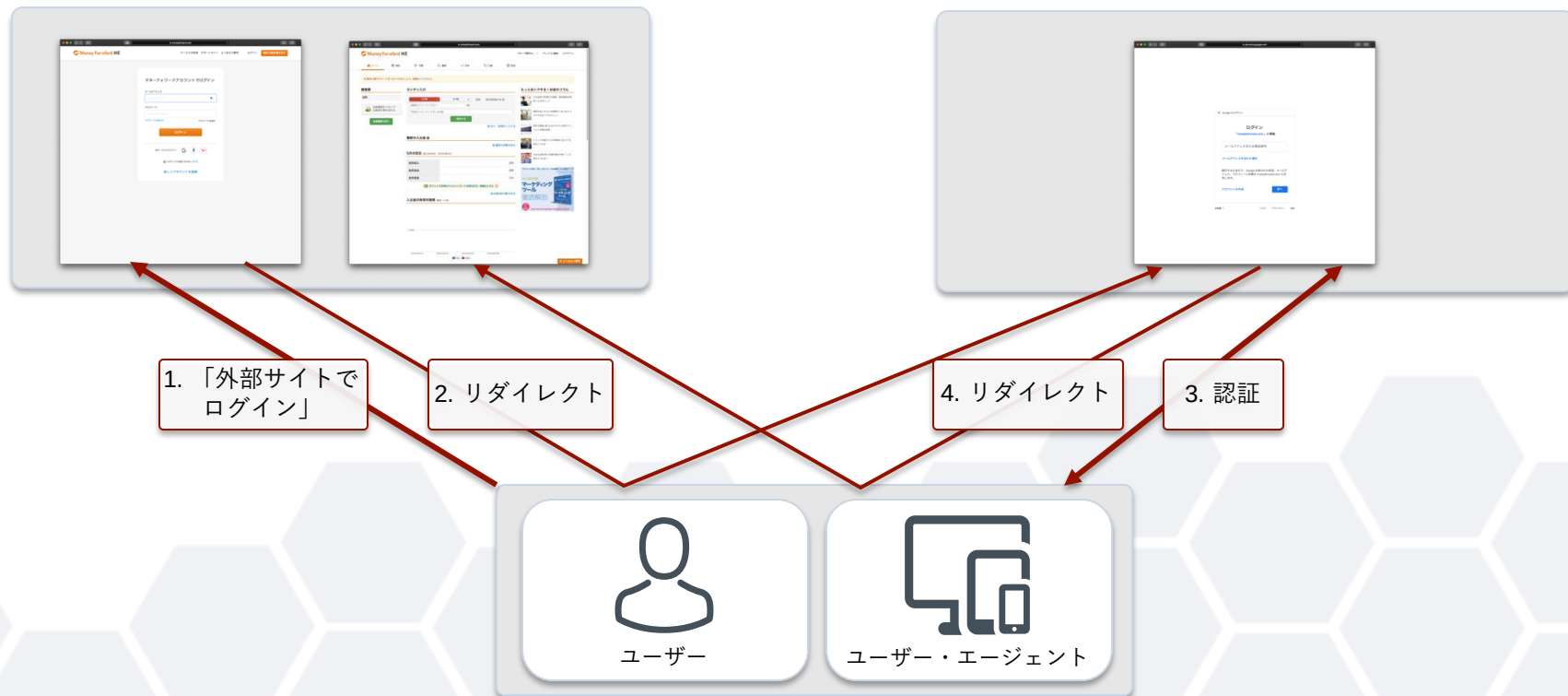
これからのOAuth/OIDC



Device Authorization Grant / CIBA

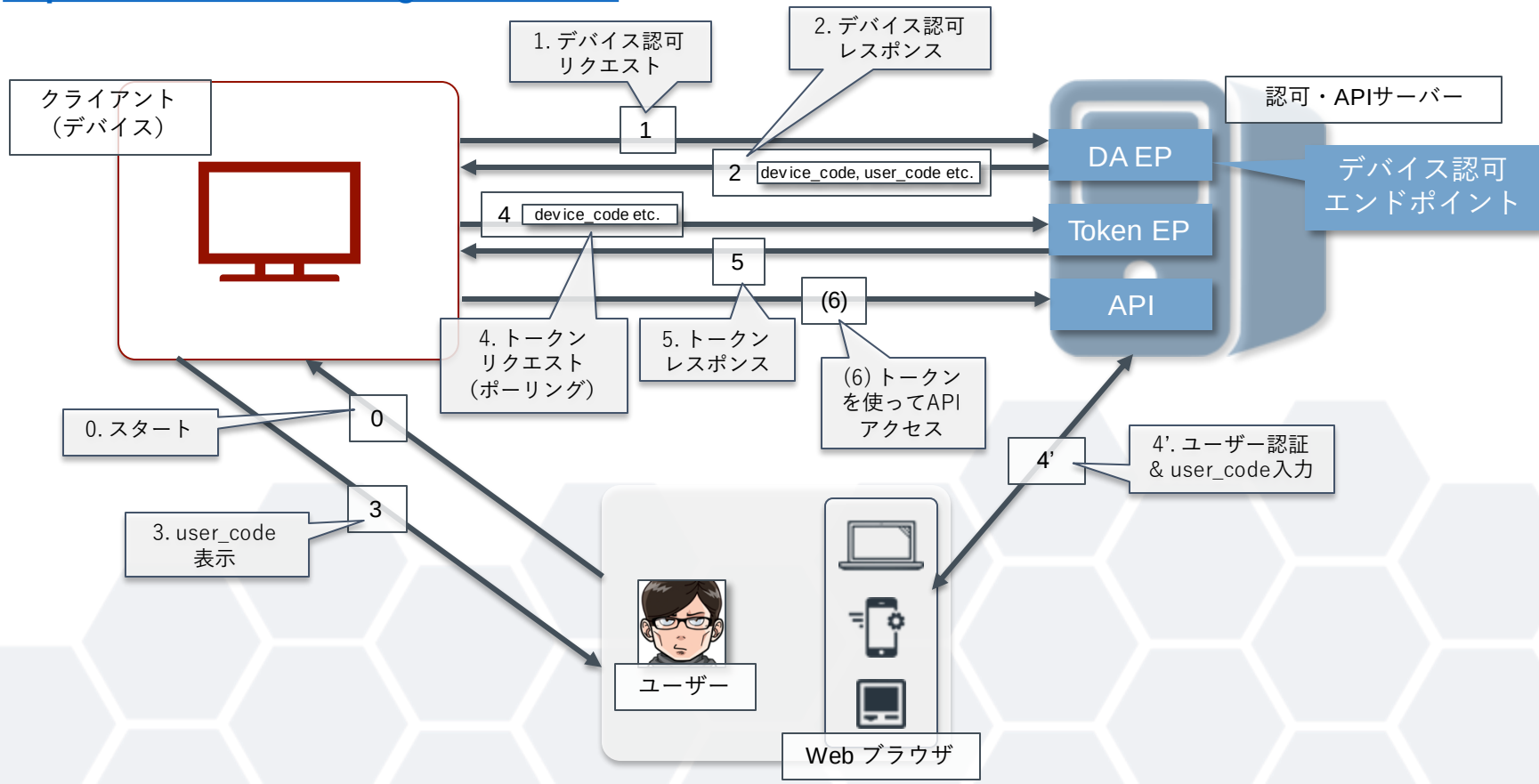
UTILIZING END-USER DEVICES

“Redirect Flow”

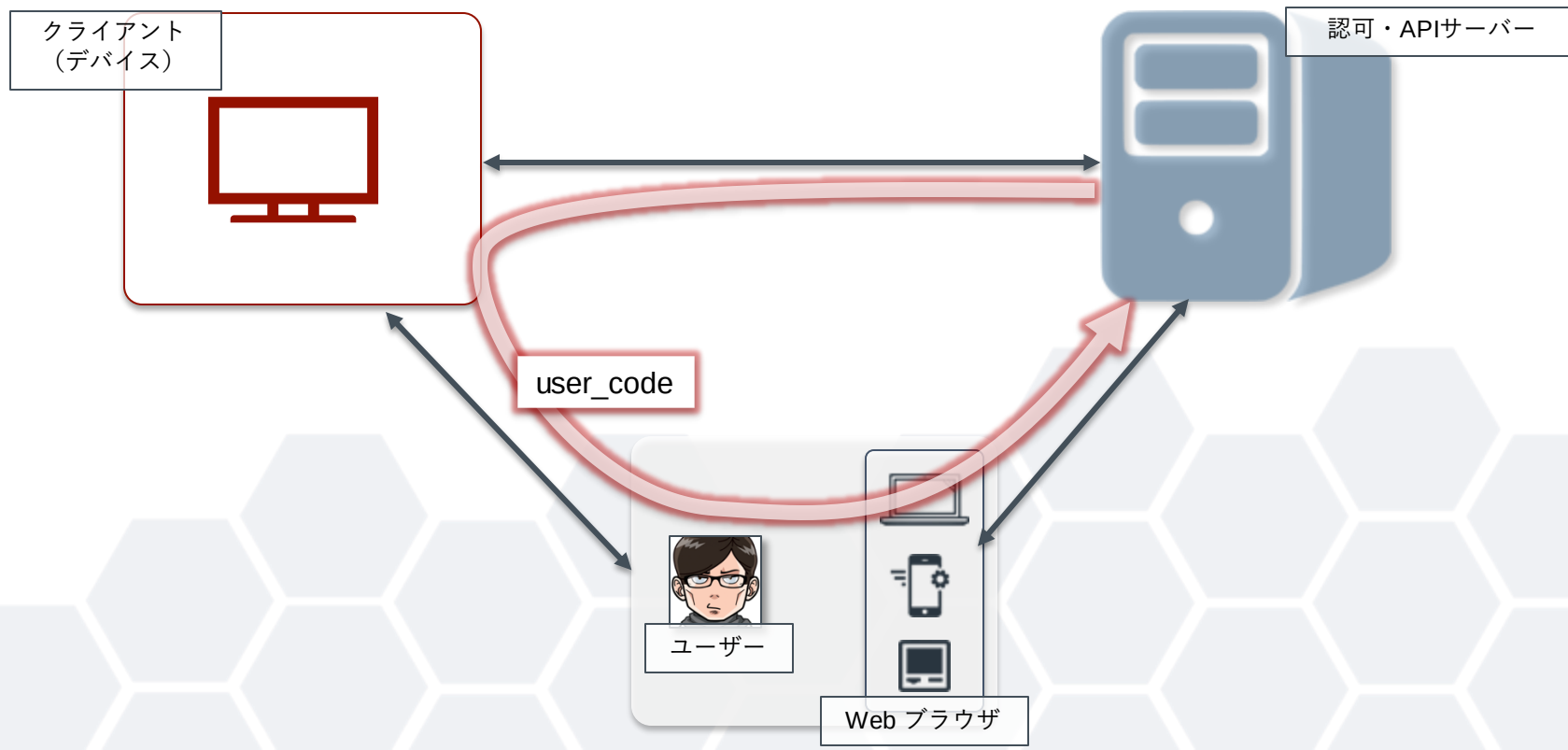


RFC 8628: OAuth 2.0 Device Authorization Grant

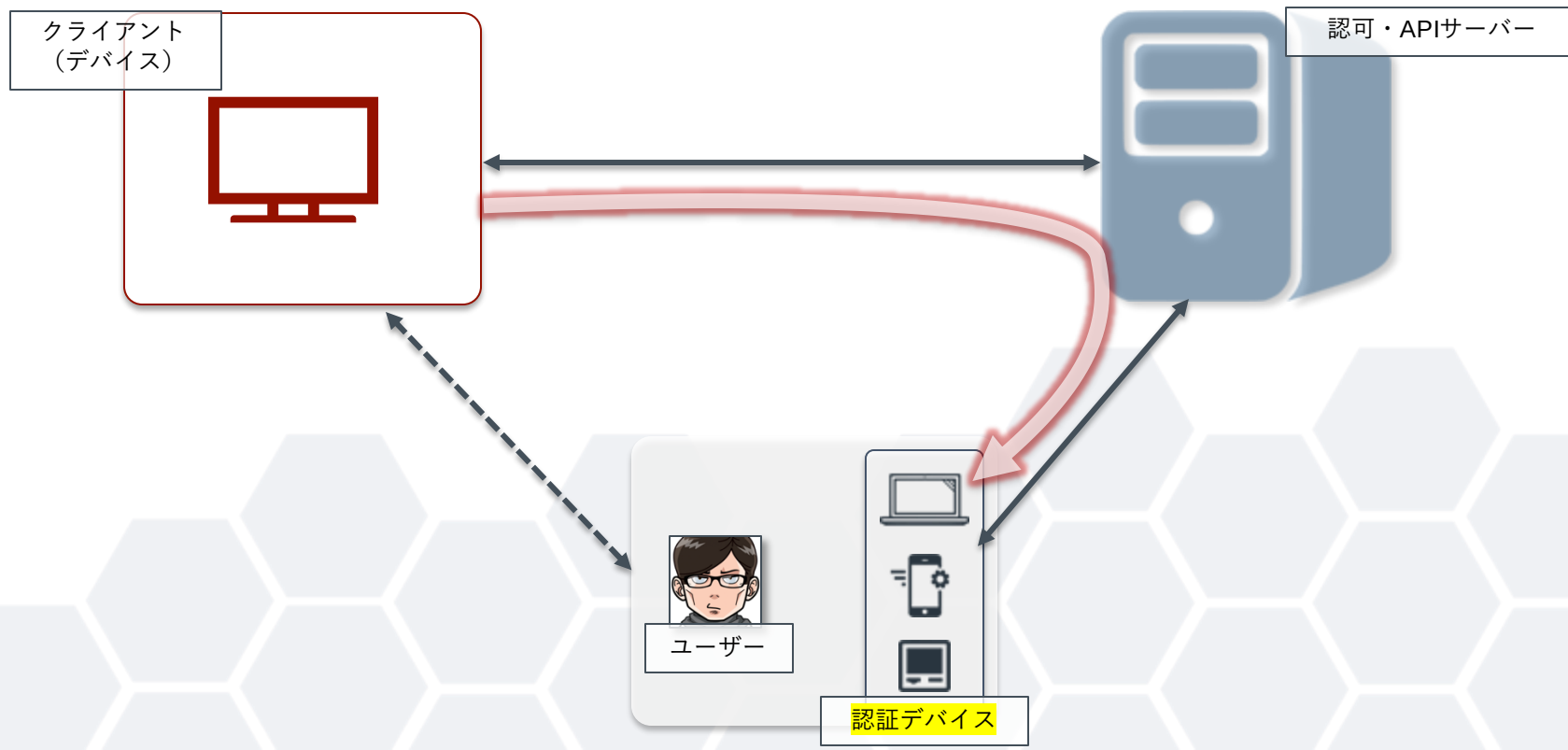
<https://datatracker.ietf.org/doc/rfc8628/>



ある意味では、ユーザーによる「リダイレクト」



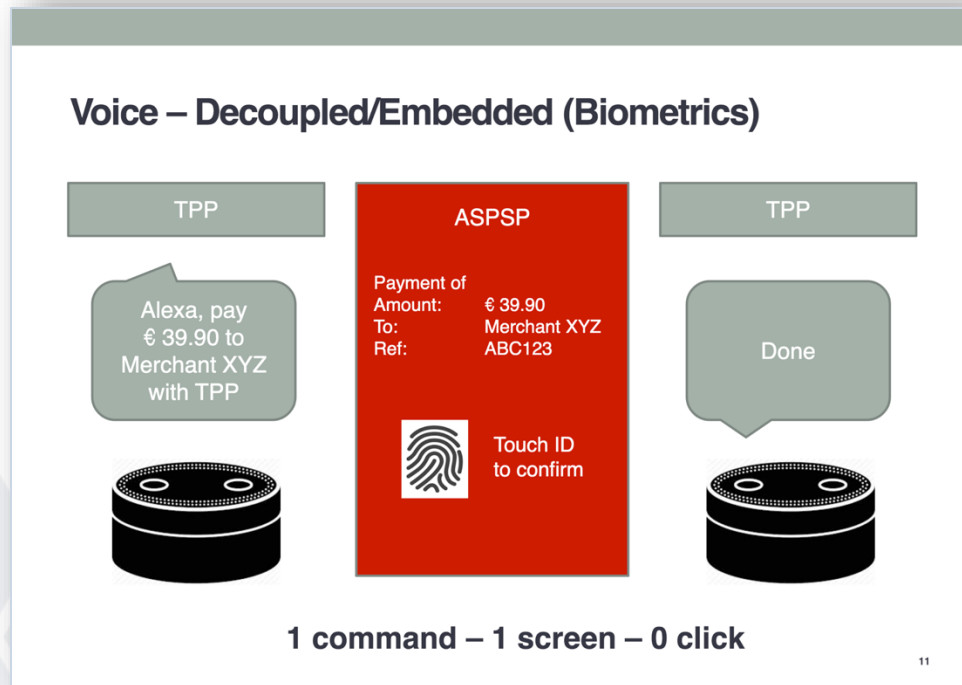
ユーザーを介さない認証・認可フローはどうか？



“Decoupled Flow”

- 「アレクサ、お店に
5,000円払って」

→ 手元のケータイに
決済サービスAppから
通知



“Decoupled Flow”

- TVショッピングとかで、
 - 視聴者が電話で購入希望を伝える
 - オペレーターに「ではお客さまのスマートフォンに注文内容をお送りします。ご確認をお願いします」と言われる
 - 視聴者は手元のスマートフォンにきた、決済サービスAppからの通知をみて承認する
 - 店舗への決済が行われる



“Decoupled Flow”

- 仮に [Ponta](#) の ID と決済できるとこのIDが紐づいてたら
 - コンビニ店員「Pontaカードカモン」
 - 客「はい」
 - 店員「XXX円です」
 - 客「CIBAPay(ダッサ)」
 - 店員「はい(おチ)」
 - 客のスマホ「(XXX円をローソンに支払います。よろしいですか?)」
 - 客「おk」
 - 店員「支払いおわた。レシートどうぞ」
- ぐらいは出来そうです



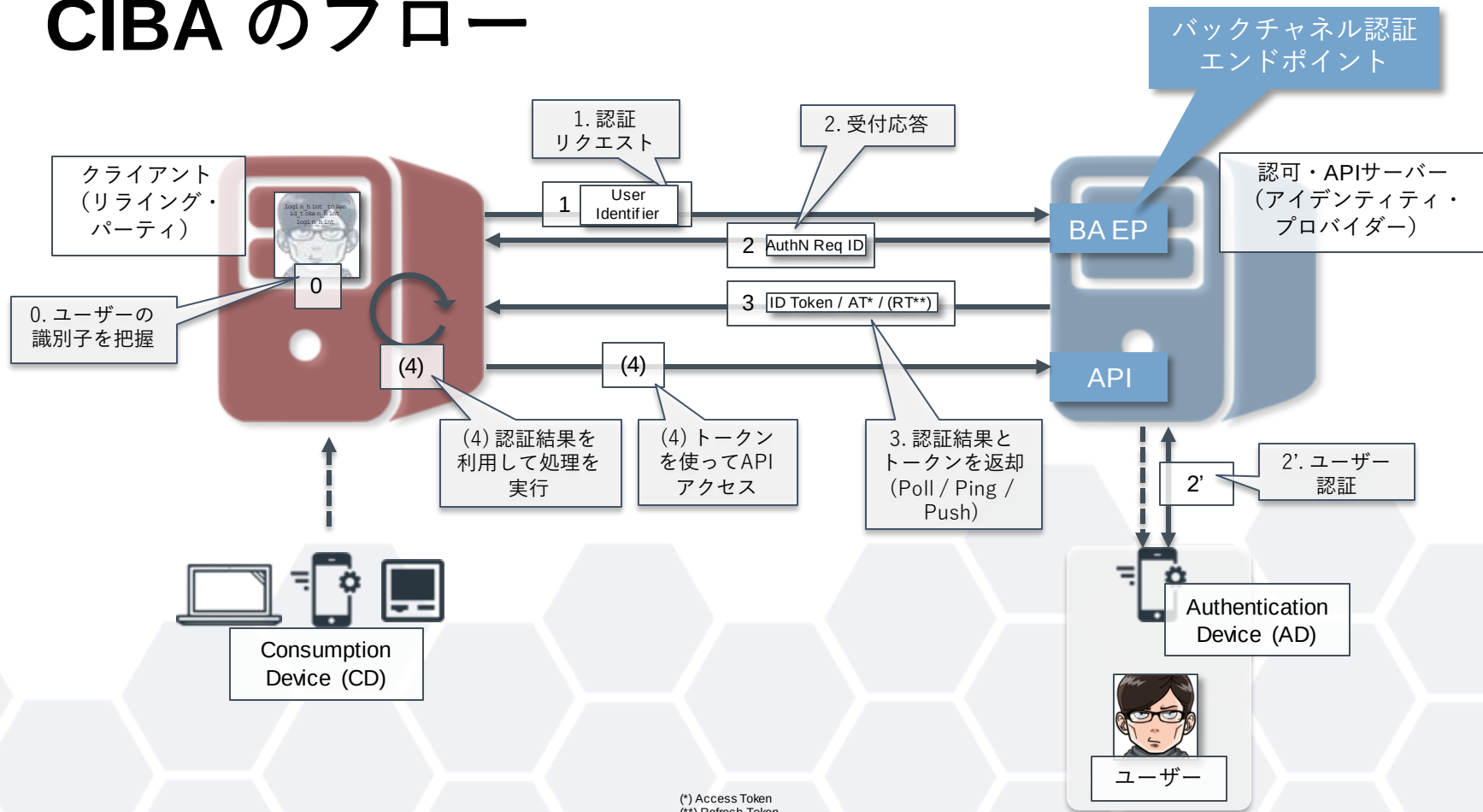
OpenID Connect Client Initiated Backchannel Authentication (CIBA) Flow - Core 1.0

https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html

- OpenID Foundation 傘下のWGにて策定中の新たなオープン仕様
 - 「サービスを利用するデバイス」と「認証を行うデバイス」を分離
- より良い顧客体験 (CX) の提供とセキュリティ強化に有用



CIBA のフロー



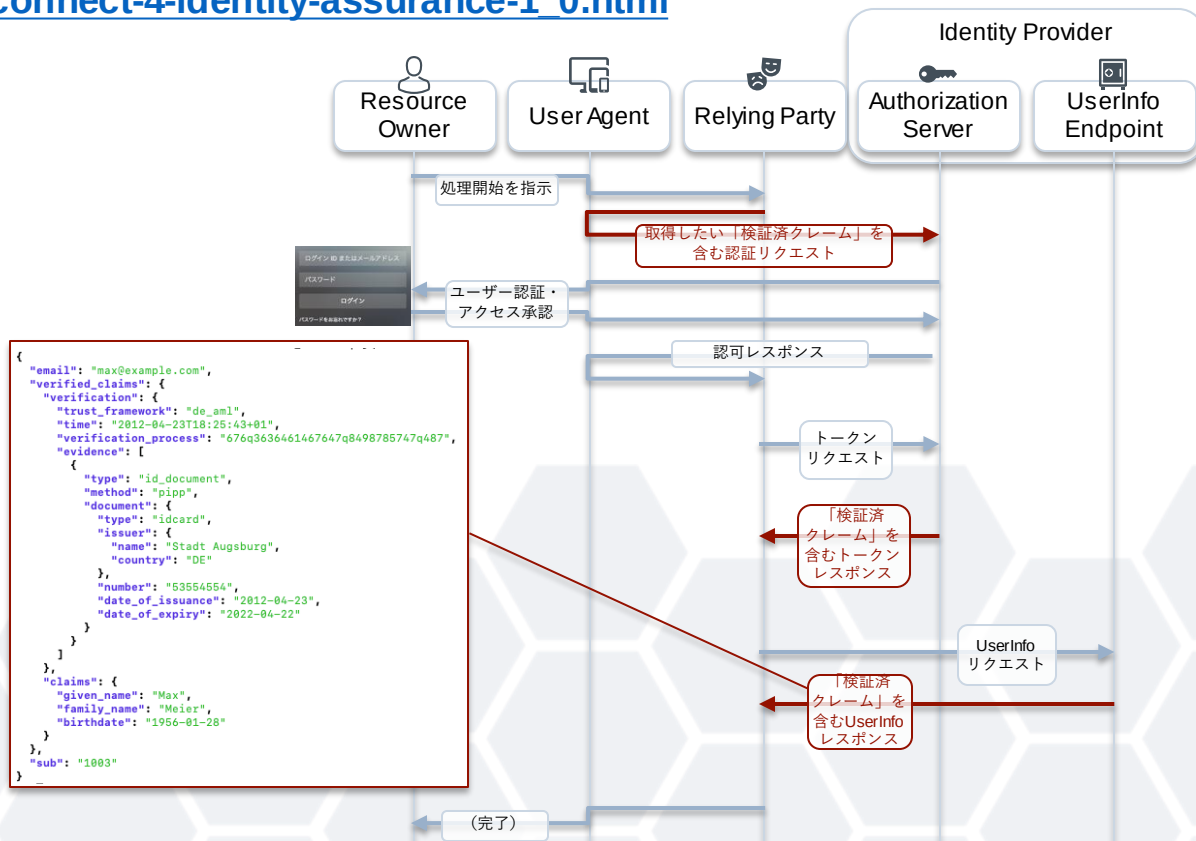
IDA

ASSURED IDENTITY

OpenID Connect for Identity Assurance 1.0

https://openid.net/specs/openid-connect-4-identity-assurance-1_0.html

- “verified_claims”（新規）
 - 「検証済クレーム」
 - OIDCのクレームとして定義
- RPは「取得したい検証済クレームとその内容」を、認証リクエストのclaimsパラメータを用いて指定
- IdPはIDトークンやUserInfoレスポンスとして、検証済クレームを返却



まとめ



まとめ

- 2010年代前半に確立したOAuth 2.0 / OpenID Connectは2020年の環境においても十分通用する
- 一方で現在も、活用領域の拡大・高度化につれて、関連仕様の策定や適用方法の更新が続いている
- 2015年以降の仕様・プラクティスは、新たなサービスや今後のアーキテクチャを考える上での参考として役立つ（かも）

Authleteは本日紹介した仕様のすべてに対応

https://www.authlete.com/ja/legal/spec_sheet/



仕様

機能	Version	説明
標準仕様	1.1 ~	• RFC 6749: The OAuth 2.0 Authorization Framework • RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage • RFC 7099: OAuth 2.0 Token Revocation • RFC 7636: Proof Key for Code Exchange by OAuth Public Clients • RFC 7662: OAuth 2.0 Token Introspection • OAuth 2.0 Multiple Response Type Encoding Practices • OAuth 2.0 Form Post Response Mode • OpenID Connect Core 1.0 • OpenID Connect Discovery 1.0
	2.0 ~	• RFC 7523: JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants • OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens • Financial grade API - Part 1: Read-Only API Security Profile • Financial grade API - Part 2: Read and Write API Security Profile • UK Open Banking Security Profile
	2.1 ~	• RFC 7591: OAuth 2.0 Dynamic Client Registration Protocol • RFC 7592: OAuth 2.0 Dynamic Client Registration Management Protocol • RFC 8628: OAuth 2.0 Device Authorization Grant • OpenID Connect Dynamic Client Registration 1.0 • OpenID Connect Client Initiated Backchannel Authentication Flow - Core 1.0 • Financial-grade API: JWT Secured Authorization Response Mode for OAuth 2.0 (AARM) • Financial grade API: Client Initiated Backchannel Authentication Profile
クライアント認証方式	1.1 ~	• OAuth 2.0 Pushed Authorization Requests • OAuth 2.0 Rich Authorization Requests • Resource Indicators for OAuth 2.0 • OpenID Connect for Identity Assurance 1.0
	2.0 ~	• none • client_secret_basic (RFC 6749) • client_secret_post (RFC 6749) • client_secret_jwt (RFC 7523) • private_key_jwt (RFC 7523) • tls_client_auth (MTLS) • self_signed_tls_client_auth (MTLS)
エンドポイント	1.1 ~	• Authorization Endpoint (RFC 6749) • Discovery Endpoint (OIDC Discovery 1.0) • Introspection Endpoint (RFC 7662) • NWK Set Endpoint (RFC 7517) • Revocation Endpoint (RFC 7099) • Token Endpoint (RFC 6749) • UserInfo Endpoint (OIDC Core 1.0)
	2.1 ~	• Backchannel Authentication Endpoint (CIBA Core 1.0) • Device Authorization Endpoint (RFC 8628)
	2.2 ~	• Pushed Authorization Request Endpoint (PAR)

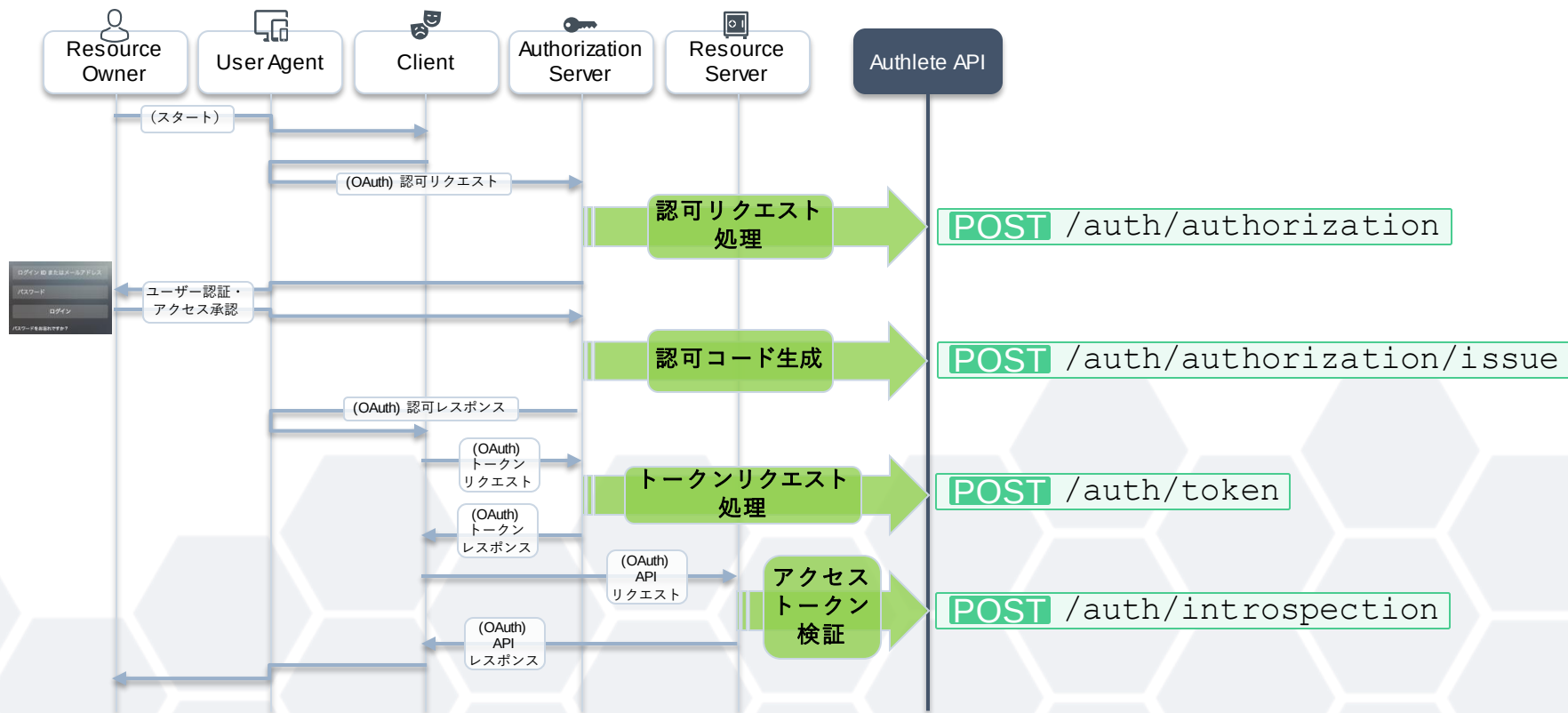
認可種別 (Grant Type)	1.1 ~	• authorization_code (RFC 6749) • implicit (RFC 6749) • password (RFC 6749) • client_credentials (RFC 6749) • refresh (RFC 6749)
	2.1 ~	• urn:ietf:params:oauth:grant-type:cbza (CIBA) • urn:ietf:params:oauth:grant-type:device_code (RFC 8628)
応答種別 (Response Type)	1.1 ~	• code (RFC 6749) • token (RFC 6749) • id_token (Multiple Response Type) • code token (Multiple Response Type) • code id_token (Multiple Response Type) • id_token token (Multiple Response Type) • id_token id_token token (Multiple Response Type) • none (Multiple Response Type)
	1.1 ~	• query (Multiple Response Type) • fragment (Multiple Response Type) • form_post (Form Post Response Mode)
応答形式 (Response Mode)	2.1 ~	• jwt (AARM) • query_jwt (AARM) • fragment_jwt (AARM) • form_post_jwt (AARM)
	1.1 ~	• H256 • H256A • H256B • H256C • H256D • H256E • H256F • H256G • H256H • H256I • H256J • H256K • H256L • H256M • H256N • H256O • H256P • H256Q • H256R • H256S • H256T • H256U • H256V • H256W • H256X • H256Y • H256Z • H256AA • H256AB • H256AC • H256AD • H256AE • H256AF • H256AG • H256AH • H256AI • H256AJ • H256AK • H256AL • H256AM • H256AN • H256AO • H256AP • H256AQ • H256AR • H256AS • H256AT • H256AU • H256AV • H256AW • H256AX • H256AY • H256AZ • H256BA • H256BB • H256BC • H256BD • H256BE • H256BF • H256BG • H256BH • H256BI • H256BJ • H256BK • H256BL • H256BM • H256BN • H256BO • H256BP • H256BQ • H256BR • H256BS • H256BT • H256BU • H256BV • H256BW • H256BX • H256BY • H256BZ • H256CA • H256CB • H256CC • H256CD • H256CE • H256CF • H256CG • H256CH • H256CI • H256CJ • H256CK • H256CL • H256CM • H256CN • H256CO • H256CP • H256CQ • H256CR • H256CS • H256CT • H256CU • H256CV • H256CW • H256CX • H256CY • H256CZ • H256DA • H256DB • H256DC • H256DD • H256DE • H256DF • H256DG • H256DH • H256DI • H256DJ • H256DK • H256DL • H256DM • H256DN • H256DO • H256DP • H256DQ • H256DR • H256DS • H256DT • H256DU • H256DV • H256DW • H256DX • H256DY • H256DZ • H256EA • H256EB • H256EC • H256ED • H256EE • H256EF • H256EG • H256EH • H256EI • H256EJ • H256EK • H256EL • H256EM • H256EN • H256EO • H256EP • H256EQ • H256ER • H256ES • H256ET • H256EU • H256EV • H256EW • H256EX • H256EY • H256EZ • H256FA • H256FB • H256FC • H256FD • H256FE • H256FF • H256FG • H256FH • H256FI • H256FJ • H256FK • H256FL • H256FM • H256FN • H256FO • H256FP • H256FQ • H256FR • H256FS • H256FT • H256FU • H256FV • H256FW • H256FX • H256FY • H256FZ • H256GA • H256GB • H256GC • H256GD • H256GE • H256GF • H256GG • H256GH • H256GI • H256GJ • H256GK • H256GL • H256GM • H256GN • H256GO • H256GP • H256GQ • H256GR • H256GS • H256GT • H256GU • H256GV • H256GW • H256GX • H256GY • H256GZ • H256HA • H256HB • H256HC • H256HD • H256HE • H256HF • H256HG • H256HH • H256HI • H256HJ • H256HK • H256HL • H256HM • H256HN • H256HO • H256HP • H256HQ • H256HR • H256HS • H256HT • H256HU • H256HV • H256HW • H256HX • H256HY • H256HZ • H256IA • H256IB • H256IC • H256ID • H256IE • H256IF • H256IG • H256IH • H256II • H256IJ • H256IK • H256IL • H256IM • H256IN • H256IO • H256IP • H256IQ • H256IR • H256IS • H256IT • H256IU • H256IV • H256IW • H256IX • H256IY • H256IZ • H256JA • H256JB • H256JC • H256JD • H256JE • H256JF • H256JG • H256JH • H256JI • H256JJ • H256JK • H256JL • H256JM • H256JN • H256JO • H256JP • H256JQ • H256JR • H256JS • H256JT • H256JU • H256JV • H256JW • H256JX • H256JY • H256JZ • H256KA • H256KB • H256KC • H256KD • H256KE • H256KF • H256KG • H256KH • H256KI • H256KJ • H256KL • H256KM • H256KN • H256KO • H256KP • H256KQ • H256KR • H256KS • H256KT • H256KU • H256KV • H256KW • H256KX • H256KY • H256KZ • H256LA • H256LB • H256LC • H256LD • H256LE • H256LF • H256LG • H256LH • H256LI • H256LJ • H256LK • H256LL • H256LM • H256LN • H256LO • H256LP • H256LQ • H256LR • H256LS • H256LT • H256LU • H256LV • H256LW • H256LX • H256LY • H256LZ • H256MA • H256MB • H256MC • H256MD • H256ME • H256MF • H256MG • H256MH • H256MI • H256MJ • H256MK • H256ML • H256MM • H256MN • H256MO • H256MP • H256MQ • H256MR • H256MS • H256MT • H256MU • H256MV • H256MW • H256MX • H256MY • H256MZ • H256NA • H256NB • H256NC • H256ND • H256NE • H256NF • H256NG • H256NH • H256NI • H256NJ • H256NK • H256NL • H256NM • H256NN • H256NO • H256NP • H256NQ • H256NR • H256NS • H256NT • H256NU • H256NV • H256NW • H256NX • H256NY • H256NZ • H256OA • H256OB • H256OC • H256OD • H256OE • H256OF • H256OG • H256OH • H256OI • H256OJ • H256OK • H256OL • H256OM • H256ON • H256OO • H256OP • H256OQ • H256OR • H256OS • H256OT • H256OU • H256OV • H256OW • H256OX • H256OY • H256OZ • H256PA • H256PB • H256PC • H256PD • H256PE • H256PF • H256PG • H256PH • H256PI • H256PJ • H256PK • H256PL • H256PM • H256PN • H256PO • H256PP • H256PQ • H256PR • H256PS • H256PT • H256PU • H256PV • H256PW • H256PX • H256PY • H256PZ • H256QA • H256QB • H256QC • H256QD • H256QE • H256QF • H256QG • H256QH • H256QI • H256QJ • H256QK • H256QL • H256QM • H256QN • H256QO • H256QP • H256QQ • H256QR • H256QS • H256QT • H256QU • H256QV • H256QW • H256QX • H256QY • H256QZ • H256RA • H256RB • H256RC • H256RD • H256RE • H256RF • H256RG • H256RH • H256RI • H256RJ • H256RK • H256RL • H256RM • H256RN • H256RO • H256RP • H256RQ • H256RR • H256RS • H256RT • H256RU • H256RV • H256RW • H256RX • H256RY • H256RZ • H256SA • H256SB • H256SC • H256SD • H256SE • H256SF • H256SG • H256SH • H256SI • H256SJ • H256SK • H256SL • H256SM • H256SN • H256SO • H256SP • H256SQ • H256SR • H256SS • H256ST • H256SU • H256SV • H256SW • H256SX • H256SY • H256SZ • H256TA • H256TB • H256TC • H256TD • H256TE • H256TF • H256TG • H256TH • H256TI • H256TJ • H256TK • H256TL • H256TM • H256TN • H256TO • H256TP • H256TQ • H256TR • H256TS • H256TT • H256TU • H256TV • H256TW • H256TX • H256TY • H256TZ • H256UA • H256UB • H256UC • H256UD • H256UE • H256UF • H256UG • H256UH • H256UI • H256UJ • H256UK • H256UL • H256UM • H256UN • H256UO • H256UP • H256UQ • H256UR • H256US • H256UT • H256UU • H256UV • H256UW • H256UX • H256UY • H256UZ • H256VA • H256VB • H256VC • H256VD • H256VE • H256VF • H256VG • H256VH • H256VI • H256VJ • H256VK • H256VL • H256VM • H256VN • H256VO • H256VP • H256VQ • H256VR • H256VS • H256VT • H256VU • H256VV • H256VW • H256VX • H256VY • H256VZ • H256WA • H256WB • H256WC • H256WD • H256WE • H256WF • H256WG • H256WH • H256WI • H256WJ • H256WK • H256WL • H256WM • H256WN • H256WO • H256WP • H256WQ • H256WR • H256WS • H256WT • H256WU • H256WV • H256WW • H256WX • H256WY • H256WZ • H256XA • H256XB • H256XC • H256XD • H256XE • H256XF • H256XG • H256XH • H256XI • H256XJ • H256XK • H256XL • H256XM • H256XN • H256XO • H256XP • H256XQ • H256XR • H256XS • H256XT • H256XU • H256XV • H256XW • H256XX • H256XY • H256XZ • H256YA • H256YB • H256YC • H256YD • H256YE • H256YF • H256YG • H256YH • H256YI • H256YJ • H256YK • H256YL • H256YM • H256YN • H256YO • H256YP • H256YQ • H256YR • H256YS • H256YT • H256YU • H256YV • H256YW • H256YX • H256YY • H256YZ • H256ZA • H256ZB • H256ZC • H256ZD • H256ZE • H256ZF • H256ZG • H256ZH • H256ZI • H256ZJ • H256ZK • H256ZL • H256ZM • H256ZN • H256ZO • H256ZP • H256ZQ • H256ZR • H256ZS • H256ZT • H256ZU • H256ZV • H256ZW • H256ZX • H256ZY • H256ZZ

Authlete 固有

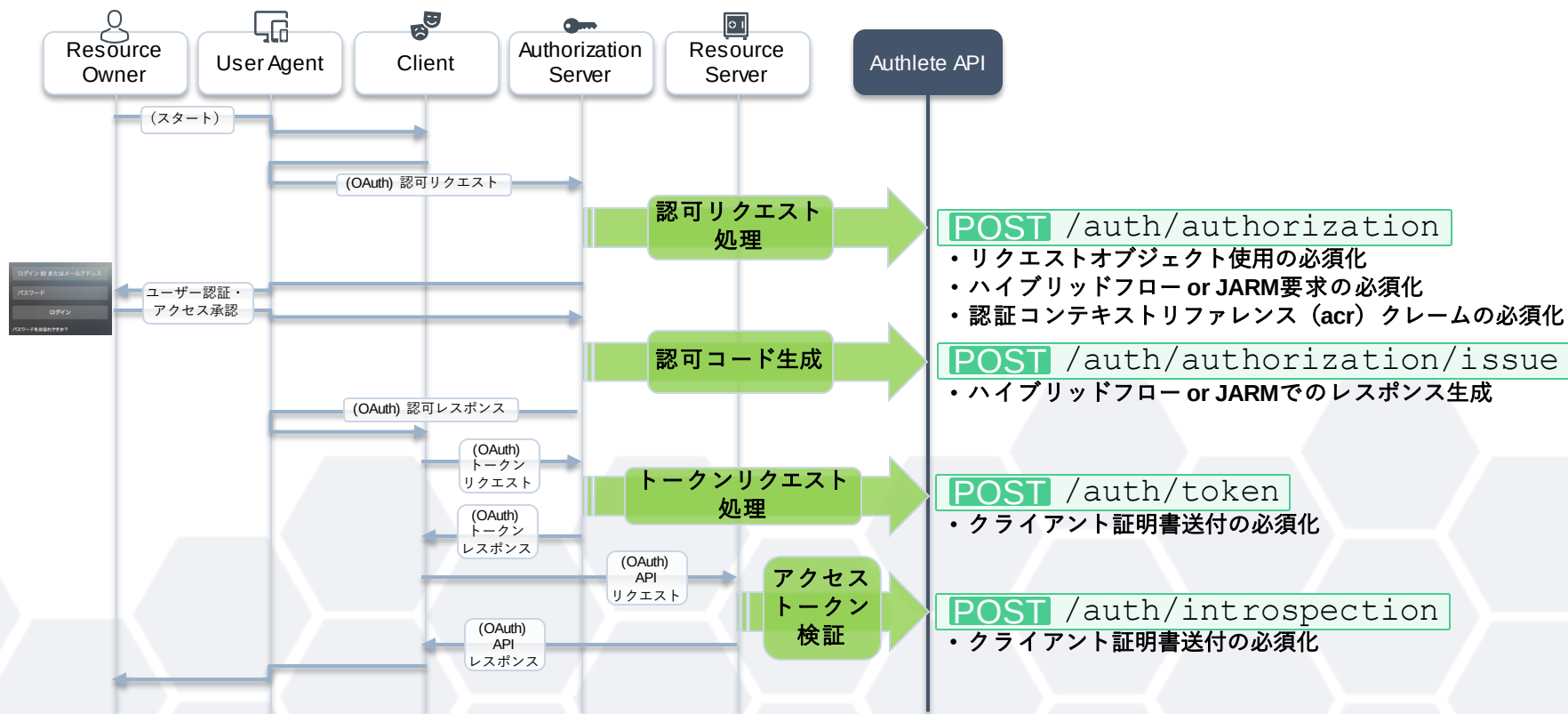
トークン有効期限設
定

AUTHLETE

プロトコル処理とトークン管理をAuthleteが代行



AuthleteにおけるFAPI Part 2サポートの例



リソース

- 弊社川崎 @darutk の記事
 - <https://qiita.com/TakahikoKawasaki>
- Authlete API チュートリアル
 - <https://www.authlete.com/ja/developers/tutorial/>
- プレゼンテーション資料
 - <https://www.slideshare.net/tkudo>
 - <https://speakerdeck.com/takahikokawasaki/>

Thank You

www.authlete.com

www.linkedin.com/in/tatsuokudo

