AUTHLETE

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

AUTHLETE

OAuth 2.0 and OpenID Connect in the early 2010s

OAuth 2.0とは

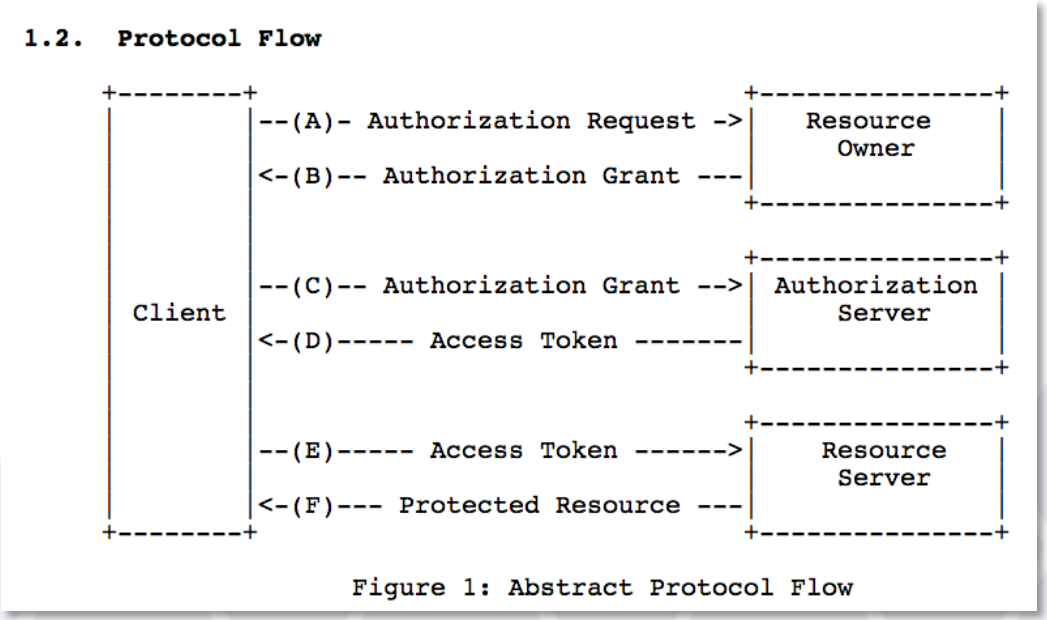
- トークンを用いたAPIアクセス権移譲のフレームワーク
  - RFC 6749: トークン付与プロセスと基本的な付与フロー
    （認可コード、クライアントクレデンシャルなど）
- 広範なユースケースに適用可能
  - トークン付与方式やトークン形式の拡張
  - ユーザー、サーバー、クライアント、デバイス、…

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
「認可グラント」を提供）

「認可グラント」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）



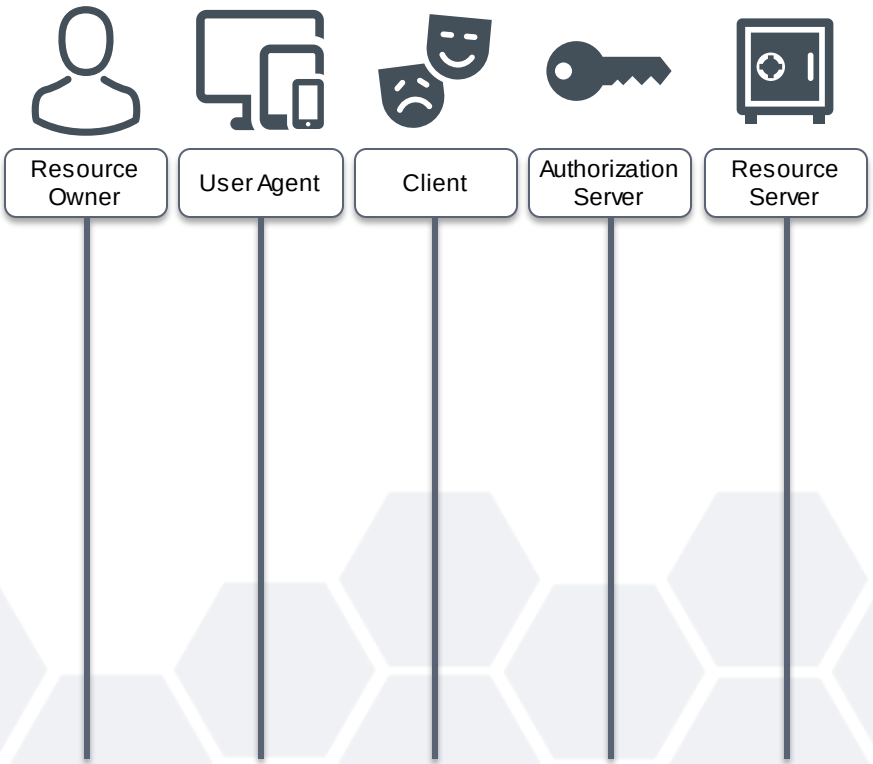
1.2. Protocol Flow

```
+--------+                                +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Figure 1: Abstract Protocol Flow

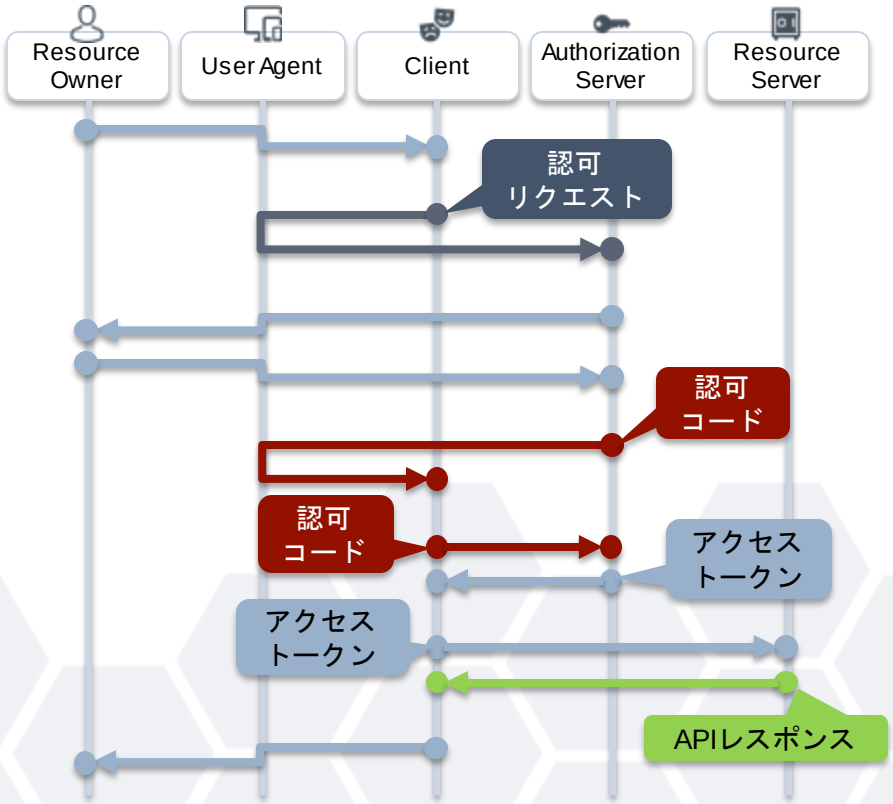
Source: RFC 6749 1.2. Protocol Flow https://tools.ietf.org/html/rfc6749

“OAuth Dance”



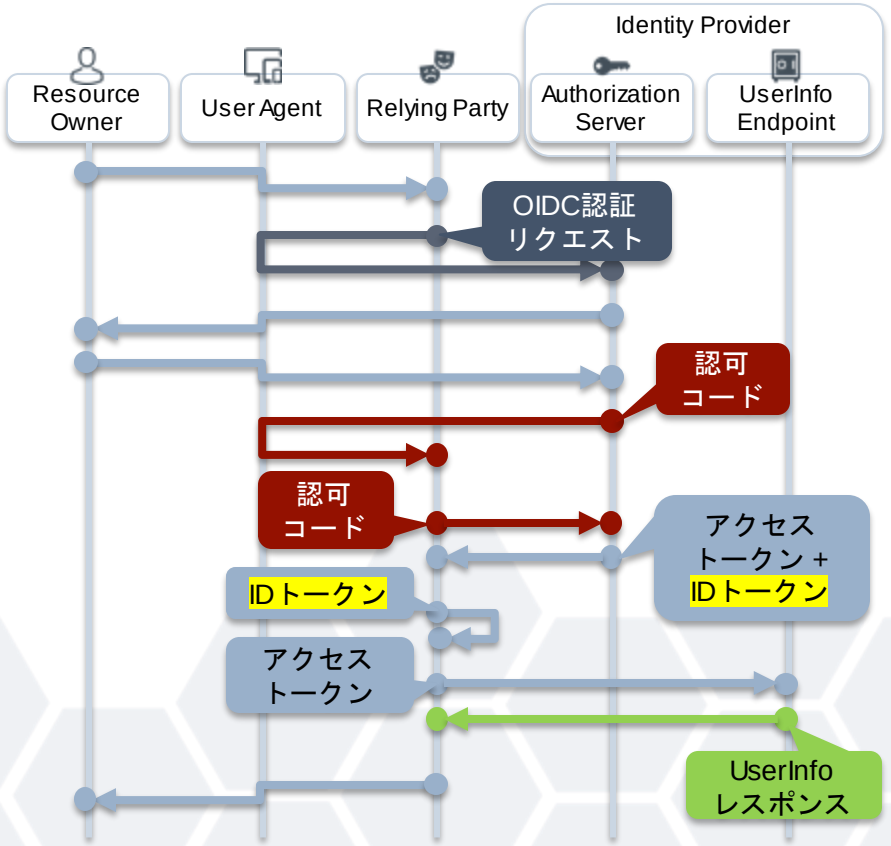
- **Resource Owner**
  e.g. end user
- **User Agent**
  e.g. Web browser
- **Client**
  e.g. Web application using APIs
- **Authorization Server**
  e.g. user authentication server
- **Resource Server**
  e.g. API server

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
- （アクセストークンつきのAPIリクエストを受け付けてレスポンスを返す）

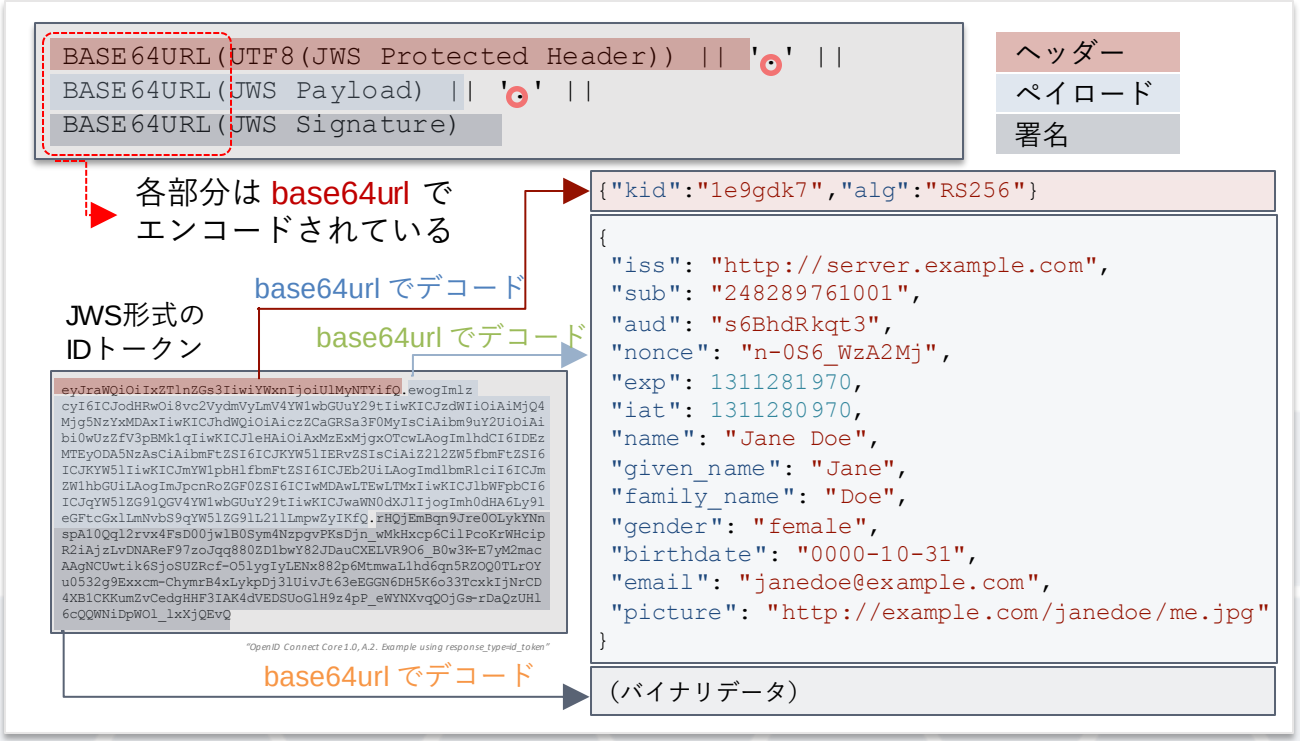
OpenID Connect 認可コードフロー



- 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

JWT (JSON Web Token) を用いたトークン表現

- 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

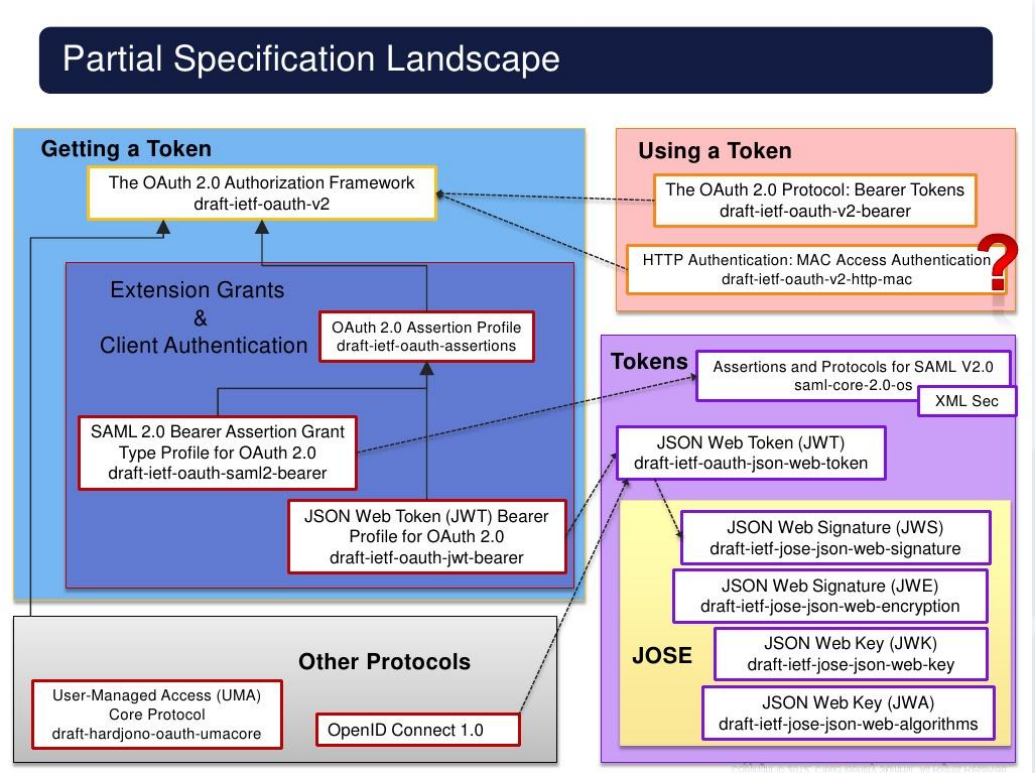


JWS Signed JWT の例

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

AUTHLETE

OAuth/OIDC in 2020

AUTHLETE
Experts in API Authorization

PKCE / Resource Indicators / MTLS / private_key_jwt

OAUTH 2.0 SECURITY BCP

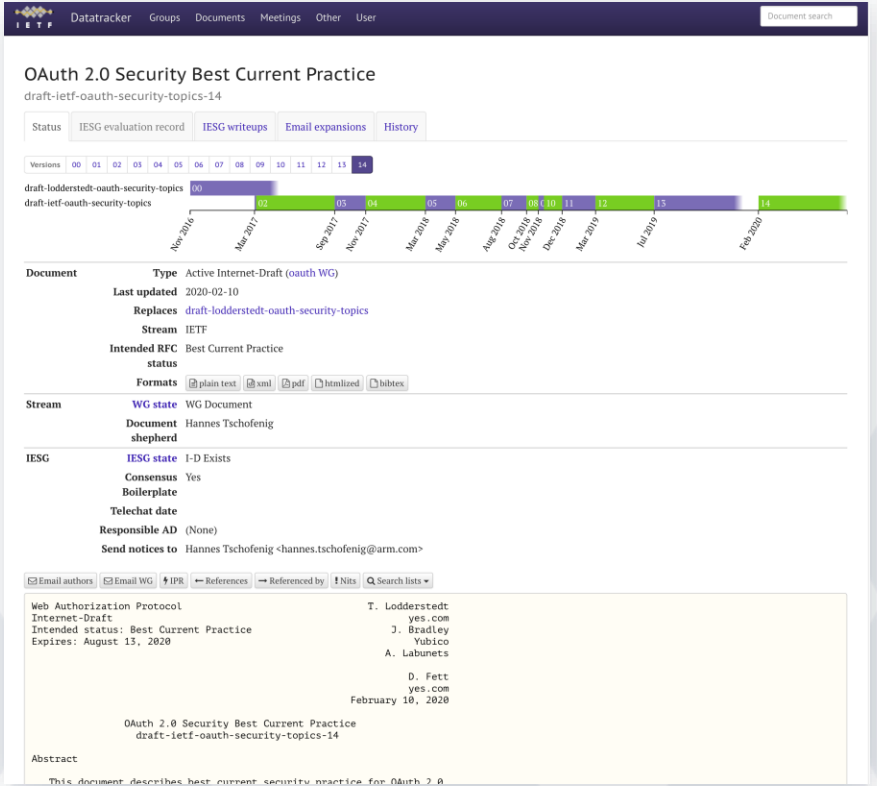
OAuth 2.0 Security Best Current Practice (BCP)

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/

AUTHLETE

OAuth 2.0 Security BCPの構成

Table of Contents

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

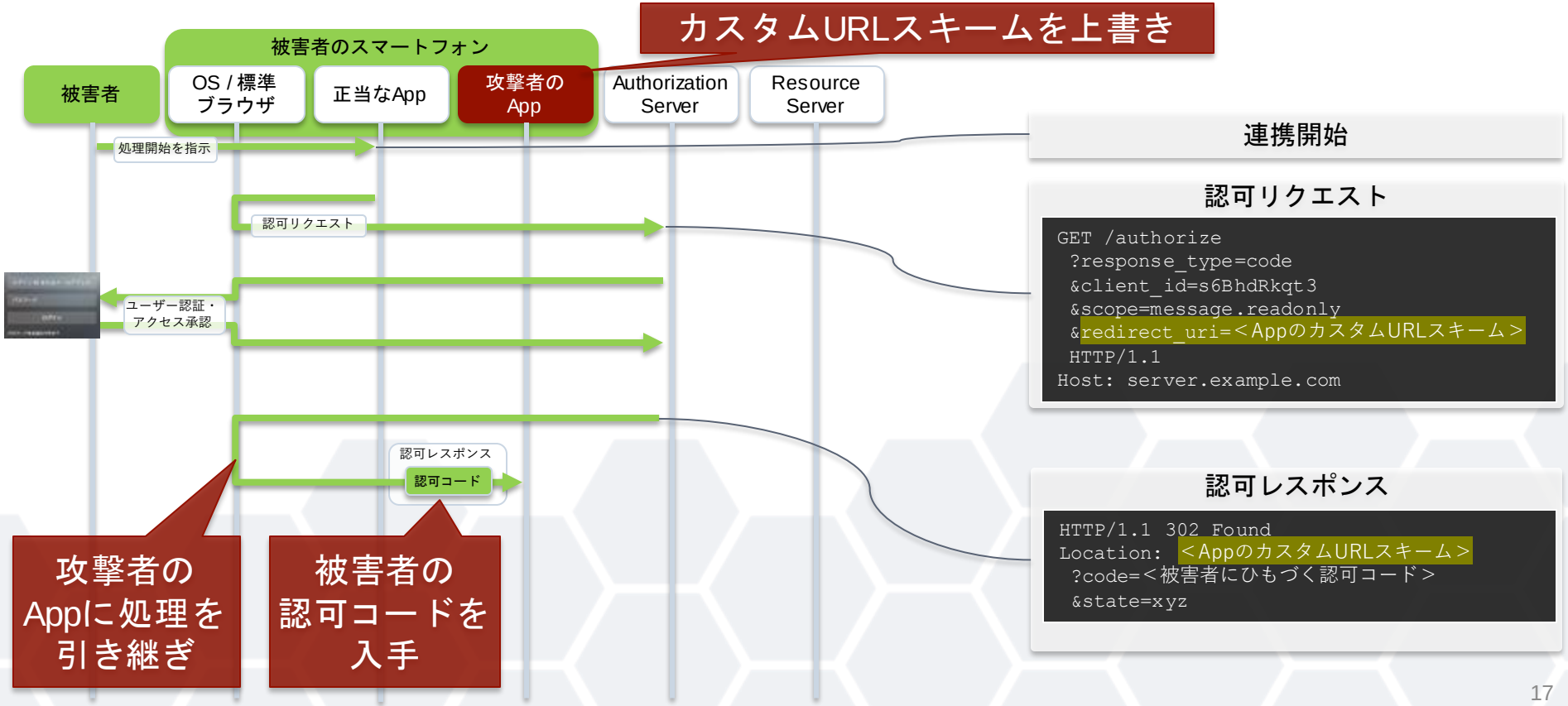
2.1. リダイレクトベースのフローの保護

- 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

AUTHLETE

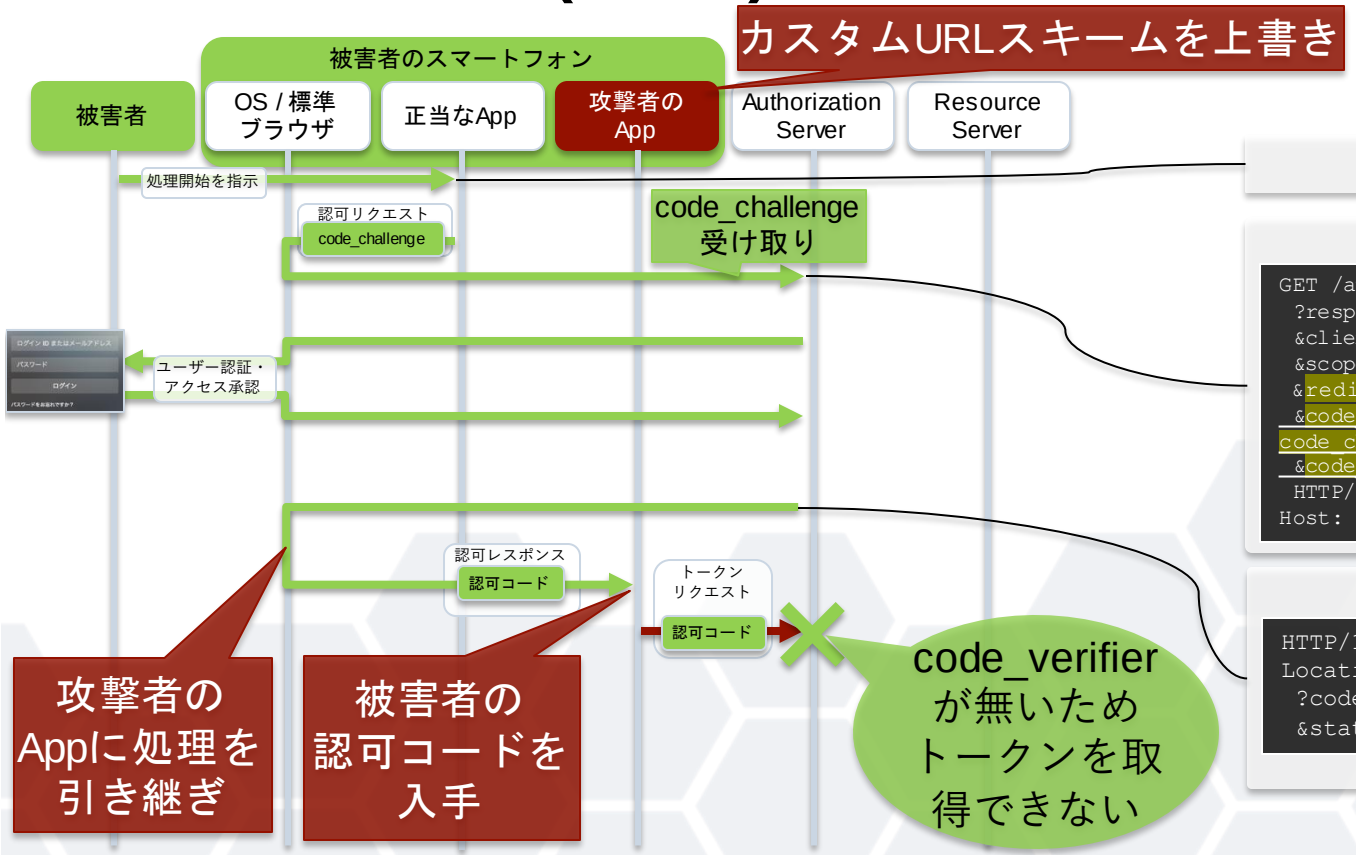
Native Appsにおける戻し先の乗っ取り

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



AUTHLETE

RFC 7636: Proof Key for Code Exchange by OAuth Public Clients (PKCE) https://datatracker.ietf.org/doc/rfc7636/



連携開始

認可リクエスト

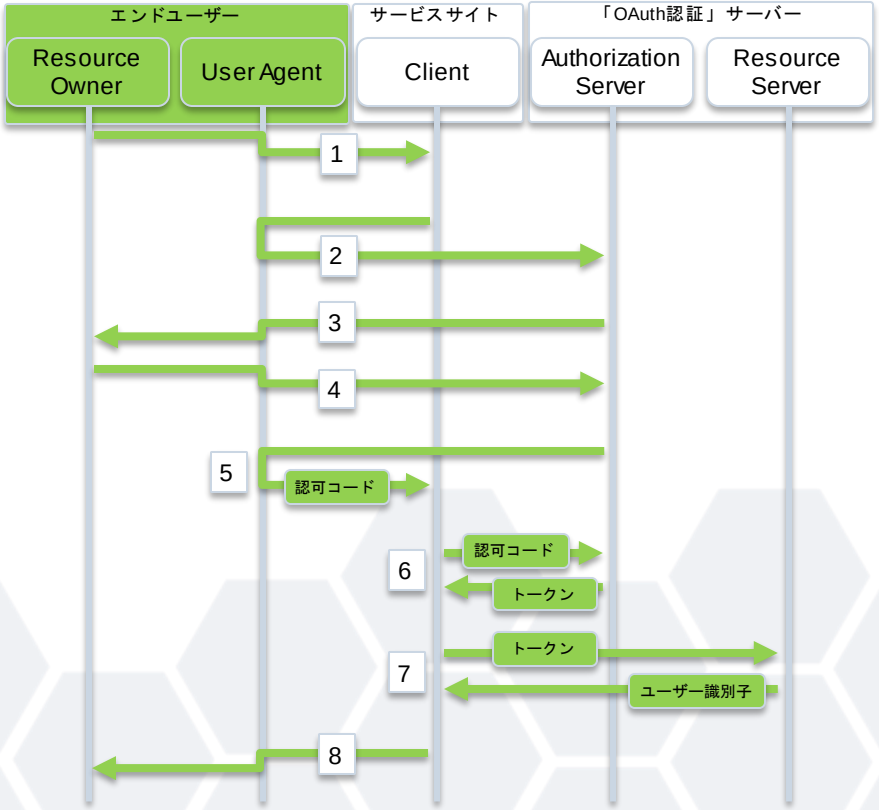
```
GET /authorize
 ?response_type=code
 &client_id=s6BhdRkqt3
 &scope=message.readonly
 &redirect_uri=<Appのカスタムurlスキーム>
 &code_challenge=<code_verifierから
code_challenge_methodに従って生成した値>
 &code_challenge_method=S256
 HTTP/1.1
Host: server.example.com
```

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
```

CSRF +「OAuth認証」

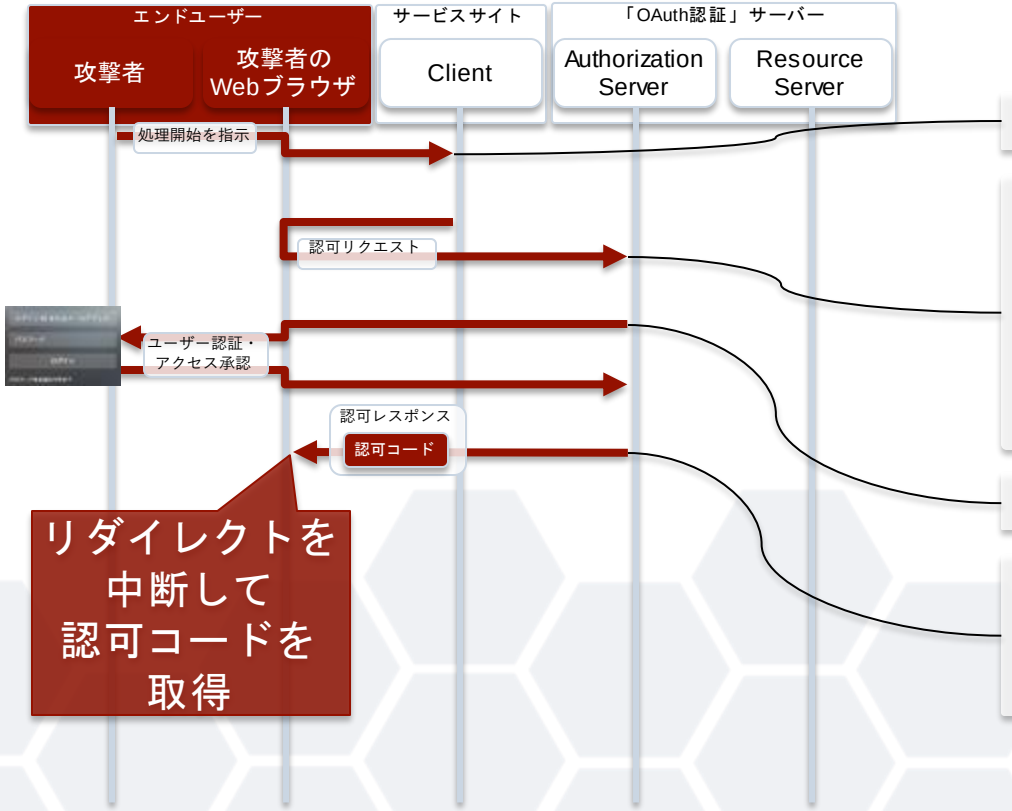
アクセストークンを用いてユーザー識別子を取得



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



「IDひもづけ開始」

認可リクエスト

```
GET /authorize
 ?response_type=code
 &client_id=s6BhdRkqt3
 &scope=message.readonly
 &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
 HTTP/1.1
Host: server.example.com
```

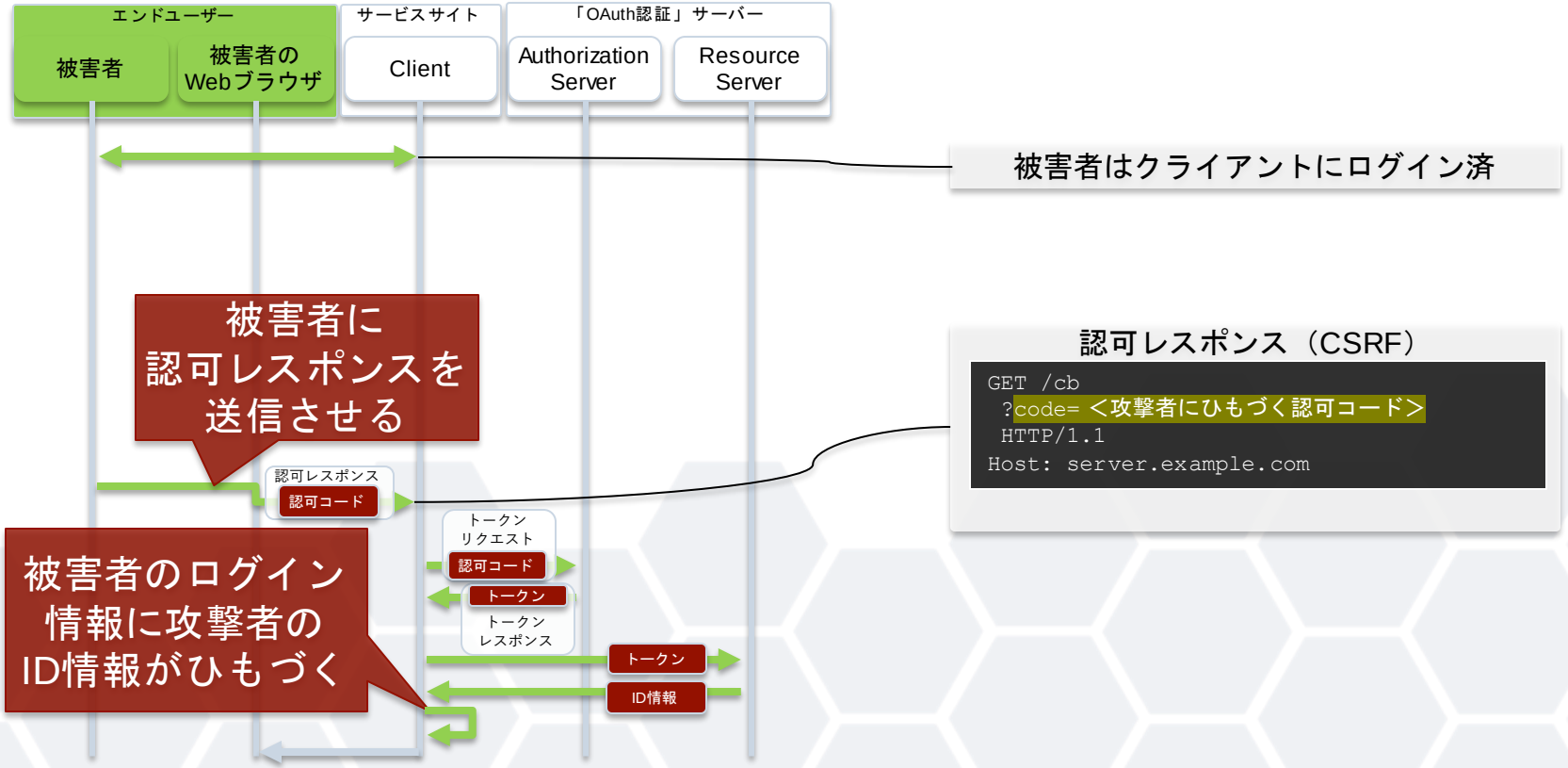
攻撃者自身としてユーザー認証 & 同意確認

認可レスポンス

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb
 ?code=<攻撃者にひもづく認可コード>
```

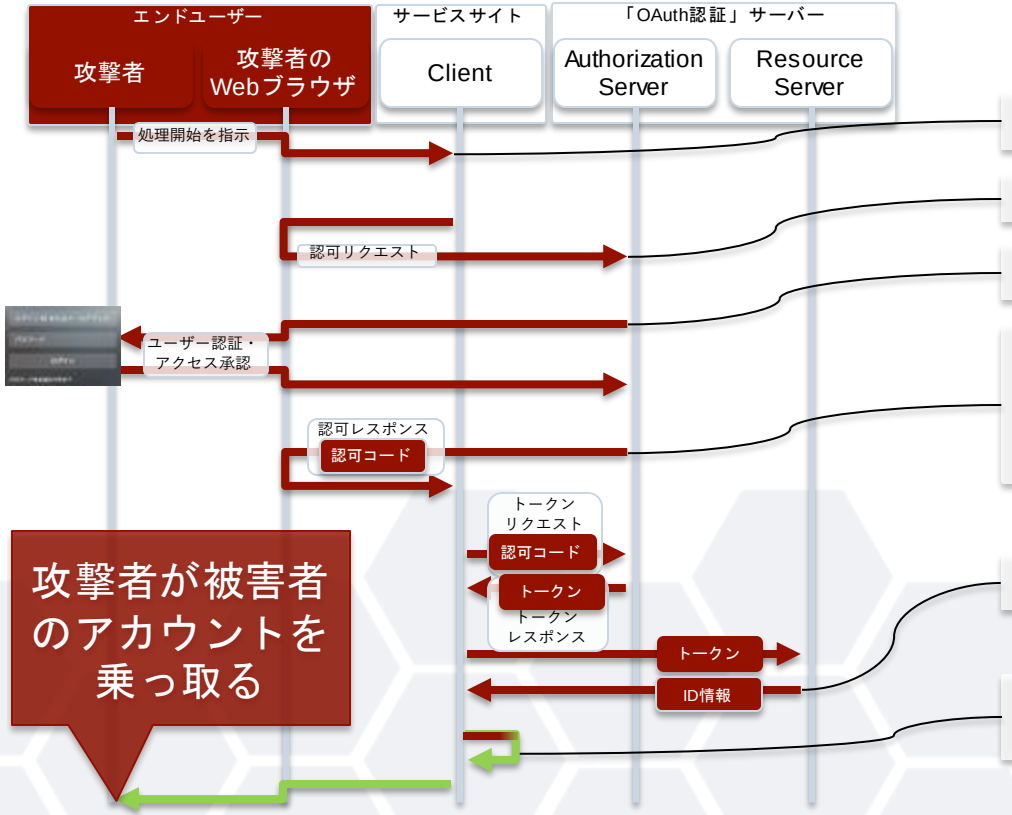
CSRF +「OAuth認証」

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



「OAuth認証開始」

認可リクエスト

攻撃者自身としてユーザー認証 & 同意確認

認可レスポンス

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb
 ?code=<攻撃者にひもづく認可コード>
```

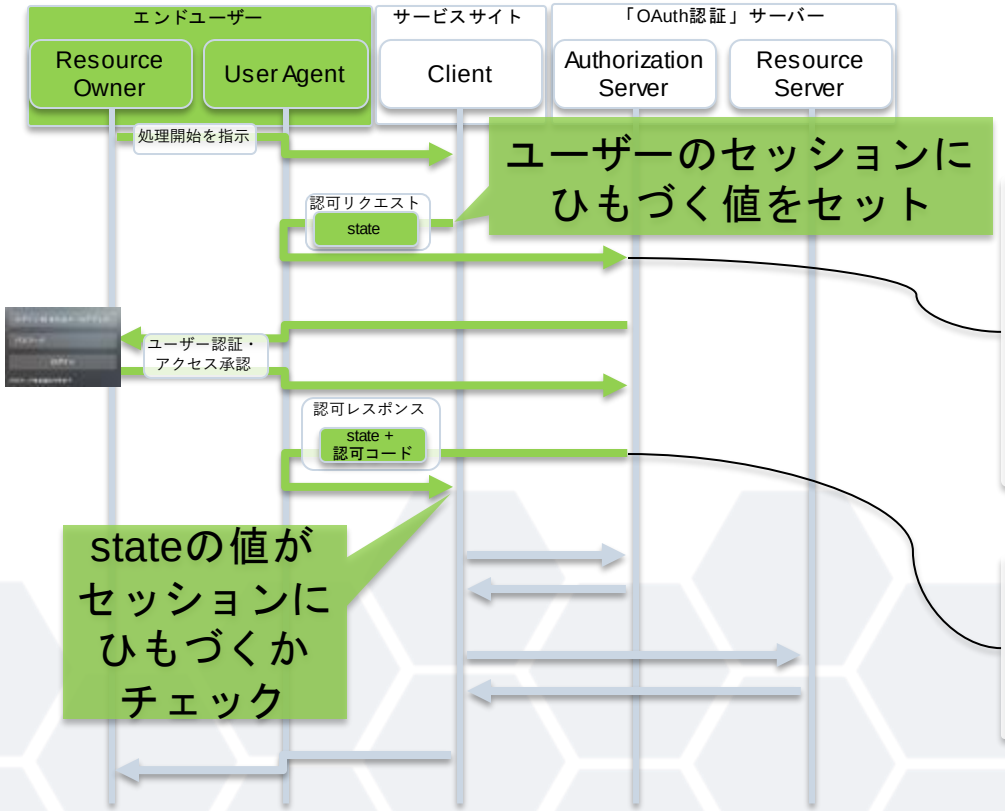
攻撃者のID情報を返却

「攻撃者のID情報」をもとに
被害者としてのログインセッションを生成

AUTHLETE

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」が同一であるかをクライアントが確認



認可リクエスト

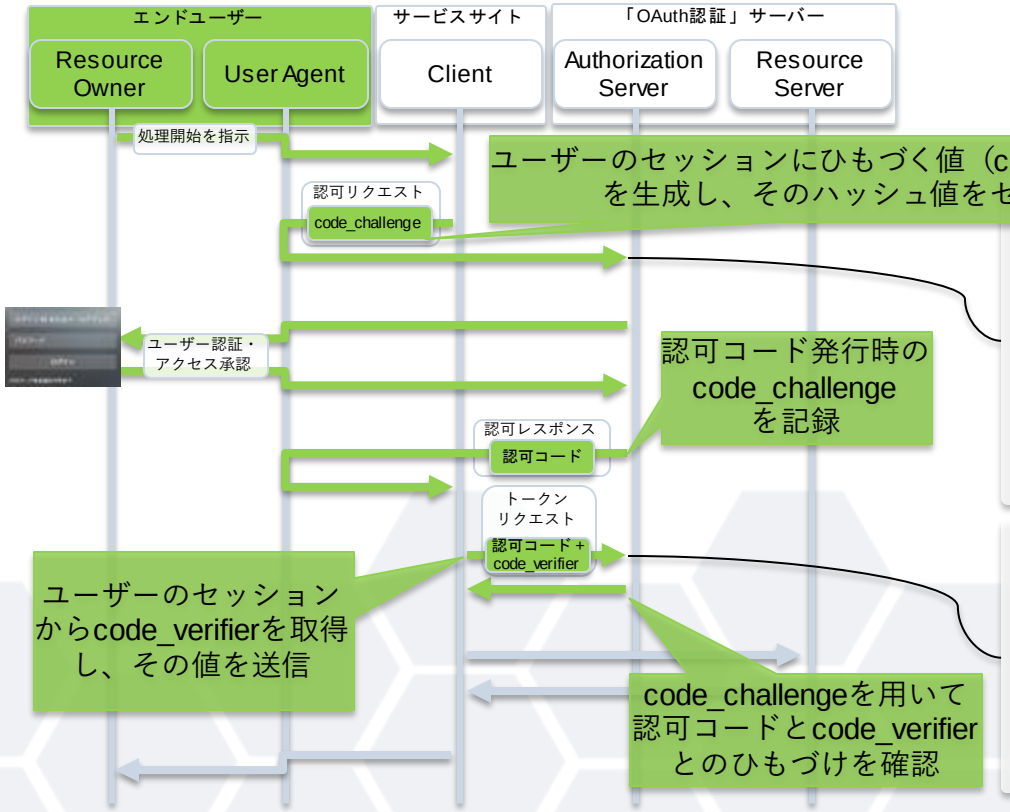
```
GET /authorize
 ?response_type=code
 &client_id=s6BhdRkqt3
 &scope=message.readonly
 &state=3af23asl0989gnv
 &redirect_uri= https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
 HTTP/1.1
Host: server.example.com
```

認可レスポンス

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb
 ?code=<エンドユーザーにひもづく認可コード>
 &state=3af23asl0989gnv
```

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」が同一であるかを認可サーバーが確認



認可リクエスト

```
GET /authorize
 ?response_type=code
 &client_id=s6BhdRkqt3
 &scope=message.readonly
 &code_challenge=<code_verifierからcode_challenge_methodに従って生成した値>
 &code_challenge_method=S256
 &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
 HTTP/1.1
Host: server.example.com
```

トークンリクエスト

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=<認可コード>
 &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
 &code_verifier=<最初に生成したcode_verifierの値>
```

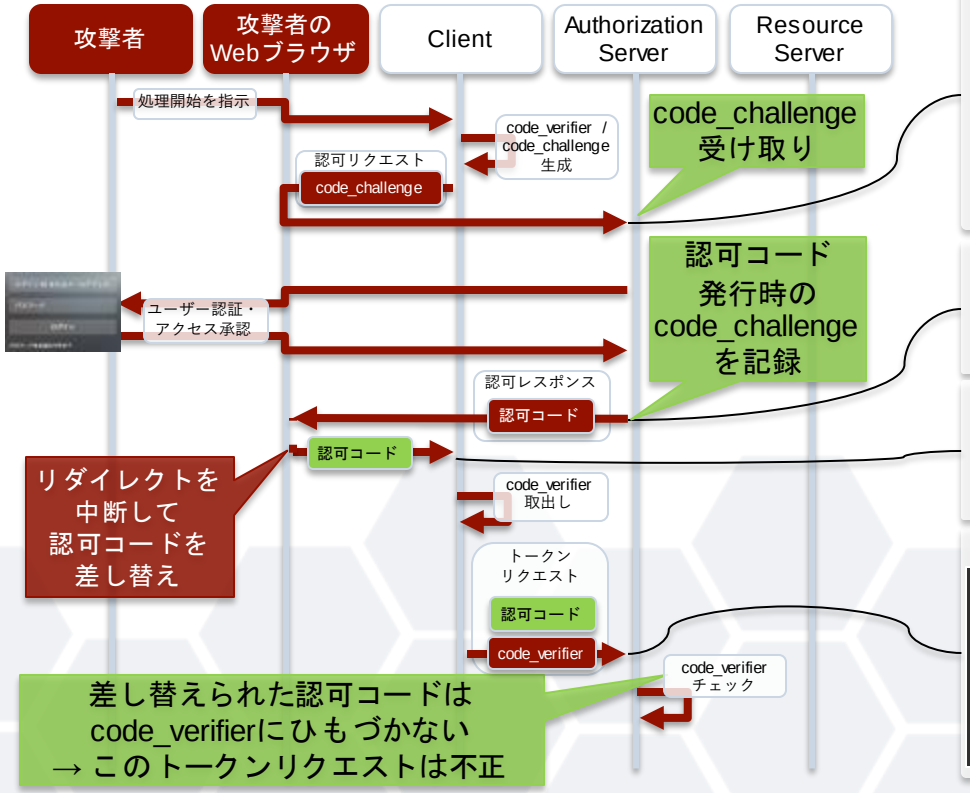
draft-ietf-oauth-security-topics-14 / 2. Recommendations

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



認可リクエスト

```
GET /authorize
 ?response_type=code
 &client_id=s6BhdRkqt3
 &scope=message.readonly
 &state=xyz
 &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
 &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: https://client.example.com/cb?code=<攻撃者にひもづく認可コード>
 &state=xyz
```

認可レスポンス（差し替え後）

```
GET /cb
 ?code=<被害者にひもづく認可コード>&state=xyz HTTP/1.1
Host: server.example.com
```

トークンリクエスト

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=<被害者にひもづく認可コード>
 &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
 &code_verifier=<最初に生成したcode_verifierの値>
```

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

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

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

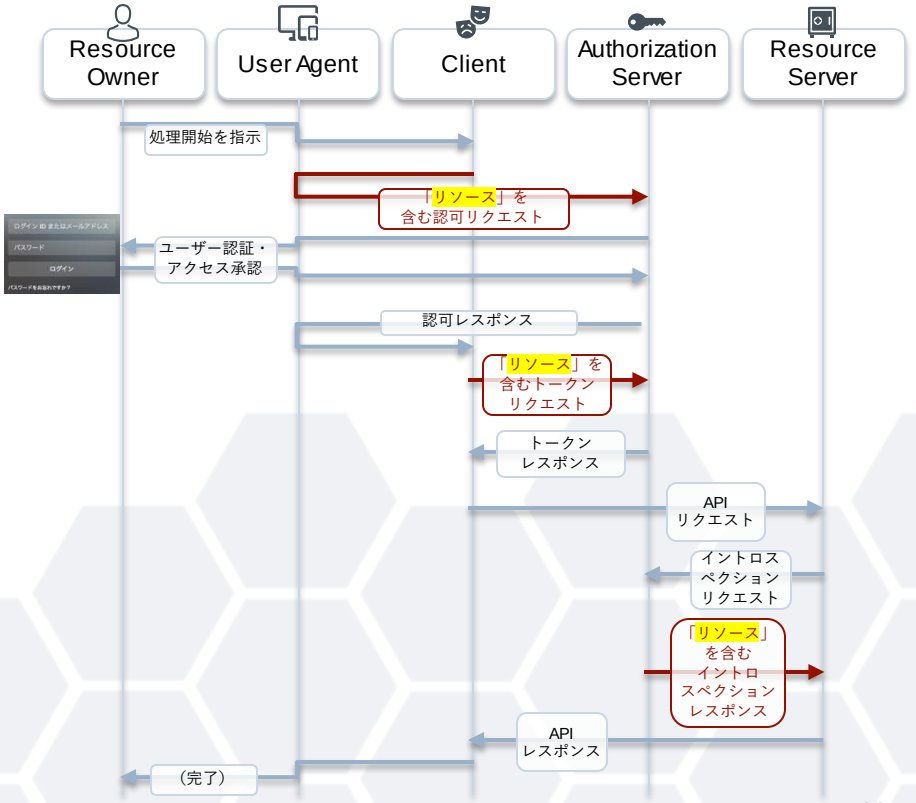
2.3. アクセストークンの権限の制限

- 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

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

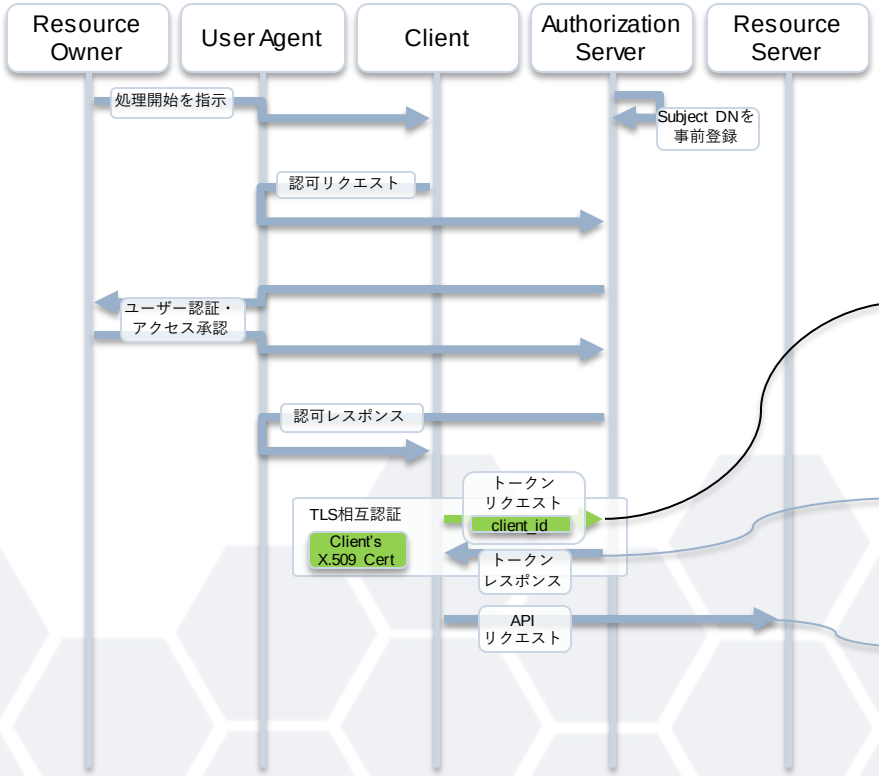
2.5. クライアント認証

- 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

AUTHLETE

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

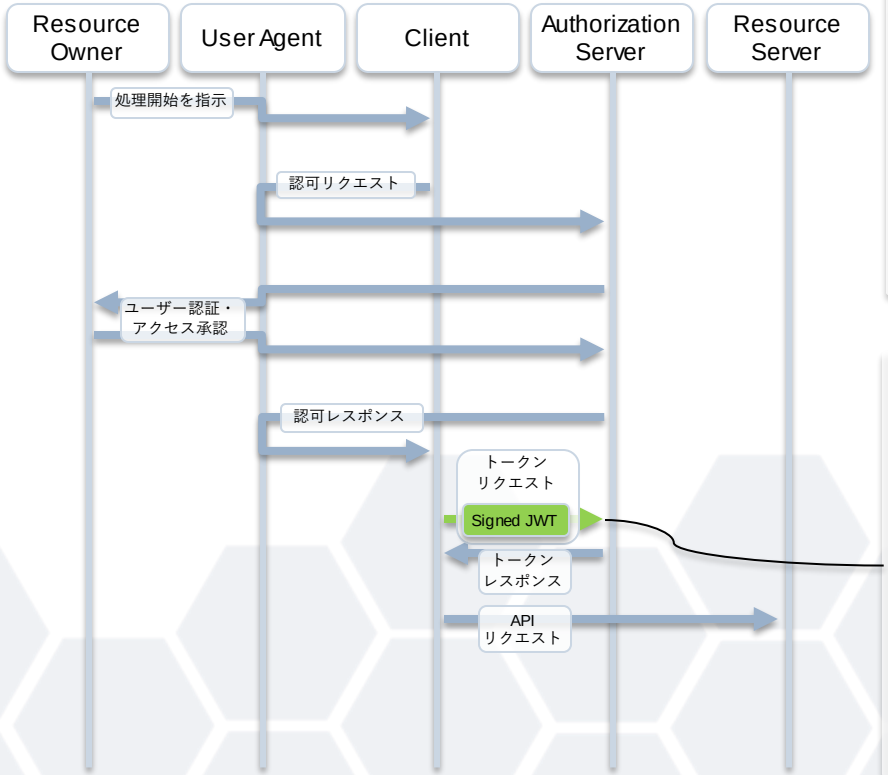
APIリクエスト

- 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.
 ew0KICAg[...omitted for brevity...].
 t-_gX8JQ[...omitted for brevity...]
```

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

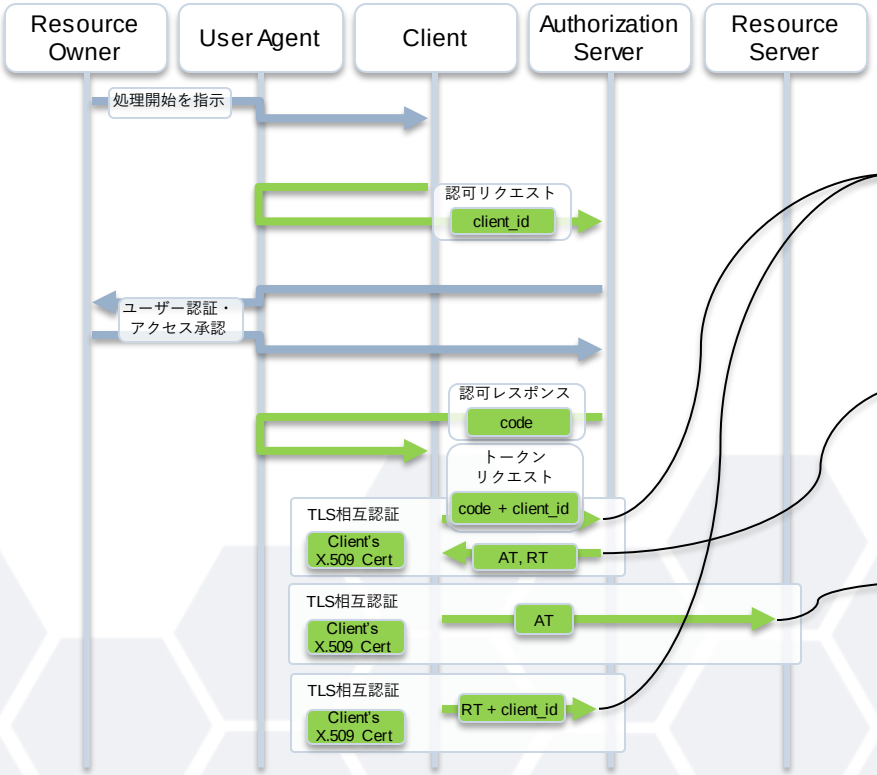
2.2. トークンリプレイの防止

- 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

AUTHLETE

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「サムプリント」を利用

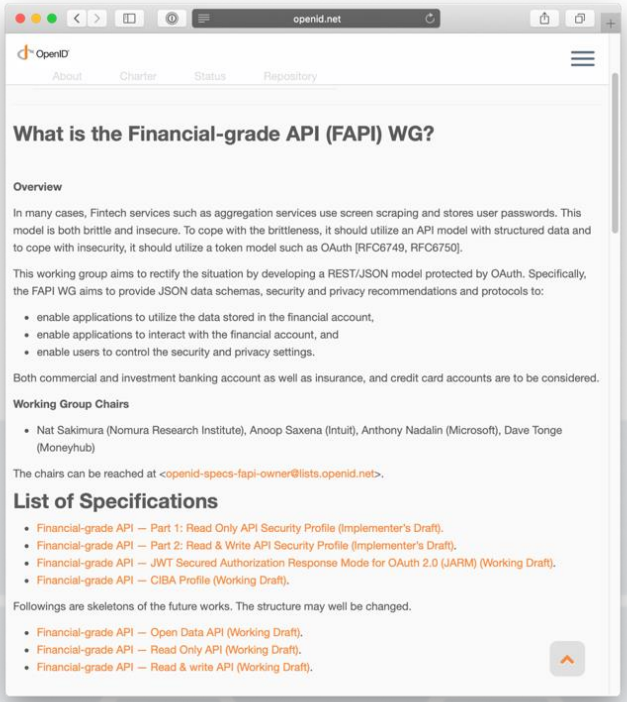
AUTHLETE
Experts in API Authorization

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



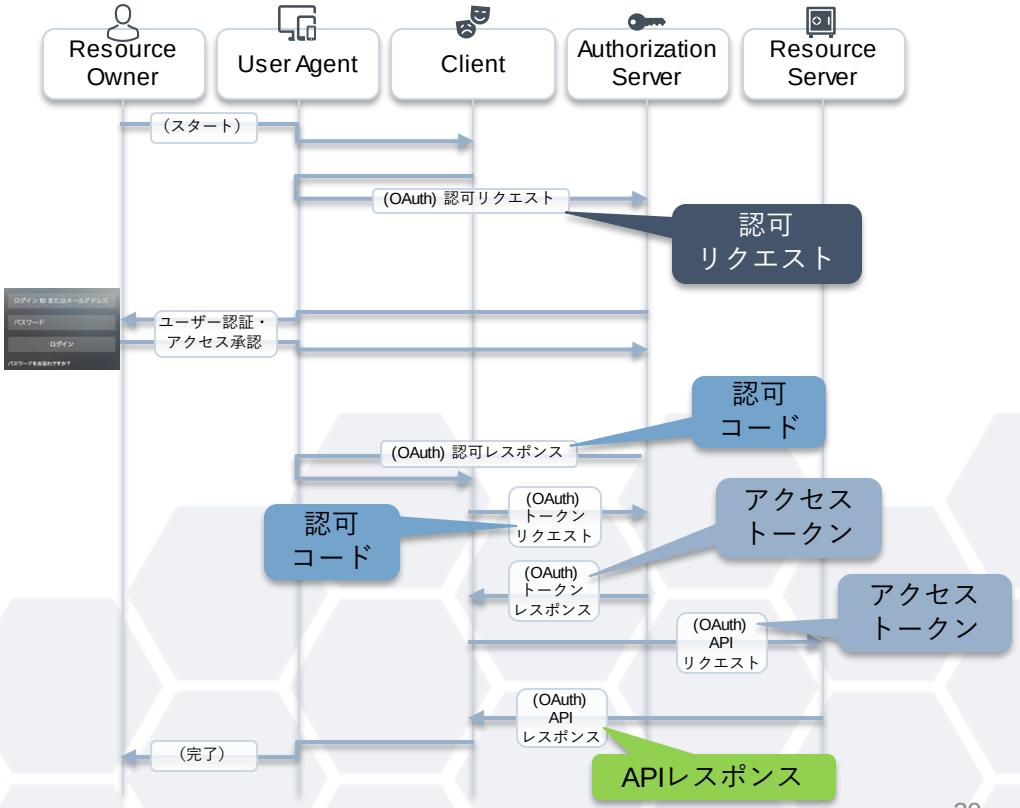
Source: https://openid.net/wg/fapi/

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

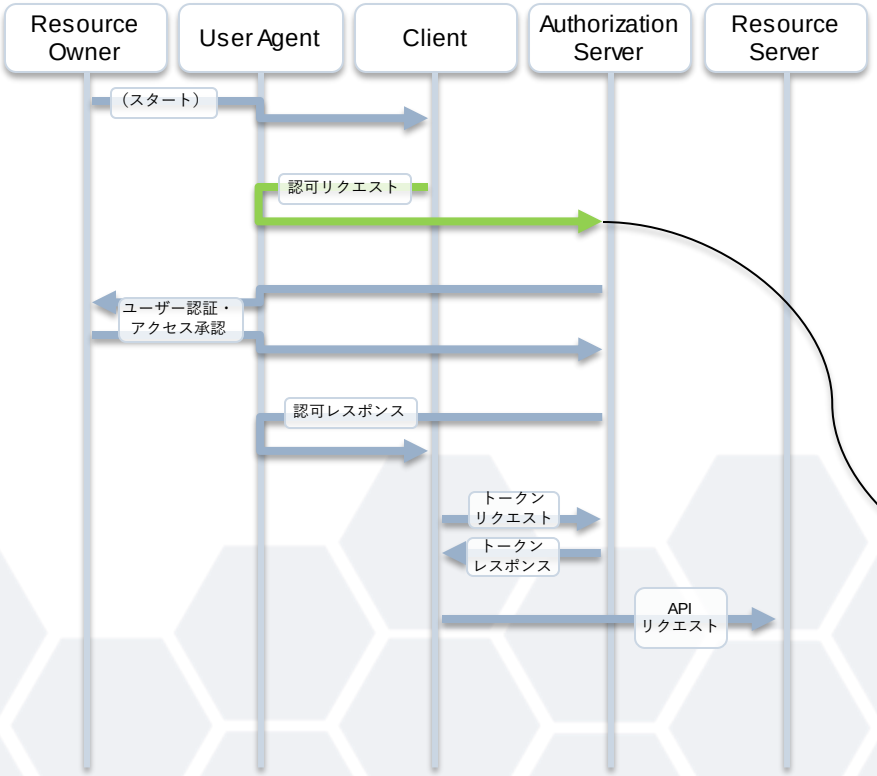
- 認可リクエスト / レスポンスの送信者詐称・改ざん防止
  - Request Objectの利用
  - Hybrid FlowもしくはJARMの利用
- 認可コードの漏洩・盗用防止
  - redirect_uriの厳密な管理
- クライアントのなりすまし防止
  - Mutual TLSもしくはJWTによるクライアント認証
- トークンの漏洩・盗用防止
  - OAUTB（トークン・バインディング）もしくはMTLS（双方向TLS接続へのバインド）の利用



AUTHLETE

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=eyJhbGciOiJSUzI1NiIsImtpZCI6Imsy
 YmRjIn0.ew0KICJpc3MiOiAiczZCaGRS…
```

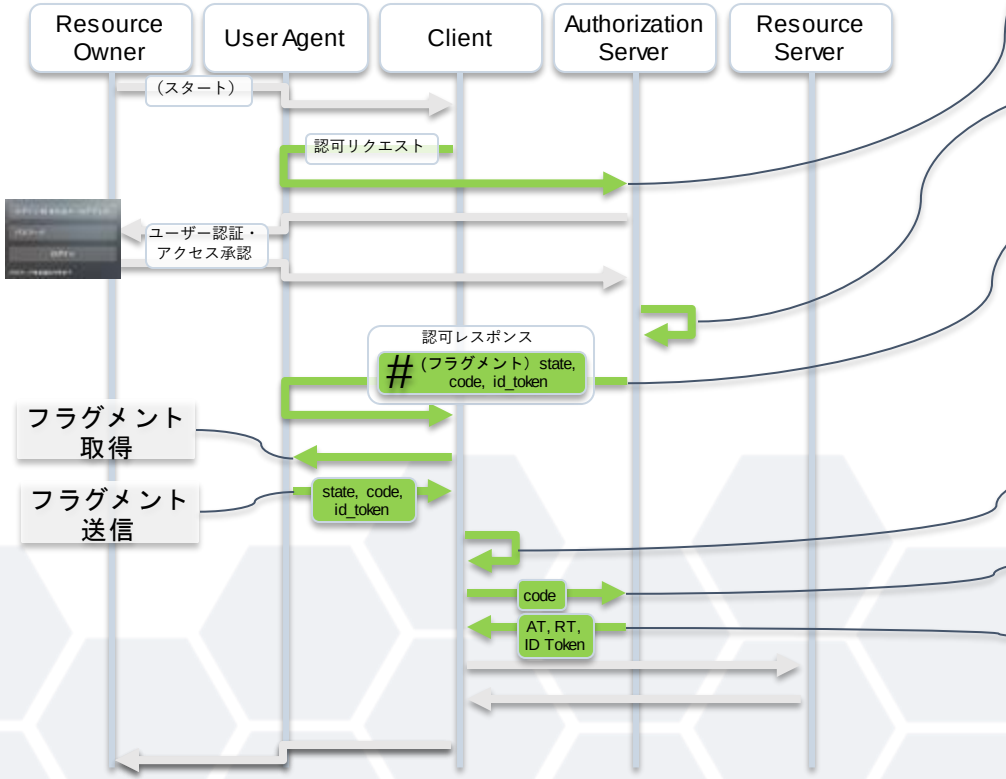
- 参照渡し

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

AUTHLETE

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.mytpp.com/cb#
 code=SplxlOBeZQQYbYS6WxSbIA
 &id_token=eyJ0 ... NiJ9.eyJ1c ...
   I6IjIifX0.DeWt4Qu ... ZXso
 &state=af0ifjsldkj
```

Detached Signatureによる検証

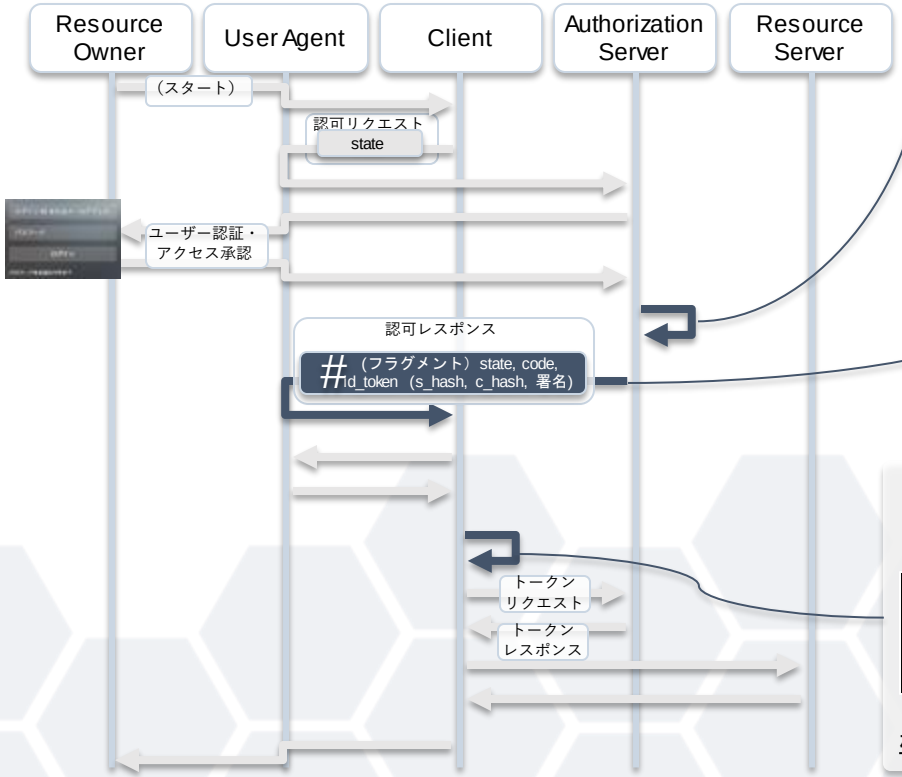
トークンリクエスト

トークンレスポンス

AUTHLETE

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=SplxlOBeZQQYbYS6WxSbIA
 &id_token=eyJ0 ... NiJ9.eyJ1c ...
   I6IjIifX0.DeWt4Qu ... ZXso
 &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

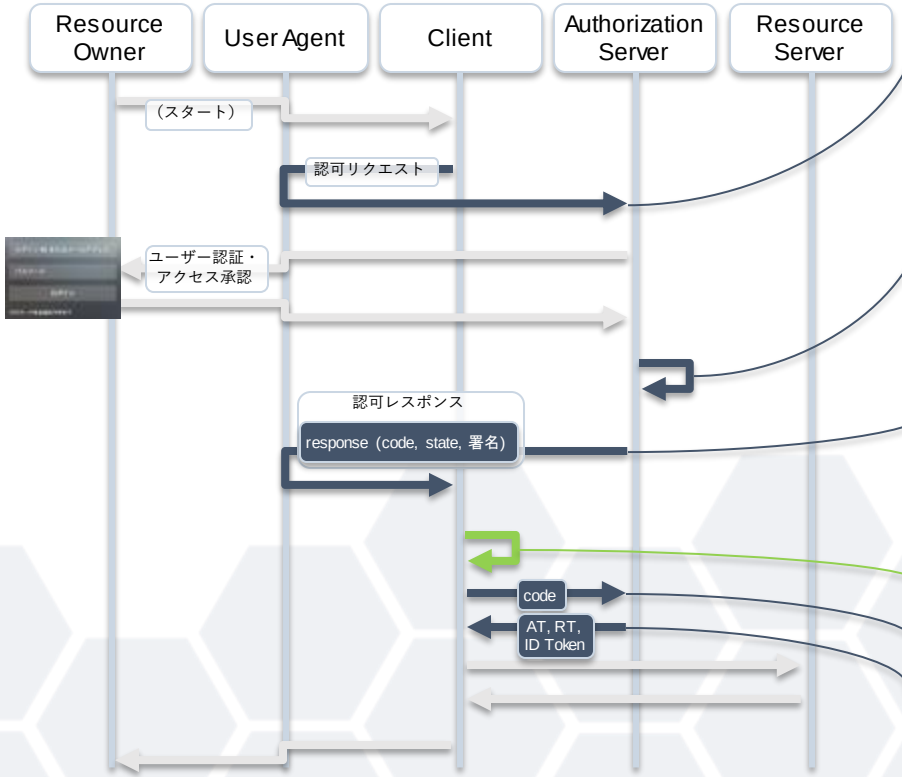
```
code=SplxlOBeZQQYbYS6WxSbIA
state=af0ifjsldkj
```

クエリパラメーターとして受け取った値

AUTHLETE

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=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
  eyJpc3MiOiJodHRwczovL2FjY291bnRzLmV4YW1wbGUuY29tIiwiYXVkI
  joiczZCaGRSa3F0MyIsImV4cCI6MTMxMTI4MTk3MCwiY29kZSI6IlB5eU
  ZhdXgybzdRMFlmWEJVMzJqaHcuNUZYU1FwdnI4YWt2OUNlUkRTZDBRQSI
  sInN0YXRlIjoiUzhOSjd1cWs1Zlk0RWpOdlBfR19GdHlKdTZwVXN2SDlq
  c1luaTlkTUFKdyJ9.
  HkdJ_TYgwBBj10C-aWuNUiA062Amq2b0_oyuc5P0aMTQphAqC2o9WbGSk
  pfuHVBowlb-zJ15tBvXDIABL_t83q6ajvjtq_pqsByiRK2dLVdUwKhW3P
  _9wjvI0K20gdoTNbNlP9Z41mhart4BqraIoI8e-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）

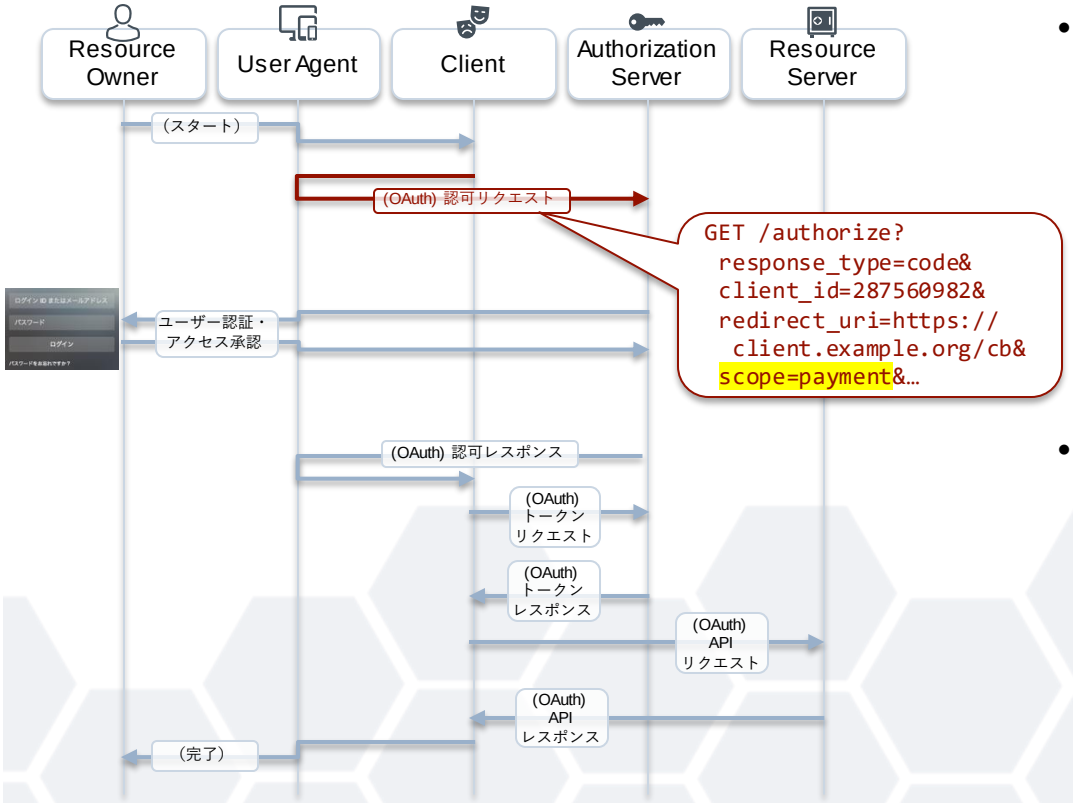
- Removed
  - OAuth 2.0 Token Binding
  - code id_token and code id_token token
  - LoA3
  - Public client support
- Separated
  - Pushed Request Object https://bitbucket.org/openid/fapi/src/master/Financial_API_Pushed_Request_Object.md
- Promoted
  - JARM as an alternative to Hybrid Flow
- Added
  -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

AUTHLETE
Experts in API Authorization

PAR / RAR

“LODGING INTENT PATTERN”

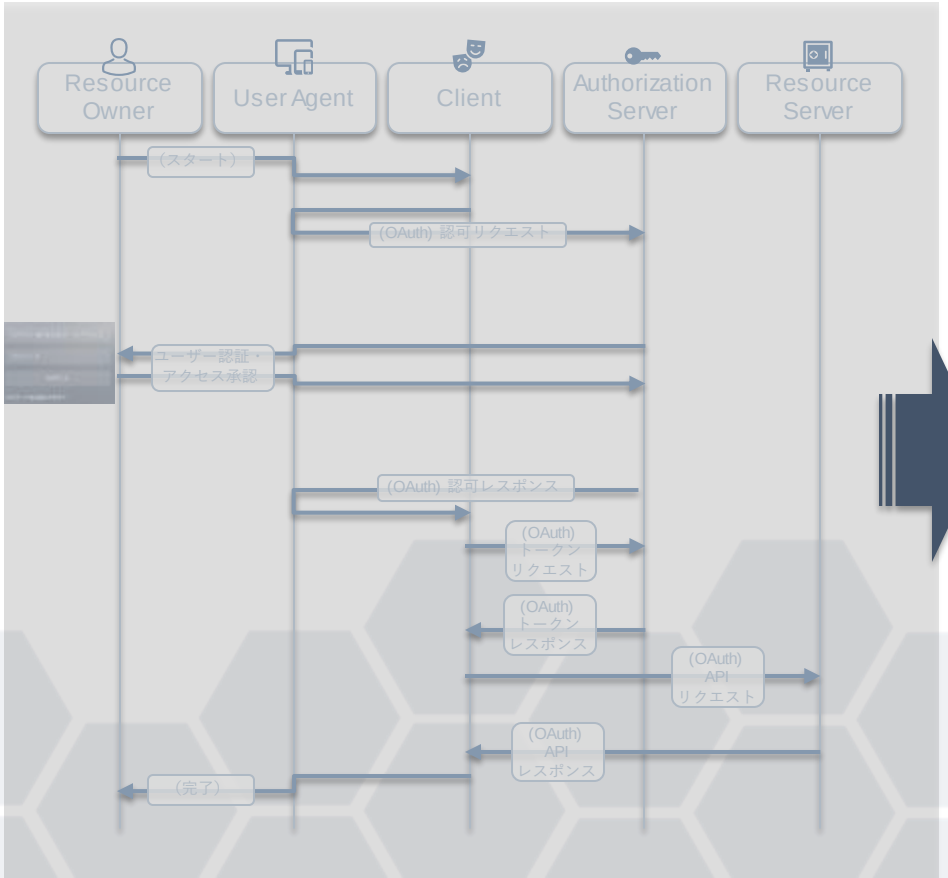
OAuth における「スコープ」(scope)

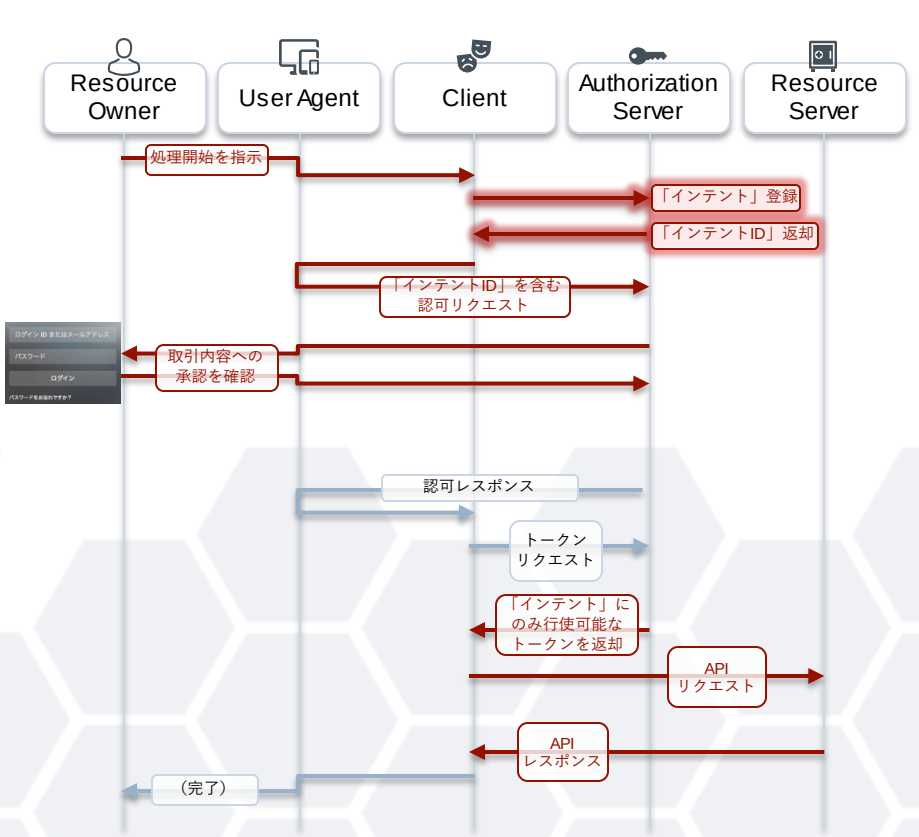


- リソースオーナー（ユーザー）がクライアントに移譲する権限の粒度
  - APIによってアクセス可能なデータの種類・範囲（e.g. 口座情報、取引履歴）
  - APIによって実行可能な機能の範囲（e.g. 参照、決済、送金）
- 「ある特定の取引」「同意した範囲でのデータ取得」のような粒度を表現するにはどうするか?

AUTHLETE

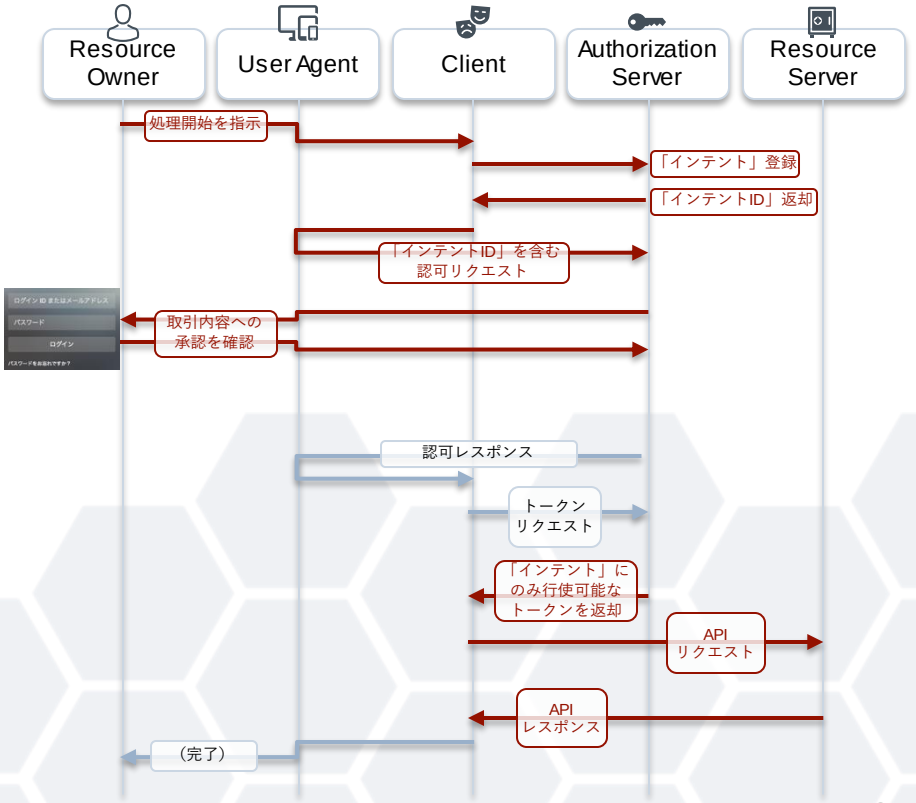
「インテント」の導入





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

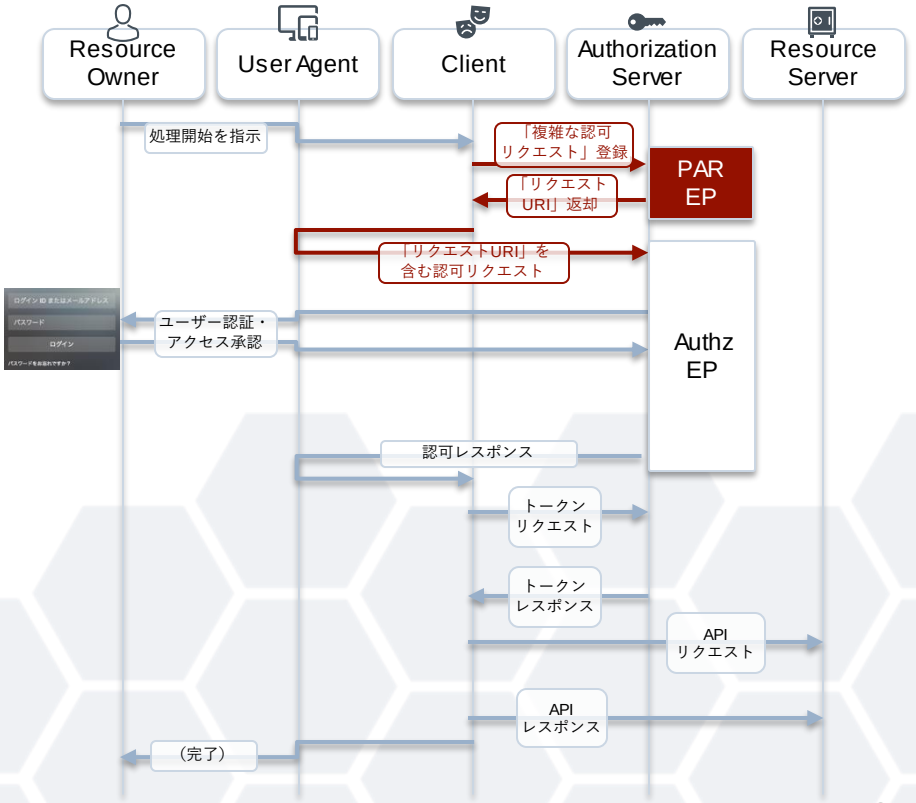
Source: https://www.slideshare.net/tkudo/osw2019/7

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/

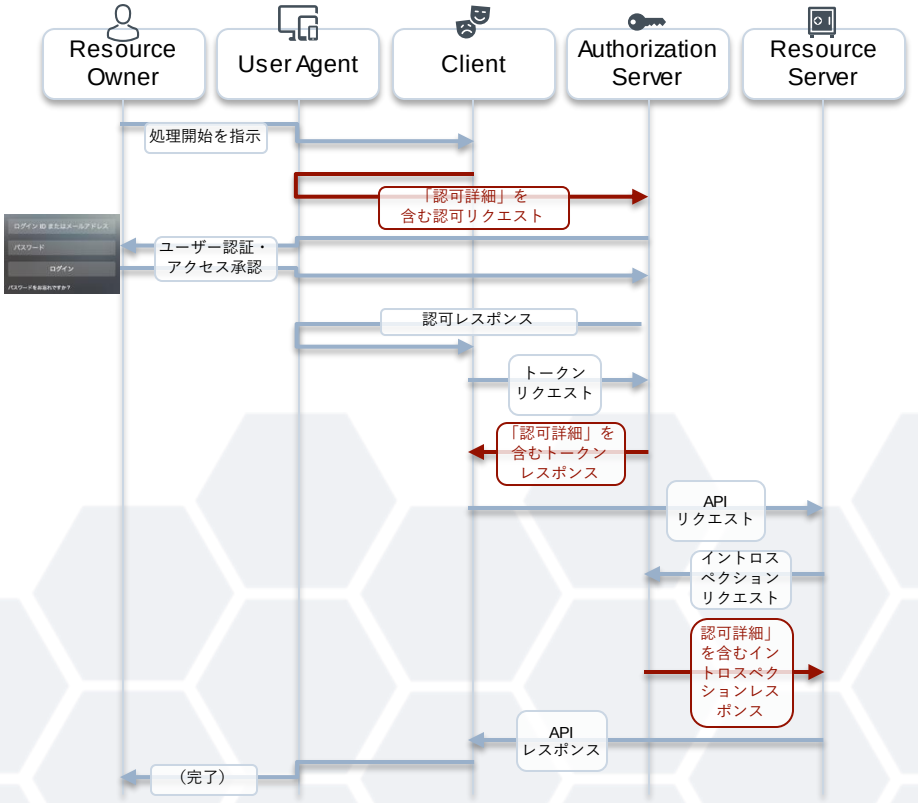
- PARエンドポイント（新規）
  - クライアントが認可サーバーに「複雑な認可リクエスト」を登録し、認可サーバーが「リクエストURI」を返却
  - 認可サーバーがクライアントを認証可能
- 認可エンドポイント（拡張）
  - クライアントが認可サーバーに、リクエストURIを含む「単純な認可リクエスト」を送信
    - https://as.example.com/authorize?request_uri=<リクエストURI>&…
  - ユーザーエージェントによって中継される情報を最小化



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



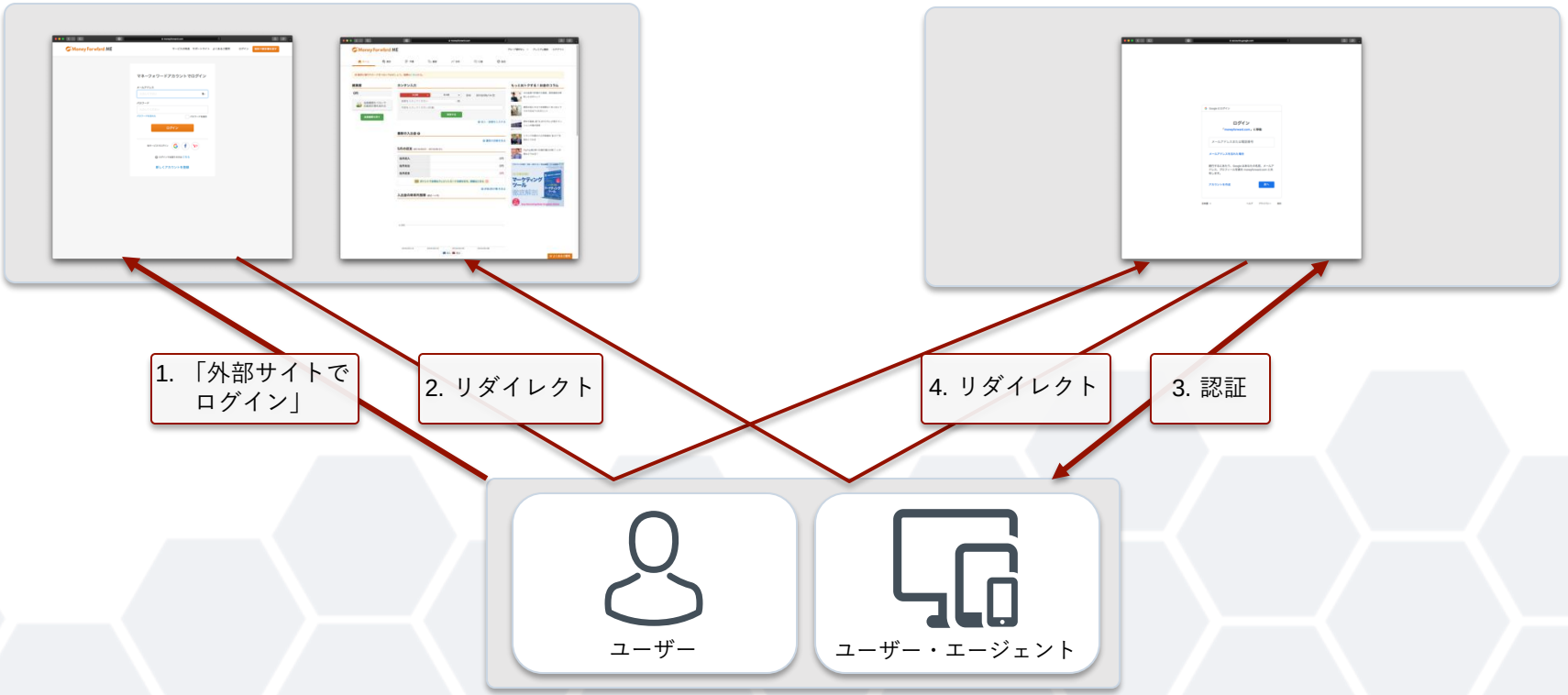
AUTHLETE

これからのOAuth/OIDC

Device Authorization Grant / CIBA

UTILIZING END-USER DEVICES

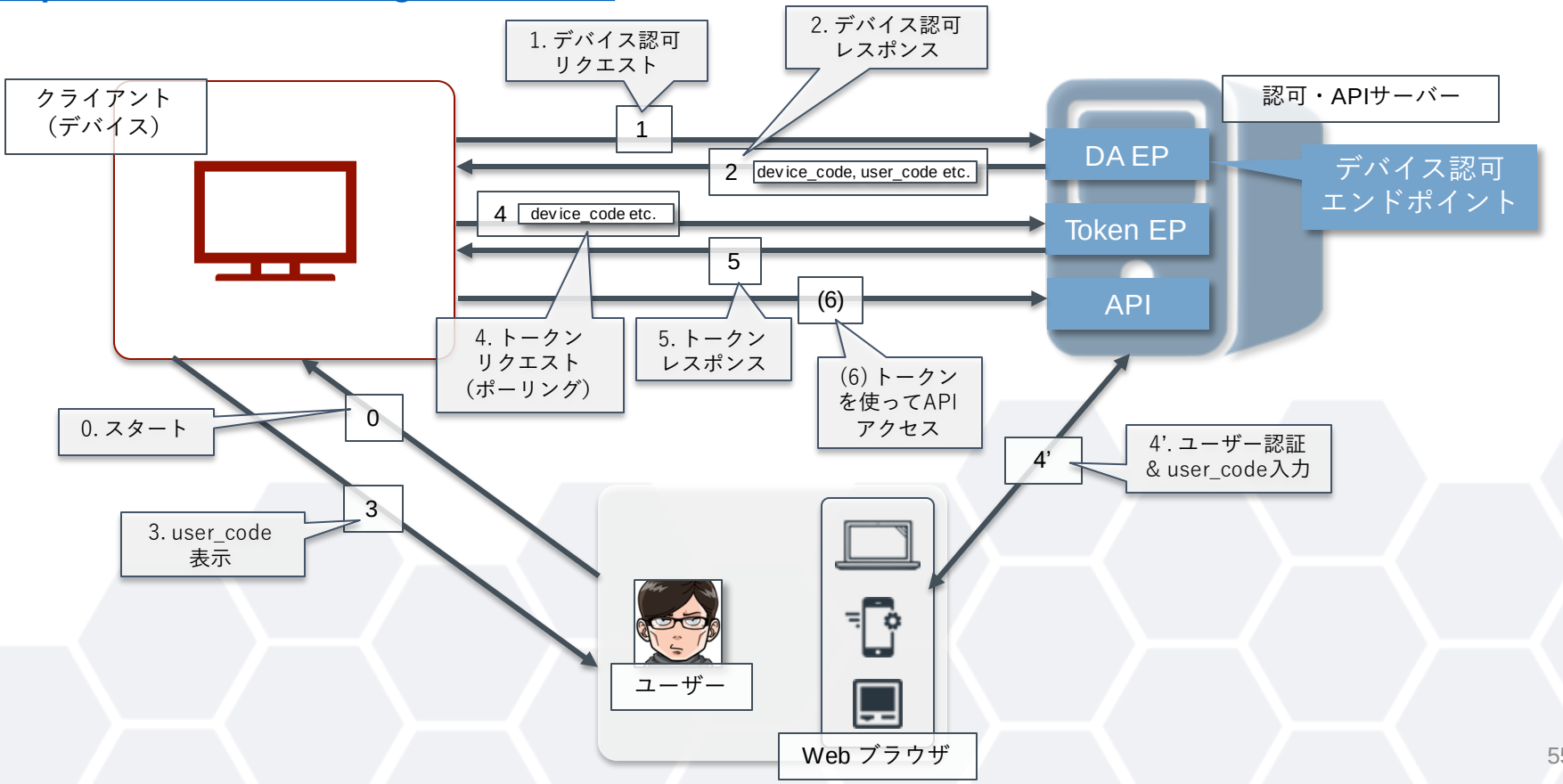
“Redirect Flow”



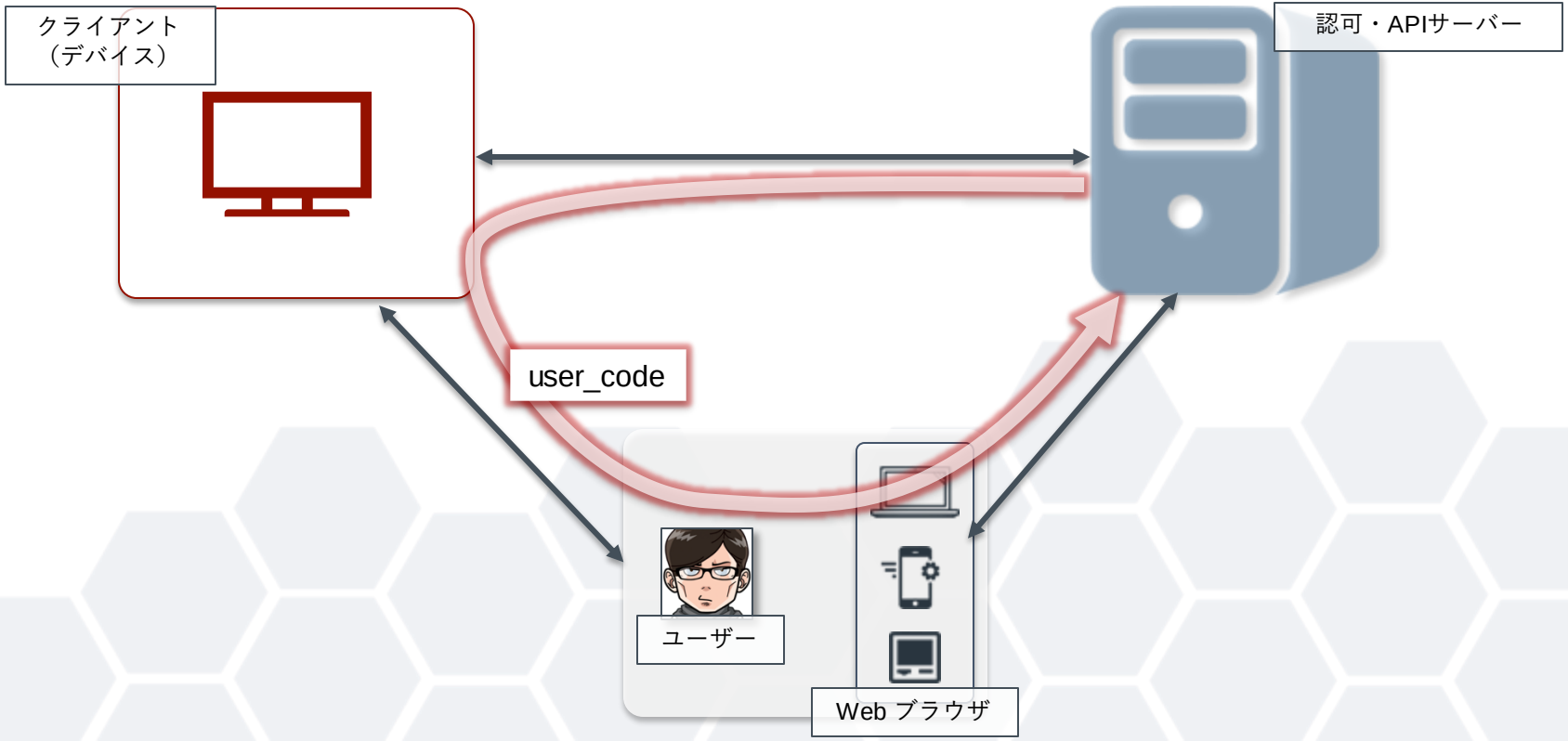
Source: マネーフォワード，Google

RFC 8628: OAuth 2.0 Device Authorization Grant

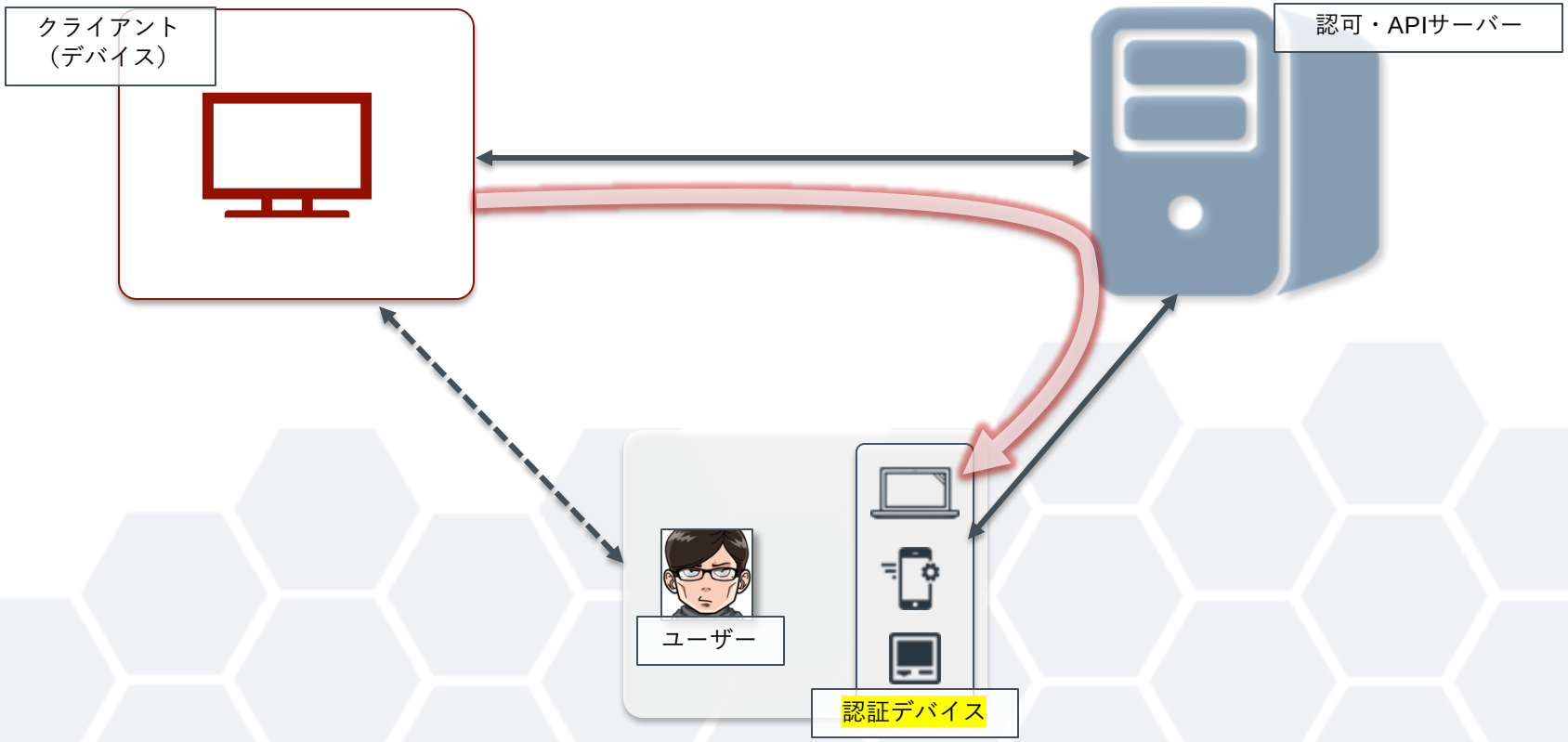
https://datatracker.ietf.org/doc/rfc8628/



ある意味では、ユーザーによる「リダイレクト」



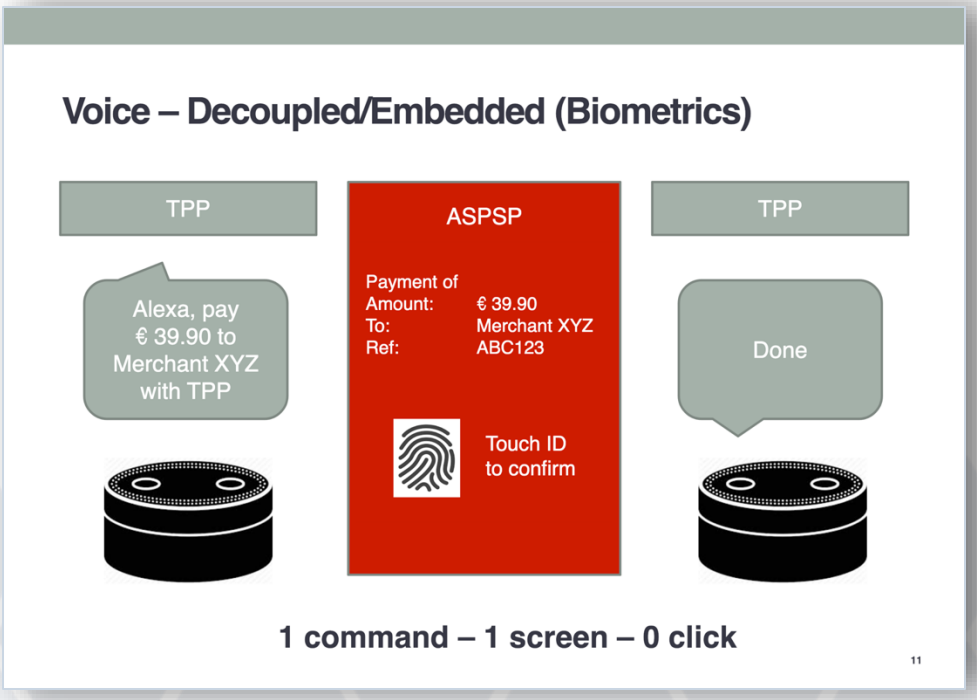
ユーザーを介さない認証・認可フローはどうか?



“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



Source: https://www.europeanpaymentscouncil.eu/document-library/minutes-and-agendas/authentication-sca-guidance-key-topic-clarification-api

“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「CIBA Pay(ﾀﾞｯｻ)」
  - 店員「はい(ﾎﾟﾁ)」
  - 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。レシートどぞ」

ぐらいは出来そうです

Source: https://ritou.hatenablog.com/entry/2018/12/29/224452

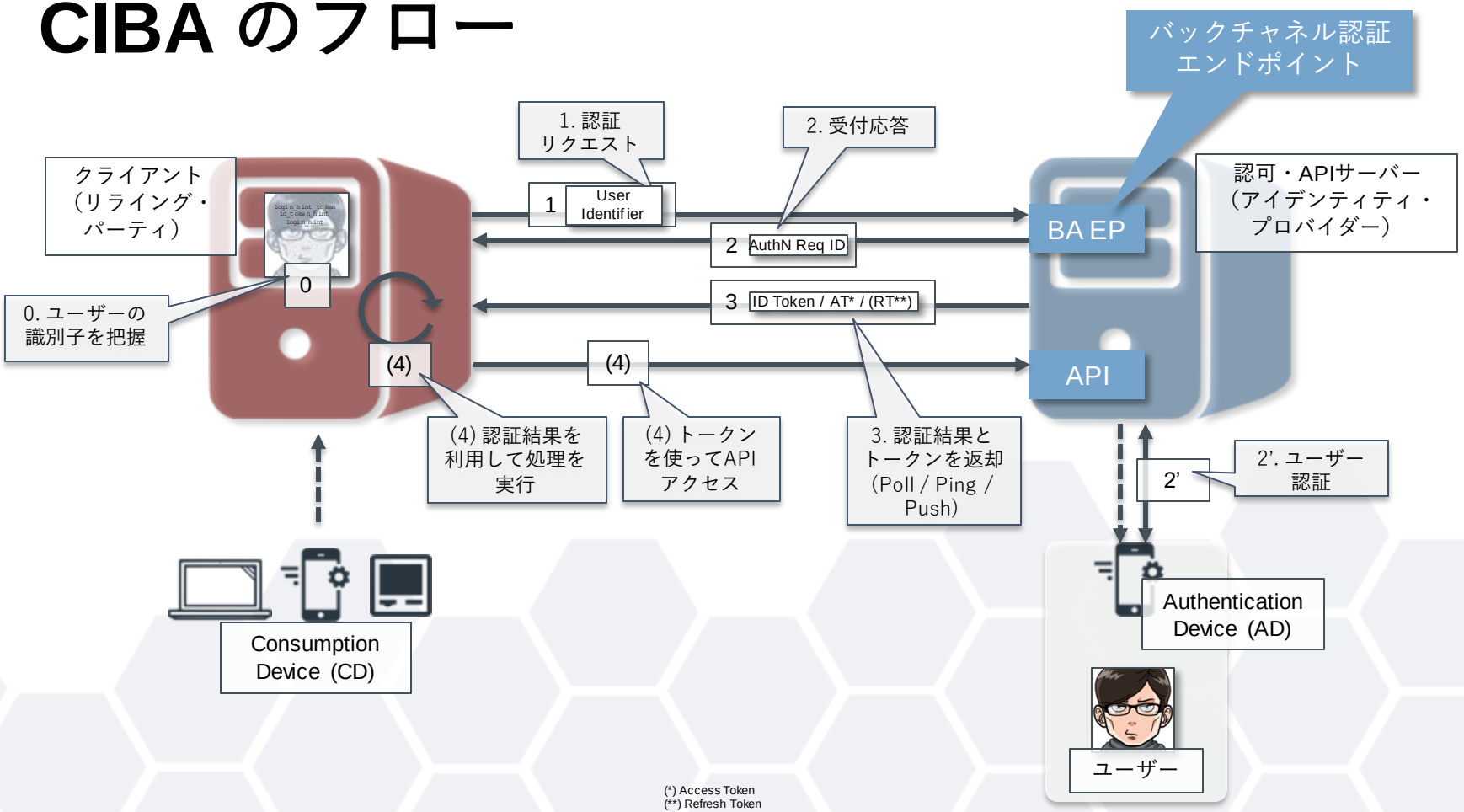
AUTHLETE

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



(*) Access Token
(**) Refresh Token

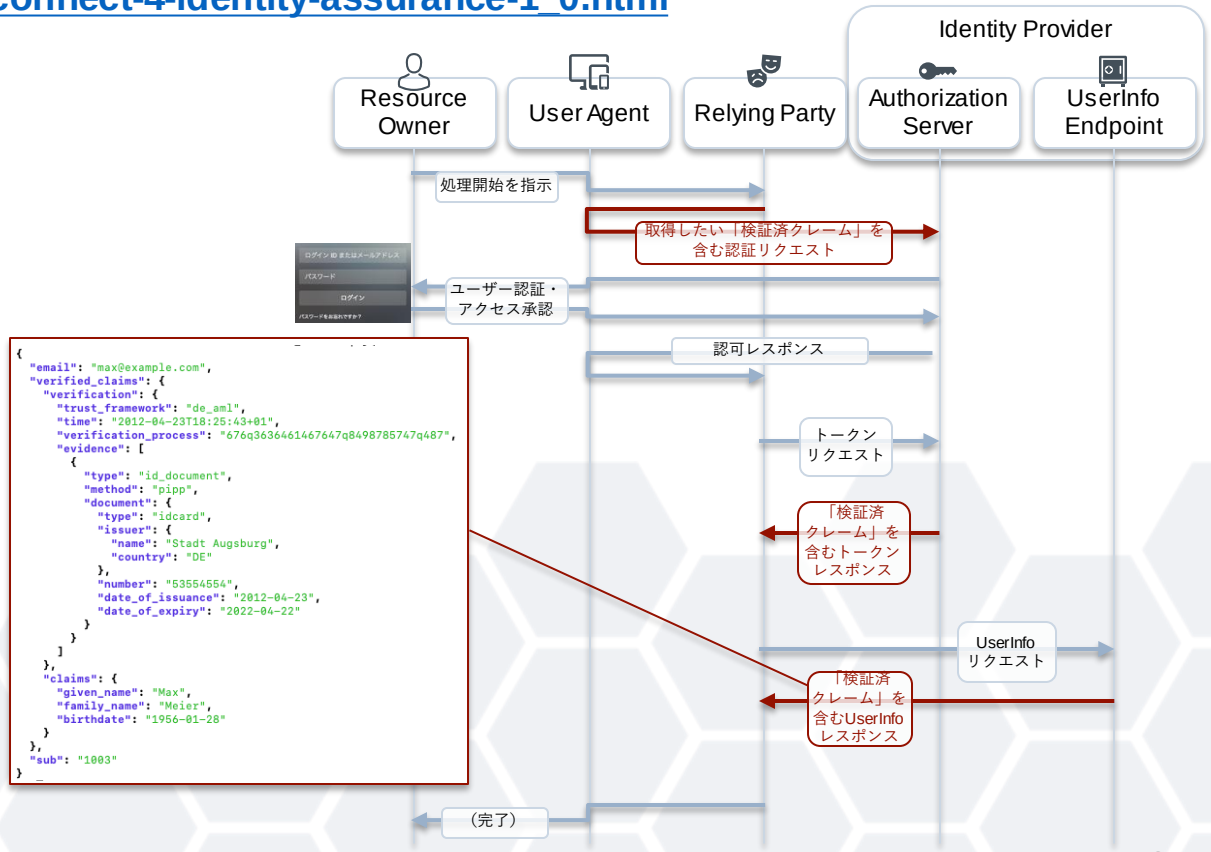
IDA

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パラメーターを用いて指定
- 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



AUTHLETE

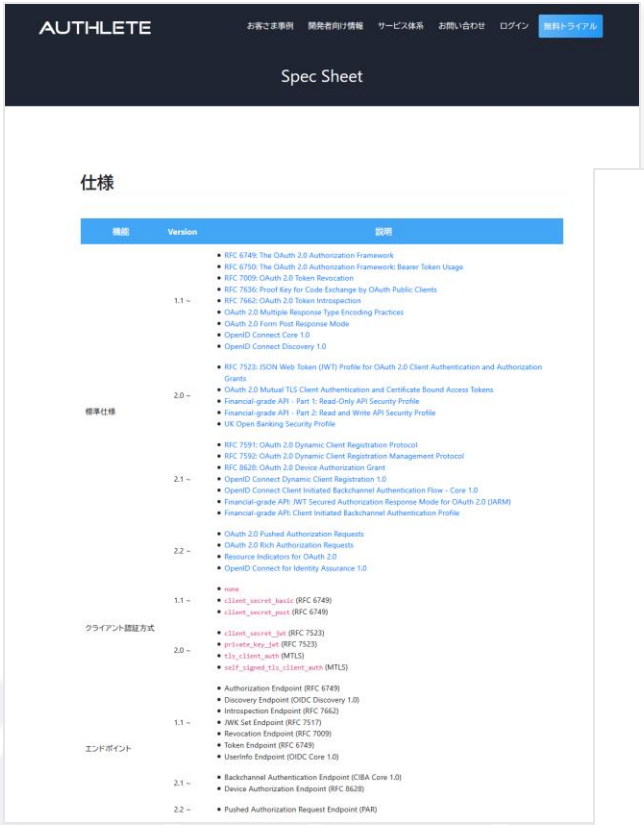
まとめ

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

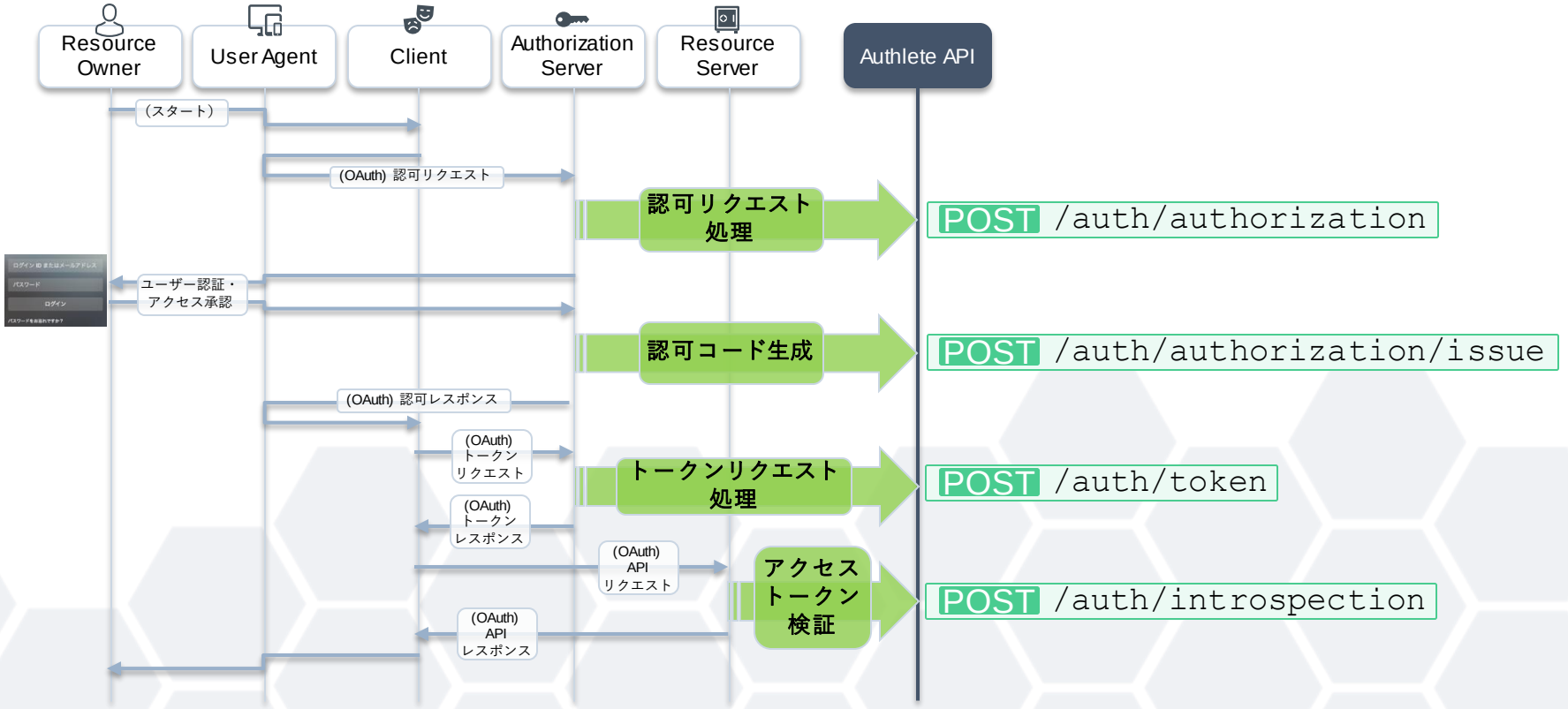
https://www.authlete.com/ja/legal/spec_sheet/



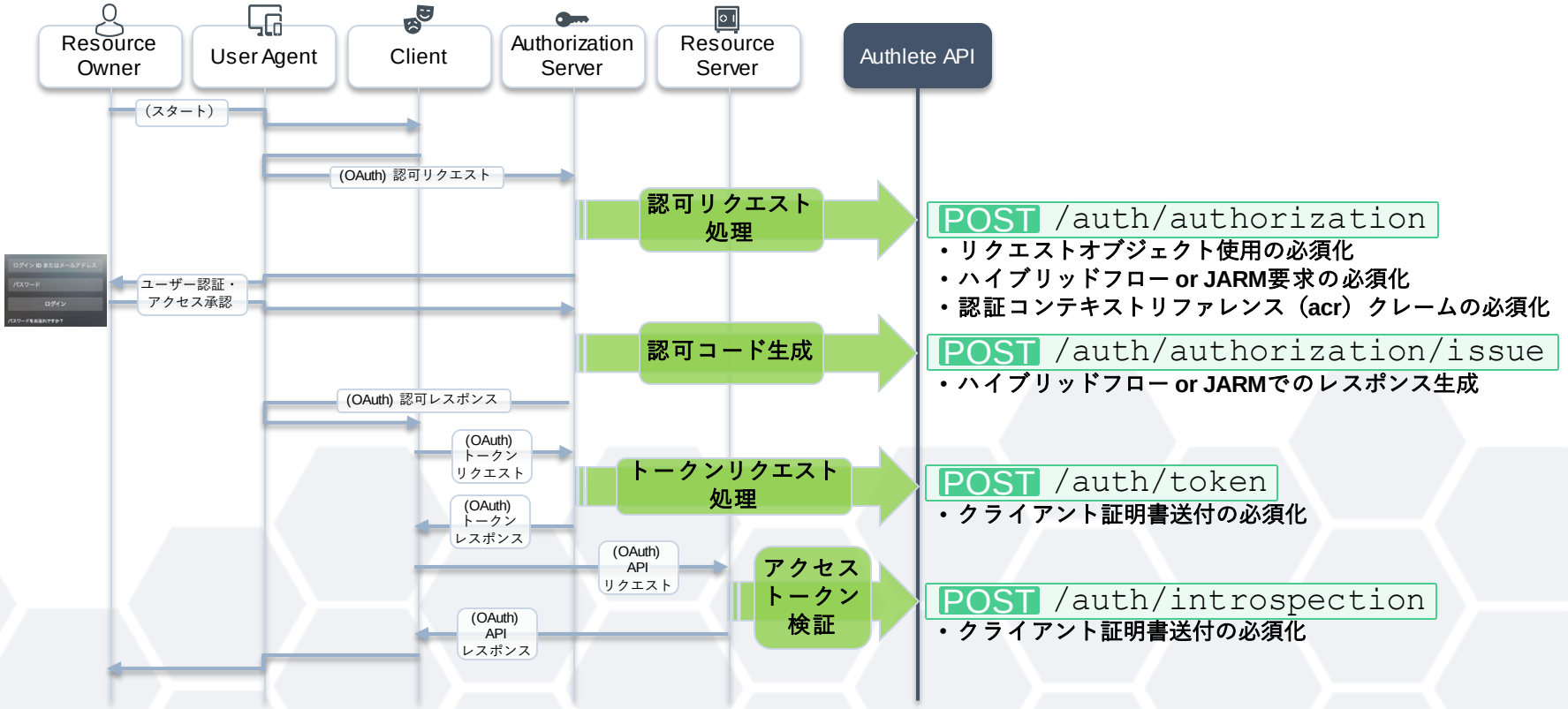
AUTHLETE

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/

AUTHLETE

Thank You

www.authlete.com

www.linkedin.com/in/tatsuokudo