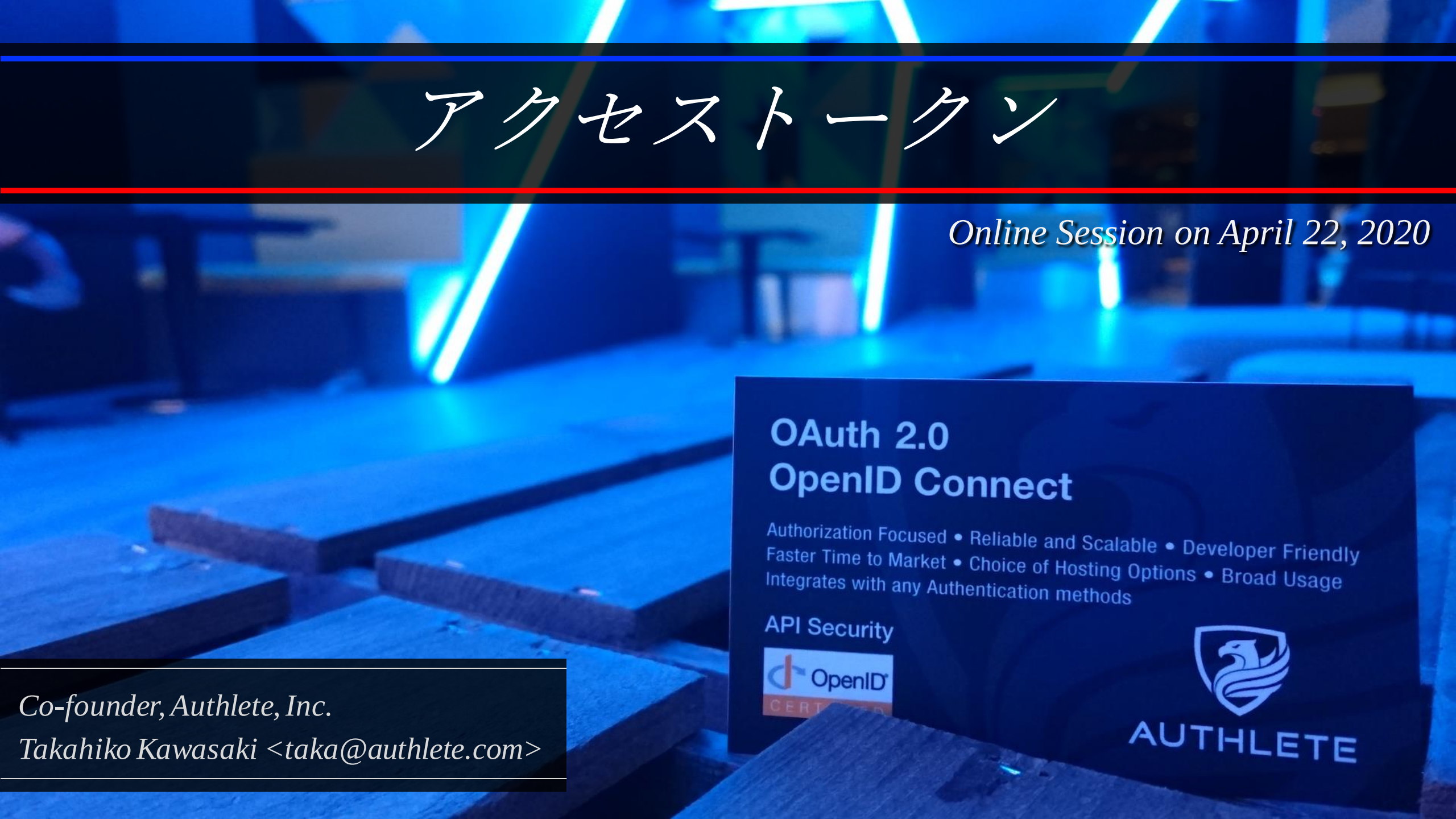
アクセストークン

Online Session on April 22, 2020



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

前提知識

- RFC 6749 (The OAuth 2.0 Authorization Framework)
- RFC 7515 ～ RFC 7519 (JWS, JWE, JWK, JWA, JWT)

『OAuth & OIDC 入門編』（前提知識おさらい動画）

https://www.youtube.com/watch?v=PKPj_MmLq5E

アクセストークン発行


OAuth 2.0
OpenID Connect
Authorization Focused • Reliable and Scalable • Developer Friendly
Faster Time to Market • Choice of Hosting Options • Broad Usage
Integrates with any Authentication methods
API Security
OpenID
AUTHLETE

ほとんどの場合、アクセストークンはトークンエンドポイントから発行される。

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

{
   "access_token":   "アクセストークン",
   "token_type":     "トークンタイプ",
   "expires_in":     有効秒数,
   "refresh_token":  "リフレッシュトークン",
   "scope":          "スコープ群"
}
```

トークンレスポンスの形式

インプリシットフローでは※アクセストークンは認可エンドポイントから発行される。

※ 正確に言うと、認可リクエストの response_type に token が含まれている場合

```
HTTP/1.1 302 Found
Location: リダイレクトURI
    #access_token=アクセストークン
    &token_type=トークンタイプ
    &expires_in=有効秒数
    &scope=スコープ群
    &state=ステート
```

インプリシットフローの
認可レスポンスの形式

CIBA※のPUSHモードでは、クライアント通知エンドポイントにアクセストークンが送られてくる。

※ Client Initiated Backchannel Authentication

```
POST クライアント通知エンドポイント HTTP/1.1
Host: 通知を受けるサーバー
Authorization: Bearer クライアント通知トークン
Content-Type: application/json

{
  "access_token":   "アクセストークン",
  "token_type":     "トークンタイプ",
  "expires_in":     有効秒数,
  "refresh_token": "リフレッシュトークン",
  "id_token":       "IDトークン",
  "auth_req_id":    "認証リクエストID"
}
```

CIBA PUSH モードの
クライアント通知の形式

アクセストークン利用


OAuth 2.0
OpenID Connect
Authorization Focused • Reliable and Scalable • Developer Friendly
Faster Time to Market • Choice of Hosting Options • Broad Usage
Integrates with any Authentication methods
API Security
OpenID
AUTHLETE

RFC 6750 (The OAuth 2.0 Authorization Framework: Bearer Token Usage) には、Protected Resource Endpoint（いわゆる Web API）へのリクエストにアクセストークンを含める方法が三つ定義されている。

① **Authorization Request Header Field** (Section 2.1)

→ HTTP ヘッダー `Authorization` にアクセストークンを埋め込む

② **Form-Encoded Body Parameter** (Section 2.2)

→ フォームパラメーター `access_token` を使う

③ **URI Query Parameter** (Section 2.3)

→ クエリーパラメーター `access_token` を使う

① Authorization Request Header Field (Section 2.1)

RFC 6750, Section 2.1 に挙げられている例

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

アクセストークン

Authorization ヘッダーに設定する値の形式

```
b64token    = 1*( ALPHA / DIGIT /
                  "-" / "." / "_" / "~" / "+" / "/" ) *"="
credentials = "Bearer" 1*SP b64token
```

→ アクセストークンの表現に使える文字を BASE64 系の文字のみに限定している。
RFC 6749 の定義（任意の文字列）よりも制限が強い。

疑問：Bearer の後に続ける文字列は次のどちらか？

① 認可サーバーから受け取ったアクセストークンそのもの
② アクセストークンに BASE64 エンコーディングを施したもの

正解は①だが、

- b64token という名称を使い、
- 仕様書の当該箇所の前段で（BASE64を使う）ベーシック認証に言及し、
- アクセストークンの実装方法にしれっと制限を加えることになるため
  （②であれば元となるアクセストークンの表現方法に制限はかからない）、

②という誤解を与えてもやむをえない。実際にそう誤解した技術者は何人もいた。

参考→ **question about the b64token syntax in draft-ietf-oauth-v2-bearer**
https://www.ietf.org/mail-archive/web/oauth/current/msg08481.html

② Form-Encoded Body Parameter (Section 2.2)

RFC 6750, Section 2.2 に挙げられている例

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

access_token=mF_9.B5f-4.1JqM
```

③ URI Query Parameter (Section 2.3)

```
GET /resource?access_token=mF_9.B5f-4.1JqM HTTP/1.1
Host: server.example.com
```

RFC 6750, Section 2.3 に挙げられている例

DPoP-bound アクセストークンの場合

DPoP, Section 6 に挙げられている例

```
GET /protectedresource HTTP/1.1
Host: resource.example.com
Authorization: DPoP eyJhbGciOiJFUzI1NiIsImtpZCI6IkJlQUxrYiJ9.eyJzdWI
 iOiJzb21lb25lQGV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbX
 BsZS5jb20iLCJhdWQiOiJodHRwczovL3Jlc291cmNlLmV4YW1wbGUub3JnIiwibmJmI
 joxNTYyMjYyNjExLCJleHAiOjE1NjIyNjYyMTYsImNuZiI6eyJqa3QiOiIwWmNPQ09S
 Wk5ZeS1EV3BxcTMwalp5SkdIVE4wZDJIZ2xCVjN1aWd1QTRJIn19.vsFiVqHCyIkBYu
 50c69bmPJsj8qYlsXfuC6nZcLl8YYRNOhqMuRXu6oSZHe2dGZY0ODNaGg1cg-kVigzY
 hF1MQ
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiJlMWozVl9iS2ljOC1MQUVCIiwiaHRtIj
 oiR0VUIiwiaHR1IjoiaHR0cHM6Ly9yZXNvdXJjZS5leGFtcGxlLm9yZy9wcm90ZWN0Z
 WRyZXNvdXJjZSIsImlhdCI6MTU2MjI2MjYxOH0.lNhmpAX1WwmpBvwhok4E74kWCiGB
 NdavjLAeevGy32H3dbF0Jbri69Nm2ukkwb-uyUI4AUg1JSskfWIyo4UCbQ
```

←アクセストークン

この例のアクセストークンはたまたま JWT になっているが DPoP-bound だからといって JWT である必要はない。

←DPoP proof JWT

エラー応答


OAuth 2.0
OpenID Connect
Authorization Focused • Reliable and Scalable • Developer Friendly
Faster Time to Market • Choice of Hosting Options • Broad Usage
Integrates with any Authentication methods
API Security
OpenID
AUTHLETE

アクセストークンが無効のとき、API はどのようなエラーを返せばよいのか？

→ RFC 6750 に準拠するなら `WWW-Authenticate` ヘッダーを使う。

アクセストークンが期限切れのときの応答例（RFC 6750, Section 3）

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example",
                         error="invalid_token",
                         error_description="The access token expired"
```

任意項目を全て含む応答例

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example",
                         error="invalid_token",
                         error_description="The access token expired",
                         error_uri="https://example.com/error/E001",
                         scope="openid profile email"
```

エラー応答に含まれる `error` パラメーターの値として、RFC 6750, Section 3.1. Error Codes に次の三つが定義されている。

`invalid_request`
The request is missing a required parameter, includes an unsupported parameter or parameter value, repeats the same parameter, uses more than one method for including an access token, or is otherwise malformed. The resource server SHOULD respond with the HTTP 400 (Bad Request) status code.
`invalid_token`
The access token provided is expired, revoked, malformed, or invalid for other reasons. The resource SHOULD respond with the HTTP 401 (Unauthorized) status code. The client MAY request a new access token and retry the protected resource request.
`insufficient_scope`
The request requires higher privileges than provided by the access token. The resource server SHOULD respond with the HTTP 403 (Forbidden) status code and MAY include the "scope" attribute with the scope necessary to access the protected resource.

アクセストークン実装の分類

RFC 6749, 1.4. Access Token より抜粋

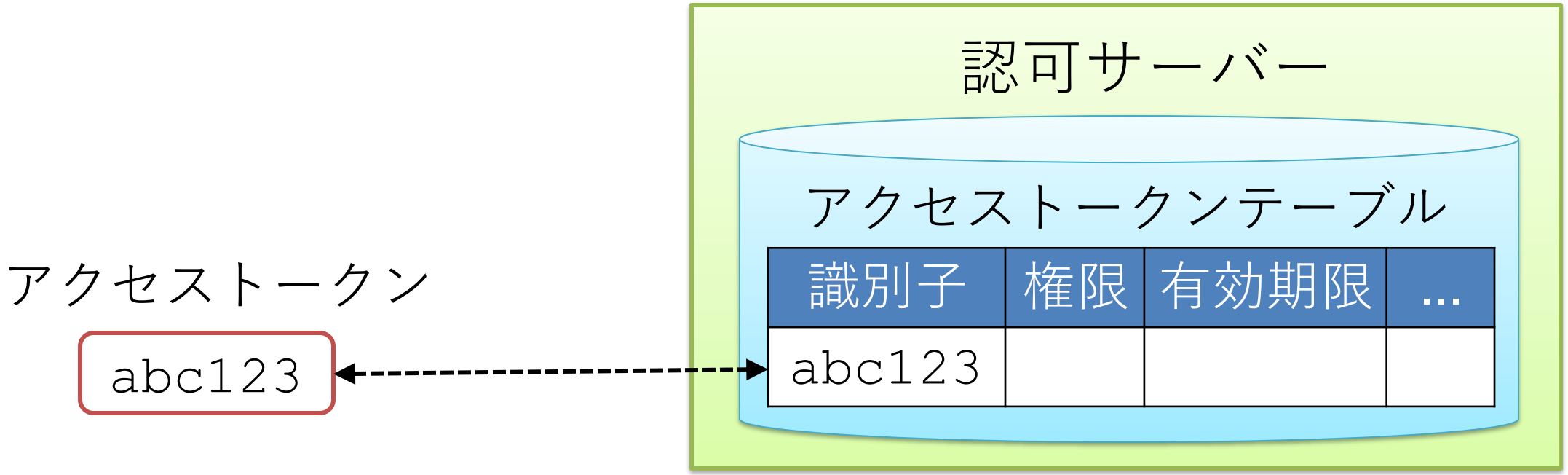
> *The token may denote an identifier used to retrieve the authorization information or may self-contain the authorization information in a verifiable manner (i.e., a token string consisting of some data and a signature). ...*

- ☑ アクセストークンの実装方法はおおむね『識別子型』と『内包型』の二つに大別することができる。
- ☑ これらを組み合わせる『ハイブリッド型』の実装もある。

識別子型

識別子型の実装では、アクセストークンに紐付く情報を認可サーバーのデータベース内に保存する。そして、そのデータレコードを一意に特定できる識別子をアクセストークンとする。

※ セキュリティのため、実際にはアクセストークンそのものではなくハッシュ値を保存することになる。



内包型

内包型の実装では、アクセストークンに紐付く情報をアクセストークン自体の中に埋め込む。

アクセストークン

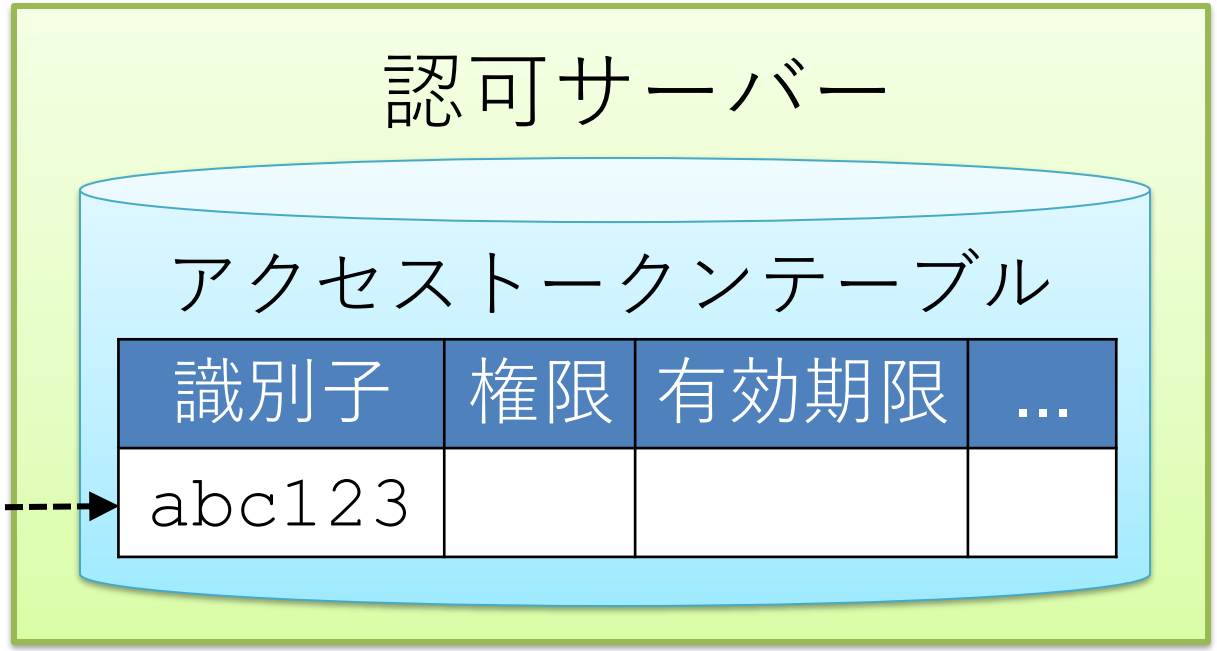
```
{
  "scope": "...",
  "client_id": "...",
  "exp": ...,
  "iat": ...,
  "sub": "...",
  "iss": "...",
  "jti": "..."
}
```

ハイブリッド型

ハイブリッド型の実装では、内包型アクセストークンを生成しつつも、
それに付随するデータを認可サーバーのデータベース内に持つ。

アクセストークン

```
{
  "scope": "...",
  "client_id": "...",
  "exp": ...,
  "iat": ...,
  "sub": "...",
  "iss": "...",
  "jti": "..."
}
```

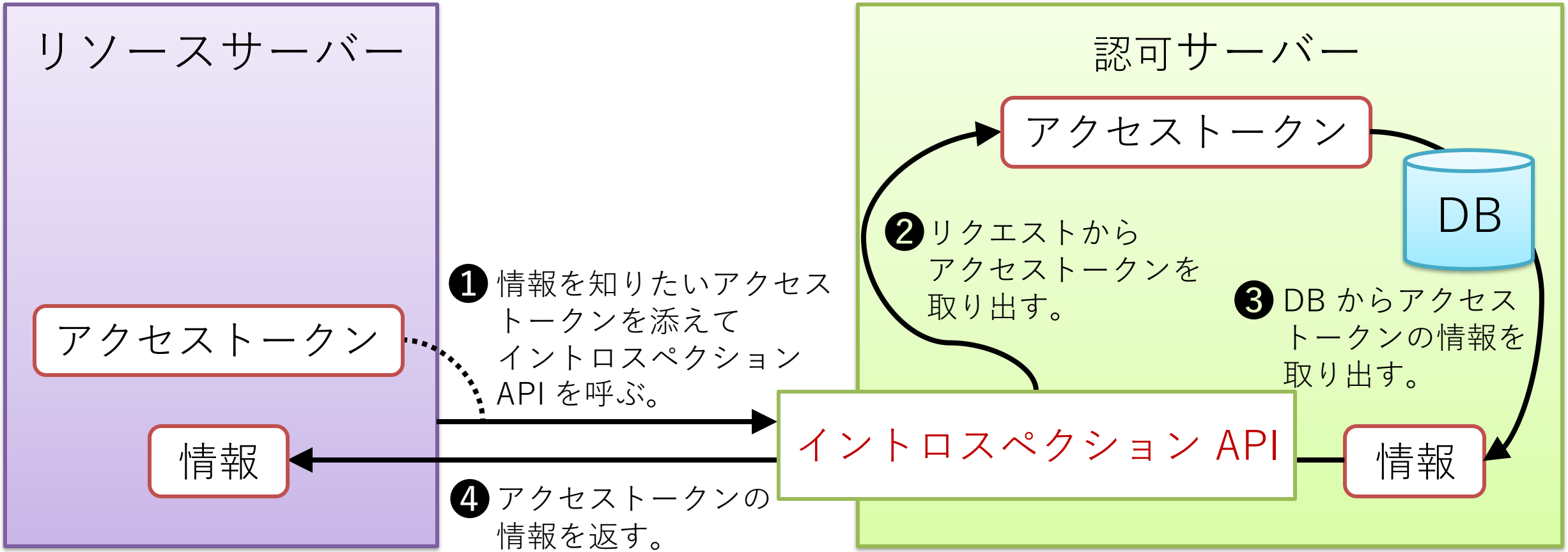


アクセストークン情報取得

識別子型アクセストークンの情報取得方法

認可サーバーが用意するイントロスペクション API に問い合わせる。

※ 認可サーバーとリソースサーバーが DB を共有しているシステムは、直接 DB を参照してアクセストークンの情報を得る。



RFC 7662 : OAuth 2.0 Token Introspection （イントロスペクション API の標準仕様）

イントロスペクションリクエスト（RFC 7662, Section 2.1）

HTTP メソッド	POST		
Content-Type	application/x-www-form-urlencoded		
パラメーター	token	必須	アクセストークンまたはリフレッシュトークン
	token_type_hint	任意	access_token または refresh_token

token_type_hint : *“The protected resource MAY pass this parameter to help the authorization server optimize the token lookup. If the server is unable to locate the token using the given hint, it MUST extend its search across all of its supported token types.”*

token_type_hint と実際に渡されたトークンの種類が異なっていても、エラーにならないばかりか、検索は続行される。

イントロスペクションレスポンス（RFC 7662, Section 2.2）

Status Code	200 OK		
Content-Type	application/json		
パラメーター	active	必須	true または false
	scope	任意	スコープ（RFC 6749, Section 3.3）
	client_id	任意	クライアントID
	username	任意	リソースオーナーの Human-Readable 識別子
	token_type	任意	トークンタイプ（RFC 6749, Section 5.1）
	exp	任意	有効期限終了時刻
	iat	任意	発行時刻
	nbf	任意	有効期限開始時刻
	sub	任意	主体識別子。通常はリソースオーナー識別子
	aud	任意	トークン利用者
	iss	任意	トークン発行者
	jti	任意	トークン識別子

RFC 7662, 2.1. Introspection Request より抜粋

To prevent token scanning attacks, the endpoint MUST also require some form of authorization to access this endpoint, such as client authentication as described in OAuth 2.0 [RFC6749] or a separate OAuth 2.0 access token such as the bearer token described in OAuth 2.0 Bearer Token Usage [RFC6750]. The methods of managing and validating these authentication credentials are out of scope of this specification.

- [x] イントロスペクションエンドポイントは、何らかの方法で保護しなければならない。ただし、その方法は仕様範囲外。

Authlete 社内の Slack

July 17th, 2017

taka 18:06

RFC 7662 (OAuth 2.0 Token Introspection) (written by @justin), 2.1. Introspection Request:
https://tools.ietf.org/html/rfc7662#section-2.1

> To prevent token scanning attacks, the endpoint MUST also require some form of authorization to access this endpoint, such as **client authentication** as described in OAuth 2.0 [RFC6749] or a separate OAuth 2.0 access token such as the bearer token described in OAuth 2.0 Bearer Token Usage [RFC6750]. The methods of managing and validating these authentication credentials are out of scope of this specification.

The examples in the paragraph excerpted above give an impression that expected API callers are legitimate **client applications** (not resource servers).

And, the following example and explanation:

> The following is a non-normative example request:
>
> POST /introspect HTTP/1.1
> Host: server.example.com
> Accept: application/json
> Content-Type: application/x-www-form-urlencoded
> Authorization: Bearer 23410913-abewfq.123483
>
> token=2YotnFZFEjr1zCsicMWpAA
>
> In this example, the protected resource **uses a client identifier and client secret** to authenticate itself to the introspection endpoint. The protected resource also sends a token type hint indicating that it is inquiring about an access token.

explicitly says that a pair of client identifier and client secret is used to call the introspection API.

（中略）

仕様書は、イントロスペクションエンドポイントの保護方法の例としてクライアント認証を挙げている。

えっ？ リソースサーバー認証じゃないの？

RFC 7662 策定者本人（Justin Richer）

justin 22:46

This is for use by resource servers but we didn't want to invent a brand new software authentication framework just for this purpose, so we allow re-use of the one that OAuth already defines.

That doesn't mean that it's a client, it just means that you have something that looks like a client ID and something that looks like a client secret

The client does **not** do the authentication, the resource server does.

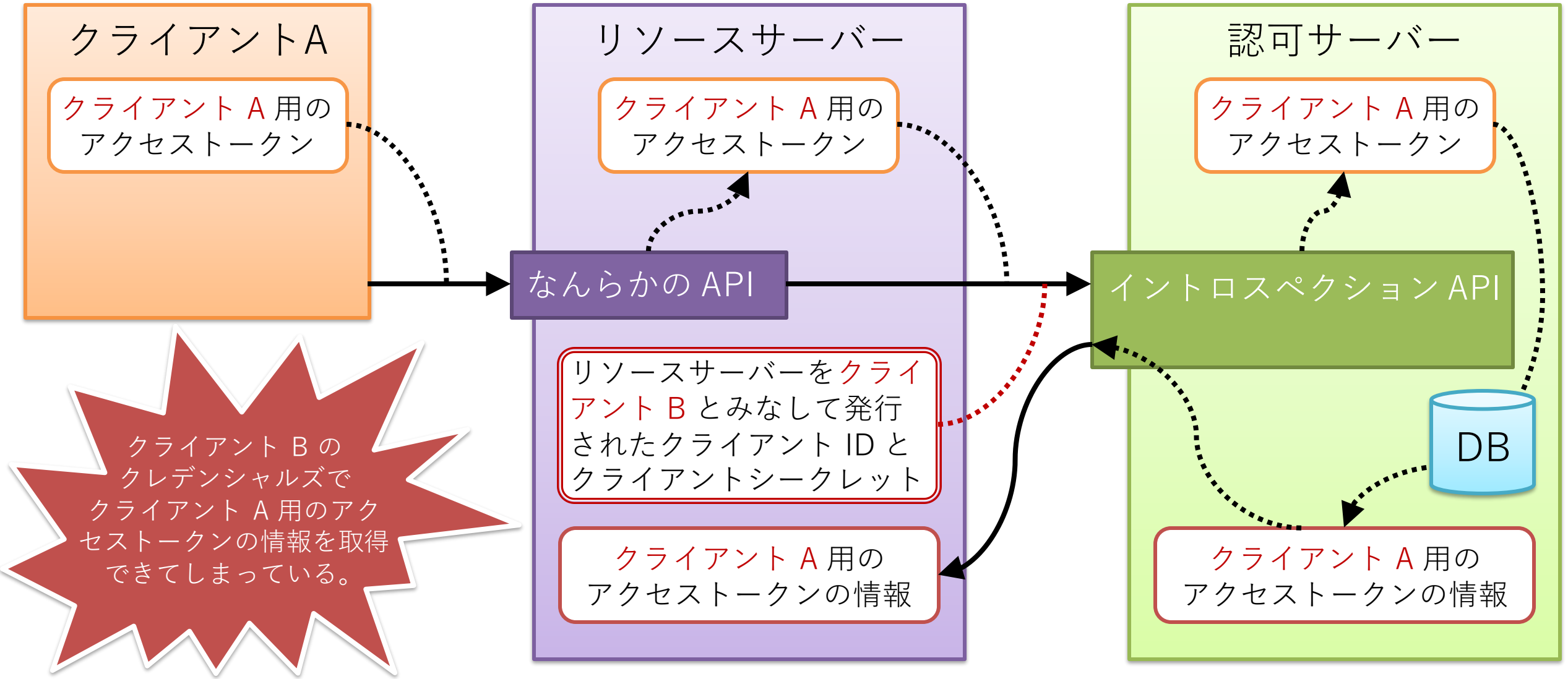
新たな認証フレームワークを発明したくはなかったので、既存のもの（クライアント認証）を流用することに。クライアントIDのように見える何かとクライアントシークレットのように見える何かでの認証だが、API Caller は OAuth クライアントではない。

RFC 8414 : OAuth 2.0 Authorization Server Metadata

`introspection_endpoint`
OPTIONAL. URL of the authorization server's OAuth 2.0 introspection endpoint [RFC7662].
`introspection_endpoint_auth_methods_supported`
OPTIONAL. JSON array containing a list of client authentication methods supported by this introspection endpoint. The valid client authentication method values are those registered in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] or those registered in the IANA "OAuth Access Token Types" registry [IANA.OAuth.Parameters]. (These values are and will remain distinct, due to Section 7.2.) If omitted, the set of supported authentication methods MUST be determined by other means.
`introspection_endpoint_auth_signing_alg_values_supported`
OPTIONAL. JSON array containing a list of the JWS signing algorithms ("alg" values) supported by the introspection endpoint for the signature on the JWT [JWT] used to authenticate the client at the introspection endpoint for the "private_key_jwt" and "client_secret_jwt" authentication methods. This metadata entry MUST be present if either of these authentication methods are specified in the "introspection_endpoint_auth_methods_supported" entry. No default algorithms are implied if this entry is omitted. The value "none" MUST NOT be used.

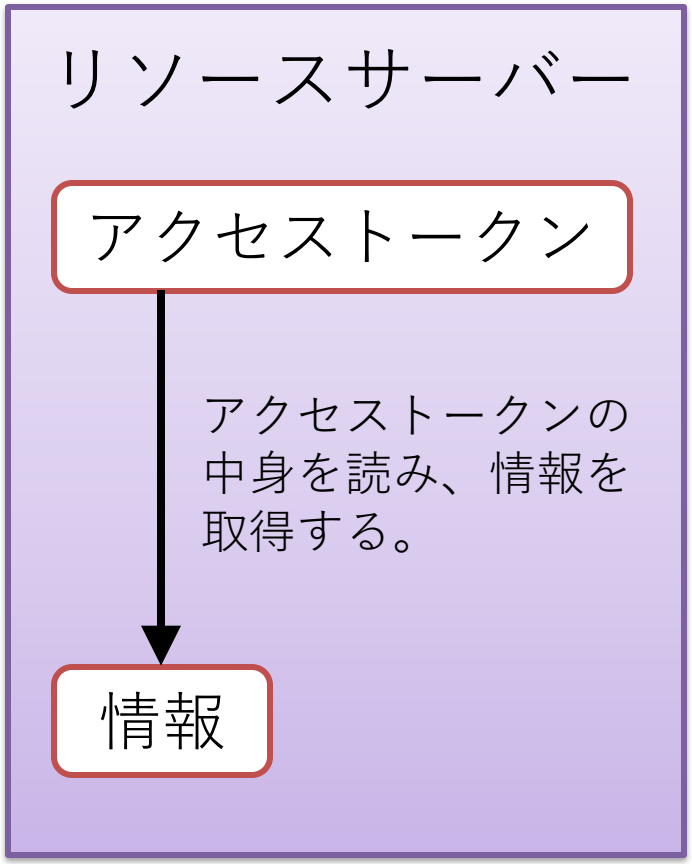
イントロスペクションエンドポイントにおける認証方法に縛りはないはずだが、
RFC 8414 のメタデータ定義はクライアント認証の流用を前提としてしまっている。

こんな状態なので、既存のクライアント管理テーブルを流用して、そこにリソースサーバー用のレコードを追加する認可サーバーの実装が出てくる。（どの実装とは言わないが…）



内包型アクセストークンの情報取得方法

アクセストークンの中身を読む。



- ✅ イントロスペクション API を呼ばずに済むので、認可サーバーとの通信が発生しない。
  - → 内包型アクセストークンの大きな利点！内包型の勝利！
  - → ...だと盲信していられるのも始めのうちだけだ！
- ✅ 内包型アクセストークンを失効させたい場合、PKI 証明書の CRL や OCSP に相当する仕組みが必要で、結局通信が発生する。（後述）
- ✅ 内包型アクセストークンに必須となる署名（後述）、場合によっては暗号化（後述）。それらの鍵の有効性確認のために結局通信が発生。

「アクセストークンの有効期間を短くして失効を諦める」という妥協をしない限り、内包型の利点はほとんどない。

内包型アクセストークンの検証

内包型アクセストークンは、暗号化されていない限りその内容は誰でも読める。そのため、何らかの工夫をしなければ簡単に偽造されてしまう。

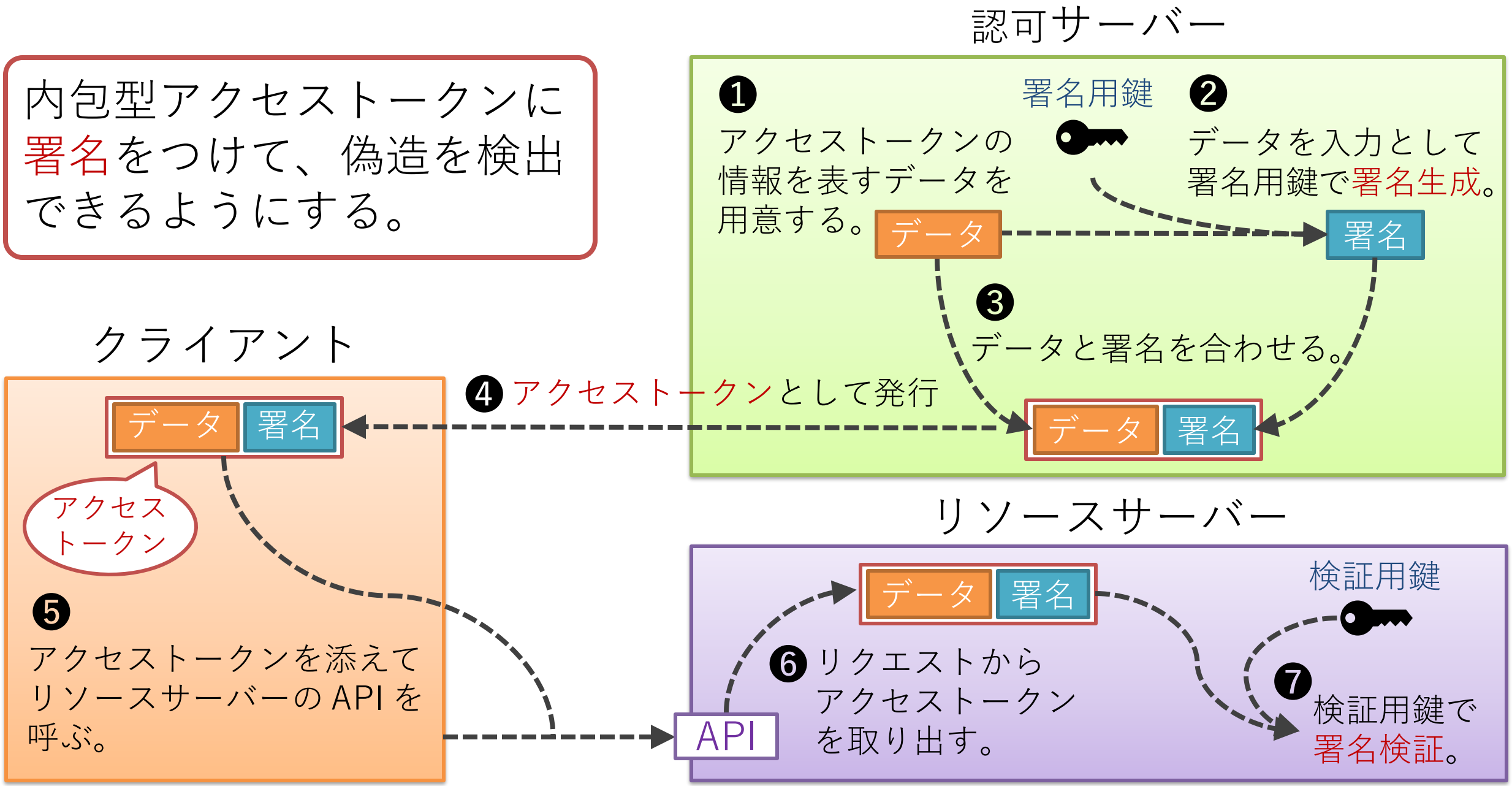
リソースサーバーは、受け取ったアクセストークンに埋め込まれている情報を鵜呑みにする前に、何かしらの方法で、そのアクセストークンが偽造されたものではないことを確認しなければならない。

RFC 6749, 1.4. Access Token より抜粋（再掲）

> *The token may denote an identifier used to retrieve the authorization information or may self-contain the authorization information* ***in a verifiable manner*** *(i.e., a token string consisting of some data and a signature). ...*

→ 「**検証可能な方法で**認可情報を内包」

内包型アクセストークンに署名をつけて、偽造を検出できるようにする。



署名付きデータの汎用形式として、RFC 7519 で規定される JWT の使い勝手が良いため、内包型アクセストークンのフォーマットとして JWT が選ばれることが多い。実際に、「アクセストークンの形式が JWT の場合」という規定を持つ仕様も存在する。

例：RFC 8705, 3.1. JWT Certificate Thumbprint Confirmation Method

> *When access tokens are represented as JSON Web Tokens (JWT) [RFC7519], the certificate hash information SHOULD be represented using the "x5t#S256" confirmation method member defined herein.*

JWT 形式のアクセストークンを標準化しようという動きもある。

JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens

https://datatracker.ietf.org/doc/draft-ietf-oauth-access-token-jwt/

JWT 型アクセストークンの考慮事項

署名アルゴリズムの選択肢は RFC 7518 : JSON Web Algorithms (JWA) の『3.1. "alg" (Algorithm) Header Parameter Values for JWS』に列挙されているものになる。ただし『署名無し』を意味する `none` は選択肢から外れる。

`HS256`	HMAC using SHA-256
`HS384`	HMAC using SHA-384
`HS512`	HMAC using SHA-512
`RS256`	RSASSA-PKCS1-v1_5 using SHA-256
`RS384`	RSASSA-PKCS1-v1_5 using SHA-384
`RS512`	RSASSA-PKCS1-v1_5 using SHA-512
`ES256`	ECDSA using P-256 and SHA-256
`ES384`	ECDSA using P-384 and SHA-384
`ES512`	ECDSA using P-521 and SHA-512
`PS256`	RSASSA-PSS using SHA-256 and MGF1 with SHA-256
`PS384`	RSASSA-PSS using SHA-384 and MGF1 with SHA-384
`PS512`	RSASSA-PSS using SHA-512 and MGF1 with SHA-512

対称鍵系署名アルゴリズム

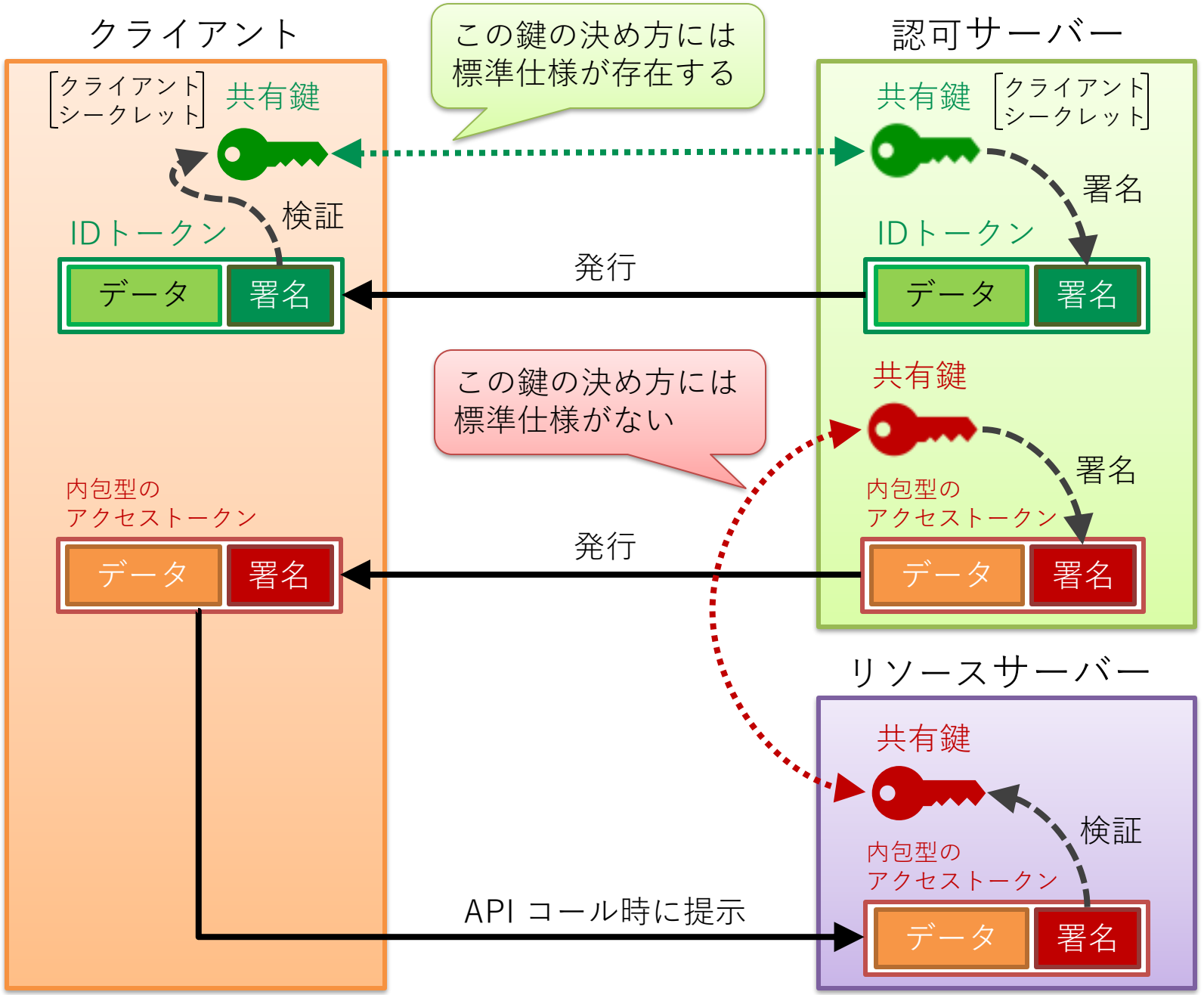
署名アルゴリズムが対称鍵系の場合、認可サーバーとリソースサーバーが同じ鍵を共有する必要がある。

但し、共有鍵 の決め方を定める標準仕様は存在しない。

【参考】

『OpenID Connect Core 1.0』の『10.1. Signing』には「IDトークンやリクエストオブジェクトなどの署名に対称鍵系アルゴリズムを用いる場合、クライアントシークレットを鍵 として用いること」という趣旨の規定がある。

※この規定は認可サーバーとクライアント間でしか成立しないので、認可サーバーとリソースサーバーの間で共有する鍵の決め方には流用できない。



非対称鍵系署名アルゴリズム

署名アルゴリズムが非対称鍵系（公開鍵系）の場合、

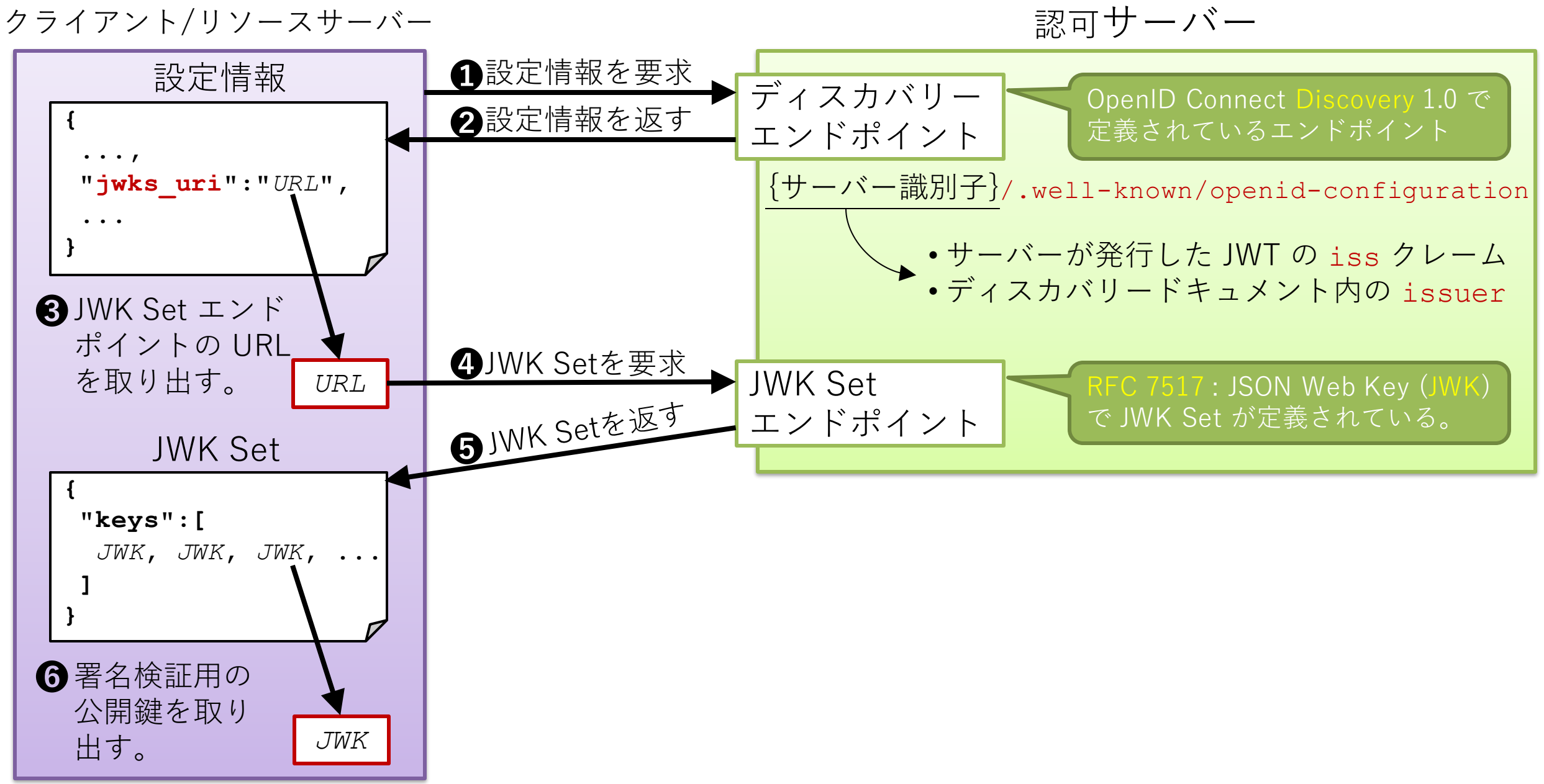
- ☑ 認可サーバーは秘密鍵を用いてアクセストークンに署名する。
- ☑ リソースサーバーは公開鍵を用いて署名を検証する。

リソースサーバーは何らかの方法で認可サーバーの公開鍵を取得する必要がある。

認可サーバーが自身の JWK Set を公開するエンドポイントを提供し、その応答の中にアクセストークンの署名検証用公開鍵を含めればよい。

JWK Set エンドポイントの URL はディスカバリードキュメント内の `jwks_uri`。

ディスカバリードキュメントの場所は `.well-known/openid-configuration`。


クライアント/リソースサーバー
認可サーバー
設定情報
{
...,
"jwks_uri":"URL",
...
}
❶設定情報を要求
❷設定情報を返す
ディスカバリー
エンドポイント
OpenID Connect Discovery 1.0 で
定義されているエンドポイント
{サーバー識別子}/.well-known/openid-configuration
• サーバーが発行した JWT の iss クレーム
• ディスカバリードキュメント内の issuer
❸JWK Set エンド
ポイントの URL
を取り出す。
URL
❹JWK Setを要求
JWK Set
エンドポイント
RFC 7517 : JSON Web Key (JWK)
で JWK Set が定義されている。
❺JWK Setを返す
JWK Set
{
"keys":[
JWK, JWK, JWK, ...
]
}
❻署名検証用の
公開鍵を取り
出す。
JWK

非対称鍵系署名アルゴリズム選定時の注意点

RFC 7518 : JSON Web Algorithms (JWA), Section 3.1

HS256	Required	対称鍵系
RS256	Recommended	
ES256	Recommended+	+は「将来重要度が上がる可能性が高い」という印
他	Optional	

IDトークンやリクエストオブジェクトなどのJWTの署名アルゴリズムを RS256 決め打ちにしている実装はかなり多い。決め打ちにすることで、認可サーバーの実装が極端に楽になるから。

しかし、RSで始まるアルゴリズムの使用は避けたほうがよい。セキュリティー上の懸念があるため、Financial-grade API Part 2 ではRS系アルゴリズムを使用禁止としている。かわりにES256とPS256を推奨。

JWT 型アクセストークンの暗号化

対称鍵系暗号アルゴリズム

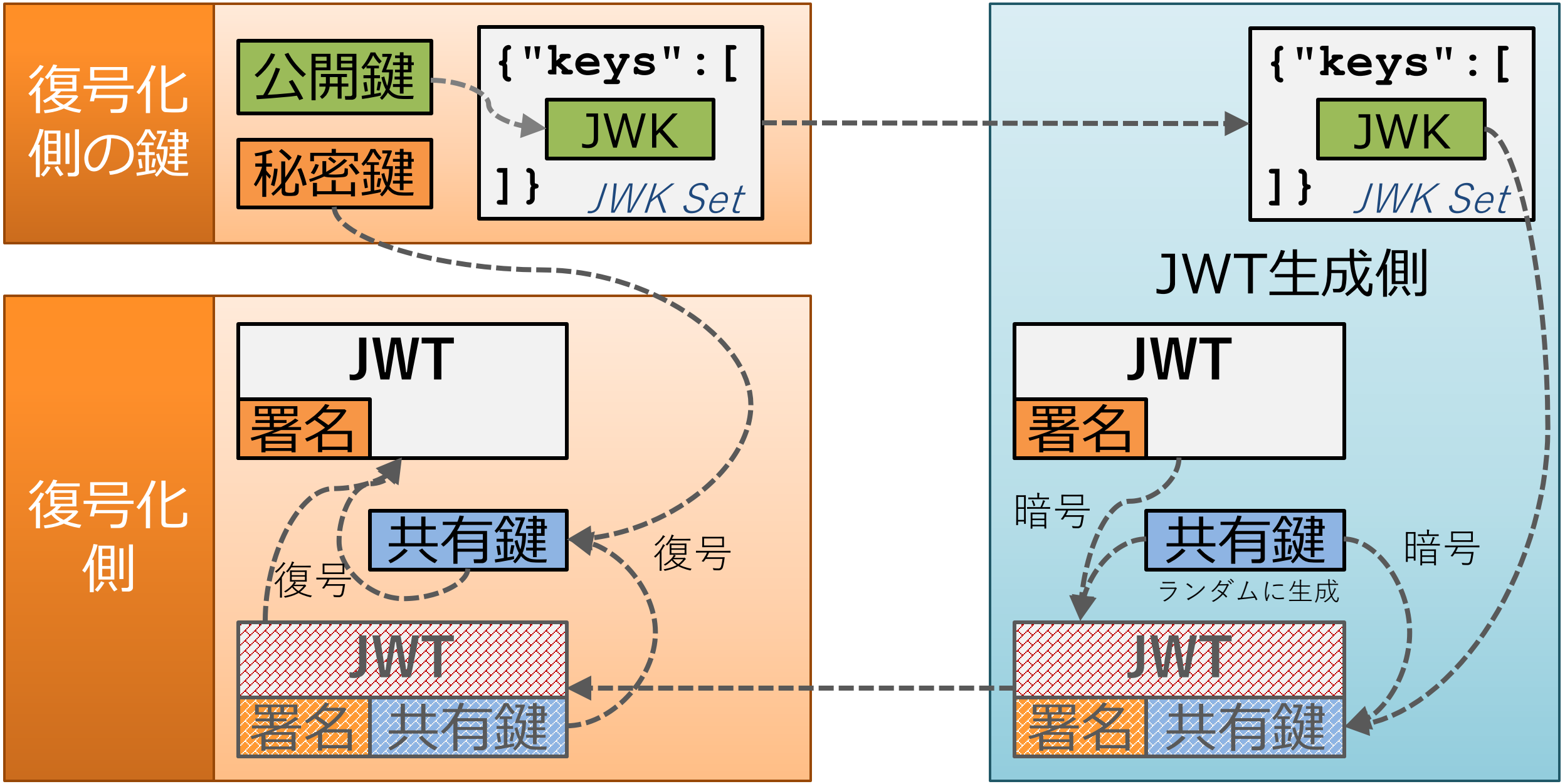
暗号アルゴリズムが対称鍵系の場合、認可サーバーとリソースサーバーが同じ鍵を共有する必要がある。しかし、共有鍵🔑の決め方を定める標準仕様は存在しない。

非対称鍵系暗号アルゴリズム

暗号の場合、署名とは逆で、公開鍵を持つのは認可サーバー、秘密鍵を持つのはリソースサーバー。

認可サーバーは何らかの方法でリソースサーバーの公開鍵を取得する必要がある。

リソースサーバーのメタデータを定める仕様（OAuth 2.0 Protected Resource Metadata）にメタデータの一つとして `jwks_uri` が含まれていたが、当仕様のドラフトの最終更新日は 2017 年 1 月 19 日で、ドラフト自体既に無効になっている。


復号化側の鍵
公開鍵
秘密鍵
{"keys":[
JWK
]}
JWK Set
{"keys":[
JWK
]}
JWK Set
JWT生成側
復号化側
JWT
署名
共有鍵
復号
復号
JWT
署名
共有鍵
暗号
暗号
ランダムに生成
JWT
署名
共有鍵
JWT
署名
共有鍵

クレーム名

スコープ、有効期限、クライアント ID などのアクセストークンの情報を、
どのようなクレーム名でペイロード部に格納すべきか？

- ☑ RFC 7662 や RFC 7519 のクレーム名・型に倣うのが無難
- ☑ クレーム名の標準化を目指す動きもある

  JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens
  https://datatracker.ietf.org/doc/draft-ietf-oauth-access-token-jwt/

仕様によっては、イントロスペクションレスポンス内のプロパティーや
JWT アクセストークンのクレームを定めている。

`aud`	RFC 8707 : Resource Indicators for OAuth 2.0
`cnf / x5t#S256`	RFC 8705 : OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens
`cnf / jkt`	OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer (DPoP)
`authorization_details`	OAuth 2.0 Rich Authorization Requests (RAR)

アクセストークン失効


OAuth 2.0
OpenID Connect
Authorization Focused • Reliable and Scalable • Developer Friendly
Faster Time to Market • Choice of Hosting Options • Broad Usage
Integrates with any Authentication methods
API Security
OpenID
AUTHLETE

識別子型アクセストークンの失効　→ データベースからレコードを削除するだけ
内包型アクセストークンの失効　　→ PKI の CRL や OCSP 相当の仕組みを運用

CRL (Certificate Revocation List)

期限切れ前に失効した証明書群のシリアル番号一覧。
難点：リストが肥大化しうる。リスト同期の方法とタイミング。

OCSP (Online Certificate Status Protocol)

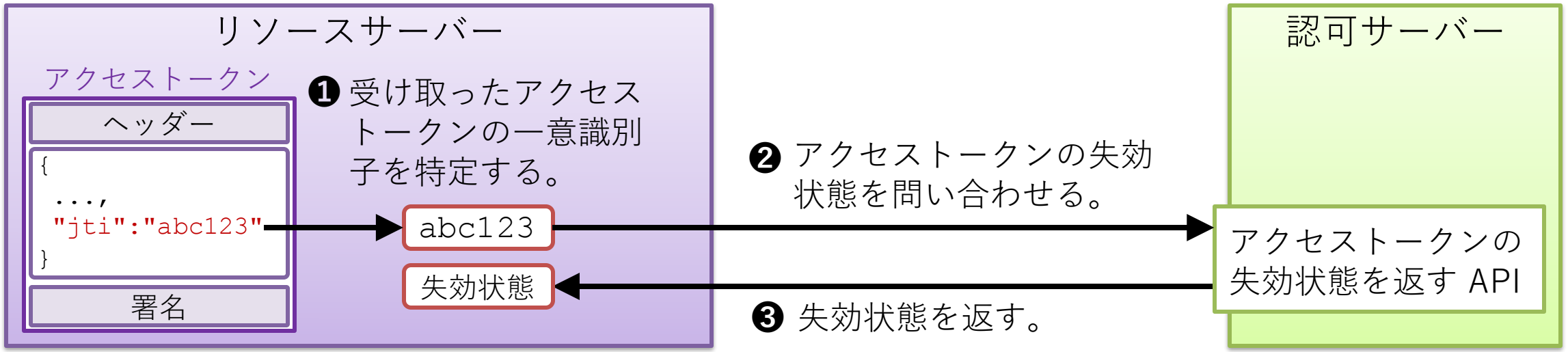
単一の証明書の失効状態をリアルタイムに問い合わせる仕組み。
難点：通信発生によるパフォーマンス問題。新たな単一障害点。

① CRL や OCSP 相当の仕組みを構築するためには、アクセストークンを一意に識別できなければならない。→ JWT の `jti` クレームを使えば実現可能。

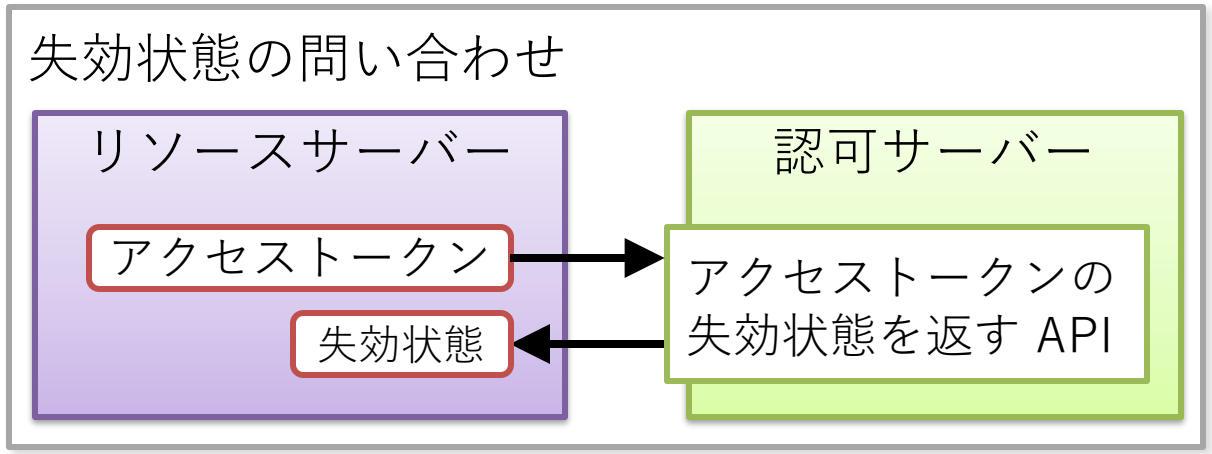
② アクセストークンを失効させるたびに、一意識別子を『失効されたアクセストークンのリスト』に追加する。当該アクセストークンの元々の有効期限が切れるまでは識別子をリスト内に保持しておく。

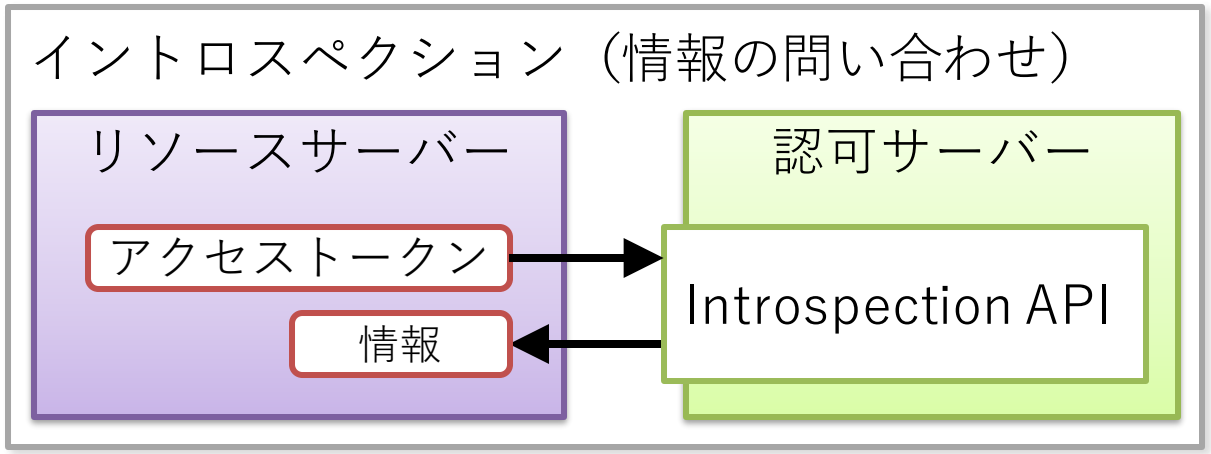
③ リソースサーバーは、受け取ったアクセストークンの失効状態を確認する。

CRL 相当	失効アクセストークンリストを取得し、当該一意識別子が含まれていないか確認
OCSP 相当	OCSP レスポンダに相当する『アクセストークン失効状態 API』に問い合わせ



失効状態の確認のために認可サーバーに問い合わせをすると、識別子型アクセストークンにおいてイントロスペクション API に問い合わせるのと同様に、ネットワーク通信が発生する。内包型アクセストークンの最大の利点が損なわれる。





というか、イントロスペクションレスポンス内の `active` に注目すれば、イントロスペクション API は『アクセストークンの失効情報を返す API』そのものであることが分かる。

よって、識別子型アクセストークンのほうが利点が多い（後述）ことを考慮すると、失効状態の問い合わせをしているようでは、内包型アクセストークンを積極的に採用する理由はほとんどない。

「アクセストークンの有効期間を短くして失効を諦める」という妥協をしない限り、内包型の利点はほとんどない。

これを差し置いても内包型を採用しなければならない理由があるとすれば、それは「何らかの事情によりリソースサーバーと認可サーバーが通信できない」場合のみ。

RFC 7009 : OAuth 2.0 Token Revocation

アクセストークン/リフレッシュトークンを
失効させるための API

リボケーションリクエスト（RFC 7009, Section 2.1. Revocation Request）

HTTP メソッド	POST		
Content-Type	application/x-www-form-urlencoded		
パラメーター	token	必須	アクセストークンまたはリフレッシュトークン
	token_type_hint	任意	access_token または refresh_token

RFC 7009, Section 2.1 に挙げられている例

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

token=45ghiukldjahdnhzdauz&token_type_hint=refresh_token
```

※ クライアント認証が必要

リボケーションレスポンス（RFC 7009, Section 2.2）

Status Code	200 OK	※ トークンが不正でもエラーにはならない。

JSONP サポート（RFC 7009, Section 2.3. Cross-Origin Support）

追加のリクエストパラメーター

パラメーター	callback	任意	JSONP 用 JavaScript 関数名

RFC 7009, Section 2.3 に挙げられているリクエストの例（注：GET メソッドを許容している）

```
https://example.com/revoke?token=4agabcdefddddafdd&
callback=package.myCallback
```

成功応答の例

```
package.myCallback();
```

エラー応答の例

```
package.myCallback({"error":"unsupported_token_type"});
```

Proof of Possession

電車の切符

盗んだ人が使えてしまう。

国際線の航空券

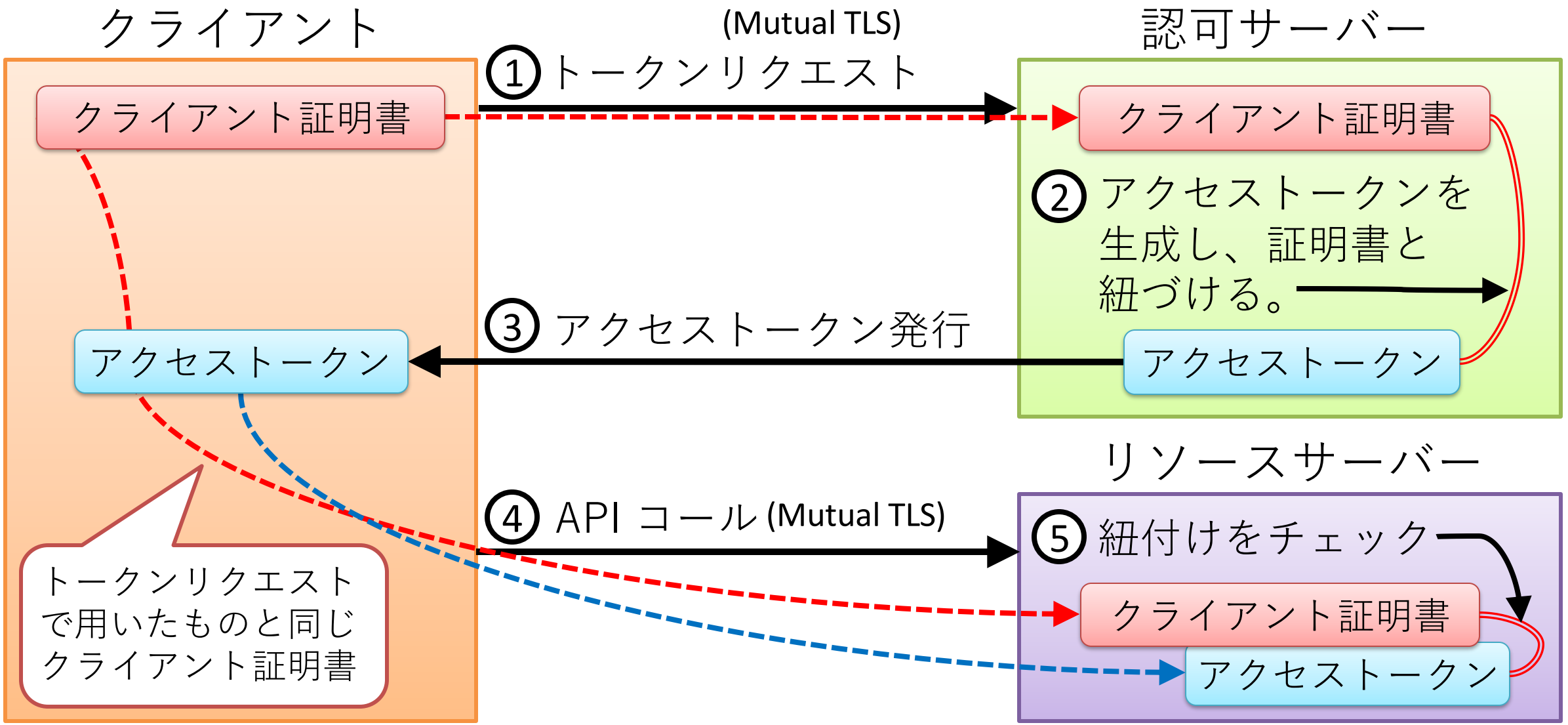
使用時にパスポートによる本人確認があるので、盗んでも使えない。

従来のアクセストークンは電車の切符のようなもの。使用時にアクセストークンの正当な所有者であることの証明『Proof of Possession』を要求すれば、アクセストークンの不正利用を減らせる。

Proof of Possession の手法を定める仕様

MTLS	RFC 8705 : OAuth 2.0 Mutual-TLS Client Authentication and Certificate Bound Access Tokens
DPoP	OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer (DPoP)

Certificate-Bound Access Token (RFC 8705, Section 3)



Q. アクセストークンとクライアント証明書の紐付け、具体的にはどうやるの？

A. クライアント証明書のハッシュ値をアクセストークンと紐付ける。

→ X.509 証明書の DER の SHA-256 ハッシュ

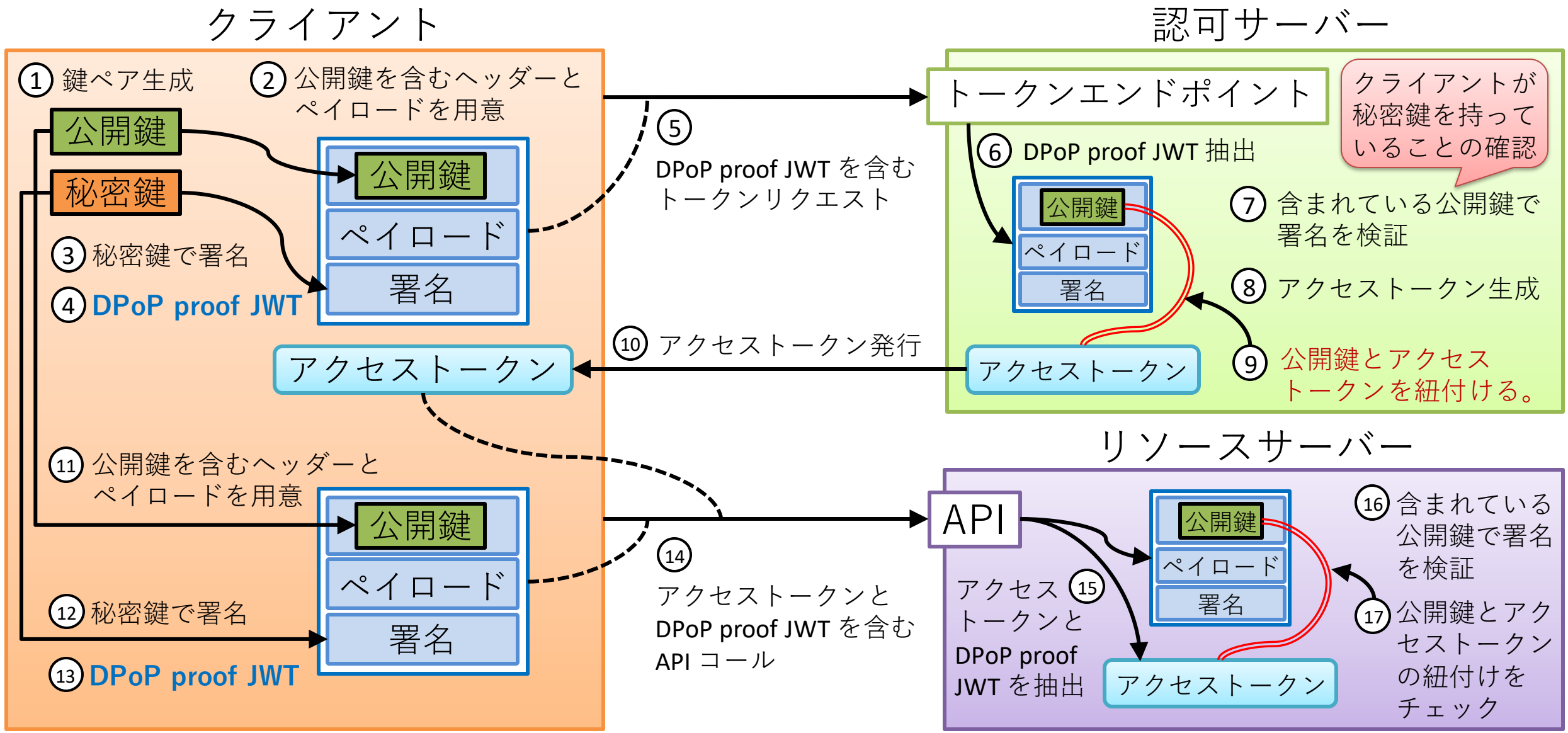
イントロスペクションレスポンスや JWT アクセストークンのペイロードは、このハッシュ値の BASE64URL を `cnf` クレーム下の `x5t#S256` の値として含む。

RFC 8705, Section 3.2 に挙げられているイントロスペクションレスポンスの例

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "active": true,
  "iss": "https://server.example.com",
  "sub": "ty.webb@example.com",
  "exp": 1493726400,
  "nbf": 1493722800,
  "cnf":{
    "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
  }
}
```

OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer (DPoP)



Q. アクセストークンと公開鍵の紐付け、具体的にはどうやるの？

A. 公開鍵のハッシュ値をアクセストークンと紐付ける。

→ JWK SHA-256 Thumbprint (RFC 7638)

イントロスペクションレスポンスや JWT アクセストークンのペイロードは、このハッシュ値の BASE64URL を `cnf` クレーム下の `jkt` の値として含む。

DPoP, Section 7 に挙げられている JWT アクセストークンのペイロードの例

```
{
  "sub":"someone@example.com",
  "iss":"https://server.example.com",
  "aud":"https://resource.example.org",
  "nbf":1562262611,
  "exp":1562266216,
  "cnf":{
      "jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I"
  }
}
```

DPoP proof JWT のヘッダーとペイロードの例（DPoP, Section 4）

```
{
  "typ":"dpop+jwt",   ← typ は固定文字列 "dpop+jwt"
  "alg":"ES256",      ← 署名アルゴリズムは非対称鍵系のみ
  "jwk": {            ↓ 公開鍵
    "kty":"EC",
    "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv":"P-256"
  }
}.{
  "jti":"-BwC3ESc6acc2lTc",
  "htm":"POST",                                  ← リクエストの HTTP メソッド
  "htu":"https://server.example.com/token",      ← リクエストの URL
  "iat":1562262616
}
```

DPoP proof JWT を伴うトークンリクエストの例（DPoP, Section 5）

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiItQndDM0VTYzZhY2MybFRjIiwiaHRtIj
 oiUE9TVCIsImh0dSI6Imh0dHBzOi8vc2VydmVyLmV4YW1wbGUuY29tL3Rva2VuIiwia
 WF0IjoxNTYyMjYyNjE2fQ.2-GxA6T8lP4vfrg8v-FdWP0A0zdrj8igiMLvqRMUvwnQg
 4PtFLbdLXiOSsX0x7NVY-FNyJK70nfbV37xRZT3Lg

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&code_verifier=bEaL42izcC-o-xBk0K2vuJ6U-y1p9r_wW2dFWIWgjz-
```

↑ DPoP proof JWT

※ 認可サーバーが DPoP をサポートしていれば、トークンレスポンスの `"token_type"` は `"DPoP"` になる。

DPoP proof JWT を伴うリソースアクセスの例（DPoP, Section 6）　（再掲）

```
GET /protectedresource HTTP/1.1
Host: resource.example.com
Authorization: DPoP eyJhbGciOiJFUzI1NiIsImtpZCI6IkJlQUxrYiJ9.eyJzdWI
 iOiJzb21lb25lQGV4YW1wbGUuY29tIiwiaXNzIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbX
 BsZS5jb20iLCJhdWQiOiJodHRwczovL3Jlc291cmNlLmV4YW1wbGUub3JnIiwibmJmI
 joxNTYyMjYyNjExLCJleHAiOjE1NjIyNjYyMTYsImNuZiI6eyJqa3QiOiIwWmNPQ09S
 Wk5ZeS1EV3BxcTMwalp5SkdIVE4wZDJIZ2xCVjN1aWd1QTRJIn19.vsFiVqHCyIkBYu
 50c69bmPJsj8qYlsXfuC6nZcLl8YYRNOhqMuRXu6oSZHe2dGZY0ODNaGg1cg-kVigzY
 hF1MQ
DPoP: eyJ0eXAiOiJkcG9wK2p3dCIsImFsZyI6IkVTMjU2IiwiandrIjp7Imt0eSI6Ik
 VDIiwieCI6Imw4dEZyaHgtMzR0VjNoUklDUkRZOXpDa0RscEJoRjQyVVFVZldWQVdCR
 nMiLCJ5IjoiOVZFNGpmX09rX282NHpiVFRsY3VOSmFqSG10NnY5VERWclUwQ2R2R1JE
 QSIsImNydiI6IlAtMjU2In19.eyJqdGkiOiJlMWozVl9iS2ljOC1MQUVCIiwiaHRtIj
 oiR0VUIiwiaHR1IjoiaHR0cHM6Ly9yZXNvdXJjZS5leGFtcGxlLm9yZy9wcm90ZWN0Z
 WRyZXNvdXJjZSIsImlhdCI6MTU2MjI2MjYxOH0.lNhmpAX1WwmpBvwhok4E74kWCiGB
 NdavjLAeevGy32H3dbF0Jbri69Nm2ukkwb-uyUI4AUg1JSskfWIyo4UCbQ
```

←アクセストークン

この例のアクセストークンはたまたま JWT になっているが DPoP-bound だからといって JWT である必要はない。

←DPoP proof JWT

Resource Indicators

RFC 8707 : Resource Indicators for OAuth 2.0

- ✅ アクセストークンの audience を指定できる
  - → イントロスペクションレスポンスや JWT アクセストークンのペイロードの `aud` の値となる。
    - ➢ 各 protected resource endpoint が、アクセストークンの `aud` 内に自身の URL を対象とする値が含まれていることを確認することが期待されている。

- ✅ `resource` リクエストパラメーター追加（複数回指定可能）
  - → アクセストークンの発行に関わるリクエストが影響を受ける。
    - ➢ 認可リクエスト
    - ➢ トークンリクエスト
    - ➢ バックチャネル認証リクエスト（CIBA）
    - ➢ デバイス認可リクエスト（RFC 8628）

⚠ なお、複数回指定可能というのは、RFC 6749 の *“Request and response parameters MUST NOT be included more than once”* という規定に明確に違反している。

→ しかし、既存実装への忖度や「当該規定は拡張仕様には適用されない」という超次元解釈により、そのまま RFC 化

resource を含む認可リクエストの例（RFC 8707, Section 2.1, Figure 2）

```
GET /as/authorization.oauth2?response_type=code
    &client_id=s6BhdRkqt3
    &state=tNwzQ87pC6llebpmac_IDeeq-mCR2wLDYljHUZUAWuI
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &scope=calendar%20contacts
    &resource=https%3A%2F%2Fcal.example.com%2F
    &resource=https%3A%2F%2Fcontacts.example.com%2F HTTP/1.1
Host: authorization-server.example.com
```

この認可リクエストにより、
右表のトークンが発行される。

トークン	対象リソース
認可コード	https://cal.example.com/ https://contacts.example.com/

resource を含むトークンリクエストの例（RFC 8707, Section 2.2, Figure 3）

```
POST /as/token.oauth2 HTTP/1.1
Host: authorization-server.example.com
Authorization: Basic czZCaGRSa3F0Mzpoc3FFelFsVW9IQUU5cHg0RlNyNHlJ
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
&code=10esc29BWC2qZB0acc9v8zAv9ltc2pko105tQauZ
&resource=https%3A%2F%2Fcal.example.com%2F
```

このトークンリクエストにより、右表のトークン群が発行される。

先に挙げた認可リクエストに対応するトークンリクエストだという前提

トークン	対象リソース
アクセストークン	https://cal.example.com/
リフレッシュトークン	https://cal.example.com/ https://contacts.example.com/

先のトークンリクエストで、例えば次のようなペイロードを持つ JWT アクセストークンが発行される。（仕様書内のアクセストークンをデコードするとこうなる↓）

```
{
  "iss": "https://authorization-server.example.com",
  "sub": "__b_c",
  "exp": 1588420826,
  "scope": "contacts",
  "aud": "https://contacts.example.com"
}
```

RFC 8707 の詳細解説はこちら↓

Resource Indicators for OAuth 2.0
https://qiita.com/TakahikoKawasaki/items/57d6242ff8b20dbff08f

Rich Authorization Requests

Rich Authorization Requests for OAuth 2.0 (RAR)

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

☑ アクセストークンに認可の詳細情報を紐付けることができる

背景：欧州金融業界の法的要件により付与権限詳細の同意取得が必須化。認可要求複雑化に対応するため、英国は認可要求詳細を表現・事前登録する方法（lodging intent パターン）を考案したが、汎用化・標準化のため、認可の詳細情報を表現する方法として RAR の策定作業が進められている。

☑ `authorization_details` パラメーター追加

→基本的に `scope` パラメーターと同じ場所で使える。

- 認可リクエスト / 認可レスポンス
- トークンリクエスト / トークンレスポンス
- バックチャネル認証リクエスト / クライアント通知（CIBA）
- デバイス認可リクエスト（RFC 8628）
- イントロスペクションレスポンス（RFC 7662）
- JWT アクセストークン

認可リクエストの例（RAR, Section 3.1）

```
GET /authorize?response_type=code
    &client_id=s6BhdRkqt3
    &state=af0ifjsldkj
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
    &code_challenge_method=S256
    &code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bwc-uCHaoeK1t8U
    &authorization_details=%5B%7B%22type%22%3A%22account%5Finformati
    on%22%2C%22actions%22%3A%5B%22list%5Faccounts%22%2C%22read%5Fbal
    ances%22%2C%22read%5Ftransactions%22%5D%2C%22locations%22%3A%5B%
    22https%3A%2F%2Fexample%2Ecom%2Faccounts%22%5D%7D%5D
    HTTP/1.1
Host: server.example.com
```

authorization_details の値を URL decode したもの。配列であることに注目。

```
[{"type":"account_information","actions":["list_accounts","read_balances"
,"read_transactions"],"locations":["https://example.com/accounts"]}]
```

トークンレスポンスの例（RAR, Section 3.4）

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
   "access_token": "2YotnFZFEjr1zCsicMWpAA",
   "token_type": "example",
   "expires_in": 3600,
   "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
   "authorization_details": [
     {
         "type": "https://www.someorg.com/payment_initiation",
         "actions": [
            "initiate",
            "status",
            "cancel"
         ],
         "locations": [
            "https://example.com/payments"
         ],
         "instructedAmount": {
            "currency": "EUR",
            "amount": "123.50"
         },
         "creditorName": "Merchant123",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   ]
}
```

イントロスペクションレスポンスの例（RAR, Section 3.6）

```
{
   "active": true,
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "exp": 1311281970,
   "acr": "psd2_sca",
   "txn": "8b4729cc-32e4-4370-8cf0-5796154d1296",
   "authorization_details": [
     {
         "type": "https://www.someorg.com/payment_initiation",
         "actions": [
            "initiate",
            "status",
            "cancel"
         ],
         "locations": [
            "https://example.com/payments"
         ],
         "instructedAmount": {
            "currency": "EUR",
            "amount": "123.50"
         },
         "creditorName": "Merchant123",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   ],
   "debtorAccount": {
      "iban": "DE40100100103307118608",
      "user_role": "owner"
   }
}
```

✅ リクエストで指定された authorization_details がそのままアクセストークンに紐付けられる。

⚠ ただし、resource パラメーター（RFC 8707）によるフィルタリングの仕組みがある。

- For all "authorization_details" w/o locations -> treat them as applicable to all resources, i.e. add them to the result set independent of the resource value
- For all "authorization_details" w/ locations -> add only those authorization_details objects with at least one location matching the resource value.

- locations プロパティーを含まない authorization_details 配列要素は、resource の値に関わらず採用
- locations プロパティーを含む authorization_details 配列要素は、一つ以上の location（locations 配列の要素）がいずれかの resource にマッチする場合のみ採用

仕様策定者と長々議論し、フィルタリングの動作詳細を詰め、仮想コードまで書いた。

[Issue 40] Authorization request with both `authorization_details` and `resources`
https://github.com/oauthstuff/draft-oauth-rar/issues/40

```
private static AuthzDetailsElement[] filterElementsByResources(
        AuthzDetailsElement[] elements, URI[] resources)
{
    // If the request does not include the 'resource' request parameters.
    if (resources == null || resources.length == 0)
    {
        // Filtering is not performed. All the elements included in the
        // 'authorization_details' array are used without being dropped.
        return elements;
    }

    // The resultant set of elements after filtering.
    List<AuthzDetailsElement> filteredElements
        = new ArrayList<AuthzDetailsElement>();

    // For each element listed in the 'authorization_details' array.
    for (AuthzDetailsElement element : elements)
    {
        // The "locations" property in the element.
        String[] locations = element.getLocations();

        // If the element does not have the "locations" property.
        if (locations == null)
        {
            // In this case, the element is selected unconditionally.
            filteredElements.add(element);
            continue;
        }

        // If the set of resources covers any one of the locations.
        if (doResourcesCoverAnyLocation(resources, locations))
        {
            // In this case, the element is selected.
            filteredElements.add(element);
            continue;
        }

        // The element has the "locations" property, but none of the
        // locations is not covered by the set of resources. In this
        // case, the element is dropped.
    }

    // Convert the list of filtered elements to an array.
    return filteredElements.toArray(
            new AuthzDetailsElement[filteredElements.size()]);
}


private static boolean doResourcesCoverAnyLocation(
        URI[] resources, String[] locations)
{
    // For each resource that corresponds to a 'resource' parameter.
    for (URI resource : resources)
    {
        // For each location in the "locations" property.
        for (String location : locations)
        {
            // If the resource covers the location.
            if (doesResourceCoverLocation(resource, location))
            {
                return true;
            }
        }
    }

    // None of the locations is covered by the set of resources.
    return false;
}


private static boolean doesResourceCoverLocation(URI resource, String location)
{
    // NOTE: The logic here is implementation-dependent
    // unless the specification defines rules.
}
```

RAR (Rich Authorization Requests for OAuth 2.0), Section 2.3. Relationship to "resource" parameter のロジックを表す仮想コード。Issue 40 内で提出したもの。`doesResourceCoverLocation()` のロジックについては、後日仕様策定者が提案するとのこと。

Pushed Authorization Requests

Pushed Authorization Requests for OAuth 2.0 (PAR)

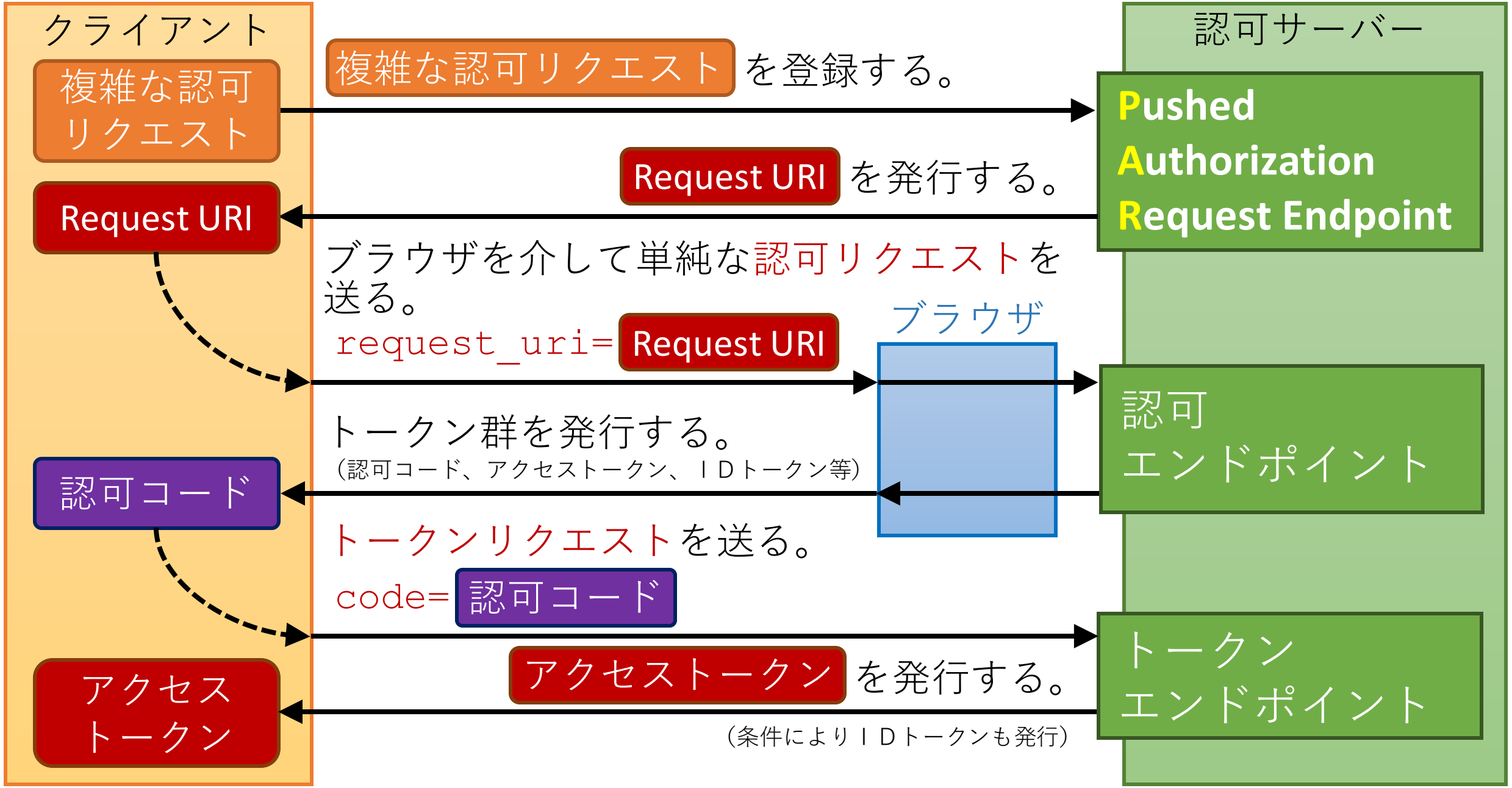
https://datatracker.ietf.org/doc/draft-ietf-oauth-par/

☑ 認可リクエストの詳細を事前に登録することができる

背景：欧州金融業界の法的要件により付与権限詳細の同意取得が必須化。認可要求複雑化に対応するため、英国は認可要求詳細を表現・事前登録する方法（lodging intent パターン）を考案したが、汎用化・標準化のため、認可要求を事前登録する方法として PAR の策定作業が進められている。

☑ Pushed Authorization Request エンドポイント追加

- → 認可リクエストを PAR エンドポイントに事前登録することにより、その認可リクエストを表す識別子『Request URI』の発行を受ける。
- → 認可リクエストの際は、発行された Request URI を `request_uri` パラメーターの値として指定する。


クライアント
認可サーバー
複雑な認可リクエスト
複雑な認可リクエスト を登録する。
Pushed Authorization Request Endpoint
Request URI を発行する。
Request URI
ブラウザを介して単純な認可リクエストを送る。
request_uri= Request URI
ブラウザ
認可エンドポイント
トークン群を発行する。
(認可コード、アクセストークン、ＩＤトークン等)
認可コード
トークンリクエストを送る。
code= 認可コード
アクセストークン を発行する。
アクセストークン
トークンエンドポイント
(条件によりＩＤトークンも発行)

PAR リクエストの例（PAR, Section 1）

```
POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3
                                                    ↑Client Type によってはクライアント認証が必要
response_type=code
&client_id=s6BhdRkqt3&state=af0ifjsldkj
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

PAR レスポンスの例（PAR, Section 1）

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

{                                                   ↓Request URI が発行されている
  "request_uri": "urn:example:bwc4JK-ESC0w8acc191e-Y1LTC2",
  "expires_in": 90
}
```

認可リクエストの例（PAR, Section 1）

```
GET /authorize?request_uri=                          ←Request URI を利用
  urn%3Aexample%3Abwc4JK-ESC0w8acc191e-Y1LTC2 HTTP/1.1
```

UserInfo


OAuth 2.0
OpenID Connect
Authorization Focused • Reliable and Scalable • Developer Friendly
Faster Time to Market • Choice of Hosting Options • Broad Usage
Integrates with any Authentication methods
API Security
OpenID
AUTHLETE

ユーザー情報エンドポイント（OpenID Connect Core 1.0, Section 5.3. UserInfo Endpoint）に、
openid をスコープとして持つアクセストークンを使ってアクセスすると、ユーザー情報が得られる。

ユーザー情報レスポンスの例（OIDC Core 1.0, Section 5.3.2.）

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "sub": "248289761001",
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "preferred_username": "j.doe",
  "email": "janedoe@example.com",
  "picture": "http://example.com/janedoe/me.jpg"
}
```

クライアントは、ユーザー情報レスポンスに含めてほしいクレーム群を、認可リクエスト時に要求することができる。注意：ユーザー情報リクエスト時ではない。

ユーザー情報レスポンスに含めてほしいクレーム群を指定する方法

① スコープを使う方法（OIDC Core 1.0, Section 5.4）

特別に事前定義されたスコープ名を scope リクエストパラメーターに含めると、対応するクレーム群を要求していることになる。例えばphone を scope に含めると、phone_number クレームと phone_number_verified クレームを要求する意となる。

スコープ名	対応するクレーム群
profile	name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, updated_at
email	email, email_verified
address	address
phone	phone_number, phone_number_verified

② claims リクエストパラメーターを使う方法（OIDC Core 1.0, Section 5.5）

claims リクエストパラメーターに渡す JSON のトップレベルプロパティーとして "userinfo" を含め、そこに規則に従ってクレーム群を並べる。カスタムクレームの要求や、"essential" / "value" / "values" の条件を付けたい場合、claims リクエストパラメーターを使うしかない。

OIDC Core 1.0, Section 5.5 に挙げられている claims リクエストパラメーターの値の例

```
{
 "userinfo":
  {
   "given_name": {"essential": true},
   "nickname": null,
   "email": {"essential": true},
   "email_verified": {"essential": true},
   "picture": null,
   "http://example.info/claims/groups": null
  },
 "id_token":
  {
   "auth_time": {"essential": true },
   "acr": {"values": ["urn:mace:incommon:iap:silver"] }
  }
}
```

クライアントは、これらのクレーム群をユーザー情報レスポンスに含めてほしいと要求している。

クライアントは、ユーザー情報レスポンスに含めてほしいクレーム群を、認可リクエスト時に要求することができる。注意：ユーザー情報リクエスト時ではない。（再掲）

→ということは、ユーザー情報エンドポイントの実装は、提示されたアクセストークンから「ユーザー情報レスポンスに含めるべきクレーム群に関する情報」を取得しなければならない。

→つまりは、「クレーム指定用スコープ群と claims JSON 内の "userinfo" オブジェクト」に相当する情報が、アクセストークンに紐付けて記憶されていなければならない。

例えば、JWT アクセストークンであれば、下記と同様の情報を何らかの表現形式で含む必要がある。

```
{
  "iss": "https://server.example.com",
  ...,
  "scope": "openid phone",
  "userinfo":{
    "given_name": {"essential": true},
    "nickname":  {"value": "Bob"}
  }
}
```

← 個人情報漏洩の可能性を示すため、claims リクエストパラメーター内に "value":"Bob" が含まれていたものと想定した例。

⚠これに相当する情報を含まない（ハイブリッド型ではない純粋な内容型の）JWT アクセストークンの実装は、ユーザー情報レスポンス内のクレームに関して、クライアントの要求に応えられない。

JWT 形式のアクセストークンのサポートを検討中に、「内包型アクセストークンはユーザー情報エンドポイントから返してほしいと要求されたクレーム群に関する情報を内容しなければならない」という結論に至ったあとの、Authlete 社内での Justin と Taka の会話を紹介

justin 04:10

@taka Your conclusion on JWt-based access token is incomplete. You do not need to put the user info into the access token, nor should you do so as it is potentially privacy-leaking. The userinfo endpoint will need to dereference the JWT to determine which user it applies to. This can be done with the `sub` claim. If the `claims` parameter is to be supported directly as you describe above, then that can be looked up underneath the `jti` claim which is transaction-specific as Joseph said. No JWT based system is fully self-contained, that I've seen. Any those that come closest use encrypted tokens (and the associated key management) to keep information relatively safe. (edited)

> @taka JWT アクセストークンに関する君の結論は完璧ではないよ。ユーザー情報をアクセストークンに含める必要はないし、潜在的な個人情報流出にもなるので、そうすべきでもない。ユーザー情報エンドポイントはどのユーザーを対象とするのか決めるために、JWT から参照を得る必要があるだろうね。これは `sub` クレームを使えばできる。君が書いたように `claims` パラメーターを直接的にサポートすべきだとしても、Joseph が言ったように、トランザクション固有の `jti` クレームに関連付けてその情報を検索することができる。JWT ベースのシステムで完全な内容型なんて無いんだよ、僕が見てきた限りではね。完全に近いであろうものは、情報を相対的に安全に保つために暗号化されたトークン（と関連する鍵管理）を使うよ。

taka 04:13

Privacy-leaking only if claims in `claims` contain `value` or `values`.

> 個人情報流出になるのは、`claims` が `value` や `values` を含んでいる場合のみかな。

justin 04:14

Not really — the fact that a field exists at all could be privacy leaking. This is less of a problem with common things like “address” and “family name” but more of an issue once you get to other resource types like medical and financial records.

そうでもないんだよ。フィールドが存在するというだけで個人情報流出になることがあるからね。これは `address` や `family_name` のような一般的なものでは問題にならないけど、医療記録や財務記録のような別のリソース型にアクセスできるとなると、途端に問題になる。

taka 04:16

"No JWT based system is fully self-contained" is an interesting statement.

「JWT ベースのシステムで完全な内包型なんて無いんだよ」というのは面白い文だね。

justin 04:16

You are right in the common case, but as Authlete is flexible enough we should consider other things

re: self-contained, what I mean by that is that inevitably the token will be used to look up something in a database someplace

一般的なケースについて言えば君の言っていることは正しいんだけど、Authlete は柔軟だから他のやり方を検討した方がいいだろうね。内包型については、言いたいのは、データベース内のどこかにある何かを検索するためにトークンを使うことは避けられないということだよ。

taka 04:17

Yes, I understand.

うん、わかってるよ。

justin 04:18

:nod: ok. There is a design pressure, that I've seen, to put as much information into the JWT itself so as to minimize lookup and network calls. This is a dangerous pattern with often unintended consequences in security and privacy because you have an all-powerful artifact that leaks everywhere it's used.

We saw this happen with SAML

Since OAuth is much more about API access we see it less, but it still can creep in

> OK。これまでにも見てきたけど、検索や通信を最小化するために JWT 自身にできるだけ情報を詰め込め、というプレッシャーが設計にはかかる。全ての使用箇所で流出していくことになる非常に強力な物を手にしてしまうため、この設計は、往々にしてセキュリティーやプライバシー上の意図しない問題を引き起こす危険なパターンなんだ。
>
> SAML でこの問題が起こるのを見てきたんだ。
>
> OAuth は API アクセスに関するものだから同じ問題は起こりにくいけど、それでも問題が紛れ込む可能性はある。

taka 04:21

So, Authlete can embed information equivalent to the userinfo property into a JWT-based access token but should dare not to do it. Thank you for your valuable comment.

> つまり、Authlete は userinfo プロパティー相当の情報を JWT アクセストークンに埋め込むこともできるけども、敢えてそうすべきではない、と。貴重なコメントありがとう。

justin 04:22

Yes, it would be technically feasible but I would not recommend it as either an implementation or a general pattern.

> そう、技術的に可能だけど、実装としても一般的なパターンとしてもお勧めしないよ。

元々識別子型アクセストークンをサポートしていた Authlete。バージョン 2.1 以降は、設定により JWT 型アクセストークンの発行も可能に。社内の議論を踏まえ、実装はどうなったのか？

設定	生成されるアクセストークン（例）
アクセストークン署名アルゴリズム 選択してください	xrvhSFnmE12pKz6Opu5gI7KkOAFUVuI8gjIZdHlfPVI
アクセストークン署名アルゴリズム ES256	eyJhbGciOiJFUzI1NiJ9.eyJzdWIiOiIxMDAxIiwic2NvcGUiOiJlbWFpbCBvcGVuaWQgcHJvZmlsZSIsImlzcyI6Imh0dHBzOi8vYXV0aGxldGUuY29tIiwiZXhwIjoxNTUzNDI3MjU1LCJpYXQiOjE1NTMzNDA4NTUsImNsaWVudF9pZCI6IjUwNjgxMTIxMjMiLCJqdGkiOiJ4cnZoU0ZubUUxMnBLejZPcHU1Z0k3S2tPQUZVVnVJOGdqSVpkSGxmUFZJIn0.bGKzVC9tVYN3H3hbnxmW6hIWKHrqXqgFz4kSDVHEGjQh_QRXvSFhBbFqwZR2W9T0ybdv-TE9lxWphRqUd92j7Q

ペイロード

```
{
  "sub": "1001",
  "scope": "email openid profile",
  "iss": "https://authlete.com",
  "exp": 1553427255,
  "iat": 1553340855,
  "client_id": "5068112123",
  "jti": "xrvhSFnmE12pKz6Opu5gI7KkOAFUVuI8gjIZdHlfPVI"
}
```

留意事項

1. 署名アルゴリズムは非対称鍵系のみ。対称鍵系のアルゴリズムの共有鍵決定方法の標準が存在しないため。
2. 暗号化はサポートしない。暗号鍵の扱いについて標準が存在しないため。
3. 認可リクエストで要求されたクレーム群に関する情報を JWT 形式アクセストークンに含めない。個人情報流出の恐れがあるため。実装をハイブリッド型にすることにより対応。
4. カスタムクレームの埋め込みをサポートする。

識別子型と内包型の比較

識別子型と内包型の比較

	識別子型	内包型
長さ	固定長	アクセストークンに紐付く情報の量による
情報の置き場所	認可サーバーの DB 内	アクセストークン内部（クライアントが保持）
情報取得方法	イントロスペクション	アクセストークンをデコード
有効性確認方法	イントロスペクション	署名検証
失効方法	データベースから削除	失効情報登録
失効状態確認方法	イントロスペクション	CRL や OCSP 相当の仕組みの運用
発行済アクセストークンの一括取得	認可サーバーへの命令	不可能
発行済アクセストークンの属性更新	認可サーバーへの命令	不可能
発行済アクセストークンの一括削除	認可サーバーへの命令	不可能
署名	不要	認可サーバーが検証用鍵を配布
暗号化	不要	リソースサーバーが暗号用鍵を配布
情報漏洩	無し	・暗号化しなければ全ての情報が漏洩 ・暗号化した上で鍵危殆化対策が必要
秘密情報との紐付け	認可サーバーの DB に保存	暗号化により対応

Thank You

お問い合わせ

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

一般	info@authlete.com
営業	sales@authlete.com
広報	pr@authlete.com
技術	support@authlete.com

@authlete_jp